

# User Manual

## DA16200 HTTP Extended APIs

### UM-WI-022

#### **Abstract**

*This document assists software developers with implementing applications for the DA16200 SDK. A certain degree of reader familiarity with programming environments, debugging tools, and software engineering process in general is assumed.*

---

---

DA16200 HTTP Extended APIs

**Contents**

**Abstract** ..... 1

**Contents** ..... 2

**Tables** ..... 2

**1 Terms and Definitions**..... 3

**2 References** ..... 3

**3 Overview**..... 4

**4 HTTP(s) API**..... 4

    4.1 API: HTTP Client ..... 4

    4.2 API: HTTP Server ..... 9

**Revision History** ..... 18

**Tables**

Table 1: Extended HTTP-Client API List ..... 4

Table 2: Extended HTTP-Server API list ..... 9

---

## DA16200 HTTP Extended APIs

### 1 Terms and Definitions

HTTP                    HyperText Transfer Protocol  
HTML                    HyperText Markup Language

### 2 References

- [1] DA16200, Datasheet, Dialog Semiconductor
- [2] DA16200, SDK Programmer Guide, Dialog Semiconductor
- [3] DA16200, Example Application Manual, Dialog Semiconductor
- [4] NetX Duo HTTP User Guide

## DA16200 HTTP Extended APIs

### 3 Overview

The DA16200 SDK offers a HTTP Server/Client module and works with two HTTP APIs:

- The original NetXDuo HTTP APIs
- The DA16200 HTTP APIs

This document describes how to use HTTP APIs on the DA16200 SDK.

<b>NOTE</b>
The HTTP Server/Client of the DA16200 SDK is based on the NetXDuo HTTP v5.10 module from ExpressLogic.
For details on how to use the NetXDuo HTTP APIs, refer to the Hypertext Transfer Protocol (NetX Duo HTTP) User's Guide (see [4]).

## 4 HTTP(s) API

### 4.1 API: HTTP Client

**Table 1: Extended HTTP-Client API List**

<pre>UINT nx_http_client_head_start(NX_HTTP_CLIENT *client_ptr,                                ULONG ip_address,                                CHAR *resource,                                CHAR *username,                                CHAR *password,                                ULONG wait_option)</pre>	
<b>Parameter</b>	<p>*client_ptr: Pointer to HTTP client</p> <p>*ip_address: HTTP Server IP address</p> <p>*resource: Pointer to resource (URL)</p> <p>*username: Pointer to username</p> <p>*password: Pointer to password</p> <p>*wait_option: Suspension option</p>
<b>Return</b>	Completion status
<b>Description</b>	This function processes the application HEAD request. The specified resource (URL) is requested from the HTTP Server at the specified IP address.

## DA16200 HTTP Extended APIs

<pre>UINT _nx_http_client_head_packet(NX_HTTP_CLIENT *client_ptr,                                 NX_PACKET **packet_ptr,                                 UINT *content_length,                                 UINT *resp_state,                                 CHAR *content_type,                                 UINT *content_type_len,                                 ULONG wait_option)</pre>	
<b>Parameter</b>	*client_ptr: Pointer to HTTP client *packet_ptr: Destination for packet pointer *content_length: <b>Content Length</b> field in response *content_type: <b>Content Type</b> field in response *content_type_len: Length of content_type *wait_option: Suspension option
<b>Return</b>	Completion status
<b>Description</b>	This function processes the application HEAD request. The specified resource (URL) is requested from the HTTP Server at the specified IP address.

<pre>UINT nx_http_client_set_secure_connection(NX_HTTP_CLIENT *client_ptr,  UINT is_secure)</pre>	
<b>Parameter</b>	*client_ptr: Pointer to HTTP client *is_secure: HTTPs or HTTP connection
<b>Return</b>	0 (NX_SUCCESS) on success
<b>Description</b>	This function sets HTTPs connection. Default connection is HTTP.

<pre>UINT nx_http_client_set_authmode(NX_HTTP_CLIENT *client_ptr,                                  UINT authmode)</pre>	
<b>Parameter</b>	*client_ptr: Pointer to HTTP client *authmode: MBEDTLS_SSL_VERIFY_NONE(0): Peer certificate is not checked MBEDTLS_SSL_VERIFY_REQUIRED(2): Peer presents a valid certificate, handshake is aborted if verification failed
<b>Return</b>	0 (NX_SUCCESS) on success
<b>Description</b>	This function sets the certificate verification mode in HTTPs connection. Default mode is None.

