EBS HTTP Managed Client Library

Introduction

The EBS HTTP Managed Client Library provides client-side HTTP functionality for embedded platforms. This allows programmers to interface with HTTP servers from within embedded applications without the need to understand the details of the HTTP protocol. The library handles all of the details for transacting with the server. Applications include Web-based devices, cloud computing, implementing REST-ful and other Service Orientated Architectures.

The EBS HTTP Managed Client supports the following features:

- Client side support for GET, POST, and PUT operations
- Server side support web server functions and to layered protociols like SOAP and GENA.
- Optional modules to support:
- Soap Client.
- Soap Server.
- Integrates with EBS's UPnP solution.
- Security (HTTP over SSL)
- Cookies
- Basic/Digest authentication
- Persistent connections
- Blocking and non-blocking operation
- Single thread and Multi-thread safe modes

Support for GET, POST, and PUT Operations

The Managed Client uses a transaction model for HTTP operations. A transaction consists of the following steps:

1. The transaction is initiated by specifying the server name, port, and type of network transport to be used (for example, standard TCP/IP or SSL). The result of initiating a transaction is receiving a handle to a Managed Client Session, which uniquely identifies this transaction in subsequent API calls.
2. A single command, such as a GET or POST command, is issued. Each command has a number of options that may be used. For example, GET allows you to optionally specify an If-Modified-Since date so that the remote server may opt to not re-send data that has not changed since the last time the data was sent.

3. In the case of a POST or PUT command, data is written to the server. The format of this data is specified as an option to the POST or PUT command.
4. The server response is read. This response includes information like:

   The status code returned by the server, which indicates whether the request was successful, if authorization is needed, whether the requested file has moved, etc.

   The MIME type of the response data that the server has sent

   The date that the request was processed

   The expiration date for caching purposes

   The new location for the file, if it has been moved
5. In the case of a POST or GET, the response data is read.
6. The transaction is ended in one of two ways:

   Graceful close - this allows the underlying network connection to stay open for future transactions. This is transparent to the application, however; new transactions must still be explicitly initiated; you can not simply re-use the session handle for more commands.

   Hard close - ensures that the underlying network connection is shut down.

Optional Add-on Modules

Each add-on module to the HTTP Managed Client is independently built and linked into the application. An add-on module is utilized by initializing a context structure and then handing that structure into the initialization routine for the Managed Client. Each add-on's context structure contains the necessary function pointers and hooks for the Managed Client to bind to it at runtime and utilize its functionality. Usually, the add-on also defines some extra functions that operate on the context structure. For example, the cookies module has functions to explicitly add and remove cookies from the context.

Persistent Connections

The transaction model described above effectively allows the HTTP Managed Client to implement persistent connections in a way that is for the most part transparent to the application. It is possible, with this model, for the Managed Client to be maintaining active network connections even when no transaction is pending. Therefore, there are API functions provided to allow the application to explicitly shut down network connections, either by host or universally.

Blocking and Non-Blocking Operation

When a transaction is initiated, blocking or non-blocking mode may be specified. In the blocking mode, all API calls which involve some kind of lower-level network I/O will not return until the operation has been completed. This means, for example, that the call to initiate a transaction will not return until the connection has been fully established and it is safe to issue a command. In the case of reading data, blocking mode may allow the

read function to return before the specified number of bytes has been read, but if there is data remaining to be received, it will not return without having received at least some data.

In non-blocking mode, these same functions will always return as soon as possible. In this mode, it is often useful to utilize the Managed Client's select functions to block for a certain amount of time waiting for certain events to occur (for example, block waiting for at least one of a set of sessions to receive some data).

Single thread and Multi-thread safe modes

The thread-safe option available for the Managed Client is a compile-time only option. Each add-on module also has its own thread-safe compile-time option which must be enabled. The Managed Client and the add-ons each require a mutex to be passed in on initialization when thread-safe mode is enabled.

Example Code

The best way to understand the interface to the Managed Client is through example. The source distribution contains several example programs. Below is a list of examples provided.

- Simple GET Example - Downloading a file using GET
- Multiple simultaneous GET Example - Downloading multiple files simultaneously using the GET method
- Secure HTTP Example - Shows how to use HTTPS to perform secure operations
- Cookies Example - Shows how to enable cookies
- POST Example - Shows how to POST data to a server
- Authentication Example - Downloading a file using authentication

API Reference

Below is a list of API's for the Managed Client.

