Renesas Synergy™ Platform

NetX Duo™ MQTT Module Guide

R11AN0344EU0100
Rev.1.00
Oct 22, 2018

Introduction

This module guide will enable you to effectively use a module in your own design. Upon completion of this guide, you will be able to add this module to your own design, configure it correctly for the target application, and write code using the included application project code as a reference and efficient starting point. References to more detailed API descriptions and suggestions of other application projects that illustrate more advanced uses of the module are available on the Renesas Synergy Knowledge Base (as described in the References section at the end of this document) and should be valuable resources for creating more complex designs.

The MQTT (Message Queue Telemetry Transport) communication protocol is based on a publisher-subscriber model. A data producer can publish information to other clients through a broker. Multiple data consumers, if interested in a topic, can subscribe to the topic through the broker. The broker is responsible for authentication and authorization of the clients and delivering published messages to its topic subscribers. In this publisher-subscriber model, multiple clients may publish data with the same topic. A client will receive the messages that are published if the client subscribes to the same topic.

Contents

1. NetX Duo MQTT Client Module Features

- Compliant with OASIS MQTT Version 3.1.1 Oct 29th, 2014. The specification can be found at: http://mqtt.org/
- Provides option to enable/disable TLS for secure communications using NetX Secure in SSP
- Supports QoS and provides the ability to choose the levels that can be selected while publishing the message
- Internally buffers and maintains queue of received messages
- Provides mechanism to register callback when new message is received.
- Provides mechanism to register callback when connection with the broker is terminated.

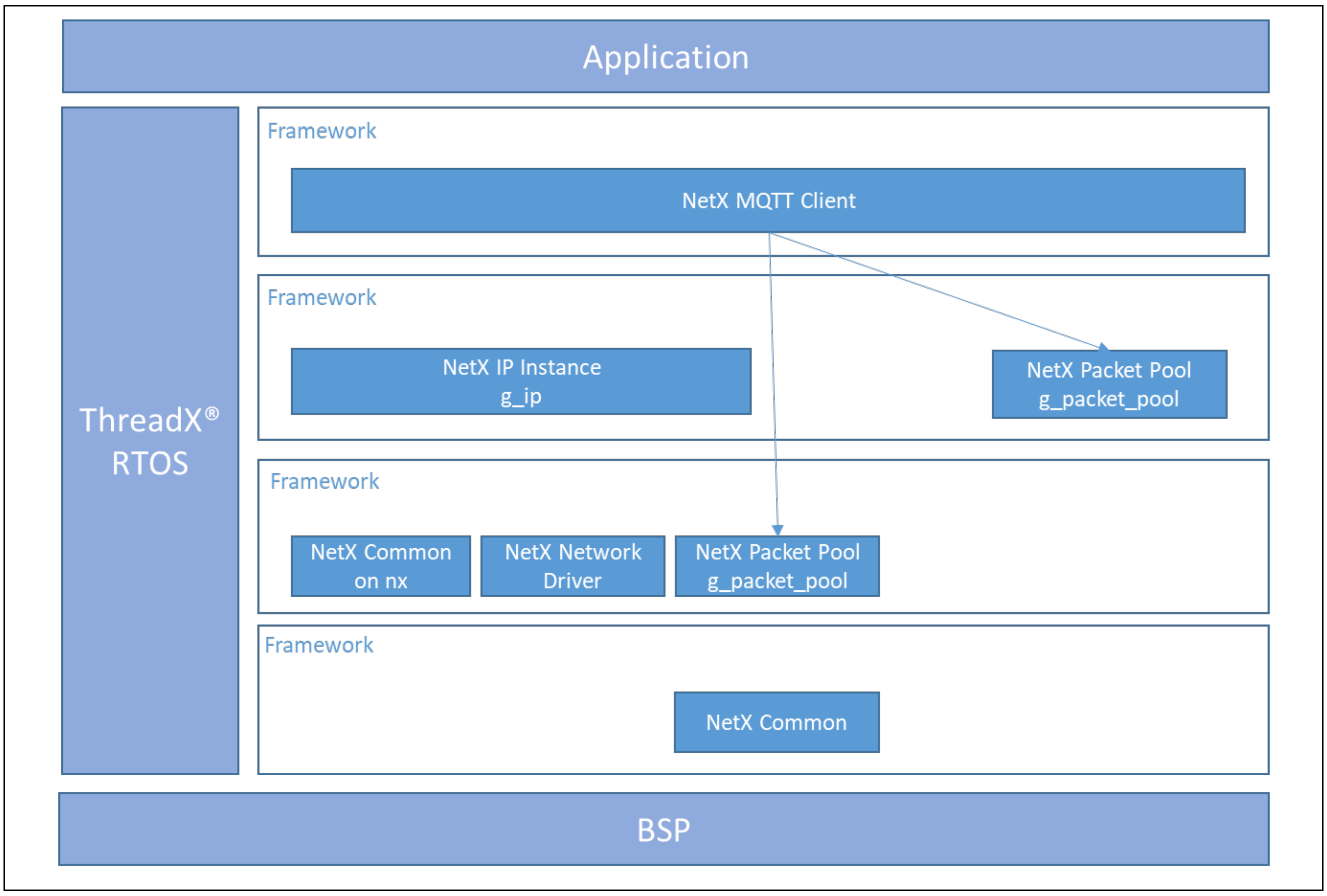


Figure 1 NetX Duo MQTT Client Module Block Diagram

2. NetX Duo MQTT Client Module APIs Overview

The NetX Duo MQTT Client Support module defines APIs for creating the MQTT Client, connecting to broker, setting up TLS security, and receiving MQTT messages. A complete list of the available APIs, an example API call, and a short description of each can be found in the table below. A table of status return values follows.

Table 1 Status Return Values

Function Name	Example API Call and Description
`nxd_mqtt_client_create`	`nxd_mqtt_client_create(&mqtt_client_secure, "my_client", CLIENT_ID_STRING, strlen(CLIENT_ID_STRING), ip_ptr, pool_ptr, (VOID*)mqtt_client_stack, sizeof(mqtt_client_stack), mqtt_thread_priority, (UCHAR*)client_memory, sizeof(client_memory));` Create an MQTT client with the specified Client ID, stack memory and stack size, and message block memory.

