User's Manual

NetX™

Web Hypertext Transfer Protocol (HTTP) and Hypertext Transfer Protocol Secure (HTTPS)

User Guide

Renesas Synergy™ Platform

Renesas Electronics
www.renesas.com

Feb 25, 2019

Notice

1. Descriptions of circuits, software and other related information in this document are provided only to illustrate the operation of semiconductor products and application examples. You are fully responsible for the incorporation or any other use of the circuits, software, and information in the design of your product or system. Renesas Electronics disclaims any and all liability for any losses and damages incurred by you or third parties arising from the use of these circuits, software, or information.
2. Renesas Electronics hereby expressly disclaims any warranties against and liability for infringement or any other claims involving patents, copyrights, or other intellectual property rights of third parties, by or arising from the use of Renesas Electronics products or technical information described in this document, including but not limited to, the product data, drawings, charts, programs, algorithms, and application examples.
3. No license, express, implied or otherwise, is granted hereby under any patents, copyrights or other intellectual property rights of Renesas Electronics or others.
4. You shall not alter, modify, copy, or reverse engineer any Renesas Electronics product, whether in whole or in part. Renesas Electronics disclaims any and all liability for any losses or damages incurred by you or third parties arising from such alteration, modification, copying or reverse engineering.
5. Renesas Electronics products are classified according to the following two quality grades: "Standard" and "High Quality". The intended applications for each Renesas Electronics product depends on the product's quality grade, as indicated below.
   "Standard": Computers; office equipment; communications equipment; test and measurement equipment; audio and visual equipment; home electronic appliances; machine tools; personal electronic equipment; industrial robots; etc.
   "High Quality": Transportation equipment (automobiles, trains, ships, etc.); traffic control (traffic lights); large-scale communication equipment; key financial terminal systems; safety control equipment; etc.
   Unless expressly designated as a high reliability product or a product for harsh environments in a Renesas Electronics data sheet or other Renesas Electronics document, Renesas Electronics products are not intended or authorized for use in products or systems that may pose a direct threat to human life or bodily injury (artificial life support devices or systems; surgical implantations; etc.), or may cause serious property damage (space system; undersea repeaters; nuclear power control systems; aircraft control systems; key plant systems; military equipment; etc.). Renesas Electronics disclaims any and all liability for any damages or losses incurred by you or any third parties arising from the use of any Renesas Electronics product that is inconsistent with any Renesas Electronics data sheet, user's manual or other Renesas Electronics document.
6. When using Renesas Electronics products, refer to the latest product information (data sheets, user's manuals, application notes, "General Notes for Handling and Using Semiconductor Devices" in the reliability handbook, etc.), and ensure that usage conditions are within the ranges specified by Renesas Electronics with respect to maximum ratings, operating power supply voltage range, heat dissipation characteristics, installation, etc. Renesas Electronics disclaims any and all liability for any malfunctions, failure or accident arising out of the use of Renesas Electronics products outside of such specified ranges.
7. Although Renesas Electronics endeavors to improve the quality and reliability of Renesas Electronics products, semiconductor products have specific characteristics, such as the occurrence of failure at a certain rate and malfunctions under certain use conditions. Unless designated as a high reliability product or a product for harsh environments in a Renesas Electronics data sheet or other Renesas Electronics document, Renesas Electronics products are not subject to radiation resistance design. You are responsible for implementing safety measures to guard against the possibility of bodily injury, injury or damage caused by fire, and/or danger to the public in the event of a failure or malfunction of Renesas Electronics products, such as safety design for hardware and software, including but not limited to redundancy, fire control and malfunction prevention, appropriate treatment for aging degradation or any other appropriate measures. Because the evaluation of microcomputer software alone is very difficult and impractical, you are responsible for evaluating the safety of the final products or systems manufactured by you.
8. Please contact a Renesas Electronics sales office for details as to environmental matters such as the environmental compatibility of each Renesas Electronics product. You are responsible for carefully and sufficiently investigating applicable laws and regulations that regulate the inclusion or use of controlled substances, including without limitation, the EU RoHS Directive, and using Renesas Electronics products in compliance with all these applicable laws and regulations. Renesas Electronics disclaims any and all liability for damages or losses occurring as a result of your noncompliance with applicable laws and regulations.
9. Renesas Electronics products and technologies shall not be used for or incorporated into any products or systems whose manufacture, use, or sale is prohibited under any applicable domestic or foreign laws or regulations. You shall comply with any applicable export control laws and regulations promulgated and administered by the governments of any countries asserting jurisdiction over the parties or transactions.
10. It is the responsibility of the buyer or distributor of Renesas Electronics products, or any other party who distributes, disposes of, or otherwise sells or transfers the product to a third party, to notify such third party in advance of the contents and conditions set forth in this document.
11. This document shall not be reprinted, reproduced or duplicated in any form, in whole or in part, without prior written consent of Renesas Electronics.
12. Please contact a Renesas Electronics sales office if you have any questions regarding the information contained in this document or Renesas Electronics products.

(Note1) "Renesas Electronics" as used in this document means Renesas Electronics Corporation and also includes its directly or indirectly controlled subsidiaries.

(Note2) "Renesas Electronics product(s)" means any product developed or manufactured by or for Renesas Electronics.

(Rev.4.0-1 November 2017)

Corporate Headquarters

TOYOSU FORESIA, 3-2-24 Toyosu,
Koto-ku, Tokyo 135-0061, Japan
www.renesas.com

Trademarks

Contact information

For further information on a product, technology, the most up-to-date version of a document, or your nearest sales office, please visit:
www.renesas.com/contact/.

Renesas Synergy Specific Information

If you are using NetX Web HTTP/HTTPS for the Renesas Synergy platform, please use the following information.

HTTP Client Authentication

Page 9: HTTP Client authentication using username and password has not been tested with SSP v1.5.0 or 1.6.0.

Installation

Page 15: If you are using Renesas Synergy SSP and the e^2 studio ISDE, Web HTTP/HTTPS will already be installed. You can ignore the HTTP Installation section.

NetX Web Hypertext Transfer Protocol (HTTP) and Hypertext Transfer Protocol Secure (HTTPS)

User Guide

Express Logic, Inc.

858.613.6640
Toll Free 888.THREADX
FAX 858.521.4259

www.expresslogic.com

Part Number: 000-1054
Revision 5.11SP1

Contents

Chapter 1

Introduction to HTTP and HTTPS

The Hypertext Transfer Protocol (HTTP) is a protocol designed for transferring content on the Web. HTTP is a simple protocol that utilizes reliable Transmission Control Protocol (TCP) services to perform its content transfer function. Because of this, HTTP is a highly reliable content transfer protocol. HTTP is one of the most used application protocols. All operations on the Web utilize the HTTP protocol.

HTTPS is the secure version of the HTTP protocol, which implements HTTP using Transport Layer Security (TLS) to secure the underlying TCP connection. Other than the additional configuration required to set up TLS, HTTPS is basically identical to HTTP in use.

General HTTP Requirements

In order to function properly, the NetX Web HTTP package requires that NetX Duo (version 5.10 or later) is installed. In addition, an IP instance must be created, and TCP must be enabled on that same IP instance. For HTTPS support, NetX Secure TLS (version 5.11 or later) must also be installed (see next section). The demo file in section “Small Example System” in **Chapter 2** demonstrates how this is done.

The HTTP Client portion of the NetX Web HTTP package has no further requirements.

The HTTP Server portion of the NetX Web HTTP package has several additional requirements. First, it requires complete access to TCP *well-known port 80* for handling all Client HTTP requests (this can be changed by the application to any other valid TCP port). The HTTP Server is also designed for use with the FileX embedded file system. If FileX is not available, the user may port the portions of FileX used to their own environment. This is discussed in later sections of this guide.

HTTPS Requirements

For HTTPS to function properly, the NetX Web HTTP package requires that NetX Duo (version 5.10 or later) and NetX Secure TLS (version 5.11 or later) are both installed. In addition, an IP instance must be created, and TCP must be enabled on that same IP instance for use with TLS. The TLS session will need to be initialized with appropriate cryptographic routines, a trusted CA certificate, and space must be allocated for certificates that will be provided by remote server hosts during the TLS handshake. The demo file in section “Small Example HTTPS System” in **Chapter 2** will demonstrate how this is done.

The HTTPS Client portion of the NetX Web HTTP package has no further requirements.

The HTTPS Server portion of the NetX Web HTTP package has several additional requirements. First, it requires complete access to TCP *well-known port 443* for handling all Client HTTPS requests (as with plaintext HTTP, this port can be changed by the application). Second, the TLS session will need to be initialized with proper cryptographic routines and a server identity certificate (or Pre-Shared Key). The HTTPS Server is also designed for use with the FileX embedded file system. If FileX is not available, the user may port the portions of FileX used to their own environment. The use of FileX is discussed in later sections of this guide.

Refer to the NetX Secure documentation for more information on configuration options for TLS.

Unless noted, all HTTP functionality described in this document also applies to HTTPS.

HTTP and HTTPS Constraints

NetX Web HTTP implements the HTTP 1.1 standard. However, here are following constraints:

1. Request pipelining is not supported
2. The HTTP Server supports both basic and MD5 digest authentication, but not MD5-sess. At present, the HTTP Client supports only basic authentication. When using TLS for HTTPS, HTTP authentication may still be used.
3. No content compression is supported.
4. TRACE, OPTIONS, and CONNECT requests are not supported.
5. The packet pool associated with the HTTP Server or Client must be large enough to hold the complete HTTP header.

6. HTTP Client services are for content transfer only—there are no display utilities provided in this package.

HTTP URL (Resource Names)

The HTTP protocol is designed to transfer content on Web. The requested content is specified by the Universal Resource Locator (URL). This is the primary component of every HTTP request. URLs always start with a "/" character and typically correspond to files on the HTTP Server. Common HTTP file extensions are shown below:

Extension	Meaning
.htm (or .html)	Hypertext Markup Language (HTML)
.txt	Plain ASCII text
.gif	Binary GIF image
.xbm	Binary Xbitmap image

HTTP Client Requests

The HTTP has a simple mechanism for requesting Web content. There is a set of standard HTTP commands that are issued by the Client after a connection has been successfully established on the TCP *well-known port 80 (port 443 for HTTPS)*. The following shows some of the basic HTTP commands:

HTTP Command	Meaning
GET *resource* HTTP/1.1	*Get the specified resource*
POST *resource* HTTP/1.1	*Get the specified resource and pass attached input to the HTTP Server*
HEAD *resource* HTTP/1.1	*Treated like a GET but not content is returned by the HTTP Server*
PUT *resource* HTTP/1.1	*Place resource on HTTP Server*
DELETE *resource* HTTP/1.1	*Delete resource on the Server*

These ASCII commands are generated internally by Web browsers and the NetX Web HTTP Client services to perform HTTP operations with an HTTP Server.

Note that the HTTP Client application should use port 80, or port 443 if HTTPS is used. Both the Client and Server HTTP APIs take the port as a parameter – the macros NX_WEB_HTTP_SERVER_PORT (port 80) and NX_WEB_HTTPS_SERVER_PORT (port 443) are defined for convenience. The HTTP Server port can also be changed at runtime using the *nx_web_http_client_set_connect_port()* service. See Chapter 4 for more details on this service.

HTTP Server Responses

The HTTP Server utilizes the same *well-known TCP port 80 (443 for HTTPS)* to send Client command responses. Once the HTTP Server processes the Client command, it returns an ASCII response string that includes a 3-digit numeric status code. The numeric response is used by the HTTP Client software to determine whether the operation succeeded or failed. Following is a list of various HTTP Server responses to Client commands:

Numeric Field	Meaning
200	*Request was successful*
400	*Request was not formed properly*
401	*Unauthorized request, client needs to send authentication*
404	*Specified resource in request was not found*
500	*Internal HTTP Server error*
501	*Request not implemented by HTTP Server*
502	*Service is not available*

For example, a successful Client request to PUT the file “test.htm” is responded with the message “HTTP/1.1 200 OK.”

HTTP Communication

As mentioned previously, the HTTP Server utilizes the *well-known TCP port 80 (443 for HTTPS)* to field Client requests. HTTP Clients may use any available TCP port for outgoing connections. The general sequence of HTTP events is as follows:

HTTP GET Request:

1. Client issues TCP connect to Server port 80 (or 443 for HTTPS).
2. If HTTPS is being used, the TCP connection is followed by a TLS handshake to authenticate the server and establish a secure channel.
3. Client sends "**GET *resource* HTTP/1.1**" request (along with other header information).
4. Server builds an "**HTTP/1.1 200 OK**" message with additional information followed immediately by the resource content (if any).
5. Server disconnects from the client (TLS is shut down if HTTPS is being used).
6. Client disconnects from the socket (TLS is shut down following the disconnection alert from the server).

HTTP PUT Request:

1. Client issues TCP connect to Server port 80 (or 443).
2. If HTTPS is being used, the TCP connection is followed by a TLS handshake to authenticate the server and establish a secure channel.
3. Client sends "PUT resource HTTP/1.1" request, along with other header information, and followed by the resource content.
4. Server builds an "HTTP/1.1 200 OK" message with additional information followed immediately by the resource content.
5. Server performs a disconnection.
6. Client performs a disconnection.

Note: as mentioned previously, the HTTP Server can change the default connect port (80 or 443) at runtime another port using the *nx_web_http_client_set_connect_port()* for web servers that use alternate ports to connect to clients.

HTTP Authentication

HTTP authentication is optional and is not required for all Web requests. There are two flavors of authentication, namely *basic* and *digest*. Basic authentication is equivalent to the *name* and *password* authentication found in many protocols. In HTTP basic authentication, the name and

passwords are concatenated and encoded in the base64 format. The main disadvantage of basic authentication is the name and password are transmitted openly in the request. This makes it somewhat easy for the name and password to be stolen. Digest authentication addresses this problem by never transmitting the name and password in the request. Instead, an algorithm is used to derive a 128-bit digest from the name, password, and other information. The NetX Web HTTP Server supports the standard MD5 digest algorithm.

When is authentication required? The HTTP Server decides if a requested resource requires authentication. If authentication is required and the Client request did not include the proper authentication, a “HTTP/1.1 401 Unauthorized” response with the type of authentication required is sent to the Client. The Client is then expected to form a new request with the proper authentication.

When HTTPS is used, the HTTPS Server can still utilize HTTP authentication. In this case, TLS is used to encrypt all HTTP traffic so using *basic* HTTP authentication does not pose a security risk. *Digest* authentication is also permitted but provides no significant security improvement over basic authentication over TLS.

HTTP Authentication Callback

As mentioned before, HTTP authentication is optional and isn’t required on all Web transfers. In addition, authentication is typically resource dependent. Access of some resources on the Server require authentication, while others do not. The NetX Web HTTP Server package allows the application to specify (via the *nx_web_http_server_create* call) an authentication callback routine that is called at the beginning of handling each HTTP Client request.

The callback routine provides the NetX Web HTTP Server with the username, password, and realm strings associated with the resource and return the type of authentication necessary. If no authentication is necessary for the resource, the authentication callback should return the value of **NX_WEB_HTTP_DONT_AUTHENTICATE**. Otherwise, if basic authentication is required for the specified resource, the routine should return **NX_WEB_HTTP_BASIC_AUTHENTICATE**. And finally, if MD5 digest authentication is required, the callback routine should return **NX_WEB_HTTP_DIGEST_AUTHENTICATE**. If no authentication is required for any resource provided by the HTTP Server, the callback is not needed, and a NULL pointer can be provided to the HTTP Server create call.

The format of the application authenticate callback routine is very simple and is defined below:

```
UINT nx_web_http_server_authentication_check(NX_WEB_HTTP_SERVER *server_ptr,
                                             UINT request_type, CHAR *resource,
                                             CHAR **name, CHAR **password,
                                             CHAR **realm);
```

The input parameters are defined as follows:

Parameter	Meaning
request_type	Specifies the HTTP Client request, valid requests are defined as: **NX_WEB_HTTP_SERVER_GET_REQUEST** **NX_WEB_HTTP_SERVER_POST_REQUEST** **NX_WEB_HTTP_SERVER_HEAD_REQUEST** **NX_WEB_HTTP_SERVER_PUT_REQUEST** **NX_WEB_HTTP_SERVER_DELETE_REQUEST**
resource	Specific resource requested.
name	Destination for the pointer to the required username.
password	Destination for the pointer to the required password.
realm	Destination for the pointer to the realm for this authentication.

The return value of the authentication routine specifies if authentication is required. `name, password, and realm` pointers are not used if **NX_WEB_HTTP_DONT_AUTHENTICATE** is returned by the authentication callback routine. Otherwise the HTTP server developer must ensure that **NX_WEB_HTTP_MAX_USERNAME** and **NX_WEB_HTTP_MAX_PASSWORD** defined in *nx_web_http_server.h* are large enough for the username and password specified in the authentication callback. These both have a default size of 20 characters.

HTTP Invalid Username/Password Callback

The optional invalid username/password callback in the NetX Web HTTP Server is invoked if the HTTP server receives an invalid username and password combination in a Client request. If the HTTP server application registers a callback with HTTP server it will be invoked if either basic or

digest authentication fails *in nx_web_http_server_get_process()*, in *nx_web_http_server_put_process(),* or *in nx_web_http_server_delete_process().*

To register a callback with the HTTP server, the following service is defined for the NetX Web HTTP Server.

```
UINT nx_web_http_server_invalid_userpassword_notify_set(
                                NX_WEB_HTTP_SERVER *http_server_ptr,
                                UINT (*invalid_username_password_callback)
                                        (CHAR *resource,
                                         ULONG *client_nx_address,
                                         UINT request_type));
```

The request types are defined as follows:

NX_WEB_HTTP_SERVER_GET_REQUEST
NX_WEB_HTTP_SERVER_POST_REQUEST
NX_WEB_HTTP_SERVER_HEAD_REQUEST
NX_WEB_HTTP_SERVER_PUT_REQUEST
NX_WEB_HTTP_SERVER_DELETE_REQUEST

HTTP Insert GMT Date Header Callback

There is an optional callback in the NetX Web HTTP Server to insert a date header in its response messages. This callback is invoked when the HTTP Server is responding to a put or get request

To register a GMT date callback with the HTTP Server, the following service is defined.

```
UINT  nx_web_http_server_gmt_callback_set(
                        NX_WEB_HTTP_SERVER *server_ptr,
                        VOID (*gmt_get)(NX_WEB_HTTP_SERVER_DATE *date)
```

The NX_WEB_HTTP_SERVER_DATE data type is defined as follows:

```
typedef struct NX_WEB_HTTP_SERVER_DATE_STRUCT
{
    USHORT          nx_web_http_server_year;       /* Year                 */
    UCHAR           nx_web_http_server_month;      /* Month                */
    UCHAR           nx_web_http_server_day;        /* Day                  */
    UCHAR           nx_web_http_server_hour;       /* Hour                 */
    UCHAR           nx_web_http_server_minute;     /* Minute               */
    UCHAR           nx_web_http_server_second;     /* Second               */
    UCHAR           nx_web_http_server_weekday;    /* Weekday              */
} NX_WEB_HTTP_SERVER_DATE;
```

HTTP Cache Info Get Callback

The HTTP Server has a callback to request the maximum age and date from the HTTP application for a specific resource. This information is used to determine if the HTTP server sends an entire page in response to a Client Get request. If the “if modified since” in the Client request is not found or does not match the “last modified” date returned by the get cache callback, the entire page is sent.

To register the callback with the HTTP server the following service is defined:

```
UINT  nx_web_http_server_cache_info_callback_set(
                              NX_WEB_HTTP_SERVER *server_ptr,
                              UINT (*cache_info_get)
                                    (CHAR *, UINT *, NX_WEB_HTTP_SERVER_DATE *))
```

HTTP Chunked Transfer Coding Support

When the total length of HTTP message cannot be determined before sending it, the Chunked Transfer Coding feature can be used to send messages as series of chunks without the “Content-Length” header field. This feature is supported in all HTTP request and response messages. As a receiver, this feature is supported, and the chunk header is processed transparently by internal logic. As a sender, the API *nx_web_http_client_request_chunked_set* and *nx_web_http_server_response_chunked_set* must be invoked by client and server respectively.

```
UINT  nx_web_http_client_request_chunked_set(NX_WEB_HTTP_CLIENT *client_ptr,
                                             UINT chunk_size,
                                             NX_PACKET *packet_ptr);

UINT  nx_web_http_server_response_chunked_set(NX_WEB_HTTP_SERVER *server_ptr,
                                              UINT chunk_size,
                                              NX_PACKET *packet_ptr);
```

For more details on these services, see their descriptions in Chapter 3 “Description of HTTP Services”.