<pre>UINT nx_http_client_set_content_heap(NX_HTTP_CLIENT *client_ptr,                                      UINT enabled)</pre>	
<b>Parameter</b>	*client_ptr: Pointer to HTTP client *enabled: Value to use HEAP to allocate TLS buffer in HTTPs connection
<b>Return</b>	0 (NX_SUCCESS) on success
<b>Description</b>	This function allows to use heap to allocate TLS buffer in HTTPs connection. Default memory area is sec_pool in DA16200 system.

## DA16200 HTTP Extended APIs

<pre>UINT nx_http_client_set_content_buflen(NX_HTTP_CLIENT *client_ptr,  size_t in,  size_t out)</pre>	
<b>Parameter</b>	*client_ptr: Pointer to HTTP client *in: Length of incoming TLS buffer in HTTPs connection *out: Length of outgoing TLS buffer in HTTPs connection
<b>Return</b>	0 (NX_SUCCESS) on success
<b>Description</b>	This function sets lengths of TLS buffer in HTTPs connection. Default TLS buffer's length is 4096(bytes).

<pre>UINT nx_http_client_set_ca_cert(NX_HTTP_CLIENT *client_ptr,                                 char *ca_cert,                                 size_t len)</pre>	
<b>Parameter</b>	*client_ptr: Pointer to HTTP client *ca_cert: Pointer to CA certificate *len: Length of CA certificate
<b>Return</b>	0 (NX_SUCCESS) on success
<b>Description</b>	This function sets CA certificate of HTTPs connection.

<pre>UINT nx_http_client_set_cert(NX_HTTP_CLIENT *client_ptr,                               char *cert,                               size_t len)</pre>	
<b>Parameter</b>	*client_ptr: Pointer to HTTP client *cert: Pointer to certificate *len: Length of certificate
<b>Return</b>	0 (NX_SUCCESS) on success
<b>Description</b>	This function sets certificate of HTTPs connection.

<pre>UINT nx_http_client_set_private_key(NX_HTTP_CLIENT *client_ptr,                                     char *private_key,                                     size_t len)</pre>	
<b>Parameter</b>	*client_ptr: Pointer to HTTP client *cert: Pointer to certificate *len: Length of certificate
<b>Return</b>	0 (NX_SUCCESS) on success
<b>Description</b>	This function sets certificate of HTTPs connection.

## DA16200 HTTP Extended APIs

<pre>UINT nx_http_client_set_host_domain(NX_HTTP_CLIENT *client_ptr,                                    const CHAR *domain,                                    UINT domain_len)</pre>	
<b>Parameter</b>	*client_ptr: Pointer to HTTP client *domain: Pointer to host domain name *domain_len: Length of host domain name
<b>Return</b>	0 (NX_SUCCESS) on success
<b>Description</b>	This function sets specific host domain of <b>Host</b> field in HTTP request. If user does not set up the <b>Host</b> field, IP address is included in HTTP request.

<pre>UINT nx_http_client_set_range(NX_HTTP_CLIENT *client_ptr,                               ULONG start,                               ULONG end)</pre>	
<b>Parameter</b>	*client_ptr: Pointer to HTTP client *start: Start of range *end: End of range
<b>Return</b>	0 (NX_SUCCESS) on success
<b>Description</b>	This function sets specific range of <b>Range</b> field in HTTP request. If user does not set up the <b>Range</b> field, this field does not exist in HTTP request.

<pre>UINT nx_http_client_set_keep_alive(NX_HTTP_CLIENT *client_ptr,                                    UINT enable)</pre>	
<b>Parameter</b>	*client_ptr: Pointer to HTTP client *enable: Value to set <b>Keep-Alive</b> of the <b>Connection</b> field
<b>Return</b>	0 (NX_SUCCESS) on success
<b>Description</b>	This function sets specific range for the <b>Range</b> field in HTTP request. If user does not set up the <b>Range</b> field, this field does not exist in HTTP request.