`nxd_mqtt_client_connect`	**`nxd_mqtt_client_connect`**`(&mqtt_client, &server_ip, NXD_MQTT_PORT, MQTT_KEEP_ALIVE_TIMER, 0, NX_WAIT_FOREVER)` Non secure connect to the MQTT broker specifying broker IP address and port, keep alive timer, and disabling the clean session option
`nxd_mqtt_client_secure_connect`	**`nxd_mqtt_client_secure_connect`**`(&mqtt_client_secure, &server_ip, NXD_MQTT_TLS_PORT, tls_setup_amazon, 600, 1, NX_WAIT_FOREVER)` Connect to broker with TLS security using the tls_setup_amazon, which is a user-defined function, to set up TLS and set TLS parameters. The clean session option is enabled. This is only available if the NetX Duo library is built with NX_SECURE_ENABLE set, and if the MQTT client property NX Secure is set.
`nxd_mqtt_client_login_set`	**`nxd_mqtt_client_login_set`**`(mqtt_client_ptr, “Username”, strlen(“Username”), ”Password”, strlen(“Password”);` Set the optional MQTT username and password. This must be called before the nxd_mqtt_client_connect or nxd_mqtt_client_secure_connect call if the broker requires username and password.
`nxd_mqtt_client_message_get`	**`nxd_mqtt_client_message_get`**`(&mqtt_client_secure, &topic, &topic_length, &message, &message_length,  &packet_ptr);` Retreive a published MQTT message for the specified topic.
`nxd_mqtt_client_receive_notify_set`	**`nxd_mqtt_client_receive_notify_set`**`(&mqtt_client_secure, my_notify_func);` Specify the function the MQTT Client thread task calls when an MQTT message is received.
`nxd_mqtt_client_subscribe`	**`nxd_mqtt_client_subscribe`**`(&mqtt_client_secure, TEST_SUBSCRIBE_TOPIC_NAME, strlen(TEST_SUBSCRIBE_TOPIC_NAME), 0);` Send a subscriber message to the broker for the specified topic for QoS (quality of service) level 0.
`nxd_mqtt_client_unsubscribe`	Send an unsubscriber message to the broker for the specified topic.
`nxd_mqtt_client_publish`	**`nxd_mqtt_client_publish`**`(&mqtt_client_secure, TEST_SUBSCRIBE_TOPIC_NAME, strlen(TEST_SUBSCRIBE_TOPIC_NAME), message_buffer, strlen(message_buffer), 0, 1, NX_WAIT_FOREVER);` Send a message to the broker for the specified topic previously subscribed to for QoS (quality of service) level 1, and the retain message option disabled.

`nxd_mqtt_client_disconnect`	**`nxd_mqtt_client_disconnect`**`(&mqtt_client);`
	
	`Disconnect from the MQTT broker.`

nxd_mqtt_client_disconnect_notify_set	**nxd_mqtt_client_disconnect_notify_set**(mqtt_client_ptr, my_disconnect_notify); Specify the user defined function for the MQTT Client thread task to call if the broker initiates disconnecting from the client.
nxd_mqtt_client_delete	**nxd_mqtt_client_delete**(mqtt_client_ptr); Delete the MQTT instance, clear transmit and message queue messages
nxd_mqtt_client_will_message_set	**nxd_mqtt_client_will_message_set**(mqtt_client_ptr, will_topic, will_topic_length, “will_message”, strlen(“will_message”), 0, 1); Set the optional MQTT Client will message without the retain will message option, for QOS 1. If a will message is needed, this must be called before connecting to the broker.

3. NetX Duo MQTT Client Module Operational Overview

MQTT (Message Queue Telemetry Transport) is a protocol based on the NetX Duo TCP/IP stack. It is based on a publisher-subscriber model. A client can publish information to other clients through an MQTT Server (broker). A client, if interested in a topic, can subscribe to the topic through the broker. A broker is responsible for delivering published messages to its clients who subscribe to the topic. In this publisher-subscriber model, multiple clients may publish data with the same topic. A client will receive a message that was published if the client subscribes to the same topic.

The following figure provides an overview of the MQTT Client publisher-subscriber model:

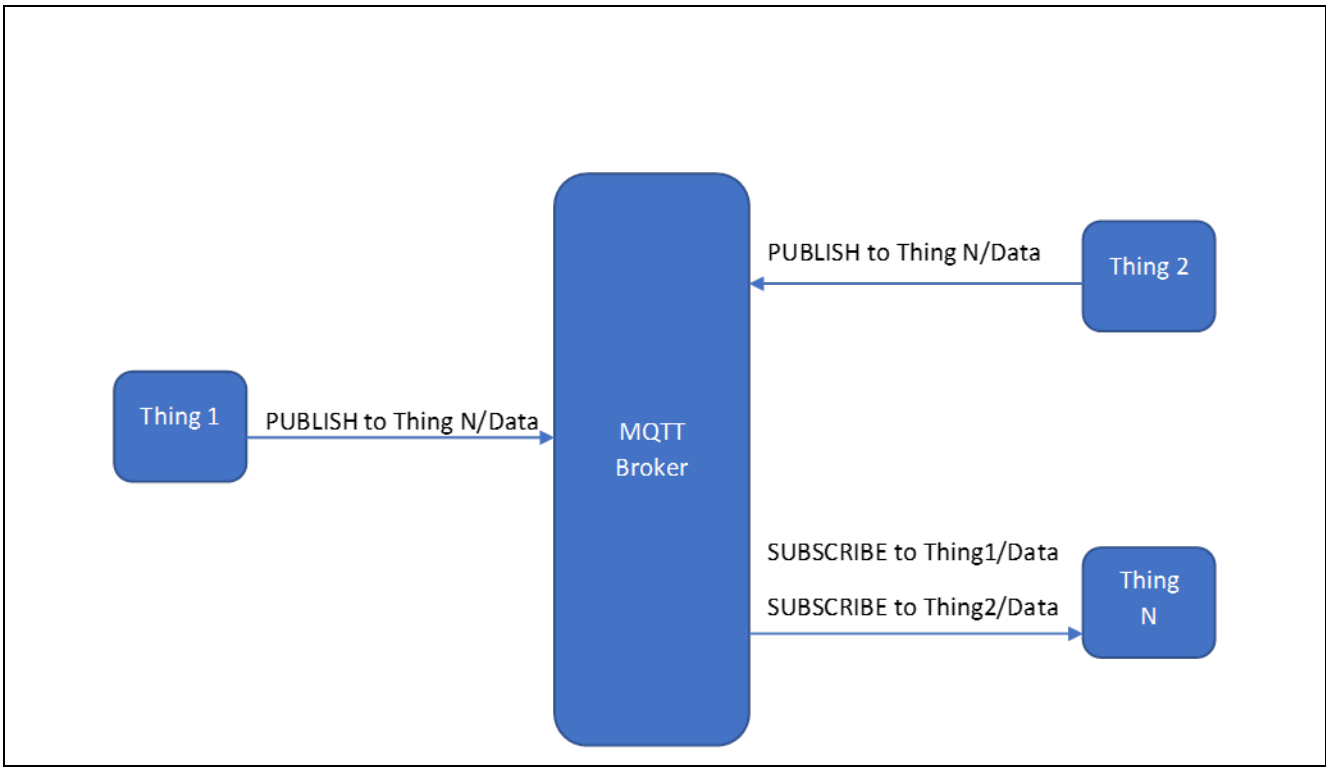


Figure 2 MQTT Client Publisher-Subscriber Model

The NetX Duo MQTT client module can be used in the normal mode or the secure mode.