HTTP_ManagedClientInit

Initialize an HTTP Managed Client instance.

HTTP_ManagedClientInitMT

Initialize a thread-safe HTTP Managed Client instance.

HTTP_ManagedClientDestroy

Destroy an HTTP Managed Client instance.

HTTP_ManagedClientCloseAll

Close any open connections

HTTP_ManagedClientCloseHost

Close any open connections to a particular host

HTTP_ManagedClientCloseStale

Close any open connections that have been reset or have timed out.

HTTP_ManagedClientStartTransaction

Initiate a transaction with an HTTP server.

HTTP_ManagedClientGet

Send a GET request

HTTP_ManagedClientPut

Send a PUT request

HTTP_ManagedClientPost

Send a POST request

HTTP_ManagedClientRequestEx

Send a request

HTTP_ManagedClientRequestHeaderDone

HTTP_ManagedClientReadResponseInfo

Get information about the server's response.

HTTP_ManagedClientReadResponseInfoEx

Get information about the server's response.

HTTP_ManagedClientReadSelect

Wait for input on a managed client session

HTTP_ManagedClientWriteSelect

Wait for a managed client session to be ready to send request/data

HTTP_ManagedClientSelect

Select sessions for read-ready, write-ready or error condition

HTTP_ManagedClientWrite

Write data to a remote server.

HTTP_ManagedClientWriteDone

Indicate that all data for a given request has been sent.

HTTP_ManagedClientRead

Read data from a session

HTTP_ManagedClientReadFrom

Read a datagram-type HTTP response

HTTP_ManagedClientFinishTransaction

Finish a transaction

HTTP_ManagedClientCloseSession

Close a session and its associated connection

HTTP_ManagedClientGetSessionSocket

get the socket associated with the http session

```
int HTTP_ManagedClientInit
( HTTPManagedClient* client,
const HTTP_CHAR* userAgent,
const HTTP_CHAR* acceptTypes,
unsigned useKeepAlive,
HTTPCookieContext* cookieContext,
HTTPAuthContext* authContext,
RTP_HANDLE sslContext,
unsigned sslEnabled,
HTTP_INT32 writeBufferSize,
HTTP_INT32 maxTotalConnections,
HTTP_INT32 maxHostConnections )
```

Initialize an HTTP Managed Client instance.

Description

This function must be called before any other managed client calls to create a context in which the client will operate. Multiple independent client contexts may be created from within the same application.

Parameters:

client - pointer to uninitialized HTTPManagedClient structure
userAgent - string to use to identify client to remote hosts
acceptTypes - string to send to server to indicate what data types we accept
useKeepAlive - boolean: if non-zero, try to keep connections open to speed things up
cookieContext - optional: initialized instance of HTTPCookieContext for cookie support
authContext - optional: initialized instance of HTTPAuthContext for authentication
sslContext - optional: initialized RTSSL context if secure HTTP is to be utilitized
sslEnabled - boolean: if sslContext is valid this MUST be non-zero; otherwise 0
writeBufferSize - number of bytes to buffer when sending
maxTotalConnections - max total connections to open simultaneously
maxHostConnections - max connections allowed to any one remote host

Returns:

0 on success, negative on failure

See Also:

HTTP_ManagedClientDestroy

```
int HTTP_ManagedClientInitMT
( HTTPManagedClient* client,
const HTTP_CHAR* userAgent,
const HTTP_CHAR* acceptTypes,
unsigned useKeepAlive,
HTTPCookieContext* cookieContext,
HTTPAuthContext* authContext,
RTP_HANDLE sslContext,
unsigned sslEnabled,
HTTP_INT32 writeBufferSize,
HTTP_INT32 maxTotalConnections,
HTTP_INT32 maxHostConnections,
RTP_MUTEX lock )
```

Initialize a thread-safe HTTP Managed Client instance.

Description

This function must be called before any other managed client calls to create a context in which the client will operate. Multiple independent client contexts may be created from within the same application. The HTTP_MANAGED_CLIENT_THREADSAFE symbol must be defined at compile time (enabling thread-safe mode) to use this function. If threadsafe mode is enabled, then this function must be used in place of HTTP_ManagedClientInit.