<pre>UINT nx_http_client_get_response_header(NX_HTTP_CLIENT *client_ptr,   NX_PACKET **header_packet_ptr,   UINT *content_length,   UINT *resp_state,   CHAR *content_type,   UINT *content_type_len)</pre>	
<b>Parameter</b>	*client_ptr: Pointer to HTTP client *header_packet_ptr: Destination for packet pointer *content_length: Content Length in response *resp_state: Response State in response *content_type: Content type in response *content_type_len: Length of content type in response
<b>Return</b>	Completion status
<b>Description</b>	This function returns HTTP header in a response packet. It can be called when application wants to get header part in a response.

DA16200 HTTP Extended APIs

<pre>UINT nx_http_client_add_header_field_1(NX_HTTP_CLIENT *client_ptr,  const CHAR *field,  UINT field_len)</pre>	
<b>Parameter</b>	*client_ptr: Pointer to HTTP client *field: String of field value to add to header field_len: String length of field value
<b>Return</b>	Completion status
<b>Description</b>	This function allows user to add a field to the header.

<pre>UINT nx_http_client_add_header_field_2(NX_HTTP_CLIENT *client_ptr,  const CHAR *field,  UINT field_len)</pre>	
<b>Parameter</b>	*client_ptr: Pointer to HTTP client *field: String of field value to add to header field_len: String length of field value
<b>Return</b>	Completion status
<b>Description</b>	This function allows user to add a field to the header.

<pre>VOID nx_http_client_set_dump_log(int debug_mode)</pre>	
<b>Parameter</b>	debug_mode: 1(enable), 0(disable)
<b>Return</b>	No return value.
<b>Description</b>	This function is used only for debugging. Setting the true value prints the dump log of request and response.



DA16200 HTTP Extended APIs

4.2 API: HTTP Server

Table 2: Extended HTTP-Server API list

<pre>UINT nx_http_server_callback_data_send(NX_HTTP_SERVER *server_ptr,  UINT logical_connection,  VOID *data_ptr, ULONG data_length)</pre>	
<b>Parameter</b>	<p>*server_ptr: Pointer to HTTP Server control block</p> <p>*logical_connection: The internal logical connection for the HTTP Server. This can be used by the application as an index into buffers and/or data structures specific for each Client connection. Its value ranges from 0 through NX_HTTP_SERVER_MAX_CLIENT - 1</p> <p>*data_ptr: Pointer to the data to send</p> <p>*data_length: Number of bytes to send</p>
<b>Return</b>	<p>*NX_SUCCESS (0x00): Successfully sent HTTP Server data</p> <p>*NX_HTTP_ERROR (0xE0): HTTP Server internal error</p>
<b>Description</b>	<p>This service sends the data in the supplied packet from the application's callback routine. This is typically used to send dynamic data associated with GET/POST requests. Note that if this function is used, the callback routine is responsible for sending the entire response in the proper format. In addition, the callback routine must return the status of NX_HTTP_CALLBACK_COMPLETED.</p>

<pre>UINT nx_http_server_callback_generate_response_header(     NX_HTTP_SERVER *server_ptr,     UINT logical_connection,     NX_PACKET **packet_pptr,     CHAR *status_code,     UINT content_length,     CHAR *content_type,     CHAR* additional_header)</pre>	
<b>Parameter</b>	<p>*server_ptr: Pointer to HTTP Server control block</p> <p>*logical_connection: The internal logical connection for the HTTP Server. This can be used by the application as an index into buffers and/or data structures specific for each Client connection. Its value ranges from 0 through NX_HTTP_SERVER_MAX_CLIENT - 1</p> <p>*packet_pptr: Pointer a packet pointer allocated for message</p> <p>*status_code: Indicate status of resource</p> <p>*content_length: Size of content in bytes</p> <p>*content_type: Type of HTTP e.g. "text/plain"</p> <p>*additional_header: Pointer to additional header text</p>
<b>Return</b>	<p>*NX_SUCCESS (0x00): Successfully created HTML header</p> <p>*status: Completion status of internal NetX calls (e.g. nx_packet_allocate)</p>
<b>Description</b>	<p>This service calls the internal function nx_http_server_generate_response_header when the HTTP server responds to Client get, put, and delete requests. It should be used in HTTP server callback functions when the HTTP server application is designing its response to the Client.</p>