- **NetX Duo MQTT client module Normal Mode Operational Description**
  In Normal mode, the communication between the MQTT client and broker is not secure.
- **NetX Duo MQTT client module Secure Mode Operational Description**
  In Secure mode, the communication between the MQTT client and broker is secured using the TLS protocol. In the thread pane, the TLS protocol is represented by "Add NetX Duo TLS common [Optional]" block.
  Depending on the use case, a client may choose one of the 3 QoS levels when publishing a message:
  — QoS 0: The message is delivered at most once. Messages sent with QoS 0 may be lost.
  — QoS 1: The message is delivered at least once. Messages sent with QoS 1 may be delivered more than once.
  — QoS 2: The message is delivered exactly once. Messages sent with QoS 2 is guaranteed to be delivered, with no duplication.
  Note: This implementation of MQTT client does not support QoS level 2 messages.

Since QoS 1 and QoS 2 are guaranteed to be delivered, the broker keeps track the state of QoS 1 and QoS 2 messages sent to each client. This is particularly important for clients that expect QoS1 or QoS 2 messages. The client may be disconnected from the broker (for example when the client reboots, or the communication link is temporarily lost). The broker must store QoS 1 and QoS 2 messages so the messages can be delivered later once the client is reconnected to the broker.

However, the client may choose not to receive any stale messages from the broker after reconnection. The client can do so by initiating the connection with the `clean_session` flag set to `NX_TRUE` (1) in the `nxd_mqtt_client_connect` API. In this case, upon receiving the MQTT CONNECT message, the broker shall discard any session information associated with this client, including undelivered or unconfirmed QoS 1 or QoS 2 messages.

If the `clean_session` flag is to `NX_FALSE`, the server shall resend the QoS 1 and QoS 2 messages. The MQTT Client also resends any un-acknowledged messages if `clean_session` is set to `NX_TRUE`. This acknowledgment is different from the TCP socket layer ACK, although that happens as well. The MQTT client sends an MQTT acknowledgment message to the broker upon receipt of a message, and gets one back when it publishes a message.

Incoming MQTT messages are stored in the receive queue of the MQTT client instance. The application retrieves these messages by calling the `nxd_mqtt_client_message_get` API, which returns both the topic and the topic message. The application must ensure to provide a large enough buffer for each. The oldest message in the queue is returned to the caller first. `nxd_mqtt_client_message_get` is non-blocking. If the MQTT client receive queue is empty, it returns immediately with an `NXD_MQTT_NO_MESSAGE` (0x1000A) status. This should not be handled as an error, but that the receive queue is empty.

To avoid polling the receive queue for incoming messages, the application can register a receive message callback function with the MQTT client by calling the `nxd_mqtt_client_recieve_notify_set` API. The callback function is defined as:

```
VOID (*receive_notify_callback)(NXD_MQTT_CLIENT *client_ptr, UINT
message_count);
```

As the MQTT client receives messages from the broker, it invokes the callback function if the function is set. The callback function passes the pointer to the client control block and a message count value. The message count value indicates the number of MQTT messages in the receive queue. Note that this callback function executes in the MQTT client thread context. Therefore, the callback function should not execute any procedures that may block the MQTT client thread. The callback function should trigger the application thread to call the `nxd_mqtt_client_message_get` API to retrieve the messages. This is demonstrated in the module guide project.

To disconnect the MQTT client service, the application shall use the service `nxd_mqtt_client_disconnect` and `nxd_mqtt_client_delete` APIs respectively. Calling `nxd_mqtt_client_disconnect` disconnects the TCP connection to the broker. It releases messages already received and stored in the receive queue. However, it does not release QoS level 1 messages in the transmit queue. QoS level 1 messages are retransmitted upon connection, assuming the `clean_session` flag is set to `NX_FALSE`.

The broker may initiate the disconnect from the client. The application can be notified of the disconnect request by registering a disconnect notify function with the MQTT Client. This is done by calling the `nxd_mqtt_client_disconnect_notify_set` API.

To delete an MQTT client, call the `nxd_mqtt_client_delete` API. This releases all message blocks in the transmit queue and the receive queue. Unacknowledged QoS level 1 messages are also deleted.

Using Secure communication

To secure the communication between MQTT client and broker, TLS protocol is required. In the thread pane, TLS protocol is represented by an "Add NetX Duo TLS common [Optional]" block. Adding NetX Duo TLS Common block enables the TLS support.

MQTT with TLS/NetX Duo Secure

When you use TLS with MQTT Client, it is strongly recommended that the TLS setup callback in the `nxd_mqtt_client_secure_connect` call contain all of the TLS set up, including creating the TLS instance, defining the local certificates, allocating memory for remote certificate processing, and optional callbacks such as timestamp and certificate authentication.

Regardless of whether the MQTT Client was able to connect via TCP successfully, or whether the TLS session was successfully started, the application MUST call `nxd_mqtt_client_disconnect` to properly clear and reset the TLS session before attempting to reconnect again.

If the session was terminated improperly, `nxd_mqtt_client_disconnect` must still be called for the same reason.

For SSP 1.3.x, the TLS setup callback will need to call a `memset` on the `NXD_SECURE_TLS_SESSION` data block before (re)creating a TLS session (`nxd_secure_tls_session_create`).

Below is the definition of the nxd_mqtt_client_secure_connect API with the TLS setup input:

```
      UINT nxd_mqtt_client_secure_connect(NXD_MQTT_CLIENT *client_ptr,
                                          NXD_ADDRESS *server_ip,
                                          UINT server_port,
                                          UINT (*tls_setup)(
                                               NXD_MQTT_CLIENT *client_ptr,
                                               NX_SECURE_TLS_SESSION *session_ptr,
                                               NX_SECURE_X509_CERT *,
                                               NX_SECURE_X509_CERT *),
                                           UINT keepalive,
                                           UINT clean_session,
                                           ULONG wait_option)
```

Add this logic to the `tls_setup` callback function (assuming the MQTT Client instance name is g_mqtt_client0):

```
      session_ptr = &(g_mqtt_client0.nxd_mqtt_tls_session);
      memset(session_ptr, 0, sizeof(NX_SECURE_TLS_SESSION));
      status = nxd_secure_tls_session_create(….)
```

If `memset` is not called, the TLS `nxd_secure_tls_session_create` call may not succeed. In SSP 1.4.0, it is no longer be necessary to call `memset`, but we still strongly recommend putting all TLS setup, including TLS creation, in the callback. It may seem wasteful to completely delete and recreate a TLS session. But the manner in which TLS is integrated into MQTT Client makes this the most sensible and reliable method to guarantee successful reconnection attempts.