HTTP Multipart Support

Multipurpose Internet Mail Extensions (MIME) was originally intended for the SMTP protocol, but its use has spread to HTTP. MIME allows

messages to contain mixed message types (e.g. image/jpg and text/plain) within the same message. The NetX Web HTTP Server has services to determine content type in HTTP messages containing MIME from the Client. To enable HTTP multipart support and use these services, the configuration option **NX_WEB_HTTP_MULTIPART_ENABLE** must be defined.

```
UINT  nx_web_http_server_get_entity_header(NX_WEB_HTTP_SERVER *server_ptr,
                                           NX_PACKET **packet_pptr,
                                           UCHAR *entity_header_buffer,
                                           ULONG buffer_size);

UINT  nx_web_http_server_get_entity_content(NX_WEB_HTTP_SERVER *server_ptr,
                                            NX_PACKET **packet_pptr,
                                            ULONG *available_offset,
                                            ULONG *available_length)
```

For more details on the use of these services, see their description in Chapter 3 “Description of HTTP Services”.

HTTP Multi-Thread Support

The NetX Web HTTP Client services can be called from multiple threads simultaneously. However, read or write requests for a particular HTTP Client instance should be done in sequence from the same thread.

If using HTTPS, NetX Web HTTP Client services may be called from multiple threads but due to the added complexity of the underlying TLS functionality each thread should have a single, independent HTTP Client instance (NX_WEB_HTTP_CLIENT control structure).

HTTP RFCs

NetX Web HTTP is compliant with RFC1945 “Hypertext Transfer Protocol/1.0”, RFC 2616 “Hypertext Transfer Protocol – HTTP/1.1”, RFC 2581 “TCP Congestion Control”, RFC 1122 “Requirements for Internet Hosts”, and related RFCs.

For HTTPS, NetX Web HTTP is compliant with RFC 2818 “HTTP over TLS”.

Chapter 2

Installation and Use of HTTP and HTTPS

This chapter contains a description of various issues related to installation, setup, and usage of the NetX Web HTTP component.

Product Distribution

HTTP for NetX is shipped on a single CD-ROM compatible disk. The package includes three source files, two include files, and a file that contains this document, as follows:

nx_web_http_common.h	Common header file for NetX Web HTTP
nx_web_http_client.h	Header file for HTTP Client for NetX Web
nx_web_http_server.h	Header file for HTTP Server for NetX Web
nx_web_http_client.c	C Source file for HTTP Client for NetX Web
nx_web_http_server.c	C Source file for HTTP Server for NetX Web
nx_tcpserver.c	C Source file for multiple TCP sockets
nx_md5.c	MD5 digest algorithms
filex_stub.h	Stub file if FileX is not present
nx_web_http.pdf	Description of HTTP for NetX Web
demo_netx_web_http.c	NetX Web HTTP demonstration

HTTP Installation

In order to use NetX Web HTTP, the entire distribution mentioned previously should be copied to the same directory where NetX Duo is installed. For example, if NetX Duo is installed in the directory "*threadx**arm7**green*" then the *nx_web_http_client.h* and *nx_web_http_client.*c for NetX Web HTTP Client applications, and *nx_web_http_server.h*, *nx_web_http_server.c,* and *nx_tcpserver.c* for NetX Web HTTP Server applications. For both client and server applications, nx_web_http_common.h must be in this directory as well. *nx_md5.c* should also be copied into this directory if digest authentication is being used. For the demo 'ram driver' application HTTP Client and Server files should be copied into the same directory.

If using TLS, you should have a separate NetX Secure directory containing the TLS source files.

Using HTTP

Using NetX Web HTTP is easy. Basically, the application code must include *nx_web_http_client.h* and/or *nx_web_http_server.h* after it includes *tx_api.h, fx_api.h,* and *nx_api.h* (*nx_web_http_common.h* is automatically included). Those headers enable the application to use ThreadX, FileX, and NetX Duo, respectively. For HTTPS support, the headers must be included after the *nx_secure_tls.h* file is included to bring in TLS support.

Once the HTTP header files are included, the application code is then able to make the HTTP function calls specified later in this guide. The application must also link with *nx_web_http_client.c* for HTTP(S) clients, *nx_web_http_server.c* and *nx_tcpserver.c* for HTTP(S) servers, and *netx_md5.c* (for digest authentication) in the build process. These files must be compiled in the same manner as other application files and its object form must be linked along with the files of the application. This is all that is required to use NetX Web HTTP.

Note that if NX_WEB_HTTP_DIGEST_ENABLE is not specified in the build process, the *md5.c* file does not need to be added to the application. Similarly, if no HTTP Client capabilities are required, the *nx_web_http_client.c* file may be omitted and if no HTTP Server capabilities are required, *nx_web_http_server.c* may be omitted.

Note that unless NX_WEB_HTTPS_ENABLE is defined in order to enable HTTPS (instead of using only plaintext HTTP) then NetX Secure TLS does not need to be in the build.

Note also that since HTTP utilizes NetX TCP services, TCP must be enabled with the *nx_tcp_enable()* call prior to using HTTP.

Note that when using HTTPS with NetX Secure TLS, TLS must be initialized with *nx_secure_tls_initialize()* prior to calling HTTPS routines.

Small Example System

An example of how easy it is to use NetX Web HTTP is described in Figure 1.1 below. In this example, the HTTP include file *nx_web_http_client.h and nx_web_http_server.h* are brought in at line 11

(*netx_web_http_common.h* is included automatically). Next, the HTTP Server is created in "*tx_application_define*" at line 208. Note that the HTTP Server control block "*Server*" was defined as a global variable at line 28 previously. After successful creation, an HTTP Server is started at line 232. At line 245 the HTTP Client is created. And finally, the Client writes the file at line 252 and reads the file back at line 291.

```
1     /* This is a small demo of HTTP on the high-performance NetX Duo TCP/IP stack.
2        This demo relies on ThreadX, NetX DUo, and FileX to show a simple HTML
3        transfer from the client and then back from the server.  */
4
5     #include  "tx_api.h"
6     #include  "fx_api.h"
7     #include  "nx_api.h"
8     #include  “nx_crypto.h”
9     #include  “nx_secure_tls_api.h”
10    #include  “nx_secure_x509.h”
11    #include  "nx_web_http_client.h"
12    #include  "nx_web_http_server.h"
13    #define     DEMO_STACK_SIZE         4096
14
15
16    /* Define the ThreadX and NetX object control blocks...  */
17
18    TX_THREAD               thread_0;
19    TX_THREAD               thread_1;
20    NX_PACKET_POOL          pool_0;
21    NX_PACKET_POOL          pool_1;
22    NX_IP                   ip_0;
23    NX_IP                   ip_1;
24    FX_MEDIA                ram_disk;
25
26    /* Define HTTP objects.  */
27
28    NX_WEB_HTTP_SERVER      my_server;
29    NX_WEB_HTTP_CLIENT      my_client;
30
31    /* Define the counters used in the demo application...  */
32
33    ULONG                   error_counter;
34
35
36    /* Define the RAM disk memory.  */
37
38    UCHAR                   ram_disk_memory[32000];
39
40    /* Include cryptographic routines for TLS. */
41    extern const NX_SECURE_TLS_CRYPTO nx_crypto_tls_ciphers;
42
43    /* Define TLS data for HTTPS. */
44    CHAR crypto_metadata[8928 * NX_WEB_HTTP_SESSION_MAX];
45    UCHAR tls_packet_buffer[16500];
46
47    /* Define certificate containers. The server certificate is used to identify the NetX
48       Web HTTPS server and the trusted certificate is used by the client to verify the
49       server’s identity certificate.  */
50    NX_SECURE_X509_CERT server_certificate;
51    NX_SECURE_X509_CERT trusted_certificate;
52
53    /* Remote certificates need both an NX_SECURE_X509_CERT container and an associated
54       buffer. The number of certificates depends on the remote host, but usually at least
55       two certificates will be sent – the identity certificate for the host and the
56       certificate that issued the identity certificate. */
57    NX_SECURE_X509_CERT remote_certificate, remote_issuer;
58    UCHAR remote_cert_buffer[2000];
59    UCHAR remote_issuer_buffer[2000];
60
61
62
```

```
63    /* Certificate information for server and client (see NetX Secure TLS reference on X.509
64       certificates for more information). Arrays are populated with binary versions Of
65       certificates and keys and the corresponding “len” variables are assigned the lengths
66       of that data. Trusted certificates do not need a private key. */
67     const UCHAR server_cert_der[] = { … };
68     const UINT  server_cert_derlen = … ;
69     const UCHAR server_cert_key_der[] = { … };
70     const UINT  server_cert_key_derlen = … ;
71
72     const UCHAR trusted_cert_der[] = { … };
73     const UINT  trusted_cert_derlen = … ;
74
75
76     /* Define function prototypes.  */
77
78     void    thread_0_entry(ULONG thread_input);
79     VOID    _fx_ram_driver(FX_MEDIA *media_ptr) ;
80     void    _nx_ram_network_driver(struct NX_IP_DRIVER_STRUCT *driver_req);
81     UINT    authentication_check(NX_WEB_HTTP_SERVER *server_ptr, UINT request_type,
82                 CHAR *resource, CHAR **name, CHAR **password, CHAR **realm);
83     UINT tls_setup_callback(NX_WEB_HTTP_CLIENT *client_ptr,
84                             NX_SECURE_TLS_SESSION *tls_session)
85
86     /* Define the application's authentication check.  This is called by
87        the HTTP server whenever a new request is received.  */
88     UINT  authentication_check(NX_WEB_HTTP_SERVER *server_ptr, UINT request_type,
89                 CHAR *resource, CHAR **name, CHAR **password, CHAR **realm)
90     {
91
92         /* Just use a simple name, password, and realm for all
93            requests and resources.  */
94         *name =     "name";
95         *password = "password";
96         *realm =    "NetX Web HTTP demo";
97
98         /* Request basic authentication.  */
99         return(NX_WEB_HTTP_BASIC_AUTHENTICATE);
100    }
101
102    /* Define the TLS setup callback for HTTPS Client. This function is invoked when the
103       HTTPS client is started - all TLS per-session initialization should go in here. */
104    UINT tls_setup_callback(NX_WEB_HTTP_CLIENT *client_ptr,
105                            NX_SECURE_TLS_SESSION *tls_session)
106    {
107    UINT status;
108
109       /* Initialize and create TLS session. */
110       nx_secure_tls_session_create(tls_session, &nx_crypto_tls_ciphers,
111                                            crypto_metadata, sizeof(crypto_metadata));
112
113
114       /* Allocate space for packet reassembly. */
115       nx_secure_tls_session_packet_buffer_set(tls_session, tls_packet_buffer,
116                                               sizeof(tls_packet_buffer));
117
118
119       /* Add a CA Certificate to our trusted store for verifying incoming server
120          certificates. */
121       nx_secure_x509_certificate_initialize(&trusted_certificate, trusted_cert_der,
122                                             trusted_cert_der_len, NX_NULL, 0, NULL, 0,
123                                             NX_SECURE_X509_KEY_TYPE_NONE);
124       nx_secure_tls_trusted_certificate_add(tls_session, &trusted_certificate);
125
126       /* Need to allocate space for the certificate coming in from the remote host. */
127       nx_secure_tls_remote_certificate_allocate(tls_session, &remote_certificate,
128                                                 remote_cert_buffer,
129                                                 sizeof(remote_cert_buffer));
130       nx_secure_tls_remote_certificate_allocate(tls_session,
131                                         &remote_issuer, remote_issuer_buffer,
132                                         sizeof(remote_issuer_buffer));
133
134       return(NX_SUCCESS);
135    }
136
137
138
```

```
139   /* Define main entry point.  */
140
141    int main()
142    {
143
144        /* Enter the ThreadX kernel.  */
145        tx_kernel_enter();
146    }
147
148
149
150   /* Define what the initial system looks like.  */
151    void    tx_application_define(void *first_unused_memory)
152    {
153
154    CHAR    *pointer;
155
156
157        /* Setup the working pointer.  */
158        pointer =  (CHAR *) first_unused_memory;
159
160        /* Create the main thread.  */
161        tx_thread_create(&thread_0, "thread 0", thread_0_entry, 0,
162                pointer, DEMO_STACK_SIZE,
163                2, 2, TX_NO_TIME_SLICE, TX_AUTO_START);
164        pointer = pointer + DEMO_STACK_SIZE;
165
166        /* Initialize the NetX system.  */
167        nx_system_initialize();
168
169        /* Create packet pool.  */
170        nx_packet_pool_create(&pool_0, "NetX Packet Pool 0",
171                                                    600, pointer, 8192);
172        pointer = pointer + 8192;
173
174        /* Create an IP instance.  */
175        nx_ip_create(&ip_0, "NetX IP Instance 0", IP_ADDRESS(1, 2, 3, 4),
176                            0xFFFFFF00UL, &pool_0, _nx_ram_network_driver,
177                            pointer, 4096, 1);
178        pointer =  pointer + 4096;
179
180        /* Create another packet pool. */
181        nx_packet_pool_create(&pool_1, "NetX Packet Pool 1", 600, pointer, 8192);
182        pointer = pointer + 8192;
183
184        /* Create another IP instance.  */
185        nx_ip_create(&ip_1, "NetX IP Instance 1", IP_ADDRESS(1, 2, 3, 5),
186                            0xFFFFFF00UL, &pool_1, _nx_ram_network_driver,
187                            pointer, 4096, 1);
188        pointer = pointer + 4096;
189
190        /* Enable ARP and supply ARP cache memory for IP Instance 0.  */
191        nx_arp_enable(&ip_0, (void *) pointer, 1024);
192        pointer = pointer + 1024;
193
194        /* Enable ARP and supply ARP cache memory for IP Instance 1.  */
195        nx_arp_enable(&ip_1, (void *) pointer, 1024);
196        pointer = pointer + 1024;
197
198        /* Enable TCP processing for both IP instances.  */
199        nx_tcp_enable(&ip_0);
200        nx_tcp_enable(&ip_1);
201
202        /* Open the RAM disk.  */
203        fx_media_open(&ram_disk, "RAM DISK",
204                        _fx_ram_driver, ram_disk_memory, pointer, 4096) ;
205        pointer += 4096;
206
207        /* Create the NetX Web HTTP Server.  */
208        nx_web_http_server_create(&my_server, "My HTTP Server", &ip_1,
209                                  NX_WEB_HTTPS_SERVER_PORT, &ram_disk,
210                                  pointer, 4096, &pool_1, authentication_check, NX_NULL);
211        pointer = pointer + 4096;
212
213
214
```

```
215    /* The TLS server needs an identity certificate which is imported as a binary DER-
216           encoded X.509 certificate and it's associated private key (e.g. DER-encoded PKCS#1
217           RSA private key). */
218        nx_secure_x509_certificate_initialize(&server_certificate, server_cert_der,
219                                server_cert_der_len, NX_NULL, 0,
220                                server_cert_key_der, server_cert_key_der_len,
221                                NX_SECURE_X509_KEY_TYPE_RSA_PKCS1_DER);
222
223        /* Setup TLS session data for the TCP server. This enables TLS and HTTPS for the
224           server.  */
225        nx_web_http_server_secure_configure(&my_server, &nx_crypto_tls_ciphers,
226                                crypto_metadata, sizeof(crypto_metadata),
227                                tls_packet_buffer, sizeof(tls_packet_buffer),
228                                &server_certificate, NX_NULL, 0, NX_NULL, 0, NX_NULL, 0);
229
230
231        /* Start the HTTP Server.  */
232        nx_web_http_server_start(&my_server);
233    }
234
235
236    /* Define the test thread.  */
237    void    thread_0_entry(ULONG thread_input)
238    {
239
240    NX_PACKET   *my_packet;
241    UINT        status;
242
243
244        /* Create an HTTP client instance.  */
245        status = nx_web_http_client_create(&my_client, "My Client", &ip_0, &pool_0, 600);
246
247        /* Check status.  */
248        if (status)
249            error_counter++;
250
251        /* Prepare to send the simple 103-byte HTML file to the Server over HTTPS.  */
252        status = nx_web_http_client_put_secure_start(&my_client, IP_ADDRESS(1,2,3,5),
253                                        NX_WEB_HTTPS_SERVER_PORT, "/test.htm", "name",
254                                        "password", 103, tls_setup_callback, 50);
255
256        /* Check status.  */
257        if (status)
258            error_counter++;
259
260        /* Allocate a packet.  */
261        status = nx_web_http_client_request_packet_allocate(&pool_0, &my_packet,
262                                                            NX_TCP_PACKET, NX_WAIT_FOREVER);
263
264        /* Check status.  */
265        if (status != NX_SUCCESS)
266            return;
267
268        /* Build a simple 103-byte HTML page.  */
269        nx_packet_data_append(my_packet, "<HTML>\r\n", 8,
270                              &pool_0, NX_WAIT_FOREVER);
271        nx_packet_data_append(my_packet,
272                    "<HEAD><TITLE>NetX HTTP Test</TITLE></HEAD>\r\n", 44,
273                              &pool_0, NX_WAIT_FOREVER);
274        nx_packet_data_append(my_packet, "<BODY>\r\n", 8,
275                              &pool_0, NX_WAIT_FOREVER);
276        nx_packet_data_append(my_packet, "<H1>NetX Test Page</H1>\r\n", 25,
277                              &pool_0, NX_WAIT_FOREVER);
278        nx_packet_data_append(my_packet, "</BODY>\r\n", 9,
279                              &pool_0, NX_WAIT_FOREVER);
280        nx_packet_data_append(my_packet, "</HTML>\r\n", 9,
281                              &pool_0, NX_WAIT_FOREVER);
282
283        /* Complete the PUT by writing the total length.  */
284        status =  nx_web_http_client_put_packet(&my_client, my_packet, 50);
285
286        /* Check status.  */
287        if (status)
288            error_counter++;
289
290        /* Now GET the file back!  */
291        status =  nx_web_http_client_get_secure_start(&my_client, IP_ADDRESS(1,2,3,5),
292                                NX_WEB_HTTPS_SERVER_PORT, "/test.htm", "name", "password",
293                                tls_setup_callback, 50);
294
295
```

```
296         /* Check status.  */
297         if (status)
298             error_counter++;
299
300         /* Get a packet.  */
301         status =  nx_web_http_client_response_body_get(&my_client, &my_packet, 20);
302
303         /* Check for an error.  */
304         if ((status) || (my_packet -> nx_packet_length != 103))
305             error_counter++;
306
307         /* Check to see if we have a packet.  */
308         if (status == NX_SUCCESS)
309         {
310
311             /* Yes, release it!  */
312             nx_packet_release(my_packet);
313         }
314
315         /* Make sure TLS shuts down properly. */
316         nx_web_http_client_delete(&my_client);
317
318         /* Flush the media.  */
319         fx_media_flush(&ram_disk);
320
321
322     }
```

Figure 1.1 Example of HTTPS use with NetX and NetX Secure TLS

Configuration Options

There are several configuration options for building HTTP for NetX. Following is a list of all options, where each is described in detail. The default values are listed but can be redefined prior to inclusion of *nx_web_http_client.h and nx_web_http_server.h*:

Define	Meaning
NX_DISABLE_ERROR_CHECKING	Defined, this option removes the basic HTTP error checking. It is typically used after the application has been debugged.
NX_WEB_HTTP_SERVER_PRIORITY	The priority of the HTTP Server thread. By default, this value is defined as 16 to specify priority 16.
NX_WEB_HTTP_NO_FILEX	Defined, this option provides a stub for FileX dependencies. The HTTP Client will function without any change if this option is defined. The HTTP Server will need to either be modified or the user will have to create a handful of FileX services in order to function properly.
NX_WEB_HTTP_TYPE_OF_SERVICE	Type of service required for the HTTP TCP requests. By default, this value is defined as NX_IP_NORMAL to indicate normal IP packet service.
NX_WEB_HTTP_SERVER_THREAD_TIME_SLICE	The number of timer ticks the Server thread is allowed to run before yielding to threads of the same priority. The default value is 2.

NX_WEB_HTTP_FRAGMENT_OPTION
Fragment enable for HTTP TCP requests. By default, this value is NX_DONT_FRAGMENT to disable HTTP TCP fragmenting.

NX_WEB_HTTP_SERVER_WINDOW_SIZE
Server socket window size. By default, this value is 2048 bytes.

NX_WEB_HTTP_TIME_TO_LIVE Specifies the number of routers this packet can pass before it is discarded. The default value is set to 0x80.

NX_WEB_HTTP_SERVER_TIMEOUT Specifies the number of ThreadX ticks that internal services will suspend for. The default value is set to 10 seconds (10 * *NX_IP_PERIODIC_RATE*).

.

NX_WEB_HTTP_SERVER_TIMEOUT_ACCEPT
Specifies the number of ThreadX ticks that internal services will suspend for in internal *nx_tcp_server_socket_accept()* calls. The default value is set to (10 * *NX_IP_PERIODIC_RATE*).

NX_WEB_HTTP_SERVER_TIMEOUT_DISCONNECT
Specifies the number of ThreadX ticks that internal services will suspend for in internal *nx_tcp_socket_disconnect()* calls. The default value is set to 10 seconds (10 * *NX_IP_PERIODIC_RATE*).