DA16200 HTTP Extended APIs

<pre>UINT nx_http_server_callback_packet_send(NX_HTTP_SERVER *server_ptr,  UINT logical_connection,  NX_PACKET *packet_ptr)</pre>	
<b>Parameter</b>	<p>*server_ptr: Pointer to HTTP Server control block</p> <p>*logical_connection: The internal logical connection for the HTTP Server. This can be used by the application as an index into buffers and/or data structures specific for each Client connection. Its value ranges from 0 through NX_HTTP_SERVER_MAX_CLIENT - 1</p> <p>*packet_ptr: Pointer to the packet to send</p>
<b>Return</b>	<p>*NX_SUCCESS (0x00): Successfully sent HTTP Server packet</p> <p>*status: Completion status of internal NetX calls</p> <p>*NX_PTR_ERROR (0x07): Invalid pointer input</p>
<b>Description</b>	<p>This service sends a complete HTTP server response from an HTTP callback. HTTP server sends the packet with the NX_HTTP_SERVER_TIMEOUT_SEND. The HTTP header and data must be appended to the packet. If the return status indicates an error, the HTTP application must release the packet.</p> <p>The callback should return NX_HTTP_CALLBACK_COMPLETED.</p> <p>See nx_http_server_callback_generate_response_header for a more detailed example.</p>

<pre>UINT nx_http_server_callback_response_send(NX_HTTP_SERVER *server_ptr,  UINT logical_connection,  CHAR *header,  CHAR *information,  CHAR *additional_info)</pre>	
<b>Parameter</b>	<p>*server_ptr: Pointer to HTTP Server control block</p> <p>*logical_connection: The internal logical connection for the HTTP Server. This can be used by the application as an index into buffers and/or data structures specific for each Client connection. Its value ranges from 0 through NX_HTTP_SERVER_MAX_CLIENT - 1</p> <p>*header: Pointer to the response header string</p> <p>*information: Pointer to the information string</p> <p>*additional_info: Pointer to the additional information string</p>
<b>Return</b>	<p>*NX_SUCCESS (0x00): Successfully sent HTTP Server packet</p> <p>*status: Completion status of internal NetX calls</p> <p>*NX_PTR_ERROR (0x07): Invalid pointer input</p>
<b>Description</b>	<p>This service sends the supplied response information from the application's callback routine. This is typically used to send custom responses associated with GET/POST requests. Note that if this function is used, the callback routine must return the status of NX_HTTP_CALLBACK_COMPLETED.</p>

DA16200 HTTP Extended APIs

<pre> UINT nx_http_server_content_get(NX_HTTP_SERVER *server_ptr,                                 UINT logical_connection,                                 NX_PACKET *packet_ptr,                                 ULONG byte_offset,                                 CHAR *destination_ptr,                                 UINT destination_size,                                 UINT *actual_size)         </pre>	
<b>Parameter</b>	<p>*server_ptr: Pointer to HTTP Server control block</p> <p>*packet_ptr: Pointer to the HTTP Client request packet. Note that this packet must not be released by the request notify callback</p> <p>*logical_connection: The internal logical connection for the HTTP Server. This can be used by the application as an index into buffers and/or data structures specific for each Client connection. Its value ranges from 0 through NX_HTTP_SERVER_MAX_CLIENT - 1</p> <p>*byte_offset: Number of bytes to offset into the content area</p> <p>*destination_ptr: Pointer to the destination area for the content</p> <p>*destination_size: Maximum number of bytes available in the destination area</p> <p>*actual_size: Pointer to the destination variable that will be set to the actual size of the content copied</p>
<b>Return</b>	<p>*NX_SUCCESS (0x00): Successful HTTP Server content get</p> <p>*NX_HTTP_ERROR (0xE0): HTTP Server internal error</p> <p>*NX_HTTP_DATA_END (0xE7): End of request content</p> <p>*NX_HTTP_TIMEOUT (0xE1): HTTP Server timeout in getting next packet of content</p> <p>*NX_PTR_ERROR (0x07): Invalid pointer input</p> <p>*NX_CALLER_ERROR (0x11): Invalid caller of this service</p>
<b>Description</b>	<p>This function should be called from the application's request notify callback specified during HTTP Server creation (nx_http_server_create).</p>