For more details about TLS protocol, please see the NetX Duo TLS Secure Module Guide.

Multiple Instances of MQTT Client per Device

For SSP 1.4.0 and earlier, a device cannot safely run multiple instances of the MQTT Client. This is because the MQTT Client in these releases assumes global variables. That should be remedied in a subsequent release.

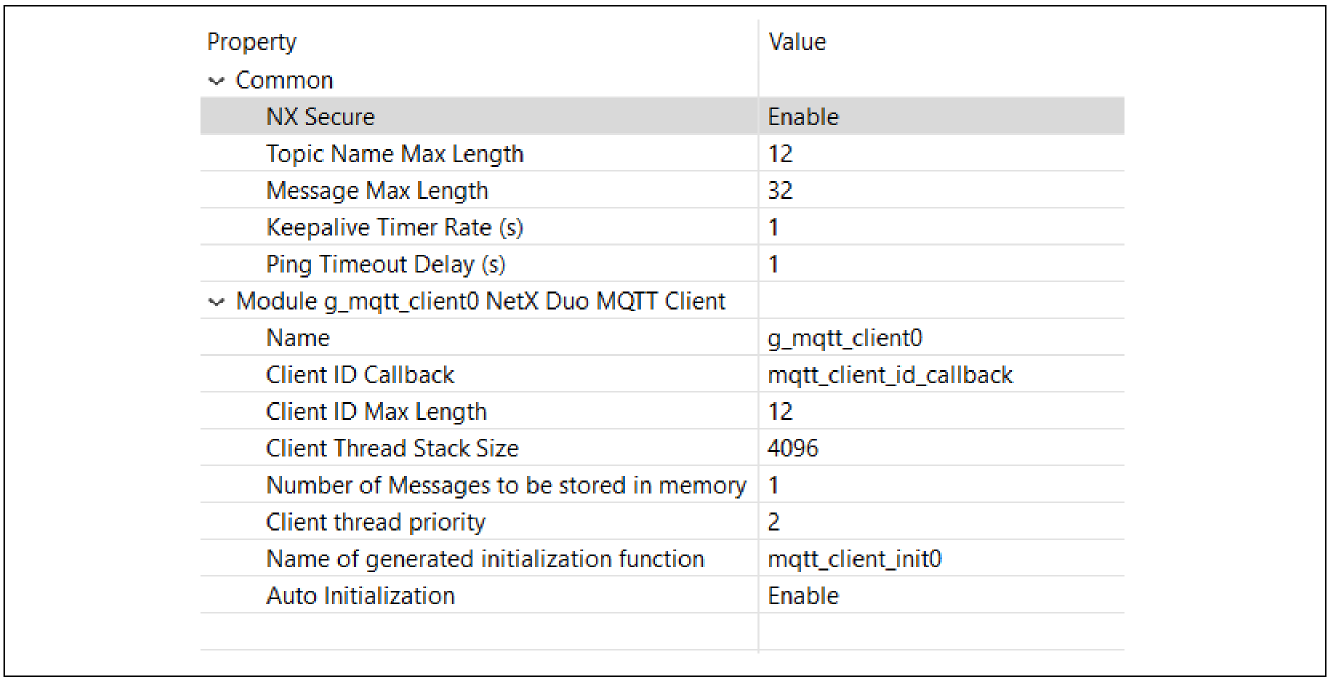
3.1 NetX Duo MQTT Client Module Important Operational Notes and Limitations

3.1.1 NetX Duo MQTT Client Module Operational Notes

The NetX Duo MQTT Client component is added by clicking on the (+) sign in the thread pane window -> **X-Ware -> NetX Duo -> Protocols -> NetX Duo MQTT Client**.

Adding the NetX Duo MQTT Client component to a project automatically adds the option to add the NetX Duo TLS component required for secure MQTT.

The MQTT Client properties are listed in the table below:



Property	Value
Common	
NX Secure	Enable
Topic Name Max Length	12
Message Max Length	32
Keepalive Timer Rate (s)	1
Ping Timeout Delay (s)	1
Module g_mqtt_client0 NetX Duo MQTT Client	
Name	g_mqtt_client0
Client ID Callback	mqtt_client_id_callback
Client ID Max Length	12
Client Thread Stack Size	4096
Number of Messages to be stored in memory	1
Client thread priority	2
Name of generated initialization function	mqtt_client_init0
Auto Initialization	Enable

Figure 3 NetX Duo MQTT Client block configurable properties

In the above figure, “Common” properties are those configurable options in the NetX Duo MQTT Client that are common to all instances of the MQTT client in the project. The “Module” properties are specific to each instance of MQTT Client in the project.

Common Properties

- NX Secure: Enables/disables TLS support. If this property is set to Enabled, the MQTT Client is built with TLS support. Note that enabling the property requires adding the NetX Duo TLS component to the project to supply the necessary source code to the project, or the project will not build. If set to Disabled, adding the NetX Duo TLS component has no effect though the project will still build and run.
- Topic Name Max Length: The maximum topic length (in bytes) the application is going to subscribe to. The default is 12 bytes.
- Message Max Length: The maximum message length (in bytes) the application is going to send or receive. The default is 32 bytes.
- Keepalive Timer Rate: This timer is used to keep track of the time since last MQTT control message was sent, and sends out an MQTT PINGREQ message before the keep-alive time expires. The default value is 1 second.
- Ping Timeout Delay: The time MQTT client waits for PINGRESP from the broker for after it sends out MQTT PINGREQ. The default value is 1 second.

Module Properties

- Name: Name of the MQTT client instance
- Name of generated initialization function: Name of initialization function which creates MQTT client instance. The default is the auto-generated function auto generated function `mqtt_client_init0`.
- Auto Initialization: Enable/disable call to initialization function. If disabled, the application thread entry function must obtain the Client ID and create the MQTT Client instance.
- Client ID Callback: Callback function provided by user for the MQTT Client thread task to obtain a unique client ID. If Auto Initialization is disabled, this and the Client ID length have no effect.
- Client ID Max Length: Maximum Length in bytes of the client ID.
- Client Thread Stack Size: MQTT Client thread stack size in bytes.
- Number of Messages to be stored in Memory: MQTT client uses memory area to store messages. The memory needed for MQTT client operation depends on the amount of data being sent or received. The minimal memory size is the size of a single `MQTT_MESSAGE_BLOCK` instance which is 60 bytes. The default value is 1 `MQTT_MESSAGE_BLOCK` or 60 bytes. However, this is not a good choice if there will be multiple messages

received before the application can receive them. Transmitted messages cannot be released until the TCP socket receives an ACK for the data, or if the QoS level is 1 or higher and the MQTT Client has received an ACK from the MQTT server. So, the module guide project uses 6 message blocks. That number can probably be reduced to 3 or 4.