The mutex passed into this function must not be used for any other purpose while this managed client instance is in operation. When the managed client is destroyed, the application must also dispose of the mutex in an appropriate manner.

Parameters:

client - pointer to uninitialized HTTPManagedClient structure
userAgent - string to use to identify client to remote hosts
acceptTypes - string to send to server to indicate what data types we accept
useKeepAlive - boolean: if non-zero, try to keep connections open to speed things up
cookieContext - optional: initialized instance of HTTPCookieContext for cookie support
authContext - optional: initialized instance of HTTPAuthContext for authentication
sslContext - optional: initialized RTSSL context if secure HTTP is to be utilitized
sslEnabled - boolean: if sslContext is valid this MUST be non-zero; otherwise 0
writeBufferSize - number of bytes to buffer when sending
maxTotalConnections - max total connections to open simultaneously
maxHostConnections - max connections allowed to any one remote host
lock - a mutex to use for thread synchronization

Returns:

0 on success, negative on failure

See Also:

HTTP_ManagedClientDestroy

void HTTP ManagedClientDestroy
(HTTPManagedClient* client)

Destroy an HTTP Managed Client instance.

Description

This function must be called once all managed client operations have been completed to free any resources being used by the managed client.

Parameters:

client - the client instance to destroy

Returns:

nothing

void HTTP ManagedClientCloseAll
(HTTPManagedClient* client)

Close any open connections

Description

Closes any connections that were left open because of the keepAlive option. The application can be assured that all connections (from this managed client, anyhow) are closed when this function returns.

Parameters:

client - the client whose connections to close

Returns:

nothing

```
void HTTP_ManagedClientCloseHost
( HTTPManagedClient* client,
const HTTP_CHAR* hostName,
HTTP_UINT16 port )
```

Close any open connections to a particular host

Description

Closes any connections to specific host that were left open because of the keepAlive option.

Parameters:

client - the managed client instance
hostName - the host name to close
port - the port number of open connections or 0 for all

Returns:

nothing

void HTTP ManagedClientCloseStale
(HTTPManagedClient* client)

Close any open connections that have been reset or have timed out.

Description

Calling this function periodically will assure that the managed client in quest is a good citizen in terms of not holding onto sockets (connected or otherwise) for too long.

Parameters:

client - the managed client instance

Returns:

nothing

```
int HTTP_ManagedClientStartTransaction
( HTTPManagedClient* client,
const HTTP_CHAR* host,
HTTP_UINT16 port,
HTTP_INT16 ipType,
HTTPManagedSessionType type,
unsigned blocking,
HTTPManagedClientSession** session )
```

Initiate a transaction with an HTTP server.

Description

Every HTTP operation provided by the managed client can be seen as a single transaction, or request/response pair. Each transaction takes place within the context of an HTTPManagedClientSession. An HTTPManagedClientSession corresponds to a single connection to a remote server. The managed client is designed to handle the specifics of establishing and closing connections automatically; thus, while calling HTTP_ManagedClientStartTransaction may cause a new connection to be established, it may just as easily find an existing open connection and pass it back through the session pointer (see "session" argument below).

This function, therefore, should not be viewed as explicitly opening a connection, but rather as setting up the managed client to send an HTTP request to a given remote host.

The session returned by this function is only good for a single transaction. Once the transaction is complete (or has been aborted), HTTP_ManagedClientFinishTransaction or HTTP_ManagedClientCloseSession (to physically close the connection) must be called.

If the blocking option is set, this function may block until a connection is established with the remote host. If it is not, this function will return immediately; but in this case, the application should use HTTP_ManagedClientSelect to determine when a request can be sent.

Parameters:

client - a managed client instance
host - the host name of the server
port - the port on the remote host, or 0 for the default
ipType - Type of address IPv4 or IPv6
type - the type of connection (TCP, secure TCP, UDP mulitcast, etc.)
blocking - boolean: use blocking I/O or non-blocking
session - pointer to HTTPManagedClientSession pointer to return the session

Returns:

0 on success, negative on failure

```
int HTTP_ManagedClientGet
( HTTPManagedClientSession* session,
const HTTP_CHAR* path,
RTP_TIMESTAMP* ifModifiedSince )
```

Send a GET request

Description

Downloads a file using the HTTP GET method, with optional if-modified-since date.