.

NX_WEB_HTTP_SERVER_TIMEOUT_RECEIVE
Specifies the number of ThreadX ticks that internal services will suspend for in internal *nx_tcp_socket_receive()* calls. The default value is set to 10 seconds (10 * *NX_IP_PERIODIC_RATE*).

NX_WEB_HTTP_SERVER_TIMEOUT_SEND
Specifies the number of ThreadX ticks that internal services will suspend for in internal *nx_tcp_socket_send()* calls. The default value is set to 10 seconds (10 * *NX_IP_PERIODIC_RATE*).

NX_WEB_HTTP_MAX_HEADER_FIELD
Specifies the maximum size of the HTTP header field. The default value is 256.

NX_WEB_HTTP_MULTIPART_ENABLE
If defined, enables HTTP Server to support multipart HTTP requests.

NX_WEB_HTTP_SERVER_MAX_PENDING
Specifies the number of connections that can be queued for the HTTP Server. The default value is set to twice the maximum number of server sessions.

NX_WEB_HTTP_MAX_RESOURCE Specifies the number of bytes allowed in a client supplied *resource name*. The default value is set to 40.

NX_WEB_HTTP_MAX_NAME Specifies the number of bytes allowed in a client supplied *username*. The default value is set to 20.

NX_WEB_HTTP_MAX_PASSWORD Specifies the number of bytes allowed in a client supplied *password*. The default value is set to 20.

NX_WEB_HTTP_SERVER_SESSION_MAX
Specifies the number of simultaneous sessions for an HTTP or HTTPS server. A TCP socket and a TLS session (if HTTPS is enabled) are allocated for each session. The default value is set to 2.

NX_WEB_HTTPS_ENABLE If defined, this macro enables TLS and HTTPS. Leave undefined to free up resources if only plaintext HTTP is desired. By default, this macro is not defined.

NX_WEB_HTTP_SERVER_MIN_PACKET_SIZE
Specifies the minimum size of the packets in the pool specified at server creation. The minimum size is needed to ensure the complete HTTP header can be contained in one packet. The default value is set to 600.

NX_WEB_HTTP_CLIENT_MIN_PACKET_SIZE
Specifies the minimum size of the packets in the pool specified at Client creation. The minimum size is needed to ensure the complete HTTP header can be contained in one packet. The default value is set to 600.

NX_WEB_HTTP_SERVER_RETRY_SECONDS
Set the Server socket retransmission timeout in seconds. The default value is set to 2.

NX_WEB_HTTP_ SERVER_RETRY_MAX

This sets the maximum number of retransmissions on Server socket. The default value is set to 10.

NX_WEB_HTTP_ SERVER_RETRY_SHIFT

This value is used to set the next retransmission timeout. The current timeout is multiplied by the number of retransmissions thus far, shifted by the value of the socket timeout shift. The default value is set to 1 for doubling the timeout.

NX_WEB_HTTP_SERVER_RETRY_TRANSMIT_QUEUE_DEPTH

This specifies the maximum number of packets that can be enqueued on the Server socket retransmission queue. If the number of packets enqueued reaches this number, no more packets can be sent until one or more enqueued packets are released. The default value is set to 20.

Chapter 3

Description of HTTP Services

This chapter contains a description of all NetX Web HTTP services (listed below) in alphabetical order.

In the “Return Values” section in the following API descriptions, values in **BOLD** are not affected by the **NX_DISABLE_ERROR_CHECKING** define that is used to disable API error checking, while non-bold values are completely disabled.

HTTP Client services:

nx_web_http_client_connect
Connect to a plaintext HTTP server

nx_web_http_client_create
Create an HTTP Client Instance

nx_web_http_client_delete
Delete an HTTP Client instance

nx_web_http_client_delete_start
Send a plaintext HTTP DELETE request.

nx_web_http_client_delete_secure_start
Send an encrypted HTTPS DELETE request.

nx_web_http_client_get_start
Start an HTTP GET request

nx_web_http_client_get_secure_start
Start a secure HTTPS GET request

nx_web_http_client_head_start
Start an HTTP HEAD request

nx_web_http_client_head_secure_start
Start a secure HTTPS HEAD request

nx_web_http_client_request_packet_allocate
Allocate a HTTP(S) packet

nx_web_http_client_post_start
Start an HTTP POST request

nx_web_http_client_post_secure_start
Start a secure HTTPS POST request

nx_web_http_client_put_start
Start an HTTP PUT request

nx_web_http_client_put_secure_start
Start a secure HTTPS PUT request

nx_web_http_client_put_packet
Send next resource data packet

nx_web_http_client_request_chunked_set
Set chunked transfer for HTTP(S) request

nx_web_http_client_request_header_add
Add custom HTTP(S) header to HTTP(S) request

nx_web_http_client_request_initialize
Create a *custom HTTP(S) request*

nx_web_http_client_request_packet_send
Send HTTP(S) request packet to remote server

nx_web_http_client_request_send
Send HTTP(S) request header to remote server

nx_web_http_client_response_body_get
Get response data packets

nx_web_http_client_response_header_callback_set
Set a callback to be invoked when a response header is parsed

nx_web_http_client_secure_connect
Connect to an encrypted HTTPS server

HTTP Server services:

nx_web_http_server_cache_info_callback_set
Set callback to retrieve age and last modified date of specified URL

nx_web_http_server_callback_data_send
Send HTTP data from callback function

nx_web_http_server_callback_generate_response_header
Create response header in callback functions

nx_web_http_server_callback_packet_send
Send an HTTP packet from an HTTP callback

nx_web_http_server_callback_response_send
Send response from callback function

nx_web_http_server_content_get
Get content from the request

nx_web_http_server_content_get_extended
Get content from the request; supports empty (zero Content Length) requests

nx_web_http_server_content_length_get
Get length of content in the request

nx_web_http_server_create
Create an HTTP Server instance

nx_web_http_server_delete
Delete an HTTP Server instance

nx_web_http_server_get_entity_content
Return size and location of entity content in URL

nx_web_http_server_get_entity_header
Extract URL entity header into specified buffer

nx_web_http_server_gmt_callback_set
Set callback to retrieve GMT date and time

nx_web_http_server_invalid_userpassword_notify_set
Set callback for when invalid username and password is received in a Client request

nx_web_http_server_mime_maps_additional_set
Define additional mime maps for HTML

nx_web_http_server_response_packet_allocate
Allocate a HTTP(S) packet

nx_web_http_server_packet_content_find
Extract content length in HTTP header and set pointer to start of content data

nx_web_http_server_packet_get
Receive client packet directly

nx_web_http_server_param_get
Get parameter from the request

nx_web_http_server_query_get
Get query from the request

nx_web_http_client_response_chunked_set
Set chunked transfer for HTTP(S)

nx_web_http_server_secure_configure
Configure an HTTP Server to use HTTPS with TLS

nx_web_http_server_start
Start the HTTP or HTTPS Server

nx_web_http_server_stop
Stop the HTTP Server

nx_web_http_server_type_get
Extract HTTP type e.g. text/plain from header

nx_web_http_client_connect

Open a plaintext socket to an HTTP server for custom requests

Prototype

```
UINT nx_web_http_client_connect(NX_WEB_HTTP_CLIENT *client_ptr,
                                NXD_ADDRESS *server_ip,
                                UINT server_port, ULONG wait_option);
```

Description

This method is for **plaintext** HTTP.

This service opens a plaintext TCP socket to an HTTP server but does not send any requests. Requests are created with n*x_web_http_client_request_initialize()* and sent using *nx_web_http_client_request_send()*. Custom HTTP headers may be added to the request using *nx_web_http_client_request_header_add()*.

Use of this service enables an application to add any number of custom headers to the request. This allows for customized HTTP requests intended for specific applications.

Note that the `nx_web_http_client_*_start` methods are provided for convenience (e.g. `nx_web_http_client_get_start`) and handle the request generation and socket connection. You can use those services instead of *`nx_web_http_client_connect`* and the related routines if you do not need custom HTTP headers in your requests.

Input Parameters

client_ptr	Pointer to HTTP Client control block.
server_ip	IP address of remote HTTP server.
server_port	Port on remote HTTP server (e.g. 80 for HTTP).
wait_option	Defines how long the service will wait for underlying network operations. The wait options are defined as follows: **timeout value** (0x00000001 through 0xFFFFFFFE) **NX_WAIT_FOREVER** (0xFFFFFFFF) Selecting NX_WAIT_FOREVER causes the calling thread to suspend indefinitely until the HTTP Server responds to the request. Selecting a numeric value (0x1-0xFFFFFFFE) specifies the maximum number of timer-ticks to stay suspended while waiting for the HTTP Server response.

Return Values

NX_SUCCESS	(0x00)	Successful connection of TCP socket.
NX_PTR_ERROR	(0x07)	Invalid pointer input
NX_WEB_HTTP_NOT_READY	(0x3000A)	Another request is already in progress.

Allowed From

Threads

Example

```
NXD_ADDRESS server_ip_addr;

/* Connect to a remote HTTP server. */
server_ip_addr.nxd_ip_version = NX_IP_VERSION_V4;
server_ip_addr.nxd_ip_address.v4 = IP_ADDRESS(1,2,3,5);
nx_web_http_client_connect(&my_client, &server_ip_address,
                           NX_WEB_HTTP_SERVER_PORT, NX_WAIT_FOREVER);


/* Create a new GET request on the HTTP client instance. */
nx_web_http_client_request_initialize(&my_client,
                                      NX_WEB_HTTP_METHOD_GET,
                                      "https://192.168.1.150/test.txt ",
                                      0, /* Used by PUT and POST only */
                                      NX_FALSE,
                                      NX_NULL, /* username */
                                      NX_NULL, /* password */
                                      NX_WAIT_FOREVER);

/* Add a custom header to the GET request we just created. */
status = nx_web_http_client_request_header_add(&my_client, “Server”, 6,
                             “NetX Web HTTPS Server”, 21, NX_WAIT_FOREVER);

/* Start the GET operation to get a response from the HTTPS server. */
status = nx_web_http_client_request_send(&my_client, 1000);

/* At this point, we need to handle the response from the server by repeatedly
calling nx_web_http_client_response_body_get until the entire response is
retrieved. */

get_status = NX_SUCCESS;

while(get_status != NX_WEB_HTTP_GET_DONE)
{
   get_status = nx_web_http_client_response_body_get(&my_client, &receive_packet,
                                                     NX_WAIT_FOREVER);
   /* Process response packets… */
}
```

nx_web_http_client_create

Create an HTTP Client Instance

Prototype

```
UINT nx_web_http_client_create(NX_WEB_HTTP_CLIENT *client_ptr,
            CHAR *client_name, NX_IP *ip_ptr, NX_PACKET_POOL *pool_ptr,
            ULONG window_size);
```

Description

This service creates an HTTP Client instance on the specified IP instance. The Client instance can be used for either HTTP or HTTPS. See the services *nx_web_http_client_connect()* and *nx_web_http_client_secure_connect()* for more information on starting an HTTP or HTTPS instance. Also refer to the API for *nx_web_http_client_get_**, *nx_web_http_client_put_**, *nx_web_http_client_post_** for simple invocations of GET, PUT, and POST methods.

Input Parameters

client_ptr	Pointer to HTTP Client control block.
client_name	Name of HTTP Client instance.
ip_ptr	Pointer to IP instance.
pool_ptr	Pointer to default packet pool. Note that the packets in this pool must have a payload large enough to handle the complete response header. This is defined by *NX_WEB_HTTP_CLIENT_MIN_PACKET_SIZE* in *nx_web_http_client.h*.
window_size	Size of the Client's TCP socket receive window.

Return Values

NX_SUCCESS	(0x00)	Successful HTTP Client create
NX_PTR_ERROR	(0x16)	Invalid HTTP, ip_ptr, or packet pool pointer
NX_WEB_HTTP_POOL_ERROR	(0x30009)	Invalid payload size in packet pool

Allowed From

Initialization, Threads

Example

```
/* Create the HTTP Client instance “my_client” on “ip_0”.  */
status = nx_web_http_client_create(&my_client, “my client”, &ip_0, &pool_0,
                                   8192);


/* If status is NX_SUCCESS an HTTP Client instance was successfully
   created.  */
```

nx_web_http_client_delete

Delete an HTTP Client Instance

Prototype

```
UINT nx_web_http_client_delete(NX_WEB_HTTP_CLIENT *client_ptr);
```

Description

This service deletes a previously created HTTP Client instance.

Input Parameters

client_ptr	Pointer to HTTP Client control block.

Return Values

NX_SUCCESS	(0x00)	Successful HTTP Client delete
NX_PTR_ERROR	(0x16)	Invalid HTTP pointer
NX_CALLER_ERROR	(0x11)	Invalid caller of this service

Allowed From

Threads

Example

```
/* Delete the HTTP Client instance "my_client."  */
status =  nx_web_http_client_delete(&my_client);

/* If status is NX_SUCCESS an HTTP Client instance was successfully
   deleted.  */
```

nx_web_http_client_delete_start

Start a plaintext HTTP DELETE request

Prototype

```
UINT nx_web_http_client_delete_start(NX_WEB_HTTP_CLIENT *client_ptr,
            NXD_ADDRESS ip_address, UINT server_port, CHAR *resource,
            CHAR *host, CHAR *username, CHAR *password,
            ULONG wait_option);
```

Description

This method is for **plaintext** HTTP.

This service attempts to send a DELETE request for the resource specified by “resource” pointer on the previously created HTTP Client instance. If this routine returns NX_SUCCESS, the application can then call *nx_web_http_client_response_body_get()* to retrieve the server’s response.

Note that the resource string can refer to a local file e.g. “/index.htm” or it can refer to another URL e.g. http://abc.website.com/index.htm if the HTTP Server indicates it supports referring GET requests.

Input Parameters

client_ptr	Pointer to HTTP Client control block.
ip_address	IP address of the HTTP Server.
server_port	Port on remote HTTP Server.
host	Null-terminated string of the server’s domain name. This string is transmitted in the HTTP Host header field. The host string cannot be NULL.
resource	Pointer to URL string for requested resource.
username	Pointer to optional user name for authentication.
password	Pointer to optional password for authentication.
wait_option	Defines how long the service will wait for the HTTP Client get start request. The wait options are defined as follows: **time out value** (0x00000001 through 0xFFFFFFFE) **NX_WAIT_FOREVER** (0xFFFFFFFF) Selecting NX_WAIT_FOREVER causes the calling thread to suspend indefinitely until the HTTP Server responds to the request.

Selecting a numeric value (0x1-0xFFFFFFFE) specifies the maximum number of timer-ticks to stay suspended while waiting for the HTTP Server response.

Return Values

NX_SUCCESS	(0x00)	Successfully sent HTTP Client DELETE request message
NX_WEB_HTTP_ERROR	(0x30000)	Internal HTTP Client error
NX_WEB_HTTP_NOT_READY	(0x3000A)	HTTP Client not ready
NX_WEB_HTTP_FAILED	(0x30002)	HTTP Client error communicating with the HTTP Server.
NX_WEB_HTTP_AUTHENTICATION_ERROR	(0x3000B)	Invalid name and/or password.
NX_PTR_ERROR	(0x07)	Invalid pointer input
NX_CALLER_ERROR	(0x11)	Invalid caller of this service.

Allowed From

Threads

Example

```
/* Connect to a remote HTTP server. */
server_ip_addr.nxd_ip_version = NX_IP_VERSION_V4;
server_ip_addr.nxd_ip_address.v4 = IP_ADDRESS(1,2,3,5);

/* Start the DELETE operation on the HTTP Client "my_client."  */
status =  nx_web_http_client_delete_start(&my_client, &server_ip_address,
                                          NX_WEB_HTTP_SERVER_PORT, "/TEST.HTM",
                                         "host.com", "myname", "mypassword",
                                         1000);


/* If status is NX_SUCCESS, the DELETE request for TEST.HTM is started and is so
   far successful. The client must now call nx_web_http_client_response_body_get
   to retrieve the response from the server. */
```

nx_web_http_client_delete_secure_start

Start an encrypted HTTPS DELETE request

Prototype

```
UINT nx_web_http_client_delete_secure_start(
            NX_WEB_HTTP_CLIENT *client_ptr,
            NXD_ADDRESS ip_address, UINT server_port, CHAR *resource,
            CHAR *host, CHAR *username, CHAR *password,
            UINT (*tls_setup)(NX_WEB_HTTP_CLIENT *client_ptr,
                              NX_SECURE_TLS_SESSION *tls_session),
            ULONG wait_option);
```

Description

This method is for **TLS-secured** HTTPS.

This service attempts to send a DELETE request for the resource specified by “resource” pointer on the previously created HTTP Client instance. If this routine returns NX_SUCCESS, the application can then call *nx_web_http_client_response_body_get()* to retrieve the server’s response.

Note that the resource string can refer to a local file e.g. “/index.htm” or it can refer to another URL e.g. http://abc.website.com/index.htm if the HTTP Server indicates it supports referring GET requests.

Input Parameters

client_ptr	Pointer to HTTP Client control block.
ip_address	IP address of the HTTP Server.
server_port	Port on remote HTTP Server.
resource	Pointer to URL string for requested resource. resource must be NULL-terminated.
host	Null-terminated string of the server’s domain name. This string is transmitted in the HTTP Host header field. The host string cannot be NULL.
username	Pointer to optional user name for authentication.
password	Pointer to optional password for authentication.
tls_setup	Callback used to setup TLS configuration. The application defines this callback to initialize TLS cryptography and credentials (e.g. certificates).
wait_option	Defines how long the service will wait for the HTTP Client get start request. The wait options are defined as follows:

time out value	(0x00000001 through 0xFFFFFFFE)

NX_WAIT_FOREVER (0xFFFFFFFF)

Selecting NX_WAIT_FOREVER causes the calling thread to suspend indefinitely until the HTTP Server responds to the request.

Selecting a numeric value (0x1-0xFFFFFFFE) specifies the maximum number of timer-ticks to stay suspended while waiting for the HTTP Server response.

Return Values

NX_SUCCESS	(0x00)	Successfully sent HTTP Client DELETE request message
NX_WEB_HTTP_ERROR	(0x30000)	Internal HTTP Client error
NX_WEB_HTTP_NOT_READY	(0x3000A)	HTTP Client not ready
NX_WEB_HTTP_FAILED	(0x30002)	HTTP Client error communicating with the HTTP Server.
NX_WEB_HTTP_AUTHENTICATION_ERROR	(0x3000B)	Invalid name and/or password.
NX_PTR_ERROR	(0x07)	Invalid pointer input
NX_CALLER_ERROR	(0x11)	Invalid caller of this service.

Allowed From

Threads

Example

```
/* Connect to a remote HTTP server. */
server_ip_addr.nxd_ip_version = NX_IP_VERSION_V4;
server_ip_addr.nxd_ip_address.v4 = IP_ADDRESS(1,2,3,5);

/* Start the DELETE operation on the HTTP Client "my_client."  */
status =  nx_web_http_client_delete_secure_start(&my_client, &server_ip_addr,
                                    NX_WEB_HTTPS_SERVER_PORT, "/TEST.HTM",
                                    "host.com", "myname", "mypassword",
                                    my_tls_setup_function, 1000);


/* If status is NX_SUCCESS, the DELETE request for TEST.HTM is started and is so
   far successful. The client must now call nx_web_http_client_response_body_get
   to retrieve the server's response. */
```

nx_web_http_client_get_start

Start a plaintext HTTP GET request

Prototype

```
UINT nx_web_http_client_get_start(NX_WEB_HTTP_CLIENT *client_ptr,
            NXD_ADDRESS ip_address, UINT server_port, CHAR *resource,
            CHAR *host, CHAR *username, CHAR *password,
            ULONG wait_option);
```

Description

This method is for **plaintext** HTTP.

This service attempts to GET the resource specified by “resource” pointer on the previously created HTTP Client instance. If this routine returns NX_SUCCESS, the application can then make multiple calls to *nx_web_http_client_response_body_get()* to retrieve packets of data corresponding to the requested resource content.

Note that the resource string can refer to a local file e.g. “/index.htm” or it can refer to another URL e.g. http://abc.website.com/index.htm if the HTTP Server indicates it supports referring GET requests.

Input Parameters

client_ptr	Pointer to HTTP Client control block.
ip_address	IP address of the HTTP Server.
server_port	Port on remote HTTP Server.
resource	Pointer to URL string for requested resource.
host	Null-terminated string of the server’s domain name. This string is transmitted in the HTTP Host header field. The host string cannot be NULL.
username	Pointer to optional user name for authentication.
password	Pointer to optional password for authentication.
wait_option	Defines how long the service will wait for the HTTP Client get start request. The wait options are defined as follows: **time out value** (0x00000001 through 0xFFFFFFFE) **NX_WAIT_FOREVER** (0xFFFFFFFF) Selecting NX_WAIT_FOREVER causes the calling thread to suspend indefinitely until the HTTP Server responds to the request.