- Client Thread Priority: MQTT Client thread priority.
- Name of Generated Initialization function: Name of the function that will call the `nxd_mqtt_client_create` API. If Auto Initialization is disabled, this has no effect. If it is, Synergy will create this function automatically.
- Auto Initialization: This determines if the function specified in the Name of Generated Initialization function option is called. If set to Enable, it will invoke this function. Otherwise if set to Disable, the application must call the `nxd_mqtt_client_create` API before using any NetX Duo MQTT Client services.

Setting a unique Client ID

As mentioned previously, an MQTT client instance is created using `nxd_mqtt_client_create()` API. If the MQTT Client application is letting the ISDE create the MQTT client, then it must define the Client ID Callback. This will be called before the ISDE calls `nxd_mqtt_client_create` internally with a defined Client ID string. The MQTT Client component allows you to set the Client ID Callback and Client ID Max Length in the list of MQTT Client properties (see above Module Properties).

The Client ID should be unique and is one of the parameters the MQTT broker uses to identify the client.

The prototype for Client ID callback is as below:

```
void mqtt_client_id_callback(char * p_client_id, uint32_t *
p_client_id_length);
```

`p_client_id` is a pointer to the Client ID, so it is an output parameter which will be filled in by this callback function. `p_client_id_length` is a pointer to the length of the Client ID, so it is an input and output parameter. This mechanism enables the Client ID to be determined at run time instead of at compile time.

If Client ID Callback is left empty in the properties pane of e^2 studio, a compiler error occurs. NULL is an acceptable entry. If your application prefers to create the MQTT Client directly, set this callback to NULL and set **Auto Initialization** to Disabled. Then when your application calls the `nxd_mqtt_client_create` API, provide the Client ID string directly, and the length of it as the input parameters:

```
/* Create MQTT client instance. */

nxd_mqtt_client_create(&mqtt_client, "my_client", CLIENT_ID_STRING,
             strlen(CLIENT_ID_STRING), ip_ptr, pool_ptr,
             (VOID*)mqtt_client_stack, sizeof(mqtt_client_stack),
             MQTT_THREAD_PRIORTY,
             (UCHAR*)client_memory, sizeof(client_memory));
```

Below is the sample reference implementation of Client ID callback function which copies MAC address to Client ID.

```
void mqtt_client_id_callback(char *p_client_id, uint32_t
*p_client_id_length)
{
    uint32_t id_length;
    UCHAR mac_id[6]  = {0x01, 0x02, 0x03, 0x04, 0x05, 0x06};

    if (*p_client_id_length < sizeof(mac_id))
    {
        id_length = *p_client_id_length;
    }
    else
    {
        id_length = sizeof(mac_id);
    }

     /* Copy MAC address to CLient ID and update client ID length */
     memcpy(p_client_id, mac_id, id_length);

    return;
}
```

Note: It is possible to have an MQTT session with a zero length Client ID string. If an MQTT Client supplies a zero-byte Client ID, the Client MUST also set the `clean_session` input in the `nxd_mqtt_client_connect` API to `NX_TRUE` (1) as per the MQTT protocol. If the Client supplies a zero-byte Client ID with `clean_session` set to `NX_FALSE` (0), the Server will respond to the CONNECT Packet with a `CONNACK` return code 0x02 (Identifier rejected) and then close the Network Connection.

3.1.2 NetX Duo MQTT Client Module Limitations

- NetX Duo MQTT Client does not support sending or receiving QoS level 2 messages.
- NetX Duo MQTT Client does not support chained packets.

4. Including the NetX Duo MQTT Client Module in an Application

This section describes how to include the NetX Duo MQTT Client module in an application using the SSP configurator.

Note: It is assumed that you are familiar with creating a project, adding threads, adding a stack to a thread and configuring a block within the stack. If you are unfamiliar with any of these items, refer to the first few chapters of the *SSP User's Manual* to learn how to manage each of these important steps in creating SSP-based applications.

To add the NetX Duo MQTT Client Module to an application, simply add it to a thread using the stacks selection sequence given in the following table. The default name for the NetX Duo MQTT Client Module is g_mqtt_client0. This name can be changed in the associated Properties window.

Table 2 NetX Duo MQTT Client Module Selection Sequence

Resource	ISDE Tab	Stacks Selection Sequence
g_mqtt_client0 NetX Duo MQTT Client	Threads	New Stack> X-Ware> NetX Duo> Protocols> NetX Duo MQTT Client

As shown in the following figure, when the NetX Duo MQTT Client module is added to the thread stack, the configurator automatically adds any needed lower-level modules. Any modules needing additional configuration information have the box text highlighted in Red. Modules with a Gray band are individual modules that stand alone. Modules with a Blue band are shared or common; they need only be added once and can be used by multiple stacks. Modules with a Pink band can require the selection of lower-level modules; these are either optional or recommended. (This is indicated in the block with the inclusion of this text.) If the addition of lower-level modules is required, the module description include **Add** in the text. Clicking on any Pink banded modules brings up the **New** icon and displays possible choices.