DA16200 HTTP Extended APIs

<pre> UINT nx_http_server_content_get_extended(NX_HTTP_SERVER *server_ptr,  UINT logical_connection,  NX_PACKET *packet_ptr,  ULONG byte_offset,  CHAR *destination_ptr,  UINT destination_size, UINT *actual_size)                 </pre>	
<b>Parameter</b>	<p>*server_ptr: Pointer to HTTP Server control block</p> <p>*logical_connection: The internal logical connection for the HTTP Server. This can be used by the application as an index into buffers and/or data structures specific for each Client connection. Its value ranges from 0 through NX_HTTP_SERVER_MAX_CLIENT - 1</p> <p>*packet_ptr: Pointer to the HTTP Client request packet. Note that this packet must not be released by the request notify callback</p> <p>*byte_offset: Number of bytes to offset into the content area</p> <p>*destination_ptr: Pointer to the destination area for the content</p> <p>*destination_size: Maximum number of bytes available in the destination area</p> <p>*actual_size: Pointer to the destination variable that will be set to the actual size of the content copied</p>
<b>Return</b>	<p>*NX_SUCCESS (0x00): Successful HTTP content get</p> <p>*NX_HTTP_ERROR (0xE0): HTTP Server internal error</p> <p>*NX_HTTP_DATA_END (0xE7): End of request content</p> <p>*NX_HTTP_TIMEOUT (0xE1): HTTP Server timeout in getting next packet</p> <p>*NX_PTR_ERROR (0x07): Invalid pointer input</p> <p>*NX_CALLER_ERROR (0x11): Invalid caller of this service</p>
<b>Description</b>	<p>This service is almost identical to nx_http_server_content_get; it attempts to retrieve the specified amount of content from the POST or PUT HTTP Client request. However, it handles requests with Content Length of zero value ('empty request') as a valid request. It should be called from the application's request notify callback specified during HTTP Server creation (nx_http_server_create).</p>

DA16200 HTTP Extended APIs

<pre> UINT nx_http_server_create(NX_HTTP_SERVER *http_server_ptr,     CHAR *http_server_name,     NX_IP *ip_ptr,     VOID *stack_ptr,     ULONG stack_size,     NX_PACKET_POOL *pool_ptr,     NX_HTTP_SERVER_PARAMETERS *parameters,     UINT (*authentication_check)(NX_HTTP_SERVER *server_ptr,         UINT logical_connection,         UINT request_type,         CHAR *resource,         CHAR **name,         CHAR **password,         CHAR **realm),     UINT (*request_notify)(NX_HTTP_SERVER *server_ptr,         UINT logical_connection,         UINT request_type,         CHAR *resource,         NX_PACKET *packet_ptr)     )         </pre>	
<b>Parameter</b>	<p>*http_server_ptr: Pointer to HTTP Server control block</p> <p>*http_server_name: Pointer to HTTP Server's name</p> <p>*ip_ptr: Pointer to previously created IP instance</p> <p>*stack_ptr: Pointer to HTTP Server thread stack area</p> <p>*stack_size: Pointer to HTTP Server thread stack size</p> <p>*authentication_check: Function pointer to application's authentication checking routine. If this parameter is specified, this routine is called for each HTTP Client request. If this parameter is NULL, no authentication is performed.</p> <p>*request_notify: Function pointer to application's request notify routine. If this parameter is specified, this routine is called prior to the HTTP server processing of the request. This allows the resource name to be redirected or fields within a resource to be updated prior to completing the HTTP Client request.</p>
<b>Return</b>	<p>*NX_SUCCESS (0x00): Successful HTTP Server create</p> <p>*NX_HTTP_ERROR (0xE0): HTTP Server create error</p> <p>*NX_PTR_ERROR (0x07): Invalid HTTP Server, IP, media, stack, or packet pool pointer</p> <p>*NX_HTTP_POOL_ERROR (0xE9): Packet payload of pool is not large enough to contain complete HTTP request</p>
<b>Description</b>	<p>This service creates an HTTP Server instance, which runs in the context of its own ThreadX thread. The optional authentication_check and request_notify application callback routines give the application software control over the basic operations of the HTTP Server.</p>