Selecting a numeric value (0x1-0xFFFFFFFE) specifies the maximum number of timer-ticks to stay suspended while waiting for the HTTP Server response.

Return Values

NX_SUCCESS	(0x00)	Successfully sent HTTP Client GET start message
NX_WEB_HTTP_ERROR	(0x30000)	Internal HTTP Client error
NX_WEB_HTTP_NOT_READY	(0x3000A)	HTTP Client not ready
NX_WEB_HTTP_FAILED	(0x30002)	HTTP Client error communicating with the HTTP Server.
NX_WEB_HTTP_AUTHENTICATION_ERROR	(0x3000B)	Invalid name and/or password.
NX_PTR_ERROR	(0x07)	Invalid pointer input
NX_CALLER_ERROR	(0x11)	Invalid caller of this service.

Allowed From

Threads

Example

```
/* Connect to a remote HTTP server. */
server_ip_addr.nxd_ip_version = NX_IP_VERSION_V4;
server_ip_addr.nxd_ip_address.v4 = IP_ADDRESS(1,2,3,5);

/* Start the GET operation on the HTTP Client "my_client."  */
status =  nx_web_http_client_get_start(&my_client, &server_ip_addr,
                                       NX_WEB_HTTP_SERVER_PORT, "/TEST.HTM",
                                       "host.com", "myname", "mypassword",
                                       1000);


/* If status is NX_SUCCESS, the GET request for TEST.HTM is started and is so
   far successful. The client must now call nx_web_http_client_response_body_get
   multiple times to retrieve the content associated with TEST.HTM. */
```

nx_web_http_client_get_secure_start

Start an encrypted HTTPS GET request

Prototype

```
UINT nx_web_http_client_get_secure_start(
            NX_WEB_HTTP_CLIENT *client_ptr,
            NXD_ADDRESS ip_address, UINT server_port, CHAR *resource,
            CHAR *host, CHAR *username, CHAR *password,
            UINT (*tls_setup)(NX_WEB_HTTP_CLIENT *client_ptr,
                              NX_SECURE_TLS_SESSION *tls_session),
            ULONG wait_option);
```

Description

This method is for **TLS-secured** HTTPS.

This service attempts to GET the resource specified by “resource” pointer on the previously created HTTP Client instance. If this routine returns NX_SUCCESS, the application can then make multiple calls to *nx_web_http_client_response_body_get()* to retrieve packets of data corresponding to the requested resource content.

Note that the resource string can refer to a local file e.g. “/index.htm” or it can refer to another URL e.g. http://abc.website.com/index.htm if the HTTP Server indicates it supports referring GET requests.

Input Parameters

client_ptr	Pointer to HTTP Client control block.
ip_address	IP address of the HTTP Server.
server_port	Port on remote HTTP Server.
resource	Pointer to URL string for requested resource.
host	Null-terminated string of the server’s domain name. This string is transmitted in the HTTP Host header field. The host string cannot be NULL.
username	Pointer to optional user name for authentication.
password	Pointer to optional password for authentication.
tls_setup	Callback used to setup TLS configuration. The application defines this callback to initialize TLS cryptography and credentials (e.g. certificates).
wait_option	Defines how long the service will wait for the HTTP Client get start request. The wait options are defined as follows:

time out value	(0x00000001 through 0xFFFFFFFE)

NX_WAIT_FOREVER (0xFFFFFFFF)

Selecting NX_WAIT_FOREVER causes the calling thread to suspend indefinitely until the HTTP Server responds to the request.

Selecting a numeric value (0x1-0xFFFFFFFE) specifies the maximum number of timer-ticks to stay suspended while waiting for the HTTP Server response.

Return Values

NX_SUCCESS	(0x00)	Successfully sent HTTP Client GET start message
NX_WEB_HTTP_ERROR	(0x30000)	Internal HTTP Client error
NX_WEB_HTTP_NOT_READY	(0x3000A)	HTTP Client not ready
NX_WEB_HTTP_FAILED	(0x30002)	HTTP Client error communicating with the HTTP Server.
NX_WEB_HTTP_AUTHENTICATION_ERROR	(0x3000B)	Invalid name and/or password.
NX_PTR_ERROR	(0x07)	Invalid pointer input
NX_CALLER_ERROR	(0x11)	Invalid caller of this service.

Allowed From

Threads

Example

```
/* Connect to a remote HTTP server. */
server_ip_addr.nxd_ip_version = NX_IP_VERSION_V4;
server_ip_addr.nxd_ip_address.v4 = IP_ADDRESS(1,2,3,5);

/* Start the GET operation on the HTTP Client "my_client."  */
status =  nx_web_http_client_get_secure_start(&my_client, &server_ip_addr,
                                    NX_WEB_HTTPS_SERVER_PORT, "/TEST.HTM",
                                    "host.com", "myname", "mypassword",
                                    my_tls_setup_function, 1000);


/* If status is NX_SUCCESS, the GET request for TEST.HTM is started and is so far
   successful. The client must now call nx_web_http_client_response_body_get()
   multiple times to retrieve the content associated with TEST.HTM. */
```

nx_web_http_client_head_start

Start a plaintext HTTP HEAD request

Prototype

```
UINT nx_web_http_client_head_start(NX_WEB_HTTP_CLIENT *client_ptr,
            NXD_ADDRESS ip_address, UINT server_port, CHAR *resource,
            CHAR *host, CHAR *username, CHAR *password,
            ULONG wait_option);
```

Description

This method is for **plaintext** HTTP.

This service attempts to retrieve the HEAD metadata for the resource specified by “resource” pointer on the previously created HTTP Client instance. If this routine returns NX_SUCCESS, the application can then call *nx_web_http_client_response_body_get()* to retrieve the response.

Note that the resource string can refer to a local file e.g. “/index.htm” or it can refer to another URL e.g. http://abc.website.com/index.htm if the HTTP Server indicates it supports referring GET requests.

Input Parameters

client_ptr	Pointer to HTTP Client control block.
ip_address	IP address of the HTTP Server.
server_port	Port on remote HTTP Server.
resource	Pointer to URL string for requested resource.
host	Null-terminated string of the server’s domain name. This string is transmitted in the HTTP Host header field. The host string cannot be NULL.
username	Pointer to optional user name for authentication.
password	Pointer to optional password for authentication.
wait_option	Defines how long the service will wait for the HTTP Client get start request. The wait options are defined as follows: **time out value** (0x00000001 through 0xFFFFFFFE) **NX_WAIT_FOREVER** (0xFFFFFFFF) Selecting NX_WAIT_FOREVER causes the calling thread to suspend indefinitely until the HTTP Server responds to the request.

Selecting a numeric value (0x1-0xFFFFFFFE) specifies the maximum number of timer-ticks to stay suspended while waiting for the HTTP Server response.

Return Values

NX_SUCCESS	(0x00)	Successfully sent HTTP Client HEAD request message
NX_WEB_HTTP_ERROR	(0x30000)	Internal HTTP Client error
NX_WEB_HTTP_NOT_READY	(0x3000A)	HTTP Client not ready
NX_WEB_HTTP_FAILED	(0x30002)	HTTP Client error communicating with the HTTP Server.
NX_WEB_HTTP_AUTHENTICATION_ERROR	(0x3000B)	Invalid name and/or password.
NX_PTR_ERROR	(0x07)	Invalid pointer input
NX_CALLER_ERROR	(0x11)	Invalid caller of this service.

Allowed From

Threads

Example

```
/* Connect to a remote HTTP server. */
server_ip_addr.nxd_ip_version = NX_IP_VERSION_V4;
server_ip_addr.nxd_ip_address.v4 = IP_ADDRESS(1,2,3,5);

/* Start the HEAD operation on the HTTP Client "my_client."  */
status =  nx_web_http_client_head_start(&my_client, &server_ip_addr,
                                        NX_WEB_HTTP_SERVER_PORT, "/TEST.HTM",
                                       "host.com", "myname", "mypassword",
                                       1000);


/* If status is NX_SUCCESS, the HEAD request for TEST.HTM is started and is so
   far successful. The client must now call nx_web_http_client_response_body_get
   to retrieve the response from the server. */
```

nx_web_http_client_head_secure_start

Start an encrypted HTTPS HEAD request

Prototype

```
UINT nx_web_http_client_head_secure_start(
            NX_WEB_HTTP_CLIENT *client_ptr,
            NXD_ADDRESS ip_address, UINT server_port, CHAR *resource,
            CHAR *host, CHAR *username, CHAR *password,
            UINT (*tls_setup)(NX_WEB_HTTP_CLIENT *client_ptr,
                              NX_SECURE_TLS_SESSION *tls_session),
            ULONG wait_option);
```

Description

This method is for **TLS-secured** HTTPS.

This service attempts to retrieve the HEAD metadata for the resource specified by “resource” pointer on the previously created HTTP Client instance. If this routine returns NX_SUCCESS, the application can then call *nx_web_http_client_response_body_get()* to retrieve the server’s response corresponding to the requested resource content.

Note that the resource string can refer to a local file e.g. “/index.htm” or it can refer to another URL e.g. http://abc.website.com/index.htm if the HTTP Server indicates it supports referring GET requests.

Input Parameters

client_ptr	Pointer to HTTP Client control block.
ip_address	IP address of the HTTP Server.
server_port	Port on remote HTTP Server.
resource	Pointer to URL string for requested resource.
host	Null-terminated string of the server’s domain name. This string is transmitted in the HTTP Host header field. The host string cannot be NULL.
username	Pointer to optional user name for authentication.
password	Pointer to optional password for authentication.
tls_setup	Callback used to setup TLS configuration. The application defines this callback to initialize TLS cryptography and credentials (e.g. certificates).
wait_option	Defines how long the service will wait for the HTTP Client get start request. The wait options are defined as follows:

time out value	(0x00000001 through 0xFFFFFFFE)

NX_WAIT_FOREVER (0xFFFFFFFF)

Selecting NX_WAIT_FOREVER causes the calling thread to suspend indefinitely until the HTTP Server responds to the request.

Selecting a numeric value (0x1-0xFFFFFFFE) specifies the maximum number of timer-ticks to stay suspended while waiting for the HTTP Server response.

Return Values

NX_SUCCESS	(0x00)	Successfully sent HTTP Client HEAD request message
NX_WEB_HTTP_ERROR	(0x30000)	Internal HTTP Client error
NX_WEB_HTTP_NOT_READY	(0x3000A)	HTTP Client not ready
NX_WEB_HTTP_FAILED	(0x30002)	HTTP Client error communicating with the HTTP Server.
NX_WEB_HTTP_AUTHENTICATION_ERROR	(0x3000B)	Invalid name and/or password.
NX_PTR_ERROR	(0x07)	Invalid pointer input
NX_CALLER_ERROR	(0x11)	Invalid caller of this service.

Allowed From

Threads

Example

```
/* Connect to a remote HTTP server. */
server_ip_addr.nxd_ip_version = NX_IP_VERSION_V4;
server_ip_addr.nxd_ip_address.v4 = IP_ADDRESS(1,2,3,5);

/* Start the HEAD operation on the HTTP Client "my_client."  */
status =  nx_web_http_client_head_secure_start(&my_client, &server_ip_addr,
                                    NX_WEB_HTTPS_SERVER_PORT, "/TEST.HTM",
                                    "host.com", "myname", "mypassword",
                                    my_tls_setup_function, 1000);


/* If status is NX_SUCCESS, the HEAD request for TEST.HTM is started and is so
   far successful. The client must now call nx_web_http_client_response_body_get
   to retrieve the server's response. */
```

nx_web_http_client_request_packet_allocate

Allocate a HTTP(S) packet

Prototype

```
UINT nx_web_http_client_request_packet_allocate(
                                        NX_WEB_HTTP_CLIENT *client_ptr,
                                        NX_PACKET **packet_ptr,
                                        ULONG wait_option);
```

Description

This service attempts to allocates a packet for Client HTTP(S).

Input Parameters

client_ptr	Pointer to HTTP Client control block.
packet_ptr	Pointer to allocated packet.
wait_option	Defines the wait time in ticks if there are no packets available in the packet pool. The wait options are defined as follows: **NX_NO_WAIT** (0x00000000) **NX_WAIT_FOREVER** (0xFFFFFFFF) **timeout in ticks** (0x00000001 through 0xFFFFFFFE)

Return Values

NX_SUCCESS	(0x00)	Successful packet allocate
NX_NO_PACKET	(0x01)	No packet available
NX_WAIT_ABORTED	(0x1A)	Requested suspension was aborted by a call to *tx_thread_wait_abort*.
NX_INVALID_PARAMETERS	(0x4D)	Packet size cannot support protocol.
NX_PTR_ERROR	(0x07)	Invalid pointer input
NX_CALLER_ERROR	(0x11)	Invalid caller of this service.

Allowed From

Threads

Example

```
/* Allocate a packet for HTTP(S) Client and suspend for a maximum of 5 timer
   ticks if the pool is empty. */
status =  nx_web_http_client_request_packet_allocate(&my_client, &packet_ptr, 5);

/* If status is NX_SUCCESS, the newly allocated packet pointer is found in the
    variable packet_ptr.  */
```

nx_web_http_client_post_start

Start an HTTP POST request

Prototype

```
UINT nx_web_http_client_post_start(NX_WEB_HTTP_CLIENT *client_ptr,
                                   NXD_ADDRESS ip_address, UINT server_port,
                                   CHAR *resource, CHAR *host,
                                   CHAR *username,  CHAR *password,
                                   ULONG total_bytes, ULONG wait_option);
```

Description

This method is for **plaintext** HTTP.

This service attempts to send a POST request with the specified resource to the HTTP Server at the supplied IP address and port. If this routine is successful, the application code should make successive calls to the *nx_web_http_client_put_packet* routine to send the resource contents to the HTTP Server.

Note that the resource string can refer to a local file e.g. “/index.htm” or it can refer to another URL e.g. http://abc.website.com/index.htm if the HTTP Server indicates it supports referring PUT requests.

Input Parameters

client_ptr	Pointer to HTTP Client control block.
ip_address	IP address of the HTTP Server.
server_port	TCP port on the remote HTTP Server.
resource	Pointer to URL string for resource to send to Server.
host	Null-terminated string of the server’s domain name. This string is transmitted in the HTTP Host header field. The host string cannot be NULL.
username	Pointer to optional user name for authentication.
password	Pointer to optional password for authentication.
total_bytes	Total bytes of resource being sent. Note that the combined length of all packets sent via subsequent calls to *nx_web_http_client_put_packet()* must equal this value.
wait_option	Defines how long the service will wait for the HTTP Client POST start. The wait options are defined as follows: **timeout value** (0x00000001 through 0xFFFFFFFE) **NX_WAIT_FOREVER** (0xFFFFFFFF)

Selecting NX_WAIT_FOREVER causes the calling thread to suspend indefinitely until the HTTP Server responds to the request.

Selecting a numeric value (0x1-0xFFFFFFFE) specifies the maximum number of timer-ticks to stay suspended while waiting for the HTTP Server response.

Return Values

NX_SUCCESS	(0x00)	Successfully sent POST request
NX_WEB_HTTP_USERNAME_TOO_LONG	(0x30012)	Username too large for buffer
NX_WEB_HTTP_NOT_READY	(0x3000A)	HTTP Client not ready
NX_PTR_ERROR	(0x07)	Invalid pointer input
NX_SIZE_ERROR	(0x09)	Invalid total size of resource
NX_CALLER_ERROR	(0x11)	Invalid caller of this service

Allowed From

Threads

Example

```
/* Connect to a remote HTTP server. */
server_ip_addr.nxd_ip_version = NX_IP_VERSION_V4;
server_ip_addr.nxd_ip_address.v4 = IP_ADDRESS(1,2,3,5);

/* Start an HTTP POST to post the 20-byte resource "/TEST.HTM" to the HTTP Server
   at IP address 1.2.3.5.  */
status =  nx_web_http_client_post_start(&my_client, &server_ip_addr,
                                        NX_WEB_HTTP_SERVER_PORT, "/TEST.HTM",
                                        "host.com", "myname", "mypassword", 20,
                                        NX_WAIT_FOREVER);

/* If status is NX_SUCCESS, the POST operation for TEST.HTM has successfully been
   started. */
```

nx_web_http_client_post_secure_start

Start an encrypted HTTPS POST request

Prototype

```
UINT nx_web_http_client_post_secure_start(
                    NX_WEB_HTTP_CLIENT *client_ptr,
                    NXD_ADDRESS ip_address, UINT server_port,
                    CHAR *resource, CHAR *host, CHAR *username,
                    CHAR *password, ULONG total_bytes,
                    UINT (*tls_setup)(NX_WEB_HTTP_CLIENT *client_ptr,
                                      NX_SECURE_TLS_SESSION *tls_session),
                    ULONG wait_option);
```

Description

This method is for **TLS-secured** HTTPS.

This service attempts to send a POST request with the specified resource to the HTTPS Server at the supplied IP address and port. If this routine is successful, the application code should make successive calls to the *nx_web_http_client_put_packet()* routine to send the resource contents to the HTTP Server.

Note that the resource string can refer to a local file e.g. “/index.htm” or it can refer to another URL e.g. http://abc.website.com/index.htm if the HTTP Server indicates it supports referring PUT requests.

Input Parameters

client_ptr	Pointer to HTTP Client control block.
ip_address	IP address of the HTTP Server.
server_port	TCP port on the remote HTTP Server.
resource	Pointer to URL string for resource to send to Server.
host	Null-terminated string of the server’s domain name. This string is transmitted in the HTTP Host header field. The host string cannot be NULL.
username	Pointer to optional user name for authentication.
password	Pointer to optional password for authentication.
total_bytes	Total bytes of resource being sent. Note that the combined length of all packets sent via subsequent calls to *nx_web_http_client_put_packet()* must equal this value.
tls_setup	Callback used to setup TLS configuration. The application defines this callback to initialize TLS cryptography and credentials (e.g. certificates).

wait_option Defines how long the service will wait for the HTTP Client POST start. The wait options are defined as follows:

timeout value (0x00000001 through 0xFFFFFFFE)
NX_WAIT_FOREVER (0xFFFFFFFF)

Selecting NX_WAIT_FOREVER causes the calling thread to suspend indefinitely until the HTTP Server responds to the request.

Selecting a numeric value (0x1-0xFFFFFFFE) specifies the maximum number of timer-ticks to stay suspended while waiting for the HTTP Server response.

Return Values

NX_SUCCESS	(0x00)	Successfully sent POST request
NX_WEB_HTTP_USERNAME_TOO_LONG	(0x30012)	Username too large for buffer
NX_WEB_HTTP_NOT_READY	(0x3000A)	HTTP Client not ready
NX_PTR_ERROR	(0x07)	Invalid pointer input
NX_SIZE_ERROR	(0x09)	Invalid total size of resource
NX_CALLER_ERROR	(0x11)	Invalid caller of this service

Allowed From

Threads

Example

```
/* Start an HTTP PUT to place the 20-byte resource “/TEST.HTM” on the HTTPS
Server
   at IP address 1.2.3.5.  */
/* Connect to a remote HTTP server. */
server_ip_addr.nxd_ip_version = NX_IP_VERSION_V4;
server_ip_addr.nxd_ip_address.v4 = IP_ADDRESS(1,2,3,5);

status =  nx_web_http_client_put_secure_start(&my_client, &server_ip_addr,
                                    NX_WEB_HTTPS_SERVER_PORT, “/TEST.HTM”,
                                    “host.com”, “myname”, “mypassword”, 20,
                                    tls_setup, NX_WAIT_FOREVER);

/* If status is NX_SUCCESS, the POST operation for TEST.HTM has successfully been
   started. */
```

nx_web_http_client_put_start

Start an HTTP PUT request

Prototype

```
UINT nx_web_http_client_put_start(NX_WEB_HTTP_CLIENT *client_ptr,
                                  NXD_ADDRESS ip_address, UINT server_port,
                                  CHAR *resource, CHAR *host, CHAR *username,
                                  CHAR *password, ULONG total_bytes,
                                  ULONG wait_option);
```

Description

This method is for **plaintext** HTTP.

This service attempts to send a PUT request with the specified resource to the HTTP Server at the supplied IP address and port. If this routine is successful, the application code should make successive calls to the *nx_web_http_client_put_packet()* routine to send the resource contents to the HTTP Server.

Note that the resource string can refer to a local file e.g. “/index.htm” or it can refer to another URL e.g. http://abc.website.com/index.htm if the HTTP Server indicates it supports referring PUT requests.