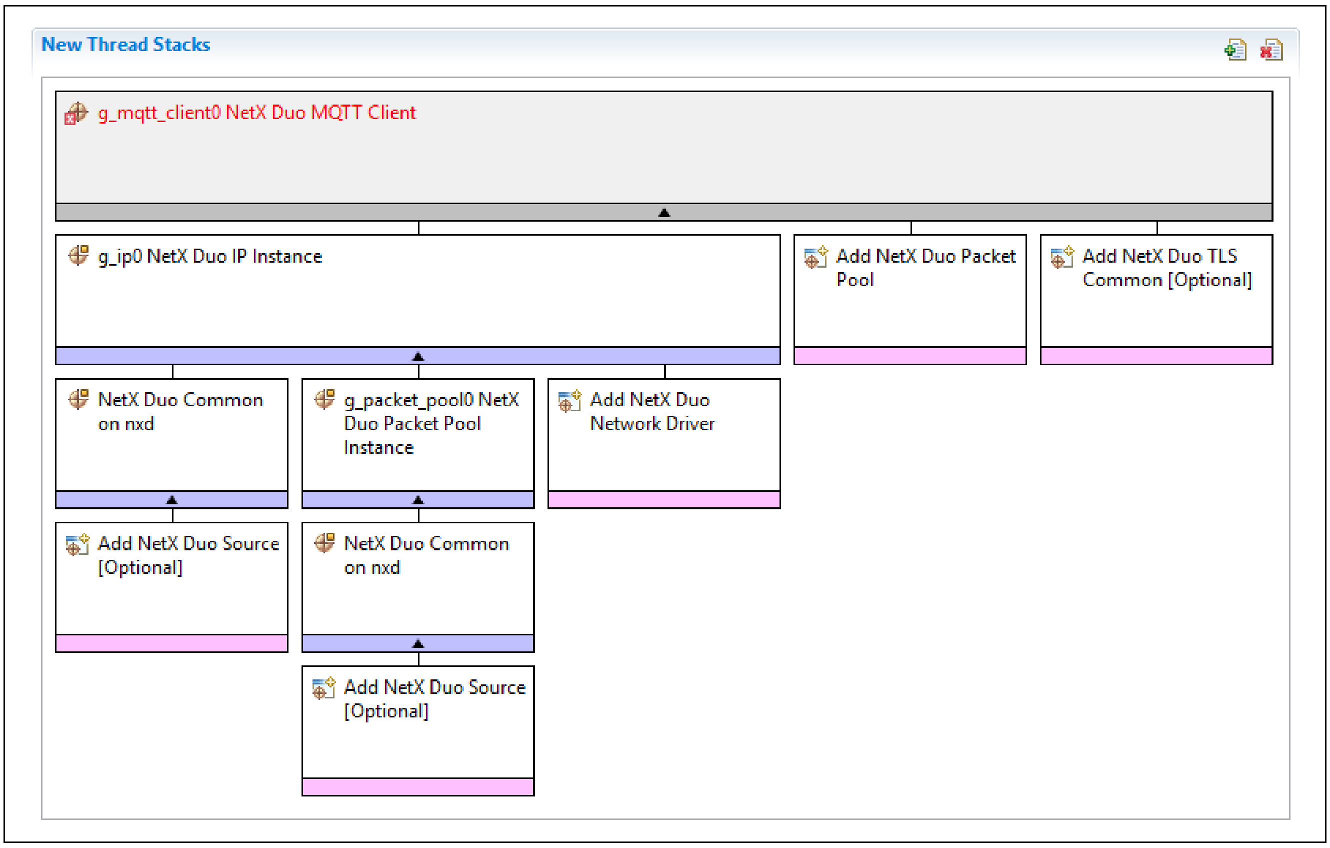

New Thread Stacks
g_mqtt_client0 NetX Duo MQTT Client
g_ip0 NetX Duo IP Instance
Add NetX Duo Packet Pool
Add NetX Duo TLS Common [Optional]
NetX Duo Common on nxd
g_packet_pool0 NetX Duo Packet Pool Instance
Add NetX Duo Network Driver
Add NetX Duo Source [Optional]
NetX Duo Common on nxd
Add NetX Duo Source [Optional]

Figure 2 NetX Duo MQTT Client Module Stack

The auto-generated code creates the IP instance, necessary to use NetX Duo library services, and the IP default packet pool. The MQTT Client can share that packet pool, or it can use its own packet pool by clicking on Add NetX Duo Packet Pool and choosing New. Set the packet payload size to the size of the messages the MQTT Client expects to send or receive. The optimal number of packets for the MQTT Client is based on expected/worst case volume of traffic for example. This is usually determined by observation. You will also need to add the NetX Duo Network Driver (NetX Ethernet, Cellular, or Wi-Fi). Use the tables provided in Section 5 for filling in these component properties.

After the auto-generated code completes, your application thread entry function runs. This is defined in the thread you created for this project in the left panel (not shown above). Now you are ready to start using the MQTT Client services. See section 6 for more details of the steps to take to connect, publish and disconnect from the MQTT Server (broker).

For details for adding a NetX Duo TLS component, please see *NetX Duo TLS Module Guide*.

5. Configuring the NetX Duo MQTT Client Module

The NetX Duo MQTT Client module must be configured by the user for the desired operation. The SSP configuration window automatically identifies (by highlighting the block in red) any required configuration selections, such as interrupts or operating modes, which must be configured for lower-level modules for successful operation. Only properties that can be changed without causing conflicts are available for modification. Other properties are 'locked' and not available for changes and are identified with a lock icon for the 'locked' property in the Properties window in the ISDE. This approach simplifies the configuration process and makes it much less error-prone than previous 'manual' approaches to configuration. The available configuration settings and defaults for all the user-accessible properties are given in the Properties tab within the SSP Configurator and are shown in the following tables for easy reference.

One of the properties most often identified as requiring a change is the interrupt priority. This configuration setting is available within the **Properties** window of the associated module. Simply select the indicated module and then view the **Properties** window. The interrupt settings are often toward the bottom of the properties list, so scroll down until they become available. Also note that the interrupt priorities listed in the **Properties** window in the ISDE indicates the validity of the setting based on the targeted MCU (CM4 or CM0+). This level of detail is not included in the following configuration properties tables but is easily visible with the ISDE when configuring interrupt-priority levels.

Note: You may want to open your ISDE, create the module, and explore the property settings in parallel with looking over the following configuration table values. This helps to orient you and can be a useful 'hands-on' approach to learning the ins and outs of developing with SSP.

Table 3 Configuration Default Settings for the NetX Duo MQTT Client Module

ISDE Property	Value	Description
NX Secure	Enable, Disable Default: Enable	NX secure selection
Topic Name Max Length	12	Topic name max length selection
Message Max Length	32	Message max length selection
Keepalive Timer Rate(s)	1	Keepalive timer rate(s) selection
Ping Timeout Delay(s)	1	Ping timeout delay(s) selection
Name	g_mqtt_client0	Module name
Name of generated initialization function	mqtt_client_init0	Name of generated initialization function selection
Auto Initialization	Enable, Disable Default: Enable	Auto initialization selection
Client ID Callback	mqtt_client_id_callback	Client ID callback selection
Client ID Max Length	12	Client ID max length selection
Client Thread Stack Size	4096	Client thread stack size selection
Number of Messages to be stored in memory	1	Number of messages to be stored in memory selection
Client thread priority	2	Client thread priority selection
Name of generated initialization function	Default: mqtt_client_init0	Function that calls the Client ID callback and create the MQTT Client instance.
Auto initialization	Enable, Disable Default: Enable	Automatically generates the MQTT Client. For example, enables the function specified in Name of generated initialization function to be called.

Note: The example settings and defaults are for a project using the Synergy S7G2 MCU Group. Other MCUs may require different configuration settings, particularly where memory is concerned.