Parameters:

session - the session over which to perform the GET
path - the file to get on the server
ifModifiedSince - instruct the server to send the file only if it has been modified after this date

Returns:

0 on success, negative on failure

Preconditions:

a session pointer must have been returned by HTTP_ManagedClientStartTransaction for each time this function is called

```
int HTTP_ManagedClientPut
( HTTPManagedClientSession* session,
const HTTP_CHAR* path,
HTTP_CHAR* contentType,
HTTP_INT32 contentLength )
```

Send a PUT request

Description

Uploads a file using the HTTP PUT method. The application should always specify the content MIME type. The contentLength is optional, however. If the contentLength is not specified, then the data is sent in chunked transfer-encoding (each call to HTTP_ManagedClientWrite will generate a single chunk). In either case, HTTP_ManagedClientWriteDone must be called once all data has been written.

Parameters:

session - a managed client instance
path - the server-side file name
contentType - the mime type of the data ("text/xml", for example)
contentLength - optional: pointer to 32-bit integer indicating the size of the file being sent; NULL if not specified

Returns:

0 on success, negative on failure

Preconditions:

a session pointer must have been returned by HTTP_ManagedClientStartTransaction for each time this function is called

```
int HTTP_ManagedClientPost
( HTTPManagedClientSession* session,
const HTTP_CHAR* path,
HTTP_CHAR* contentType,
HTTP_INT32 contentLength )
```

Send a POST request

Description

Uploads a file and requests response data using the HTTP POST method. The application should always specify the content MIME type. The contentLength is optional, however. If the contentLength is not specified, then the data is sent in chunked transfer-encoding (each call to HTTP_ManagedClientWrite will generate a single chunk). In either case, HTTP_ManagedClientWriteDone must be called once all data has been written.

Parameters:

session - a managed client instance
path - the server-side file name
contentType - the mime type of the data ("application/x-www-form-urlencoded", for example)
contentLength - optional: pointer to 32-bit integer indicating the size of the file being sent; NULL if not specified

Returns:

0 on success, negative on failure

Preconditions:

a session pointer must have been returned by HTTP_ManagedClientStartTransaction for each time this function is called

```
int HTTP_ManagedClientReadResponseInfo
( HTTPManagedClientSession* session,
HTTPResponseInfo* responseInfo )
```

Get information about the server's response.

Description

This function populates the structure pointed to by responseInfo with various information about the server response to a client-initiated request, such as the mime type of the data being received, the status code (indicating success, file moved, authentication required, file not modified, etc.).

Parameters:

session - the session for which to get info
responseInfo - uninitialized HTTPResponseInfo structure to hold info

Returns:

0 on success, negative on failure

Preconditions:

A request must have been sent over the given session using HTTP_ManagedClientGet, HTTP_ManagedClientPut, or HTTP_ManagedClientPost

```
int HTTP_ManagedClientReadResponseInfoEx
( HTTPManagedClientSession* session,
HTTPResponseInfo* responseInfo,
HTTPHeaderCallback processHeaderFn,
void* processHeaderData )
```

Get information about the server's response.

Description

This function populates the structure pointed to by responseInfo with various information about the server response to a client-initiated request, such as the mime type of the data being received, the status code (indicating success, file moved, authentication required, file not modified, etc.).

Returns:

0 on success, negative on failure

Preconditions:

A request must have been sent over the given session using HTTP_ManagedClientGet, HTTP_ManagedClientPut, or HTTP_ManagedClientPost

```
int HTTP_ManagedClientReadSelect
( HTTPManagedClientSession* session,
HTTP_INT32 timeoutMsec )
```

Wait for input on a managed client session

Description

This function blocks for at most timeoutMsec milliseconds waiting for data to arrive over a managed client session.

Returns:

non-negative on success, negative on failure

See Also:

HTTP_ManagedClientSelect, HTTP_ManagedClientRead

```
int HTTP_ManagedClientWriteSelect
( HTTPManagedClientSession* session,
HTTP_INT32 timeoutMsec )
```

Wait for a managed client session to be ready to send request/data

Description

This function blocks for at most timeoutMsec milliseconds waiting for a managed client session to enter a state were it is able to send a new request or data.