Input Parameters

client_ptr	Pointer to HTTP Client control block.
ip_address	IP address of the HTTP Server.
server_port	TCP port on the remote HTTP Server.
resource	Pointer to URL string for resource to send to Server.
host	Null-terminated string of the server’s domain name. This string is transmitted in the HTTP Host header field. The host string cannot be NULL.
username	Pointer to optional user name for authentication.
password	Pointer to optional password for authentication.
total_bytes	Total bytes of resource being sent. Note that the combined length of all packets sent via subsequent calls to *nx_web_http_client_put_packet()* must equal this value.
wait_option	Defines how long the service will wait for the HTTP Client PUT start. The wait options are defined as follows: **timeout value** (0x00000001 through 0xFFFFFFFE) **NX_WAIT_FOREVER** (0xFFFFFFFF)

Selecting NX_WAIT_FOREVER causes the calling thread to suspend indefinitely until the HTTP Server responds to the request.

Selecting a numeric value (0x1-0xFFFFFFFE) specifies the maximum number of timer-ticks to stay suspended while waiting for the HTTP Server response.

Return Values

NX_SUCCESS	(0x00)	Successfully sent PUT request
NX_WEB_HTTP_USERNAME_TOO_LONG	(0x30012)	Username too large for buffer
NX_WEB_HTTP_NOT_READY	(0x3000A)	HTTP Client not ready
NX_PTR_ERROR	(0x07)	Invalid pointer input
NX_SIZE_ERROR	(0x09)	Invalid total size of resource
NX_CALLER_ERROR	(0x11)	Invalid caller of this service

Allowed From

Threads

Example

```
/* Start an HTTP PUT to place the 20-byte resource “/TEST.HTM” on the HTTP Server
   at IP address 1.2.3.5.  */
/* Connect to a remote HTTP server. */
server_ip_addr.nxd_ip_version = NX_IP_VERSION_V4;
server_ip_addr.nxd_ip_address.v4 = IP_ADDRESS(1,2,3,5);

status =  nx_web_http_client_put_start(&my_client, &server_ip_addr,
                                       NX_WEB_HTTP_SERVER_PORT, “/TEST.HTM”,
                                       “host.com”, “myname”, “mypassword”, 20,
                                       NX_WAIT_FOREVER);

/* If status is NX_SUCCESS, the PUT operation for TEST.HTM has successfully been
   started. */
```

nx_web_http_client_put_secure_start

Start an encrypted HTTPS PUT request

Prototype

```
UINT nx_web_http_client_put_secure_start(
                    NX_WEB_HTTP_CLIENT *client_ptr,
                    NXD_ADDRESS ip_address, UINT server_port,
                    CHAR *resource, CHAR *host, CHAR *username,
                    CHAR *password, ULONG total_bytes,
                    UINT (*tls_setup)(NX_WEB_HTTP_CLIENT *client_ptr,
                                      NX_SECURE_TLS_SESSION *tls_session),
                    ULONG wait_option);
```

Description

This method is for **TLS-secured** HTTPS.

This service attempts to send a PUT request with the specified resource to the HTTPS Server at the supplied IP address and port. If this routine is successful, the application code should make successive calls to the *nx_web_http_client_put_packet()* routine to send the resource contents to the HTTP Server.

Note that the resource string can refer to a local file e.g. “/index.htm” or it can refer to another URL e.g. http://abc.website.com/index.htm if the HTTP Server indicates it supports referring PUT requests.

Input Parameters

client_ptr	Pointer to HTTP Client control block.
ip_address	IP address of the HTTP Server.
server_port	TCP port on the remote HTTP Server.
resource	Pointer to URL string for resource to send to Server.
host	Null-terminated string of the server’s domain name. This string is transmitted in the HTTP Host header field. The host string cannot be NULL.
username	Pointer to optional user name for authentication.
password	Pointer to optional password for authentication.
total_bytes	Total bytes of resource being sent. Note that the combined length of all packets sent via subsequent calls to *nx_web_http_client_put_packet()* must equal this value.
tls_setup	Callback used to setup TLS configuration. The application defines this callback to initialize TLS cryptography and credentials (e.g. certificates).

wait_option Defines how long the service will wait for the HTTP Client PUT start. The wait options are defined as follows:

timeout value (0x00000001 through 0xFFFFFFFE)
NX_WAIT_FOREVER (0xFFFFFFFF)

Selecting NX_WAIT_FOREVER causes the calling thread to suspend indefinitely until the HTTP Server responds to the request.

Selecting a numeric value (0x1-0xFFFFFFFE) specifies the maximum number of timer-ticks to stay suspended while waiting for the HTTP Server response.

Return Values

NX_SUCCESS	(0x00)	Successfully sent PUT request
NX_WEB_HTTP_USERNAME_TOO_LONG	(0x30012)	Username too large for buffer
NX_WEB_HTTP_NOT_READY	(0x3000A)	HTTP Client not ready
NX_PTR_ERROR	(0x07)	Invalid pointer input
NX_SIZE_ERROR	(0x09)	Invalid total size of resource
NX_CALLER_ERROR	(0x11)	Invalid caller of this service

Allowed From

Threads

Example

```
/* Start an HTTP PUT to place the 20-byte resource “/TEST.HTM” on the HTTPS
Server
   at IP address 1.2.3.5.  */
/* Connect to a remote HTTP server. */
server_ip_addr.nxd_ip_version = NX_IP_VERSION_V4;
server_ip_addr.nxd_ip_address.v4 = IP_ADDRESS(1,2,3,5);

status =  nx_web_http_client_put_secure_start(&my_client, &server_ip_addr,
                                    NX_WEB_HTTPS_SERVER_PORT, “/TEST.HTM”,
                                    “host.com”, “myname”, “mypassword”, 20,
                                    tls_setup, NX_WAIT_FOREVER);

/* If status is NX_SUCCESS, the PUT operation for TEST.HTM has successfully been
   started. */
```

nx_web_http_client_put_packet

Send next resource data packet

Prototype

```
UINT nx_web_http_client_put_packet(NX_WEB_HTTP_CLIENT *client_ptr,
                                   NX_PACKET *packet_ptr, ULONG wait_option);
```

Description

This service attempts to send the next packet of resource content to the HTTP Server for both PUT and POST operations. Note that this routine should be called repetitively until the combined length of the packets sent equals the “total_bytes” specified in the previous *nx_web_http_client_put_start()* or *nx_web_http_client_post_start()* call (or their corresponding secure versions).

This service also checks for a response from the server in case there was a problem establishing the HTTP (or HTTPS) connection.

Input Parameters

client_ptr	Pointer to HTTP Client control block.
packet_ptr	Pointer to next content of the resource to being sent to the HTTP Server.
wait_option	Defines how long the service will wait internally to process the HTTP Client PUT packet. The wait options are defined as follows: **timeout value** (0x00000001 through 0xFFFFFFFE) **NX_WAIT_FOREVER** (0xFFFFFFFF) Selecting NX_WAIT_FOREVER causes the calling thread to suspend indefinitely until the HTTP Server responds to the request. Selecting a numeric value (0x1-0xFFFFFFFE) specifies the maximum number of timer-ticks to stay suspended while waiting for the HTTP Server response.

Return Values

NX_SUCCESS	(0x00)	Successfully sent HTTP Client packet.
NX_WEB_HTTP_NOT_READY	(0x3000A)	HTTP Client not ready
NX_WEB_HTTP_REQUEST_UNSUCCESSFUL_CODE	(0x3000E)	Received Server error code
NX_WEB_HTTP_BAD_PACKET_LENGTH	(0x3000D)	Invalid packet length
NX_WEB_HTTP_AUTHENTICATION_ERROR	(0x3000B)	Invalid name and/or Password
NX_WEB_HTTP_INCOMPLETE_PUT_ERROR	(0x3000F)	Server responds before PUT Is complete
NX_PTR_ERROR	(0x07)	Invalid pointer input
NX_INVALID_PACKET	(0x12)	Packet too small for TCP header
NX_CALLER_ERROR	(0x11)	Invalid caller of this service

Allowed From

Threads

Example

```
/* Send a 20-byte packet representing the content of the resource
   "/TEST.HTM" to the HTTP Server.  */
status =  nx_web_http_client_put_packet(&client_ptr, packet_ptr,
                                        NX_WAIT_FOREVER);

/* If status is NX_SUCCESS, the 20-byte resource contents of TEST.HTM has
   successfully been sent. */
```

nx_web_http_client_request_chunked_set

Set chunked transfer for HTTP(S) request

Prototype

```
UINT nx_web_http_client_request_chunked_set(
                                NX_WEB_HTTP_CLIENT *client_ptr,
                                UINT chunk_size, NX_PACKET *packet_ptr);
```

Description

This service uses chunked transfer coding to send a custom HTTP(S) request to the server specified in the *nx_web_http_client_connect()* or *nx_web_http_client_secure_connect()* call which has previously established the socket connection to the remote host.

Note: If the application uses chunked transfer coding to send a request data packet, it must call this service after calling *nx_web_http_client_request_packet_allocate* (), and before call *nx_web_http_client_reqeust_packet_send* ().

Input Parameters

client_ptr	Pointer to HTTP Client control block.
chunk_size	Size of the chunk-data in octets.
packet_ptr	HTTP(S) request data packet pointer.

Return Values

NX_SUCCESS	(0x00)	Successfully set chunked.
NX_PTR_ERROR	(0x07)	Invalid pointer input

Allowed From

Threads

Example

```
/* Connect to a remote HTTPS server. */
nx_web_http_client_secure_connect(&my_client, IP_ADDRESS(1,2,3,5),
                                  NX_WEB_HTTPS_SERVER_PORT, tls_setup_callback,
                                  NX_WAIT_FOREVER);


/* Create a PUT request on the HTTP client instance. */
nx_web_http_client_request_initialize(&my_client,
                                      NX_WEB_HTTP_METHOD_PUT,
                                      "https://192.168.1.150/test.txt ",
                                      0, /* Used by PUT and POST only */
                                      NX_TRUE,
                                      NX_NULL, /* username */
                                      NX_NULL, /* password */
                                      NX_WAIT_FOREVER);

/* Start the PUT operation. */
nx_web_http_client_request_send(&my_client, 1000);

/* Create a new data packet request on the HTTP(S) client instance. */
nx_web_http_client_request_packet_allocate(&my_client,
                                           &my_packet,
                                           NX_WAIT_FOREVER);

/* Set the chunked transfer.  */
Status = nx_web_http_client_request_chunked_set(&my_client, 128, my_packet)

/* At this point, user can fill the data into my_packet. *./
nx_packet_data_append(my_packet, data_ptr, data_size,
                      packet_pool, NX_WAIT_FOREVER);

/* Send data packet request to server. */
nx_web_http_client_request_packet_send(&my_client,
                                       my_packet,
                                       0,
                                       NX_WAIT_FOREVER);
```

nx_web_http_client_request_header_add

Add a custom header to a custom HTTP request

Prototype

```
UINT nx_web_http_client_request_header_add(
                                NX_WEB_HTTP_CLIENT *client_ptr,
                                CHAR *field_name, UINT name_length,
                                CHAR *field_value, UINT value_length,
                                UINT wait_option);
```

Description

This service adds a custom header (in the form of a field name and value) to a custom HTTP request created with n*x_web_http_client_request_initialize()*.

Use of this service enables an application to add any number of custom headers to the request. This allows for customized HTTP requests intended for specific applications.

Note that the nx_web_http_client_*_start methods are provided for convenience. These functions all use this routine internally (along with *nx_web_http_client_request_initialize())* to create and send HTTP requests.

Input Parameters

client_ptr	Pointer to HTTP Client control block.
field_name	Field name string (e.g. “Content-Type”).
name_length	Length of field name string in bytes.
field_value	Field value string (e.g. “application/octet-stream”).
value_length	Length of value string in bytes.
wait_option	Defines how long the service will wait for underlying network operations. The wait options are defined as follows: **timeout value** (0x00000001 through 0xFFFFFFFE) **NX_WAIT_FOREVER** (0xFFFFFFFF) Selecting NX_WAIT_FOREVER causes the calling thread to suspend indefinitely until the HTTP Server responds to the request. Selecting a numeric value (0x1-0xFFFFFFFE) specifies the maximum number of timer-ticks to stay suspended while waiting for the HTTP

Server response.

Return Values

NX_SUCCESS	(0x00)	Successful addition of header to request.
NX_PTR_ERROR	(0x07)	Invalid pointer input

Allowed From

Threads

Example

```
/* Connect to a remote HTTPS server. */
nx_web_http_client_secure_connect(&my_client, IP_ADDRESS(1,2,3,5),
                                  NX_WEB_HTTPS_SERVER_PORT, tls_setup_callback,
                                  NX_WAIT_FOREVER);


/* Create a new GET request on the HTTP client instance. */
nx_web_http_client_request_initialize(&my_client,
                                      NX_WEB_HTTP_METHOD_GET,
                                      "https://192.168.1.150/test.txt ",
                                      0, /* Used by PUT and POST only */
                                      NX_FALSE,
                                      NX_NULL, /* username */
                                      NX_NULL, /* password */
                                      NX_WAIT_FOREVER);

/* Add a custom header to the GET request we just created. */
status = nx_web_http_client_request_header_add(&my_client, “Server”, 6,
                                  “NetX Web HTTPS Server”, 21, NX_WAIT_FOREVER);

/* Start the GET operation to get a response from the HTTPS server. */
status = nx_web_http_client_request_send(&my_client, 1000);

/* At this point, we need to handle the response from the server by repeatedly
   calling nx_web_http_client_response_body_get() until the entire response is
   retrieved. */

get_status = NX_SUCCESS;

while(get_status != NX_WEB_HTTP_GET_DONE)
{
   get_status = nx_web_http_client_response_body_get(&my_client, &receive_packet,
                                                     NX_WAIT_FOREVER);
   /* Process response packets… */
}
```

nx_web_http_client_request_initialize

Initialize a custom HTTP request

Prototype

```
UINT nx_web_http_client_request_initialize (
                  NX_WEB_HTTP_CLIENT *client_ptr,
                  UINT method, CHAR *resource, CHAR *host,
                  UINT input_size, UINT transfer_encoding_trunked,
                  CHAR *username, CHAR *password, UINT wait_option);
```

Description

This service creates a custom HTTP request and associates it with the HTTP Client instance. Unlike the simpler *nx_web_http_client_get_start()* (along with the methods for PUT, POST, and the associated secure versions of those API), the custom request is not sent until the *nx_web_http_client_request_send()* service is called.

Use of this service enables an application to add any number of custom headers to the request using the *nx_web_http_client_request_header_add()* service. This allows for customized HTTP requests intended for specific applications.

Note that the nx_web_http_client_*_start methods are provided for convenience. These functions all use this routine internally (along with *nx_web_http_client_request_send())* to create and send HTTP requests.

Input Parameters

client_ptr	Pointer to HTTP Client control block.
method	The HTTP request method to use. The options are defined as follows: **NX_WEB_HTTP_METHOD_NONE** (0x0) **NX_WEB_HTTP_METHOD_GET** (0x1) **NX_WEB_HTTP_METHOD_PUT** (0x2) **NX_WEB_HTTP_METHOD_POST** (0x3) **NX_WEB_HTTP_METHOD_DELETE** (0x4) **NX_WEB_HTTP_METHOD_HEAD** (0x5)
resource	Name of resource being transferred.
host	Null-terminated string of the server's domain name. This string is transmitted in the HTTP Host header field. The host string cannot be NULL.
input_size	Size of input data for PUT and POST. Pass 0 for other operations.
transfer_encoding_trunked	Reserved parameter for future trunked transfer support.

username	Username for protected resources.
password	Password for protected resources.
wait_option	Defines how long the service will wait for underlying network operations. The wait options are defined as follows: **timeout value** (0x00000001 through 0xFFFFFFFE) **NX_WAIT_FOREVER** (0xFFFFFFFF) Selecting NX_WAIT_FOREVER causes the calling thread to suspend indefinitely until the HTTP Server responds to the request. Selecting a numeric value (0x1-0xFFFFFFFE) specifies the maximum number of timer-ticks to stay suspended while waiting for the HTTP Server response.

Return Values

NX_SUCCESS	(0x00)	Successful initialization of request.
NX_PTR_ERROR	(0x07)	Invalid pointer input
NX_WEB_HTTP_METHOD_ERROR	(0x30014)	Some required information was missing (e.g. input_size for PUT or POST).

Allowed From

Threads

Example

```
/* Connect to a remote HTTPS server. */
nx_web_http_client_secure_connect(&my_client, IP_ADDRESS(1,2,3,5),
                                  NX_WEB_HTTPS_SERVER_PORT, tls_setup_callback,
                                  NX_WAIT_FOREVER);


/* Create a new GET request on the HTTP client instance. */
nx_web_http_client_request_initialize(&my_client,
                                      NX_WEB_HTTP_METHOD_GET,
                                      "test.txt ", “host.com”,
                                      0, /* Used by PUT and POST only */
                                      NX_FALSE,
                                      NX_NULL, /* username */
                                      NX_NULL, /* password */
                                      NX_WAIT_FOREVER);

/* Start the GET operation to get a response from the HTTPS server. */
status = nx_web_http_client_request_send(&my_client, 1000);

/* At this point, we need to handle the response from the server by repeatedly
   calling nx_web_http_client_response_body_get() until the entire response is
   retrieved. */

get_status = NX_SUCCESS;

while(get_status != NX_WEB_HTTP_GET_DONE)
{
   get_status = nx_web_http_client_response_body_get(&my_client, &receive_packet,
                                                     NX_WAIT_FOREVER);
   /* Process response packets… */
}
```

nx_web_http_client_request_packet_send

Send HTTP(S) request data packet to remote server

Prototype

```
UINT nx_web_http_client_request_packet_send(
                                NX_WEB_HTTP_CLIENT *client_ptr,
                                NX_PACKET *packet_ptr, UINT more_date,
                                ULONG wait_option);
```

Description

This service sends a custom HTTP(S) request data packet created with *nx_web_http_client_request_packet_allocate* () to the server specified in the *nx_web_http_client_connect()* or *nx_web_http_client_secure_connect()* call which has previously established the socket connection to the remote host.

Input Parameters

client_ptr Pointer to HTTP Client control block.

packet_ptr HTTP(S) request data packet pointer.

wait_option Defines how long the service will wait for underlying network operations. The wait options are defined as follows:

NX_NO_WAIT (0x00000000)
NX_WAIT_FOREVER (0xFFFFFFFF)
timeout value (0x00000001 through 0xFFFFFFFE)

Selecting NX_WAIT_FOREVER causes the calling thread to suspend indefinitely until the HTTP Server responds to the request.

Selecting a numeric value (0x1-0xFFFFFFFE) specifies the maximum number of timer-ticks to stay suspended while waiting for the HTTP Server response.

Return Values

NX_SUCCESS	(0x00)	Successful send of request data packet.
NX_PTR_ERROR	(0x07)	Invalid pointer input

Allowed From

Threads

Example

```
/* Connect to a remote HTTPS server. */
nx_web_http_client_secure_connect(&my_client, IP_ADDRESS(1,2,3,5),
                                  NX_WEB_HTTPS_SERVER_PORT, tls_setup_callback,
                                  NX_WAIT_FOREVER);


/* Create a PUT request on the HTTP client instance. */
nx_web_http_client_request_initialize(&my_client,
                                      NX_WEB_HTTP_METHOD_PUT,
                                      "https://192.168.1.150/test.txt ",
                                      128, /* Used by PUT and POST only */
                                      NX_FALSE,
                                      NX_NULL, /* username */
                                      NX_NULL, /* password */
                                      NX_WAIT_FOREVER);

/* Start the PUT operation. */
nx_web_http_client_request_send(&my_client, 1000);

/* Create a new data packet request on the HTTP(S) client instance. */
nx_web_http_client_request_packet_allocate(&my_client,
                                           &my_packet,
                                           NX_WAIT_FOREVER);

/* At this point, user can fill the data into my_packet. *./
nx_packet_data_append(my_packet, data_ptr, data_size,
                      packet_pool, NX_WAIT_FOREVER);


/* Send data packet request to server. */
status = nx_web_http_client_request_packet_send(&my_client,
                                                my_packetET,
                                                0,
                                                NX_WAIT_FOREVER);
```

nx_web_http_client_request_send

Send a custom HTTP request

Prototype

```
UINT nx_web_http_client_request_send(NX_WEB_HTTP_CLIENT *client_ptr,
                                     UINT wait_option);
```

Description

This service sends a custom HTTP request created with *nx_web_http_client_request_initialize()* to the server specified in the *nx_web_http_client_connect()* or *nx_web_http_client_secure_connect()* both of which have previously established the socket connection to the remote host.

Use of this service enables an application to add any number of custom headers to the request using the *nx_web_http_client_request_header_add()* service. This allows for customized HTTP requests intended for specific applications.

Note that the nx_web_http_client_*_start methods are provided for convenience. These functions all use this routine internally (along with *nx_web_http_client_request_initialize())* to create and send HTTP requests.

Input Parameters

client_ptr Pointer to HTTP Client control block.

wait_option Defines how long the service will wait for underlying network operations. The wait options are defined as follows:

timeout value (0x00000001 through 0xFFFFFFFE)

NX_WAIT_FOREVER (0xFFFFFFFF)

Selecting NX_WAIT_FOREVER causes the calling thread to suspend indefinitely until the HTTP Server responds to the request.