In some cases, settings other than the defaults for stack modules can be desirable. For example, you may require a longer Client ID string or to disable NX Secure.

The configurable properties for the lower-level stack modules are given in the following sections for completeness and as a reference.

Note: Most of the property settings for lower-level modules are fairly intuitive and usually can be determined by inspection of the associated properties window from the SSP configurator.

5.1 Configuration Settings for the NetX MQTT Client Lower-Level Modules

Note: The example values and defaults are for a project using the Synergy S7G2 MCU Group. Other MCUs may have different default values and available configuration settings.

Bolded entries indicate properties that may need to be modified from the default value to match your system, changed to match your network device (for example, IP address). The italicized entries are required for the MQTT Client project to run.

Typically, only a small number of settings must be modified from the default for the IP layer and lower-level drivers as indicated via the red text in the thread stack block. Notice that some of the configuration properties must be set to a certain value for proper framework operation and are locked to prevent user modification. The following table identifies all the settings within the properties section for the module.

Table 4 Configuration Settings for the NetX Duo IP Instance

ISDE Property	Value	Description
Name	g_ip0	Module name
IPv4 Address (use commas for separation)	192,168,0,2	IPv4 Address selection
Subnet Mask (use commas for separation)	255,255,255,0	Subnet Mask selection
IPv6 Global Address (use commas for separation)*	0x2001, 0x0, 0x0, 0x0, 0x0, 0x0, 0x0, 0x1	IPv6 global address selection
IPv6 Link Local Address (use commas for separation, All zeros means use MAC address)*	0x0, 0x0, 0x0, 0x0, 0x0, 0x0, 0x0, 0x0	IPv6 link local address selection. This will be configured automatically if left at the default value, based on the device MAC address.
IP Helper Thread Stack Size (bytes)	1024	IP Helper Thread Stack Size (bytes) selection
IP Helper Thread Priority	3	IP Helper Thread Priority selection
ARP	*Enable*	*ARP selection*
ARP Cache Size in Bytes	512	ARP Cache Size in Bytes selection
Reverse ARP	Enable, Disable Default: Disable	Reverse ARP selection
TCP	*Enable*	*TCP selection*
UDP	Enable, Disable Default: Enable	UDP selection
ICMP	Enable, Disable Default: Enable	ICMP selection
IGMP	Enable, Disable Default: Enable	IGMP selection
IP fragmentation	Enable, Disable Default: Disable	IP fragmentation selection
Name of generated initialization function	ip_init0	Name of generated initialization function selection
Auto Initialization	Enable, Disable Default: Enable	Auto initialization selection

Note: *Only necessary if using IPv6 network connections

Table 5 Configuration Settings for the NetX Duo Common Instance

ISDE Property	Value	Description
Name of generated initialization function	nx_common_init0	Name of generated initialization function selection
Auto Initialization	Enable, Disable Default: Enable	Auto initialization selection

Table 6 Configuration Settings for the NetX and NetX Duo Packet Pool Instance g_packet_pool0

ISDE Property	Value	Description
Name	g_packet_pool0	Module name
Packet Size in Bytes	640	Packet size selection
Number of Packets in Pool	16	Number of packets in pool selection
Name of generated initialization function	packet_pool_init0	Name of generated initialization function selection
Auto Initialization	Enable, Disable Default: Enable	Auto initialization selection

Table 7 Configuration Settings for the NetX and NetX Duo Packet Pool Instance g_packet_pool1

ISDE Property	Value	Description
Name	g_packet_pool1	Module name
Packet Size in Bytes	640	Packet size selection
Number of Packets in Pool	16	Number of packets in pool selection
Name of generated initialization function	packet_pool_init0	Name of generated initialization function selection
Auto Initialization	Enable, Disable Default: Enable	Auto initialization selection

Table 8 Configuration Settings for the NetX Port ETHER

ISDE Property	Value	Description
Parameter Checking	BSP, Enabled, Disabled Default: BSP	Enable or disable the parameter checking
Channel 0 Phy Reset Pin	IOPORT_PORT_09_PIN_03	Channel 0 Phy reset pin selection
Channel 0 MAC Address High Bits	0x00002E09	Channel 0 MAC address high bits selection
Channel 0 MAC Address Low Bits	0x0A0076C7	Channel 0 MAC address low bits selection
*Channel 1 Phy Reset Pin***	*IOPORT_PORT_08_PIN_06*	*Channel 1 Phy reset pin selection*
Channel 1 MAC Address High Bits	0x00002E09	Channel 1 MAC address high bits selection
Channel 1 MAC Address Low Bits	0x0A0076C8	Channel 1 MAC address low bits selection
Number of Receive Buffer Descriptors	8	Number of receive buffer descriptors selection
Number of Transmit Buffer Descriptors	32	Number of transmit buffer descriptors selection
Ethernet Interrupt Priority	*Priority 0 (highest), Priority 1:2, Priority 3 (CM4: valid, CM0+: lowest- not valid if using ThreadX), Priority 4:14 (CM4: valid, CM0+: invalid), Priority 15 (CM4 lowest - not valid if using ThreadX, CM0+: invalid)* *Default: Disabled*	*Ethernet interrupt priority selection. Change from Disabled for the network driver to send and receive packets.*
Name	g_sf_el_nx	Module name
*Channel***	*1*	*Channel selection*
Callback	NULL	Callback selection

Note: **This is specific to the SK-S7G2 MCU. The DK-S7G2 default values need not be modified.
For Wi Fi networks, please refer to the Synergy Wi-Fi Application Project for SK-S7G2 - Application Project on the Renesas Gallery for more information on setting network parameters.

5.2 NetX Duo MQTT Client Module Clock Configuration

The ETHERC peripheral module uses PCLKA as its clock source. The PCLKA frequency is set using the SSP configurator clock tab prior to a build, or by using the CGC interface at run-time.

5.3 NetX Duo MQTT Client Module Pin Configuration

The ETHERC peripheral module uses pins on the MCU device to communicate to external devices. I/O pins must be selected and configured by the external device as required. The following table illustrates the method for selecting the pins within the SSP configuration window and the subsequent table illustrates an example selection for the I^{2}C pins.

Note: The selected operation mode determines the peripheral signals available and the MCU pins required.

Table 9 Pin Selection for the ETHERC Module

Resource	ISDE Tab	Pin selection Sequence
ETHERC	Pins	Select Peripherals > Connectivity:ETHERC > ETHERC1.RMII

Note: The selection sequence assumes ETHERC1 is the desired hardware target for the driver.