## DA16200 HTTP Extended APIs

<pre>UINT nx_http_server_get_entity_content(NX_HTTP_SERVER *server_ptr,  UINT logical_connection,  NX_PACKET **packet_pptr,  ULONG *available_offset,  ULONG *available_length)</pre>	
<b>Parameter</b>	<p>*server_ptr: Pointer to HTTP Server</p> <p>*logical_connection: The internal logical connection for the HTTP Server. This can be used by the application as an index into buffers and/or data structures specific for each Client connection. Its value ranges from 0 through NX_HTTP_SERVER_MAX_CLIENT - 1</p> <p>*packet_pptr: Pointer to location of packet pointer</p> <p>*available_offset: Pointer to offset of entity data from the packet prepend pointer</p> <p>*available_length: Pointer to length of entity data</p>
<b>Return</b>	<p>*NX_SUCCESS (0x00): Successfully retrieved size and location of entity content</p> <p>*NX_HTTP_BOUNDARY_ALREADY_FOUND (0xF4): Content for the HTTP server internal multipart markers that are already found</p> <p>*NX_HTTP_ERROR (0xE0): HTTP Server internal error</p> <p>*NX_HTTP_TIMEOUT (0xE1): Time expired waiting for next packet/rest of Client message</p> <p>*NX_PTR_ERROR (0x07): Invalid pointer input</p>
<b>Description</b>	<p>This service determines the location of the start of data within the current multipart entity in the received Client messages, and the length of data not including the boundary string. Internally HTTP server updates its own offsets so that this function can be called again on the same Client datagram for messages with multiple entities. The packet pointer is updated to the next packet where the Client message is a multi-packet datagram.</p> <p>Note that NX_HTTP_MULTIPART_ENABLE must be enabled to use this service.</p> <p>See nx_http_server_get_entity_header for more details.</p>

## DA16200 HTTP Extended APIs

<pre>UINT nx_http_server_get_entity_header(NX_HTTP_SERVER *server_ptr,                                      UINT logical_connection,                                      NX_PACKET **packet_pptr,                                      UCHAR *entity_header_buffer,                                      ULONG buffer_size)</pre>	
<b>Parameter</b>	<p>*server_ptr: Pointer to HTTP Server</p> <p>*logical_connection: The internal logical connection for the HTTP Server. This can be used by the application as an index into buffers and/or data structures specific for each Client connection. Its value ranges from 0 through NX_HTTP_SERVER_MAX_CLIENT - 1</p> <p>*packet_pptr: Pointer to location of packet pointer</p> <p>*entity_header_buffer: Pointer to location to store entity header</p> <p>*buffer_size: Size of input buffer</p>
<b>Return</b>	<p>*NX_SUCCESS (0x00): Successfully retrieved entity header</p> <p>*NX_HTTP_NOT_FOUND (0xE6): Entity header field not found</p> <p>*NX_HTTP_TIMEOUT (0xE1): Time to receive next packet for multipacket client message is expired</p> <p>*NX_PTR_ERROR (0x07): Invalid pointer input</p> <p>*NX_CALLER_ERROR (0x11): Invalid caller of this service</p>
<b>Description</b>	<p>This service retrieves the entity header into the specified buffer. Internally HTTP Server updates its own pointers to locate the next multipart entity in a Client datagram with multiple entity headers. The packet pointer is updated to the next packet where the Client message is a multi-packet datagram.</p> <p>Note that NX_HTTP_MULTIPART_ENABLE must be enabled to use this service.</p>

DA16200 HTTP Extended APIs

<pre>UINT nx_http_server_packet_content_find(NX_HTTP_SERVER *server_ptr,  UINT logical_connection,  NX_PACKET **packet_ptr,  ULONG *content_length)</pre>	
<b>Parameter</b>	<p>*server_ptr: Pointer to HTTP server instance</p> <p>*logical_connection: The internal logical connection for the HTTP Server. This can be used by the application as an index into buffers and/or data structures specific for each Client connection. Its value ranges from 0 through NX_HTTP_SERVER_MAX_CLIENT - 1</p> <p>*packet_ptr: Pointer to packet pointer for returning the packet with updated prepend pointer</p> <p>*content_length: Pointer to extracted content_length</p>
<b>Return</b>	<p>*NX_SUCCESS (0x00): HTTP content length found and packet successfully updated</p> <p>*NX_HTTP_TIMEOUT (0xE1): Time expired waiting on next packet</p> <p>*NX_PTR_ERROR (0x07): Invalid pointer input</p> <p>*NX_CALLER_ERROR (0x11): Invalid caller of this service</p>
<b>Description</b>	<p>This service extracts the content length from the HTTP header. It also updates the supplied packet as follows: the packet prepend pointer (start of location of packet buffer to write to) is set to the HTTP content (data) just passed the HTTP header.</p> <p>If the beginning of content is not found in the current packet, the function waits for the next packet to be received using the NX_HTTP_SERVER_TIMEOUT_RECEIVE wait option.</p> <p>Note this should not be called before calling nx_http_server_get_entity_header because it modifies the prepend pointer past the entity header.</p>