Selecting a numeric value (0x1-0xFFFFFFFE) specifies the maximum number of timer-ticks to stay suspended while waiting for the HTTP Server response.

Return Values

NX_SUCCESS	(0x00)	Successful send of request.
NX_PTR_ERROR	(0x07)	Invalid pointer input

Allowed From

Threads

Example

```
/* Connect to a remote HTTPS server. */
nx_web_http_client_secure_connect(&my_client, IP_ADDRESS(1,2,3,5),
                                  NX_WEB_HTTPS_SERVER_PORT, tls_setup_callback,
                                  NX_WAIT_FOREVER);


/* Create a new GET request on the HTTP client instance. */
nx_web_http_client_request_initialize(&my_client,
                                      NX_WEB_HTTP_METHOD_GET,
                                      "https://192.168.1.150/test.txt ",
                                      0, /* Used by PUT and POST only */
                                      NX_FALSE,
                                      NX_NULL, /* username */
                                      NX_NULL, /* password */
                                      NX_WAIT_FOREVER);

/* Start the GET operation to get a response from the HTTPS server. */
status = nx_web_http_client_request_send(&my_client, 1000);

/* At this point, we need to handle the response from the server by repeatedly
   calling nx_web_http_client_response_body_get until the entire response is
   retrieved. */

get_status = NX_SUCCESS;

while(get_status != NX_WEB_HTTP_GET_DONE)
{
   get_status = nx_web_http_client_response_body_get(&my_client, &receive_packet,
                                                     NX_WAIT_FOREVER);
   /* Process response packets… */
}
```

nx_web_http_client_response_body_get

Get next resource data packet

Prototype

```
UINT nx_web_http_client_response_body_get(
                                NX_WEB_HTTP_CLIENT *client_ptr,
                                NX_PACKET **packet_ptr, ULONG
                                wait_option);
```

Description

This service retrieves the next packet of content of the resource requested by the previous *nx_web_http_client_get_start()* or *nx_web_http_client_get_secure_start()* call. Successive calls to this routine should be made until the return status of NX_WEB_HTTP_GET_DONE is received.

Input Parameters

client_ptr Pointer to HTTP Client control block.

packet_ptr Destination for packet pointer containing partial resource content.

wait_option Defines how long the service will wait for the HTTP Client get packet. The wait options are defined as follows:

timeout value	(0x00000001 through 0xFFFFFFFE)
NX_WAIT_FOREVER	(0xFFFFFFFF)

Selecting NX_WAIT_FOREVER causes the calling thread to suspend indefinitely until the HTTP Server responds to the request.

Selecting a numeric value (0x1-0xFFFFFFFE) specifies the maximum number of timer-ticks to stay suspended while waiting for the HTTP Server response.

Return Values

NX_SUCCESS	(0x00)	Successful get of HTTP Client packet.
NX_WEB_HTTP_GET_DONE	(0x3000C)	HTTP Client get packet is done
NX_WEB_HTTP_NOT_READY	(0x3000A)	HTTP Client not in get mode.
NX_WEB_HTTP_BAD_PACKET_LENGTH	(0x3000D)	Invalid packet length
NX_WEB_HTTP_STATUS_CODE_CONTINUE	(0x3001A)	HTTP status code 100 Continue
NX_WEB_HTTP_STATUS_CODE_SWITCHING_PROTOCOLS	(0x3001B)	HTTP status code 101 Switching Protocols
NX_WEB_HTTP_STATUS_CODE_CREATED	(0x3001C)	HTTP status code 201 Created
NX_WEB_HTTP_STATUS_CODE_ACCEPTED	(0x3001D)	HTTP status code 202 Accepted
NX_WEB_HTTP_STATUS_CODE_NON_AUTH_INFO	(0x3001E)	HTTP status code 203 Non-Authoritative Information
NX_WEB_HTTP_STATUS_CODE_NO_CONTENT	(0x3001F)	HTTP status code 204 No Content
NX_WEB_HTTP_STATUS_CODE_RESET_CONTENT	(0x30020)	HTTP status code 205 Reset Content
NX_WEB_HTTP_STATUS_CODE_PARTIAL_CONTENT	(0x30021)	HTTP status code 206 Partial Content
NX_WEB_HTTP_STATUS_CODE_MULTIPLE_CHOICES	(0x30022)	HTTP status code 300 Multiple Choices
NX_WEB_HTTP_STATUS_CODE_MOVED_PERMANETLY	(0x30023)	HTTP status code 301 Moved Permanently
NX_WEB_HTTP_STATUS_CODE_FOUND	(0x30024)	HTTP status code 302 Found
NX_WEB_HTTP_STATUS_CODE_SEE_OTHER	(0x30025)	HTTP status code 303 See Other
NX_WEB_HTTP_STATUS_CODE_NOT_MODIFIED	(0x30026)	HTTP status code 304 Not Modified

NX_WEB_HTTP_STATUS_CODE_USE_PROXY
(0x30027) HTTP status code 305 Use Proxy
NX_WEB_HTTP_STATUS_CODE_TEMPORARY_REDIRECT
(0x30028) HTTP status code 307 Temporary Redirect
NX_WEB_HTTP_STATUS_CODE_BAD_REQUEST
(0x30029) HTTP status code 400 Bad Request
NX_WEB_HTTP_STATUS_CODE_UNAUTHORIZED
(0x3002A) HTTP status code 401 Unauthorized
NX_WEB_HTTP_STATUS_CODE_PAYMENT_REQUIRED
(0x3002B) HTTP status code 402 Payment Required
NX_WEB_HTTP_STATUS_CODE_FORBIDDEN
(0x3002C) HTTP status code 403 Forbidden
NX_WEB_HTTP_STATUS_CODE_NOT_FOUND
(0x3002D) HTTP status code 404 Not Found
NX_WEB_HTTP_STATUS_CODE_METHOD_NOT_ALLOWED
(0x3002E) HTTP status code 405 Method Not Allowed
NX_WEB_HTTP_STATUS_CODE_NOT_ACCEPTABLE
(0x3002F) HTTP status code 406 Not Acceptable
NX_WEB_HTTP_STATUS_CODE_PROXY_AUTH_REQUIRED
(0x30030) HTTP status code 407 Proxy Authentication Required
NX_WEB_HTTP_STATUS_CODE_REQUEST_TIMEOUT
(0x30031) HTTP status code 408 Request Time-out
NX_WEB_HTTP_STATUS_CODE_CONFLICT
(0x30032) HTTP status code 409 Conflict
NX_WEB_HTTP_STATUS_CODE_GONE
(0x30033) HTTP status code 410 Gone
NX_WEB_HTTP_STATUS_CODE_LENGTH_REQUIRED
(0x30034) HTTP status code 411 Length Required
NX_WEB_HTTP_STATUS_CODE_PRECONDITION_FAILED
(0x30035) HTTP status code 412 Precondition Failed
NX_WEB_HTTP_STATUS_CODE_ENTITY_TOO_LARGE
(0x30036) HTTP status code 413 Request Entity Too Large

NX_WEB_HTTP_STATUS_CODE_URL_TOO_LARGE		
	(0x30037)	HTTP status code 414 Request-URL Too Large
NX_WEB_HTTP_STATUS_CODE_UNSUPPORTED_MEDIA		
	(0x30038)	HTTP status code 415 Unsupported Media Type
NX_WEB_HTTP_STATUS_CODE_RANGE_NOT_SATISFY		
	(0x30039)	HTTP status code 416 Requested range not satisfiable
NX_WEB_HTTP_STATUS_CODE_EXPECTATION_FAILED		
	(0x3003A)	HTTP status code 417 Expectation Failed
NX_WEB_HTTP_STATUS_CODE_INTERNAL_ERROR		
	(0x3003B)	HTTP status code 500 Internal Server Error
NX_WEB_HTTP_STATUS_CODE_NOT_IMPLEMENTED		
	(0x3003C)	HTTP status code 501 Not Implemented
NX_WEB_HTTP_STATUS_CODE_BAD_GATEWAY		
	(0x3003D)	HTTP status code 502 Bad Gateway
NX_WEB_HTTP_STATUS_CODE_SERVICE_UNAVAILABLE		
	(0x3003E)	HTTP status code 503 Service Unavailable
NX_WEB_HTTP_STATUS_CODE_GATEWAY_TIMEOUT		
	(0x3003F)	HTTP status code 504 Gateway Time-out
NX_WEB_HTTP_STATUS_CODE_VERSION_ERROR		
	(0x30040)	HTTP status code 505 HTTP Version not supported
NX_PTR_ERROR	(0x07)	Invalid pointer input
NX_CALLER_ERROR	(0x11)	Invalid caller of this service

Allowed From

Threads

Example

```
/* Get the next packet of resource content on the HTTP Client “my_client.”
   Note that the nx_web_http_client_get_start() routine must have been called
   previously. */
status =  nx_web_http_client_response_body_get(&my_client, &next_packet, 1000);

/* If status is NX_SUCCESS, the next packet of content is pointed to
   by “next_packet”. */
```

nx_web_http_client_response_header_callback_set

Set callback to invoke when processing HTTP headers

Prototype

```
UINT nx_web_http_client_response_header_callback_set(
            NX_WEB_HTTP_CLIENT *client_ptr,
            VOID (*callback_function)(NX_WEB_HTTP_CLIENT *client_ptr,
                        CHAR *field_name, UINT field_name_length,
                        CHAR *field_value, UINT field_value_length));
```

Description

This service assigns a callback that will be invoked whenever NetX Web HTTP Client processes an HTTP header in an incoming response from a remote HTTP server. The callback is invoked once for each header in the response as it is processed. The callback allows an HTTP Client application to access each of the headers in the HTTP server response to take any desired actions beyond the basic processing that NetX Web HTTP Client does.

Input Parameters

client_ptr	Pointer to HTTP Client control block.
callback_function	Callback invoked during response header processing. The callback is invoked with the field name and value as strings (and their lengths). For example, the header “Content-Length: 100” would cause the function to be invoked with “Content-Length” for *field_name* and the string “100” for *field_value*.

Return Values

NX_SUCCESS	(0x00)	Successful assignment of callback.
NX_PTR_ERROR	(0x07)	Invalid pointer input

Allowed From

Threads

Example

```
/* Setup a callback to print out header information as it is processed. */
VOID http_response_callback(NX_WEB_HTTP_CLIENT *client_ptr, CHAR *field_name,
                            UINT field_name_length, CHAR *field_value,
                            UINT field_value_length)
{
CHAR name[100];
CHAR value[100];

    memset(name, 0, sizeof(name));
    memset(value, 0, sizeof(value));

    strncpy(name, field_name, field_name_length);
    strncpy(value, field_value, field_value_length);

    printf("Received header: \n);
    printf("\tField name: %s (%d bytes)\n", name, field_name_length);
    printf("\tValue: %s (%d bytes)\n\n", value, field_value_length);
}

/* Assign the callback to the HTTP client instance. */
nx_web_http_client_response_header_callback_set(&my_client,
                                                http_response_callback);

/* Start a GET operation to get a response from the HTTP server. */
status = nx_web_http_client_get_start(&my_client, IP_ADDRESS(1,2,3,5),
                                      NX_WEB_HTTP_SERVER_PORT, "/TEST.HTM",
                                      "myname", "mypassword", 1000);

/* When the HTTP server returns a response to the GET request, NetX Web HTTP
   Client will invoke the http_response_callback function for each header
   processed in the HTTP response. */
```

nx_web_http_client_secure_connect

Open a TLS session to an HTTPS server for custom requests

Prototype

```
UINT nx_web_http_client_secure_connect(NX_WEB_HTTP_CLIENT *client_ptr,
                  NXD_ADDRESS *server_ip, UINT server_port,
                  UINT (*tls_setup)(NX_WEB_HTTP_CLIENT *client_ptr,
                                    NX_SECURE_TLS_SESSION *tls),
                  ULONG wait_option);
```

Description

This method is for **TLS-secured** HTTPS.

This service opens a secured TLS session to an HTTPS server but does not send any requests. Requests are created with n*x_web_http_client_request_initialize()* and sent using *nx_web_http_client_request_send()*. Custom HTTP headers may be added to the request using *nx_web_http_client_request_header_add()*.

Use of this service enables an application to add any number of custom headers to the request. This allows for customized HTTP requests intended for specific applications.

Note that the nx_web_http_client_*_start methods are provided for convenience. These functions all use this routine internally (along with *nx_web_http_client_request_initialize())* to create and send HTTP requests.

Input Parameters

client_ptr	Pointer to HTTP Client control block.
server_ip	IP address of remote HTTPS server.
server_port	Port on remote HTTPS server (e.g. 443 for HTTPS).
tls_setup	Callback used to setup TLS configuration. The application defines this callback to initialize TLS cryptography and credentials (e.g. certificates).
wait_option	Defines how long the service will wait for underlying network operations. The wait options are defined as follows: **timeout value** (0x00000001 through 0xFFFFFFFE) **NX_WAIT_FOREVER** (0xFFFFFFFF) Selecting NX_WAIT_FOREVER causes the calling thread to suspend indefinitely until the HTTP Server responds to the request.

Selecting a numeric value (0x1-0xFFFFFFFE) specifies the maximum number of timer-ticks to stay suspended while waiting for the HTTP Server response.

Return Values

NX_SUCCESS	(0x00)	Successful connection of TLS session.
NX_PTR_ERROR	(0x07)	Invalid pointer input
NX_WEB_HTTP_NOT_READY	(0x3000A)	Another request is already in progress.

Allowed From

Threads

Example

```
/* Connect to a remote HTTPS server. */
/* Connect to a remote HTTP server. */
server_ip_addr.nxd_ip_version = NX_IP_VERSION_V4;
server_ip_addr.nxd_ip_address.v4 = IP_ADDRESS(1,2,3,5);

nx_web_http_client_secure_connect(&my_client, &server_ip_addr,
                          NX_WEB_HTTPS_SERVER_PORT, tls_setup_callback,
                          NX_WAIT_FOREVER);


/* Create a new GET request on the HTTP client instance. */
nx_web_http_client_request_initialize(&my_client,
                                      NX_WEB_HTTP_METHOD_GET,
                                      "https://192.168.1.150/test.txt ",
                                      0, /* Used by PUT and POST only */
                                      NX_FALSE,
                                      NX_NULL, /* username */
                                      NX_NULL, /* password */
                                      NX_WAIT_FOREVER);

/* Add a custom header to the GET request we just created. */
status = nx_web_http_client_request_header_add(&my_client, “Server”, 6,
                              “NetX Web HTTPS Server”, 21, NX_WAIT_FOREVER);

/* Start the GET operation to get a response from the HTTPS server. */
status = nx_web_http_client_request_send(&my_client, 1000);

/* At this point, we need to handle the response from the server by repeatedly
   calling nx_web_http_client_response_body_get until the entire response is
   retrieved. */

get_status = NX_SUCCESS;

while(get_status != NX_WEB_HTTP_GET_DONE)
{
   get_status = nx_web_http_client_response_body_get(&my_client, &receive_packet,
                                                     NX_WAIT_FOREVER);
   /* Process response packets… */
}
```

nx_web_http_server_cache_info_callback_set

Set the callback to retrieve URL max age and date

Prototype

```
UINT nx_web_http_server_cache_info_callback_set(
                        NX_WEB_HTTP_SERVER *server_ptr,
                        UINT (*cache_info_get)(CHAR *resource,
                                               UINT *max_age,
                                               NX_WEB_HTTP_SERVER_DATE
                                                    *date));
```

Description

This service sets the callback service invoked to obtain the maximum age and last modified date of the specified resource.

Input Parameters

server_ptr	Pointer to HTTP Server control block.
cache_info_get	Pointer to the callback
max_age	Pointer to maximum age of a resource
data	Pointer to last modified date returned.

Return Values

NX_SUCCESS	(0x00)	Successfully set the callback
NX_PTR_ERROR	(0x07)	Invalid pointer input

Allowed From

Initialization

Example

```
NX_WEB_HTTP_SERVER my_server;

UINT cache_info_get(CHAR *resource, UINT *max_age,
                    NX_WEB_HTTP_SERVER_DATE *last_modified);

/* After my_server is created with nx_web_http_server_create and before the HTTP
   server is set by nx_web_http_server_start(), set the cache info callback: */

status = nx_web_http_server_cache_info_callback_set(&my_server, cache_info_get);


/* If status is NX_SUCCESS, the callback was successfully sent. */
```

nx_web_http_server_callback_data_send

Send data from callback function

Prototype

```
UINT nx_web_http_server_callback_data_send(
                                    NX_WEB_HTTP_SERVER *server_ptr,
                                    VOID *data_ptr, ULONG data_length);
```

Description

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_WEB_HTTP_CALLBACK_COMPLETED.

Input Parameters

server_ptr	Pointer to HTTP Server control block.
data_ptr	Pointer to the data to send.
data_length	Number of bytes to send.

Return Values

NX_SUCCESS	(0x00)	Successfully sent Server data
NX_PTR_ERROR	(0x07)	Invalid pointer input

Allowed From

Threads

Example

```
UINT  my_request_notify(NX_WEB_HTTP_SERVER *server_ptr, UINT request_type,
                        CHAR *resource, NX_PACKET *packet_ptr)
{

    /* Look for the test resource!  */
    if ((request_type == NX_WEB_HTTP_SERVER_GET_REQUEST) &&
         (strcmp(resource, "/test.htm") == 0))
    {

        /* Found it, override the GET processing by sending the resource
           contents directly.  */

        nx_web_http_server_callback_data_send(server_ptr,
                              "HTTP/1.0 200 \r\nContent-Length:"
                              "103\r\nContent-Type: text/html\r\n\r\n",
                               63);

        nx_web_http_server_callback_data_send(server_ptr,
                                              "<HTML>\r\n<HEAD><TITLE>NetX"
                                              "HTTP Test </TITLE></HEAD>\r\n"
                                              :<BODY>\r\n<H1>NetX Test Page"
                                              "</H1>\r\n</BODY>\r\n</HTML>\r\n",
                                              103);

        /* Return completion status.  */
        return(NX_WEB_HTTP_CALLBACK_COMPLETED);
    }

    return(NX_SUCCESS);
}
```

nx_web_http_server_callback_generate_response_header

Create a response header in a callback function

Prototype

```
UINT nx_web_http_server_callback_generate_response_header(
                        NX_WEB_HTTP_SERVER *server_ptr,
                        NX_PACKET **packet_pptr,
                        CHAR *status_code, UINT content_length,
                        CHAR *content_type, CHAR* additional_header);
```

Description

This service is used in the HTTP(S) server callback routine (defined by the application) to generate an HTTP response header. The server callback routine is invoked when the HTTP server responds to Client GET, PUT and DELETE requests which require an HTTP response. This function takes the response information from the application and generates the appropriate response header. See the service *nx_web_http_server_create()* for more information on the server request callback routine.

Input Parameters

server_ptr	Pointer to HTTP Server control block.
packet_pptr	Pointer a packet pointer allocated for message
status_code	Indicate status of resource. Examples: **NX_WEB_HTTP_STATUS_OK** **NX_WEB_HTTP_STATUS_MODIFIED** **NX_WEB_HTTP_STATUS_INTERNAL_ERROR**
content_length	Size of content in bytes
content_type	Type of HTTP e.g. “text/plain”
additional_header	Pointer to additional header text

Return Values

NX_SUCCESS	(0x00)	Successfully created HTML header
NX_PTR_ERROR	(0x07)	Invalid pointer input

Allowed From

Threads

Example

```
CHAR demotestbuffer[] = "<html>\r\n\r\n<head>\r\n\r\n<title>Main \
         Window</title>\r\n</head>\r\n\r\n<body>Test message\r\n \
         </body>\r\n</html>\r\n";

/* my_request_notify is the application request notify callback registered with
   the HTTP server in nx_web_http_server_create, creates a response to the
   received Client request. */

UINT  my_request_notify(NX_WEB_HTTP_SERVER *server_ptr, UINT request_type,
                        CHAR *resource, NX_PACKET *recv_packet_ptr)
{

 NX_PACKET   *resp_packet_ptr;
 ULONG       string_length;
 CHAR        temp_string[30];
 ULONG       length = 0;


     length = strlen(&demotestbuffer[0]);

     /* Derive the client request type from the client request.  */
     string_length =  (ULONG) nx_web_http_server_type_get(server_ptr, server_ptr ->
                                nx_web_http_server_request_resource, temp_string);

     /* Null terminate the string. */
     temp_string[temp] = 0;

     /* Now build a response header with server status is OK and no additional
        header info.  */
     status = nx_web_http_server_callback_generate_response_header(http_server_ptr,
                                  &resp_packet_ptr, NX_WEB_HTTP_STATUS_OK,
                                  length, temp_string, NX_NULL);

     /* If status is NX_SUCCESS, the header was successfully appended. */

     /* Now add data to the packet.  */
     status = nx_packet_data_append(resp_packet_ptr, &demotestbuffer[0],
                    strlen(&demotestbuffer[0]),  server_ptr ->
                    nx_web_http_server_packet_pool_ptr, NX_WAIT_FOREVER);
     if (status != NX_SUCCESS)
     {
         nx_packet_release(resp_packet_ptr);
         return status;
     }



     /* Now send the packet! */
      status = nx_web_http_server_callback_packet_send(
                              &(server_ptr -> nx_web_http_server_socket),
                              resp_packet_ptr);

     if (status != NX_SUCCESS)
     {
        nx_packet_release(resp_packet_ptr);

        return status;
     }

     /* Let HTTP server know the response has been sent. */
    return(NX_WEB_HTTP_CALLBACK_COMPLETED);

}
```

nx_web_http_server_callback_packet_send

Send an HTTP packet from callback function

Prototype

```
UINT nx_web_http_server_callback_packet_send(
                                    NX_WEB_HTTP_SERVER *server_ptr,
                                    NX_PACKET *packet_ptr);
```

Description

This service sends a complete HTTP server response from an HTTP callback. HTTP server will send the packet with the NX_WEB_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.