Returns:

non-negative on success, negative on failure

See Also:

HTTP_ManagedClientSelect, HTTP_ManagedClientWrite

```
int HTTP_ManagedClientSelect
( HTTPManagedClientSession** writeList,
HTTP_INT16* writeNum,
HTTPManagedClientSession** readList,
HTTP_INT16* readNum,
HTTPManagedClientSession** errList,
HTTP_INT16* errNum,
HTTP_INT32 timeoutMsec )
```

Select sessions for read-ready, write-ready or error condition

Description

This function serves a purpose very similar to the sockets API call "select". It blocks for at most timeoutMsec milliseconds (possibly less, if one of the sessions is in the specified condition). When it returns, all sessions NOT in the condition associated with each of the three arrays will have been removed from those respective arrays (and the associated sizes of the arrays modified accordingly).

If a session is in non-blocking mode, then selecting for write using this function will indicate the connection is in a state to send a request using HTTP_ManagedClientGet, HTTP_ManagedClientPut, or HTTP_ManagedClientPost

Parameters:

writeList - pointer to an array of HTTPManagedClientSession pointers to be selected for writing
writeNum - pointer to an integer storing the number of elements in writeList
readList - pointer to an array of HTTPManagedClientSession pointers to be selected for reading
readNum - pointer to an integer storing the number of elements in readList
errList - pointer to an array of HTTPManagedClientSession pointers to be selected for errors
errNum - pointer to an integer storing the number of elements in errList
timeoutMsec - maximum time to block for (milliseconds)

Returns:

non-negative on success, negative on failure

```
HTTP_INT32 HTTP_ManagedClientWrite
( HTTPManagedClientSession* session,
HTTP_UINT8* buffer,
HTTP_INT32 size )
```

Write data to a remote server.

Description

This function is called after HTTP_ManagedClientPut, or HTTP_ManagedClientPost to upload data to the server. When all data for the given request has been sent, HTTP_ManagedClientWriteDone must be called to signal that all data has been written.

Parameters:

session - session over which to send data
buffer - pointer to the data to send
size - number of bytes out of buffer to send

Returns:

number of bytes written on success, negative on failure

```
int HTTP ManagedClientWriteDone
( HTTPManagedClientSession* session )
```

Indicate that all data for a given request has been sent.

Description

This function must be called after the request is sent once all associated data has been written to the session.

Parameters:

session - session for which to signal done writing

Returns:

0 on success, negative on failure

```
HTTP_INT32 HTTP_ManagedClientRead
( HTTPManagedClientSession* session,
HTTP_UINT8* buffer,
HTTP_INT32 size )
```

Read data from a session

Description

This function must be called after the request and all associated data has been sent. If the session is in non-blocking mode, this function may return a HTTP_EWOULDBLOCK value to indicate that no data is yet available. A return value of 0 indicates that all data has been read from this session and the session should be closed using HTTP_ManagedClientCloseSession or HTTP_ManagedClientFinishTransaction.

Parameters:

session - session from which to read data
buffer - buffer into which to read data
size - max number of bytes to read

Returns:

bytes read or error code

```
HTTP_INT32 HTTP_ManagedClientReadFrom
( HTTPManagedClientSession* session,
HTTPResponseInfo* responseInfo,
HTTPHeaderCallback processHeaderFn,
void* processHeaderData,
HTTP_UINT8* buffer,
HTTP_INT32 size,
RTP_NET_ADDR* fromAddr )
```

Read a datagram-type HTTP response

Description

This function is only relevant to HTTP-over-UDP sessions.

Returns:

nothing

See Also:

HTTP_ManagedClientRead, HTTP_ManagedClientStartTransaction

```
void HTTP_ManagedClientFinishTransaction
( HTTPManagedClientSession* session )
```

Finish a transaction

Description

This function (or HTTP_ManagedClientCloseSession, but not both) must be called once a transaction is complete to allow the managed client to reclaim resources used by the session.

Parameters:

session - the session whose transction is complete

Returns:

nothing

See Also:

HTTP_ManagedClientCloseSession

void HTTP_ManagedClientCloseSession
(HTTPManagedClientSession* session)

Close a session and its associated connection

Description

This function performs the same task as HTTP_ManagedClientFinishTransaction except that it also closes the network connection associated with the given session. This ensures that this session's connection will not be used for any future transactions.

Parameters:

session - the session to close

Returns:

nothing

See Also:

HTTP_ManagedClientFinishTransaction