<pre>UINT nx_http_server_packet_get(NX_HTTP_SERVER *server_ptr,                               UINT logical_connection,                               NX_PACKET **packet_ptr)</pre>	
<b>Parameter</b>	<p>*server_ptr: Pointer to HTTP server instance</p> <p>*logical_connection: The internal logical connection for the HTTP Server. This can be used by the application as an index into buffers and/or data structures specific for each Client connection. Its value ranges from 0 through NX_HTTP_SERVER_MAX_CLIENT - 1</p> <p>*packet_ptr: Pointer to received packet</p>
<b>Return</b>	<p>*NX_SUCCESS (0x00): Successfully received next HTTP packet</p> <p>*NX_HTTP_TIMEOUT (0xE1): Time expired waiting on next packet</p> <p>*NX_PTR_ERROR (0x07): Invalid pointer input</p>
<b>Description</b>	<p>This service returns the next packet received on the HTTP server socket. The wait option to receive a packet is NX_HTTP_SERVER_TIMEOUT_RECEIVE.</p>



## DA16200 HTTP Extended APIs

<pre>UINT nx_http_server_disconnect_client(NX_HTTP_SERVER *server_ptr,                                      UINT logical_connection,                                      ULONG secs)</pre>	
<b>Parameter</b>	<p>*server_ptr: Pointer to HTTP server instance</p> <p>*logical_connection: The internal logical connection for the HTTP Server. This can be used by the application as an index into buffers and/or data structures specific for each Client connection. Its value ranges from 0 through NX_HTTP_SERVER_MAX_CLIENT - 1</p>
<b>Return</b>	<p>*NX_SUCCESS (0x00): Successfully received next HTTP packet</p> <p>*NX_OPTION_ERROR (0x0A): local_connection value is less than 0 or greater than NX_HTTP_SERVER_MAX_CLIENT-1</p> <p>*NX_PTR_ERROR (0x07): Invalid pointer input</p>
<b>Description</b>	This service disconnects the connected clients.

<pre>UINT nx_http_server_get_activity_timeout(NX_HTTP_SERVER *server_ptr,   UINT logical_connection,   ULONG *secs)</pre>	
<b>Parameter</b>	<p>*server_ptr: Pointer to HTTP server instance</p> <p>*logical_connection: The internal logical connection for the HTTP Server. This can be used by the application as an index into buffers and/or data structures specific for each Client connection. Its value ranges from 0 through NX_HTTP_SERVER_MAX_CLIENT - 1</p> <p>*secs: Set activity_timeout in seconds</p>
<b>Return</b>	<p>*NX_SUCCESS (0x00): Successfully received next HTTP packet</p> <p>*NX_OPTION_ERROR (0x0A): local_connection value is less than 0 or greater than NX_HTTP_SERVER_MAX_CLIENT-1</p> <p>*NX_PTR_ERROR (0x07): Invalid pointer input</p>
<b>Description</b>	This service returns activity_timeout in seconds set according to the keepalive option of the connected client.

<pre>UINT nx_http_server_set_activity_timeout(NX_HTTP_SERVER *server_ptr,   UINT logical_connection,   ULONG secs)</pre>	
<b>Parameter</b>	<p>*server_ptr: Pointer to HTTP server instance</p> <p>*logical_connection: The internal logical connection for the HTTP Server. This can be used by the application as an index into buffers and/or data structures specific for each Client connection. Its value ranges from 0 through NX_HTTP_SERVER_MAX_CLIENT - 1</p> <p>*secs: return activity_timeout in seconds</p>
<b>Return</b>	<p>*NX_SUCCESS (0x00): Successfully received next HTTP packet</p> <p>*NX_OPTION_ERROR (0x0A): local_connection value is less than 0 or greater than NX_HTTP_SERVER_MAX_CLIENT-1</p> <p>*NX_PTR_ERROR (0x07): Invalid pointer input</p>
<b>Description</b>	This service can set activity_timeout in seconds according to the keepalive option of the connected client.