The callback should return NX_WEB_HTTP_CALLBACK_COMPLETED.

See *nx_web_http_server_callback_generate_response_header()* for a more detailed example.

Input Parameters

server_ptr	Pointer to HTTP Server control block
packet_ptr	Pointer to the packet to send

Return Values

NX_SUCCESS	(0x00)	Successfully sent HTTP Server packet
NX_PTR_ERROR	(0x07)	Invalid pointer input

Allowed From

Threads

Example

```
/* The packet is appended with HTTP header and data and is ready to send to
   the Client directly. */

status = nx_web_http_server_callback_packet_send(server_ptr, packet_ptr);

if (status != NX_SUCCESS)
{

    nx_packet_release(packet_ptr);
}

return(NX_WEB_HTTP_CALLBACK_COMPLETED);
```

nx_web_http_server_callback_response_send

Send response from callback function

Prototype

```
UINT nx_web_http_server_callback_response_send(
                                    NX_WEB_HTTP_SERVER *server_ptr,
                                     CHAR *header,
                                     CHAR *information,
                                     CHAR additional_info);
```

Description

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_WEB_HTTP_CALLBACK_COMPLETED.

Input Parameters

server_ptr	Pointer to HTTP Server control block.
header	Pointer to the response header string.
information	Pointer to the information string.
additional_info	Pointer to the additional information string.

Return Values

NX_SUCCESS	(0x00)	Successfully sent HTTP Server response

Allowed From

Threads

Example

```
UINT  my_request_notify(NX_WEB_HTTP_SERVER *server_ptr, UINT request_type,
                        CHAR *resource, NX_PACKET *packet_ptr)
{
    /* Look for the test resource!  */
    if ((request_type == NX_WEB_HTTP_SERVER_GET_REQUEST) &&
         (strcmp(resource, "/test.htm") == 0))
    {

        /* In this example, we will complete the GET processing with
           a resource not found response.  */
         nx_web_http_server_callback_response_send(server_ptr,
                                     "HTTP/1.0 404 ",
                                     "NetX HTTP Server unable to find file: ",
                                      resource);

        /* Return completion status.  */
        return(NX_WEB_HTTP_CALLBACK_COMPLETED);
    }

    return(NX_SUCCESS);
}
```

nx_web_http_server_content_get

Get content from the request

Prototype

```
UINT nx_web_http_server_content_get(NX_WEB_HTTP_SERVER *server_ptr,
                                    NX_PACKET *packet_ptr,
                                    ULONG byte_offset,
                                    CHAR *destination_ptr,
                                    UINT destination_size,
                                    UINT *actual_size);
```

Description

This service attempts to retrieve the specified amount of content from the POST or PUT HTTP Client request. It should be called from the application's request notify callback specified during HTTP Server creation (*nx_web_http_server_create()*).

Input Parameters

server_ptr	Pointer to HTTP Server control block.
packet_ptr	Pointer to the HTTP Client request packet. Note that this packet must not be released by the request notify callback.
byte_offset	Number of bytes to offset into the content area.
destination_ptr	Pointer to the destination area for the content.
destination_size	Maximum number of bytes available in the destination area.
actual_size	Pointer to the destination variable that will be set to the actual size of the content copied.

Return Values

NX_SUCCESS	(0x00)	Successful HTTP Server content Get
NX_WEB_HTTP_ERROR	(0x30000)	HTTP Server internal error
NX_WEB_HTTP_DATA_END	(0x30007)	End of request content
NX_WEB_HTTP_TIMEOUT	(0x30001)	HTTP Server timeout in getting next packet of content
NX_PTR_ERROR	(0x07)	Invalid pointer input
NX_CALLER_ERROR	(0x11)	Invalid caller of this service

Allowed From

Threads

Example

```
/* Assuming we are in the application's request notify callback
   routine, retrieve up to 100 bytes of content starting at offset
   0.  */
status =  nx_web_http_server_content_get(&my_server, packet_ptr,
                                         0, my_buffer, 100, &actual_size);

/* If status is NX_SUCCESS, "my_buffer" contains "actual_size" bytes of
   request content. */
```

nx_web_http_server_content_get_extended

Get content from the request/supports zero length Content Length

Prototype

```
UINT nx_web_http_server_content_get_extended(
                                    NX_WEB_HTTP_SERVER *server_ptr,
                                    NX_PACKET *packet_ptr,
                                    ULONG byte_offset,
                                    CHAR *destination_ptr,
                                    UINT destination_size,
                                    UINT *actual_size);
```

Description

This service is almost identical to *nx_web_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_web_http_server_create()*).

Input Parameters

server_ptr	Pointer to HTTP Server control block.
packet_ptr	Pointer to the HTTP Client request packet. Note that this packet must not be released by the request notify callback.
byte_offset	Number of bytes to offset into the content area.
destination_ptr	Pointer to the destination area for the content.
destination_size	Maximum number of bytes available in the destination area.
actual_size	Pointer to the destination variable that will be set to the actual size of the content copied.

Return Values

NX_SUCCESS	(0x00)	Successful HTTP content get
NX_WEB_HTTP_ERROR	(0x30000)	HTTP Server internal error
NX_WEB_HTTP_DATA_END	(0x30007)	End of request content
NX_WEB_HTTP_TIMEOUT	(0x30001)	HTTP Server timeout in getting next packet
NX_PTR_ERROR	(0x07)	Invalid pointer input
NX_CALLER_ERROR	(0x11)	Invalid caller of this service

Allowed From

Threads

Example

```
/* Assuming we are in the application's request notify callback
   routine, retrieve up to 100 bytes of content starting at offset
   0.  */
status =  nx_web_http_server_content_get_extended(&my_server, packet_ptr,
                                                  0, my_buffer, 100, &actual_size);

/* If status is NX_SUCCESS, "my_buffer" contains "actual_size" bytes of
   request content. */
```

nx_web_http_server_content_length_get

Get length of content in the request/supports Content Length of zero value

Prototype

```
UINT nx_web_http_server_content_length_get(
                                NX_PACKET *packet_ptr,
                                UINT *content_length);
```

Description

This service attempts to retrieve the HTTP content length in the supplied packet. The return value indicates successful completion status and the actual length value is returned in the input pointer `content_length`. If there is no HTTP content/Content Length = 0, this routine still returns a successful completion status and the `content_length` input pointer points to a valid length (zero). It should be called from the application's request notify callback specified during HTTP Server creation (*nx_web_http_server_create()*).

Input Parameters

packet_ptr	Pointer to the HTTP Client request packet. Note that this packet must not be released by the request notify callback.
content_length	Pointer to value retrieved from Content Length field

Return Values

NX_SUCCESS	(0x00)	Successful HTTP Server Content Length Get
NX_WEB_HTTP_INCOMPLETE_PUT_ERROR	(0x3000F)	Improper HTTP header format
NX_PTR_ERROR	(0x07)	Invalid pointer input

Allowed From

Threads

Example

```
/* Assuming we are in the application's request notify callback
   routine, get the content length of the HTTP Client request.  */
ULONG content_length;

status =  nx_web_http_server_content_length_get(packet_ptr, &content_length);

/* If the "status" variable indicates successful completion, the "length"
   Variable contains the length of the HTTP Client request content area. */
```

nx_web_http_server_create

Create an HTTP Server instance

Prototype

```
UINT nx_web_http_server_create(NX_WEB_HTTP_SERVER *http_server_ptr,
      CHAR *http_server_name, NX_IP *ip_ptr, UINT server_port,
      FX_MEDIA *media_ptr, VOID *stack_ptr, ULONG stack_size,
      NX_PACKET_POOL *pool_ptr,
      UINT (*authentication_check)(NX_WEB_HTTP_SERVER *server_ptr,
            UINT request_type, CHAR *resource, CHAR **name,
            CHAR **password, CHAR **realm),
      UINT (*request_notify)(NX_WEB_HTTP_SERVER *server_ptr,
            UINT request_type, CHAR *resource, NX_PACKET *packet_ptr));
```

Description

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.

This service is used to create both plaintext HTTP servers and TLS-secured HTTPS servers. To enable HTTPS using TLS, see the service *nx_web_http_server_secure_configure()*.

Input Parameters

http_server_ptr	Pointer to HTTP Server control block.
http_server_name	Pointer to HTTP Server's name.
ip_ptr	Pointer to previously created IP instance.
server_port	TCP listening port for server instance.
media_ptr	Pointer to previously created FileX media instance.
stack_ptr	Pointer to HTTP Server thread stack area.
stack_size	Pointer to HTTP Server thread stack size.
authentication_check	Function pointer to application's authentication checking routine. If specified, this routine is called for each HTTP Client request. If this parameter is NULL, no authentication will be performed.
request_notify	Function pointer to application's request notify routine. If 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.

Return Values

NX_SUCCESS	(0x00)	Successful HTTP Server create.
NX_PTR_ERROR	(0x07)	Invalid HTTP Server, IP, media, stack, or packet pool pointer.
NX_WEB_HTTP_POOL_ERROR	(0x30009)	Packet payload of pool is not large enough to contain complete HTTP request.

Allowed From

Initialization, Threads

Example

```
/* Create an HTTP Server instance called “my_server.”  */
status =  nx_web_http_server_create(&my_server, “my server”, &ip_0,
              NX_WEB_HTTPS_SERVER_PORT, &ram_disk,
              stack_ptr, stack_size, &pool_0,
              my_authentication_check, my_request_notify);

/* If status equals NX_SUCCESS, the HTTP Server creation was successful. */
```

nx_web_http_server_delete

Delete an HTTP Server instance

Prototype

```
UINT nx_web_http_server_delete(NX_WEB_HTTP_SERVER *http_server_ptr);
```

Description

This service deletes a previously created HTTP Server instance.

Input Parameters

http_server_ptr Pointer to HTTP Server control block.

Return Values

NX_SUCCESS	(0x00)	Successful HTTP Server delete
NX_PTR_ERROR	(0x07)	Invalid HTTP Server pointer
NX_CALLER_ERROR	(0x11)	Invalid caller of this service

Allowed From

Threads

Example

```
/* Delete the HTTP Server instance called “my_server.”  */
status =  nx_web_http_server_delete(&my_server);

/* If status equals NX_SUCCESS, the HTTP Server delete was successful. */
```

nx_web_http_server_get_entity_content

Retrieve the location and length of entity data

Prototype

```
UINT nx_web_http_server_get_entity_content(
                                        NX_WEB_HTTP_SERVER *server_ptr,
                                        NX_PACKET **packet_pptr,
                                        ULONG *available_offset,
                                        ULONG *available_length);
```

Description

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, the 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.

Note that NX_WEB_HTTP_MULTIPART_ENABLE must be enabled to use this service. Also note that the application should not release the packet pointed to by packet_pptr. This is done internally by the HTTP server.

See *nx_web_http_server_get_entity_header()* for more details.

Input Parameters

server_ptr	Pointer to HTTP Server
packet_pptr	Pointer to location of packet pointer. Note that the application should not release this packet
available_offset	Pointer to offset of entity data from the packet prepend pointer
available_length	Pointer to length of entity data

Return Values

NX_SUCCESS	(0x00)	Successfully retrieved size and location of entity content
NX_WEB_HTTP_BOUNDARY_ALREADY_FOUND	(0x30016)	Content for the HTTP server internal multipart markers is already found

NX_WEB_HTTP_ERROR	(0x30000)	HTTP Server internal error
NX_PTR_ERROR	(0x07)	Invalid pointer input

Allowed From

Threads

Example

```
NX_WEB_HTTP_SERVER my_server;

UINT        offset, length;
NX_PACKET  *packet_ptr;

/* Inside the request notify callback, the HTTP server application first obtains
   the entity header to determine details about the multipart data.  If
   successful, it then calls this service to get the location of entity data:  */

status =  nx_web_http_server_get_entity_content(&my_server, &packet_ptr, *offset,
                                                &length);

/* If status equals NX_SUCCESS, offset and location determine the location of the
   entity data.  */
```

nx_web_http_server_get_entity_header

Retrieve the contents of entity header

Prototype

```
UINT nx_web_http_server_get_entity_header(
                                        NX_WEB_HTTP_SERVER *server_ptr,
                                        NX_PACKET **packet_pptr,
                                        UCHAR *entity_header_buffer,
                                        ULONG buffer_size);
```

Description

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.

Note that NX_WEB_HTTP_MULTIPART_ENABLE must be enabled to use this service. Note also that the application should not release the packet pointed to by packet_pptr.

Input Parameters

server_ptr	Pointer to HTTP Server
packet_pptr	Pointer to location of packet pointer. Note that the application should not release this packet
entity_header_buffer	Pointer to location to store entity header
buffer_size	Size of input buffer

Return Values

NX_SUCCESS	(0x00)	Successfully retrieved entity Header
NX_WEB_HTTP_NOT_FOUND	(0x30006)	Entity header field not found
NX_WEB_HTTP_TIMEOUT	(0x30001)	Time expired to receive next packet for multipacket client message
NX_PTR_ERROR	(0x07)	Invalid pointer input
NX_CALLER_ERROR	(0x11)	Invalid caller of this service
NX_WEB_HTTP_ERROR	(0x30000)	Internal HTTP error

Allowed From

Threads

Example

```
/* Buffer to hold data we are extracting from the request. */
UCHAR       buffer[1440];

/* my_request_notify() is the application request notify callback registered with
   the HTTP server in nx_web_http_server_create(), creates a response to the
   received Client request. */

UINT  my_request_notify(NX_WEB_HTTP_SERVER *server_ptr, UINT request_type,
                        CHAR *resource, NX_PACKET *packet_ptr)
{

  UINT        offset, length;
  NX_PACKET   *response_pkt;

  /* Process multipart data. */
  if(request_type == NX_WEB_HTTP_SERVER_POST_REQUEST)
  {

     /* Get the content header. */
     while(nx_web_http_server_get_entity_header(server_ptr, &packet_ptr, buffer,
                                          sizeof(buffer)) == NX_SUCCESS)
     {

        /* Header obtained successfully. Get the content data location. */
        while(nx_web_http_server_get_entity_content(server_ptr, &packet_ptr,
                                             &offset, &length) == NX_SUCCESS)
        {
             /* Write content data to buffer. */
             nx_packet_data_extract_offset(packet_ptr, offset, buffer, length,
                                          &length);
             buffer[length] = 0;
        }

      }

      /* Generate HTTP header. */
      status = nx_web_http_server_callback_generate_response_header(server_ptr,
                          &response_pkt, NX_WEB_HTTP_STATUS_OK, 800, "text/html",
                          "Server: NetX WEB HTTP 5.10\r\n");

      if(status == NX_SUCCESS)
      {
          if(nx_web_http_server_callback_packet_send(server_ptr, response_pkt) !=
                                               NX_SUCCESS)
          {
                 nx_packet_release(response_pkt);
          }
      }


  }
  else
  {
      /* Indicate we have not processed the response to client yet.*/
      return(NX_SUCCESS);
  }


  /* Indicate the response to client is transmitted. */
  return(NX_WEB_HTTP_CALLBACK_COMPLETED);
}
```

nx_web_http_server_gmt_callback_set

Set the callback to obtain GMT date and time

Prototype

```
UINT nx_web_http_server_gmt_callback_set(
                    NX_WEB_HTTP_SERVER *server_ptr,
                    VOID (*gmt_get)(NX_WEB_HTTP_SERVER_DATE *date);
```

Description

This service sets the callback to obtain GMT date and time with a previously created HTTP server. This service is invoked with the HTTP server is creating a header in HTTP server responses to the Client.

Input Parameters

server_ptr	Pointer to HTTP Server
gmt_get	Pointer to GMT callback
date	Pointer to the date retrieved

Return Values

NX_SUCCESS	(0x00)	Successfully set the callback
NX_PTR_ERROR	(0x07)	Invalid packet or parameter pointer.

Allowed From

Threads

Example

```
NX_WEB_HTTP_SERVER my_server;

VOID get_gmt(NX_WEB_HTTP_SERVER_DATE *now);

/* After the HTTP server is created by calling nx_web_http_server_create(), and
    before starting HTTP services when nx_web_http_server_start() is called, set
    the GMT retrieve callback:  */

status =  nx_web_http_server_gmt_callback_set(&my_server, gmt_get);

/* If status equals NX_SUCCESS, the gmt_get will be called to set the HTTP server
   response header date.  */
```

nx_web_http_server_invalid_userpassword_notify_set

Set the callback to handle invalid user/password

Prototype

```
UINT nx_web_http_server_invalid_userpassword_notify_set(
            NX_WEB_HTTP_SERVER *http_server_ptr,
            UINT (*invalid_username_password_callback)
                        (CHAR *resource,
                        ULONG  client_address,
                        UINT   request_type));
```

Description

This service sets the callback invoked when an invalid username and password is received in a Client get, put or delete request, either by digest or basic authentication. The HTTP server must be previously created.

Input Parameters

server_ptr	Pointer to HTTP Server
invalid_username_password_callback	Pointer to invalid user/pass callback
resource	Pointer to the resource specified by the client
client_address	Client address
request_type	Indicates client request type. May be: *NX_WEB_HTTP_SERVER_GET_REQUEST* *NX_WEB_HTTP_SERVER_POST_REQUEST* *NX_WEB_HTTP_SERVER_HEAD_REQUEST* *NX_WEB_HTTP_SERVER_PUT_REQUEST* *NX_WEB_HTTP_SERVER_DELETE_REQUEST*

Return Values

NX_SUCCESS	(0x00)	Successfully set the callback
NX_PTR_ERROR	(0x07)	Invalid pointer input

Allowed From

Threads

Example

```
NX_WEB_HTTP_SERVER my_server;
VOID invalid_username_password_callback(NX_ CHAR *resource,
                                        ULONG  client_address,
                                        UINT   request_type);

/* After the HTTP server is created by calling nx_web_http_server_create, and
   before starting HTTP services when nx_web_http_server_start() is called, set
   the invalid username password callback:  */

status =  nx_web_http_server_invalid_userpassword_notify_set( (&my_server,
                                          invalid_username_password_callback);


/* If status equals NX_SUCCESS, the invalid_username_password_callback function
   will be called when the HTTP server receives an invalid username/password.  */
```

nx_web_http_server_mime_maps_additional_set

Set additional MIME maps for HTML

Prototype

```
UINT nx_web_http_server_mime_maps_additional_set(
                        NX_WEB_HTTP_SERVER *server_ptr,
                        NX_WEB_HTTP_SERVER_MIME_MAP *mime_maps,
                        UINT mime_maps_num);
```

Description

This service allows the HTTP application developer to add additional MIME types from the default MIME types supplied by the NetX Web HTTP Server. See *nx_web_http_server_get_type()* for list of defined types.

When a client request is received, e.g. a GET request, HTTP server parses the requested file type from the HTTP header using preferentially the additional MIME map set and if no match if found, it looks for a match in the default MIME map of the HTTP server. If no match is found, the MIME type defaults to “text/plain”.

If the request notify function is registered with the HTTP server, the request notify callback can call *nx_web_http_server_type_get()* to parse the file type.

Input Parameters

server_ptr	Pointer to HTTP Server instance
mime_maps	Pointer to a MIME map array
mime_map_num	Number of MIME maps in array

Return Values

NX_SUCCESS	(0x00)	Successful HTTP Server MIME map set
NX_PTR_ERROR	(0x07)	Invalid pointer input

Allowed From

Initialization, Threads

Example

```
/* my_server is an NX_WEB_HTTP_SERVER previously created.  */

static NX_WEB_HTTP_SERVER_MIME_MAP my_mime_maps[] =
{
    {"abc",      "yourtype/abc"},
    {"xyz",      "mytype/xyz"},
};

status =  nx_web_http_server_mime_maps_additional_set(&my_server,
                                                      &my_mime_maps[0], 2);

/* If status equals NX_SUCCESS, two additional MIME types are added to the HTTP
  server MIME map set.” */
```

nx_web_http_server_response_packet_allocate

Allocate an HTTP(S) packet

Prototype

```
UINT nx_web_http_server_response_packet_allocate(
                                        NX_WEB_HTTP_SERVER *server_ptr,
                                        NX_PACKET **packet_ptr,
                                        ULONG wait_option);
```

Description

This service attempts to allocates a packet for the HTTP(S) server.