---

DA16200 HTTP Extended APIs

## Revision History

Revision	Date	Description
1.1	17-Sep-2020	Added API for controlling client connected from HTTP-Server
1.0	24-Apr-2020	Initial Release

## DA16200 HTTP Extended APIs

### Status Definitions

Status	Definition
DRAFT	The content of this document is under review and subject to formal approval, which may result in modifications or additions.
APPROVED or unmarked	The content of this document has been approved for publication.

### Disclaimer

Unless otherwise agreed in writing, the Dialog Semiconductor products (and any associated software) referred to in this document are not designed, authorized or warranted to be suitable for use in life support, life-critical or safety-critical systems or equipment, nor in applications where failure or malfunction of a Dialog Semiconductor product (or associated software) can reasonably be expected to result in personal injury, death or severe property or environmental damage. Dialog Semiconductor and its suppliers accept no liability for inclusion and/or use of Dialog Semiconductor products (and any associated software) in such equipment or applications and therefore such inclusion and/or use is at the customer's own risk.

Information in this document is believed to be accurate and reliable. However, Dialog Semiconductor does not give any representations or warranties, express or implied, as to the accuracy or completeness of such information. Dialog Semiconductor furthermore takes no responsibility whatsoever for the content in this document if provided by any information source outside of Dialog Semiconductor.

Dialog Semiconductor reserves the right to change without notice the information published in this document, including, without limitation, the specification and the design of the related semiconductor products, software and applications. Notwithstanding the foregoing, for any automotive grade version of the device, Dialog Semiconductor reserves the right to change the information published in this document, including, without limitation, the specification and the design of the related semiconductor products, software and applications, in accordance with its standard automotive change notification process.

Applications, software, and semiconductor products described in this document are for illustrative purposes only. Dialog Semiconductor makes no representation or warranty that such applications, software and semiconductor products will be suitable for the specified use without further testing or modification. Unless otherwise agreed in writing, such testing or modification is the sole responsibility of the customer and Dialog Semiconductor excludes all liability in this respect.

Nothing in this document may be construed as a license for customer to use the Dialog Semiconductor products, software and applications referred to in this document. Such license must be separately sought by customer with Dialog Semiconductor.

All use of Dialog Semiconductor products, software and applications referred to in this document is subject to Dialog Semiconductor's [Standard Terms and Conditions of Sale](#), available on the company website ([www.dialog-semiconductor.com](http://www.dialog-semiconductor.com)) unless otherwise stated.

Dialog, Dialog Semiconductor and the Dialog logo are trademarks of Dialog Semiconductor Plc or its subsidiaries. All other product or service names and marks are the property of their respective owners.

© 2020 Dialog Semiconductor. All rights reserved

## Contacting Dialog Semiconductor

### United Kingdom (Headquarters)

*Dialog Semiconductor (UK) LTD*  
Phone: +44 1793 757700

### Germany

*Dialog Semiconductor GmbH*  
Phone: +49 7021 805-0

### The Netherlands

*Dialog Semiconductor B.V.*  
Phone: +31 73 640 8822

### Email:

[enquiry@diasemi.com](mailto:enquiry@diasemi.com)

### North America

*Dialog Semiconductor Inc.*  
Phone: +1 408 845 8500

### Japan

*Dialog Semiconductor K. K.*  
Phone: +81 3 5769 5100

### Taiwan

*Dialog Semiconductor Taiwan*  
Phone: +886 281 786 222

### Web site:

[www.dialog-semiconductor.com](http://www.dialog-semiconductor.com)

### Hong Kong

*Dialog Semiconductor Hong Kong*  
Phone: +852 2607 4271

### Korea

*Dialog Semiconductor Korea*  
Phone: +82 2 3469 8200

### China (Shenzhen)

*Dialog Semiconductor China*  
Phone: +86 755 2981 3669

### China (Shanghai)

*Dialog Semiconductor China*  
Phone: +86 21 5424 9058