Note that if a subsequent NetX or HTTP Server API using this packet as input fails, such as nx_packet_data_append or nx_web_http_server_callback_packet_send, the application is responsible for releasing the packet.

Input Parameters

server_ptr	Pointer to HTTP Server control block.
packet_ptr	Pointer to allocated packet.
wait_option	Defines the wait time in ticks if there are no packets available in the packet pool. The wait options are defined as follows: **NX_NO_WAIT** (0x00000000) **NX_WAIT_FOREVER** (0xFFFFFFFF) **timeout value** (0x00000001 through 0xFFFFFFFE) Selecting NX_NO_WAIT causes the calling thread to return immediately if the request cannot be fulfilled. Selecting NX_WAIT_FOREVER causes the calling thread to suspend indefinitely until the HTTP Server responds to the request. Selecting a numeric value (0x1-0xFFFFFFFE) specifies the maximum number of timer-ticks to stay suspended while waiting for the HTTP Server response.

Return Values

NX_SUCCESS	(0x00)	Successful packet allocate
NX_NO_PACKET	(0x01)	No packet available
NX_WAIT_ABORTED	(0x1A)	Requested suspension was aborted by a call to *tx_thread_wait_abort*.
NX_INVALID_PARAMETERS	(0x4D)	Packet size cannot support protocol.
NX_PTR_ERROR	(0x07)	Invalid pointer input
NX_CALLER_ERROR	(0x11)	Invalid caller of this service.

Allowed From

Threads

Example

```
/* Allocate a packet for HTTP(S) Server and suspend for a maximum of 5 timer
   ticks if the pool is empty. */
status = nx_web_http_server_response_packet_allocate(&my_client, &packet_ptr, 5);

/* If status is NX_SUCCESS, the newly allocated packet pointer is found in the
   variable packet_ptr.  */
```

nx_web_http_server_packet_content_find

Extract content length and set pointer to start of data

Prototype

```
UINT nx_web_http_server_packet_content_find(
                                    NX_WEB_HTTP_SERVER *server_ptr,
                                    NX_PACKET **packet_ptr,
                                    UINT *content_length);
```

Description

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.

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_WEB_HTTP_SERVER_TIMEOUT_RECEIVE wait option.

Note this should not be called before calling *nx_web_http_server_get_entity_header()* because it modifies the packet prepend pointer past the entity header.

Input Parameters

server_ptr	Pointer to HTTP server instance
packet_ptr	Pointer to packet pointer for returning the packet with updated prepend pointer
content_length	Pointer to extracted `content_length`

Return Values

NX_SUCCESS	(0x00)	HTTP content length found and packet successfully updated
NX_WEB_HTTP_TIMEOUT	(0x30001)	Time expired waiting on next Packet
NX_PTR_ERROR	(0x07)	Invalid pointer input

Allowed From

Threads

Example

```
/* The HTTP server pointed to by server_ptr is previously created and started.
   The server has received a Client request packet, recv_packet_ptr, and the
   packet content find service is called from the request notify callback
   function registered with the HTTP server.  */

UINT content_length;

status = nx_web_http_server_packet_content_find(server_ptr, recv_packet_ptr,
                                                &content_length);

/* If status equals NX_SUCCESS, the content length specifies the content length
   and the packet pointer prepend pointer is set to the HTTP content (data). */
```

nx_web_http_server_packet_get

Receive the next HTTP packet

Prototype

```
UINT nx_web_http_server_packet_get(NX_WEB_HTTP_SERVER *server_ptr,
                                   NX_PACKET **packet_ptr);
```

Description

This service returns the next packet received on the HTTP server socket. The wait option to receive a packet is NX_WEB_HTTP_SERVER_TIMEOUT_RECEIVE.

Note that if successful the application is responsible for releasing the packet.

Input Parameters

server_ptr	Pointer to HTTP server instance
packet_ptr	Pointer to received packet

Return Values

NX_SUCCESS	(0x00)	Successfully received next HTTP packet
NX_WEB_HTTP_TIMEOUT	(0x30001)	Time expired waiting on next Packet
NX_PTR_ERROR	(0x07)	Invalid pointer input

Allowed From

Threads

Example

```
/* The HTTP server pointed to by server_ptr is previously created and started. */

UINT content_length;
NX_PACKET *recv_packet_ptr;

status = nx_web_http_server_packet_get(server_ptr, &recv_packet_ptr);

/* If status equals NX_SUCCESS, a Client packet is obtained. */
```

nx_web_http_server_param_get

Get parameter from the request

Prototype

```
UINT nx_web_http_server_param_get(NX_PACKET *packet_ptr,
                                  UINT param_number, CHAR *param_ptr,
                                  UINT *param_size, UINT max_param_size);
```

Description

This service attempts to retrieve the specified HTTP URL parameter in the supplied request packet. If the requested HTTP parameter is not present, this routine returns a status of NX_WEB_HTTP_NOT_FOUND. This routine should be called from the application's request notify callback specified during HTTP Server creation (*nx_web_http_server_create()*).

Input Parameters

packet_ptr	Pointer to HTTP Client request packet. Note that the application should not release this packet.
param_number	Logical number of the parameter starting at zero, from left to right in the parameter list.
param_ptr	Destination area to copy the parameter.
param_size	Return the total parameter data length (in bytes).
max_param_size	Maximum size of the parameter destination area.

Return Values

NX_SUCCESS	(0x00)	Successful HTTP Server parameter get
NX_WEB_HTTP_NOT_FOUND	(0x30006)	Specified parameter not found
NX_WEB_HTTP_IMPROPERLY_TERMINATED_PARAM	(0x30015)	Request parameter not properly terminated
NX_PTR_ERROR	(0x07)	Invalid pointer input
NX_CALLER_ERROR	(0x11)	Invalid caller of this service

Allowed From

Threads

Example

```
/* Assuming we are in the application's request notify callback
   routine, get the first parameter of the HTTP Client request.  */

status =  nx_web_http_server_param_get(request_packet_ptr, 0, param_destination,
                                       &param_size, 30);

/* If status equals NX_SUCCESS, the NULL-terminated first parameter can be found
   in "param_destination" and the size of that string can be found in the
   variable "param_size." */
```

nx_web_http_server_query_get

Get query from the request

Prototype

```
UINT nx_web_http_server_query_get(NX_PACKET *packet_ptr,
                                  UINT query_number,
                                  CHAR *query_ptr,
                                  CHAR *query_size,
                                  UINT max_query_size);
```

Description

This service attempts to retrieve the specified HTTP URL query in the supplied request packet. If the requested HTTP query is not present, this routine returns a status of NX_WEB_HTTP_NOT_FOUND. This routine should be called from the application's request notify callback specified during HTTP Server creation (*nx_web_http_server_create()*).

Input Parameters

packet_ptr	Pointer to HTTP Client request packet. Note that the application should not release this packet.
query_number	Logical number of the parameter starting at zero, from left to right in the query list.
query_ptr	Destination area to copy the query.
query_size	Return query data size (in bytes).
max_query_size	Maximum size of the query destination area.

Return Values

NX_SUCCESS	(0x00)	Successful HTTP Server query get
NX_WEB_HTTP_FAILED	(0x30002)	Query size too small.
NX_WEB_HTTP_NOT_FOUND	(0x30006)	Specified query not found
NX_WEB_HTTP_NO_QUERY_PARSED	(0x30013)	No query in Client request
NX_PTR_ERROR	(0x07)	Invalid pointer input
NX_CALLER_ERROR	(0x11)	Invalid caller of this service

Allowed From

Threads

Example

```
/* Assuming we are in the application's request notify callback
   routine, get the first query of the HTTP Client request.  */
status =  nx_web_http_server_query_get(request_packet_ptr, 0,
                                       query_destination, &query_size, 30);

/* If status equals NX_SUCCESS, the NULL-terminated first query can be found
   in "query_destination" and the length of that string can be found in the
   variable "query_size". */
```

nx_web_http_server_response_chunked_set

Set chunked transfer for HTTP(S) response

Prototype

```
UINT nx_web_http_server_response_chunked_set(
                                NX_WEB_HTTP_SERVER *server_ptr,
                                UINT chunk_size, NX_PACKET *packet_ptr);
```

Description

This service uses chunked transfer coding to send a custom HTTP(S) response data packet created with *nx_web_http_server_response_packet_allocate*() to the client.

Note: If the application uses chunked transfer coding to send a response data packet, it must call this service after calling *nx_web_http_server_response_packet_allocate*(), and before calling *nx_web_http_server_callback_packet_send*().

Input Parameters

client_ptr	Pointer to HTTP Client control block.
chunk_size	Size of the chunk-data in octets.
packet_ptr	HTTP(S) request data packet pointer.

Return Values

NX_SUCCESS	(0x00)	Successful set chunked.
NX_PTR_ERROR	(0x07)	Invalid pointer input

Allowed From

Threads

Example

```
/* Generate HTTP header.  */
nx_web_http_server_callback_generate_response_header(server_ptr,
       &response_pkt, NX_WEB_HTTP_STATUS_OK, 0,text/html",
       "Transfer-Encoding: chunked\r\n");

/* Create a new data packet response on the HTTP(S) Server instance. */
nx_web_http_server_response_packet_allocate(&my_server, &my_packet,
                                            NX_WAIT_FOREVER);

/* Set the chunked transfer.  */
Status = nx_web_http_server_response_chunked_set(&my_server, 128, my_packet)

/* At this point, user can fill the data into my_packet. *./
nx_packet_data_append(my_packet, data_ptr, data_size,
                      packet_pool, NX_WAIT_FOREVER);

/* Send data packet response to client. */
nx_web_http_server_calllback_packet_send(&my_server, my_packet);
```

nx_web_http_server_secure_configure

Configure an HTTP Server to use TLS for secure HTTPS

Prototype

```
UINT nx_web_http_server_secure_configure(
                        NX_WEB_HTTP_SERVER *http_server_ptr,
                        const NX_SECURE_TLS_CRYPTO *crypto_table,
                        VOID *metadata_buffer, ULONG metadata_size,
                        UCHAR* packet_buffer, UINT packet_buffer_size,
                        NX_SECURE_X509_CERT *identity_certificate,
                        NX_SECURE_X509_CERT *trusted_certificates[],
                        UINT trusted_certs_num,
                        NX_SECURE_X509_CERT *remote_certificates[],
                        UINT remote_certs_num,
                        UCHAR *remote_certificate_buffer,
                        UINT remote_cert_buffer_size);
```

Description

This service configures a previously created NetX Web HTTP server instance to use TLS for secure HTTPS communications. The parameters are used to configure all the possible TLS sessions with identical state so that each incoming HTTPS client experiences consistent behavior. The number of TLS sessions is controlled using the macro NX_WEB_HTTP_SESSION_MAX.

The cryptographic routine table (ciphersuite table) is shared between all TLS sessions as it just contains function pointers.

The metadata and packet reassembly buffers are each divided equally between all TLS sessions. If the buffer size is not evenly divisible by the number of sessions the remainder will be unused.

The passed-in identity certificate is used by all sessions. During TLS operation the server identity certificate is only read from so copies are not needed for each session.

The trusted certificates are added to each TLS session in the HTTPS server. These are used for Client certificate authentication which is automatically enabled when remote certificate space is provided.

The remote certificate array and buffer is shared by default between all TLS sessions. The remote certificates are used for Client certificate authentication which is automatically enabled when the remote certificate count is nonzero. Due to the buffer being shared some sessions may block during certificate validation.

To disable client certificate authentication, pass NX_NULL for the remote_certificates parameter and a value of 0 for the remote_certs_num parameter.

Return values will include any TLS error codes resulting from issues in the configuration of the TLS sessions.

Input Parameters

http_server_ptr	Pointer to HTTP Server instance.
crypto_table	Pointer to TLS ciphersuite table.
metadata_buffer	Pointer to cryptographic metadata buffer.
metadata_size	Size of cryptographic metadata buffer.
packet_buffer	TLS packet reassembly buffer.
packet_buffer	Size of TLS packet buffer – should be equal to (<desired TLS buffer size> * NX_WEB_HTTP_SESSION_MAX).
identity_certificate	TLS server identity certificate – will be used for all HTTPS server sessions.
trusted_certificates	Pointer to array of NX_SECURE_X509_CERT objects, used to validate incoming client certificates if client certificate authentication is enabled by passing a non-zero value for the *remote_certs_num* parameter.
trusted_certs_num	Number of trusted certificates in the *trusted_certificates* array.
remote_certificates	Pointer to array of NX_SECURE_X509_CERT objects, used for incoming client certificates.
remote_certs_num	Number of remote certificates. Should be the maximum number of expected certificates from clients. Client certificate authentication is enabled automatically when this is non-zero.
remote_certificate_buffer	Buffer to contain incoming remote certificates from clients if client certificate authentication is enabled.
remote_cert_buffer_size	Size of remote certificates buffer. Should be equal to (<maximum expected certificate size> * remote_certs_num).

Return Values

NX_SUCCESS	(0x00)	Successful initialization of the TLS session.
NX_NOT_CONNECTED	(0x38)	The underlying TCP socket is no longer connected.
NX_SECURE_TLS_UNRECOGNIZED_MESSAGE_TYPE	(0x102)	A received TLS message type is incorrect.
NX_SECURE_TLS_UNSUPPORTED_CIPHER	(0x106)	A cipher provided by the remote host is not supported.
NX_SECURE_TLS_HANDSHAKE_FAILURE	(0x107)	Message processing during the TLS handshake has failed.
NX_SECURE_TLS_HASH_MAC_VERIFY_FAILURE	(0x108)	An incoming message failed a hash MAC check.
NX_SECURE_TLS_TCP_SEND_FAILED	(0x109)	An underlying TCP socket send failed.
NX_SECURE_TLS_INCORRECT_MESSAGE_LENGTH	(0x10A)	An incoming message had an invalid length field.
NX_SECURE_TLS_BAD_CIPHERSPEC	(0x10B)	An incoming ChangeCipherSpec message was incorrect.
NX_SECURE_TLS_INVALID_SERVER_CERT	(0x10C)	An incoming TLS certificate is unusable for identifying the remote TLS server.
NX_SECURE_TLS_UNSUPPORTED_PUBLIC_CIPHER	(0x10D)	The public-key cipher provided by the remote host is unsupported.
NX_SECURE_TLS_NO_SUPPORTED_CIPHERS	(0x10E)	The remote host has indicated no ciphersuites that are supported by the NetX Secure TLS stack.
NX_SECURE_TLS_UNKNOWN_TLS_VERSION	(0x10F)	A received TLS message had an unknown TLS version in its header.

NX_SECURE_TLS_UNSUPPORTED_TLS_VERSION		
	(0x110)	A received TLS message had a known but unsupported TLS version in its header.
NX_SECURE_TLS_ALLOCATE_PACKET_FAILED		
	(0x111)	An internal TLS packet allocation failed.
NX_SECURE_TLS_INVALID_CERTIFICATE		
	(0x112)	The remote host provided an invalid certificate.
NX_SECURE_TLS_ALERT_RECEIVED		
	(0x114)	The remote host sent an alert indicating an error and ending the TLS session.
NX_PTR_ERROR	(0x07)	Tried to use an invalid pointer.

Allowed From

Initialization, Threads

Example

```
/* Create the HTTPS Server. */

status = nx_web_http_server_create(&my_server, "My HTTP Server", &ip_0,
                                   &ram_disk, &server_stack,
                                   sizeof(server_stack), &pool_0,
                                   authentication_check,
                                   server_request_callback);

/* Initialize device certificate (used for all sessions in HTTPS server). */
nx_secure_x509_certificate_initialize(&certificate, device_cert_der,
                                      device_cert_der_len, NX_NULL, 0,
                                      device_cert_key_der,
                                      device_cert_key_der_len,
                                      NX_SECURE_X509_KEY_TYPE_RSA_PKCS1_DER);

/* Setup TLS session for the HTTPS server. Note that since the remote_certs_num
   parameter is 0, no trusted certificates are needed, and Client certificate
   authentication is disabled. */
status = nx_web_http_server_secure_configure(&my_server, &nx_crypto_tls_ciphers,
                                             crypto_metadata,
                                             sizeof(crypto_metadata),
                                             tls_packet_buffer,
                                             sizeof(tls_packet_buffer),
                                             &certificate, NX_NULL, 0, NX_NULL,
                                             0, NX_NULL, 0);


/* Start an HTTPS Server with TLS.  */
status = nx_web_http_server_start(&my_server);

/* If status equals NX_SUCCESS, the HTTP Server has been started. */
```

nx_web_http_server_start

Start the HTTP Server

Prototype

```
UINT nx_web_http_server_start(NX_WEB_HTTP_SERVER *http_server_ptr);
```

Description

This service starts a previously created HTTP or HTTPS Server instance.

HTTPS servers share the same API as HTTP. To enable HTTPS using TLS on an HTTP server, see the service *nx_web_http_server_secure_configure().*

Input Parameters

http_server_ptr	Pointer to HTTP Server instance.

Return Values

NX_SUCCESS	(0x00)	Successful HTTP Server Start
NX_PTR_ERROR	(0x07)	Invalid pointer input

Allowed From

Initialization, Threads

Example

```
/* Start the HTTP Server instance "my_server."  */
status =  nx_web_http_server_start(&my_server);

/* If status equals NX_SUCCESS, the HTTP Server has been started. */
```

nx_web_http_server_stop

Stop the HTTP Server

Prototype

```
UINT nx_web_http_server_stop(NX_WEB_HTTP_SERVER *http_server_ptr);
```

Description

This service stops the previously create HTTP Server instance. This routine should be called prior to deleting an HTTP Server instance.

Input Parameters

http_server_ptr	Pointer to HTTP Server instance.

Return Values

NX_SUCCESS	(0x00)	Successful HTTP Server Stop
NX_PTR_ERROR	(0x07)	Invalid pointer input
NX_CALLER_ERROR	(0x11)	Invalid caller of this service

Allowed From

Threads

Example

```
/* Stop the HTTP Server instance "my_server."  */
status =  nx_web_http_server_stop(&my_server);

/* If status equals NX_SUCCESS, the HTTP Server has been stopped. */
```

nx_web_http_server_type_get

Extract file type from Client HTTP request

Prototype

```
UINT nx_web_http_server_type_get(NX_WEB_HTTP_SERVER *http_server_ptr,
                                 CHAR *name, CHAR *http_type_string,
                                 UINT *string_size);
```

Description

This service extracts the HTTP request type in the buffer *http_type_string* and its length in *string_size* from the input buffer *name*, usually the URL. If no MIME map is found, it defaults to the “text/plain” type. Otherwise it compares the extracted type against the HTTP Server default MIME maps for a match. The default MIME maps in NetX Web HTTP Server are:

html	text/html
htm	text/html
txt	text/plain
gif	image/gif
jpg	image/jpeg
ico	image/x-icon

If supplied, it will also search a user defined set of additional MIME maps. See *nx_web_http_server_mime_maps_addtional_set()* for more details on user defined maps.

Input Parameters

http_server_ptr	Pointer to HTTP Server instance
name	Pointer to buffer to search
http_type_string	Pointer to extracted HTML type string
string_size	Pointer to return extracted HTML type string length.

Return Values

NX_SUCCESS	(0x00)	Successful extraction of type
NX_PTR_ERROR	(0x07)	Invalid pointer input
NX_WEB_HTTP_EXTENSION_MIME_DEFAULT	(0x30019)	Default “text/plain” returned.

Allowed From

Application

Example

```
/* my_server is a previously created HTTP server, which starts accepting client
   requests when nx_web_http_server_start is called*/

CHAR temp_string[20];
UINT string_length;

/* Extract the HTTP type. */
    string_length =  nx_web_http_server_type_get(&my_server_ptr,
                my_server.nx_web_http_server_request_resource, temp_string);

/* If string_length is non zero, the HTTP string is extracted. */
```

For a more detailed example, see the description for *nx_web_http_server_callback_generate_response_header().*

NetX Web Hypertext Transfer Protocol (HTTP) and
Hypertext Transfer Protocol Secure (HTTPS) User Guide

Publication Date: Rev.5.13 Feb 25, 2019

Published by: Renesas Electronics Corporation

NetX Web Hypertext Transfer Protocol (HTTP) and Hypertext Transfer Protocol Secure (HTTPS) User Guide

Renesas Electronics Corporation

R11UM0095EU0513