Table 10 Pin Configuration Settings for the ETHERC1

Property	Value	Description
Operation Mode	Disabled, Custom, RMII (Default: Disabled)	Select RMII as the Operation Mode for ETHERC1
Pin Group Selection	Mixed, _A only (Default: _A only)	Pin group selection
REF50CK	P701	REF50CK Pin
TXD0	P700	TXD0 Pin
TXD1	P406	TXD1 Pin
TXD_EN	P405	TXD_EN Pin
RXD0	P702	RXD0 Pin
RXD1	P703	RXD1 Pin
RX_ER	P704	RX_ER Pin
CRS_DV	P705	CRS_DV Pin
MDC	P403	MDC Pin
MDIO	P404	MDIO Pin

Note: Example settings are for a project using the Synergy S7G2 MCU and the SK-S7G2 Kit. Other Synergy MCUs and other Synergy Kits may have different available pin configuration settings.

6. Using the NetX Duo MQTT Client Module in an Application

Below are the basic steps towards getting an MQTT Client project connected to a broker.

Add a NetX Duo MQTT Client component as follows. After adding a thread in the Threads panel, select that thread and click on the (+) in the **New Thread Stacks** panel. Then choose **X-Ware** > **NetX Duo** > **Protocols** > **NetX Duo MQTT Client**. Please see following figure.

If there is no pre-existing IP instance then adding MQTT client block will add the IP instance below it. If there is a IP instance already added to the **Thread Stacks** pane choose the “**Use**” option and select the already created IP instance.

Never create multiple IP instances in NetX Duo! The IP thread task in NetX Duo uses global variables and having multiple IP instances can result in undefined behavior.

Once an IP instance is added, the user can either add a new packet pool or use an existing packet pool which is required for MQTT Client. To attach a packet pool to the MQTT Client, click on the **Add NetX Duo Packet Pool** block and choose **New** to create a new one (default name g_packet_pool1) or click on **Use** to share the packet pool used by the IP instance. The module guide project creates a new pool, g_packet_pool1.

There is no disadvantage in creating a second packet pool, provided there is enough system memory. Usually using multiple packet pools allows for optimal memory usage and better control of packet availability. The IP instance’s packet pool (by default g_packet_pool0) is used by the network driver to receive packets and the IP instance for internal operations. Creating a new packet pool for the application itself, particularly if it sends a large number of packets out at a time, might result in better performance, since it frees up g_packet_pool0 for internal operations.

After adding packet pool, the thread pane view looks as shown below.

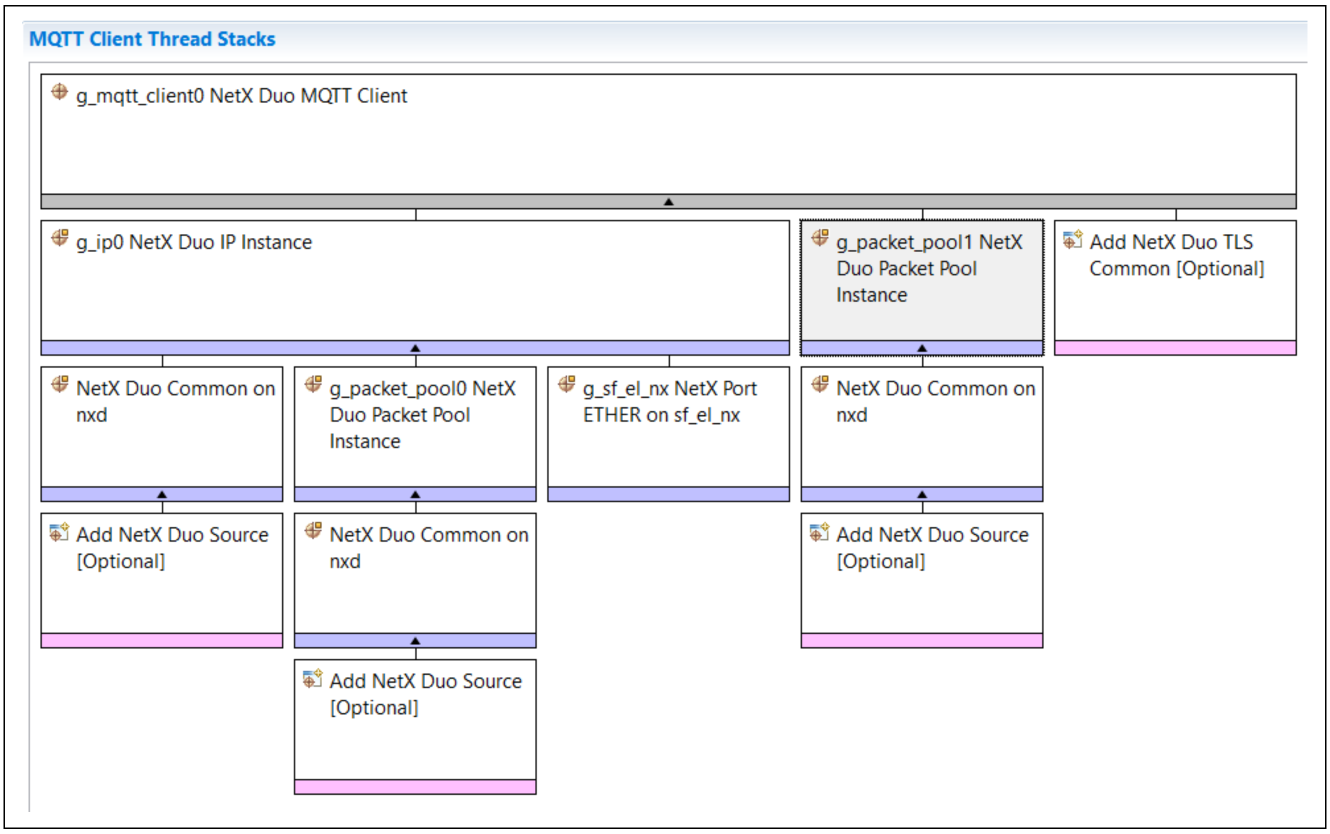

MQTT Client Thread Stacks
g_mqtt_client0 NetX Duo MQTT Client
g_ip0 NetX Duo IP Instance
g_packet_pool1 NetX Duo Packet Pool Instance
Add NetX Duo TLS Common [Optional]
NetX Duo Common on nxd
g_packet_pool0 NetX Duo Packet Pool Instance
g_sf_el_nx NetX Port ETHER on sf_el_nx
NetX Duo Common on nxd
Add NetX Duo Source [Optional]
NetX Duo Common on nxd
Add NetX Duo Source [Optional]
Add NetX Duo Source [Optional]

Figure 6 MQTT Client block with IP Instance and a second packet pool g_packet_pool1 used by the MQTT Client thread task

After adding the NetX Port Ether driver, the thread pane view looks as shown below

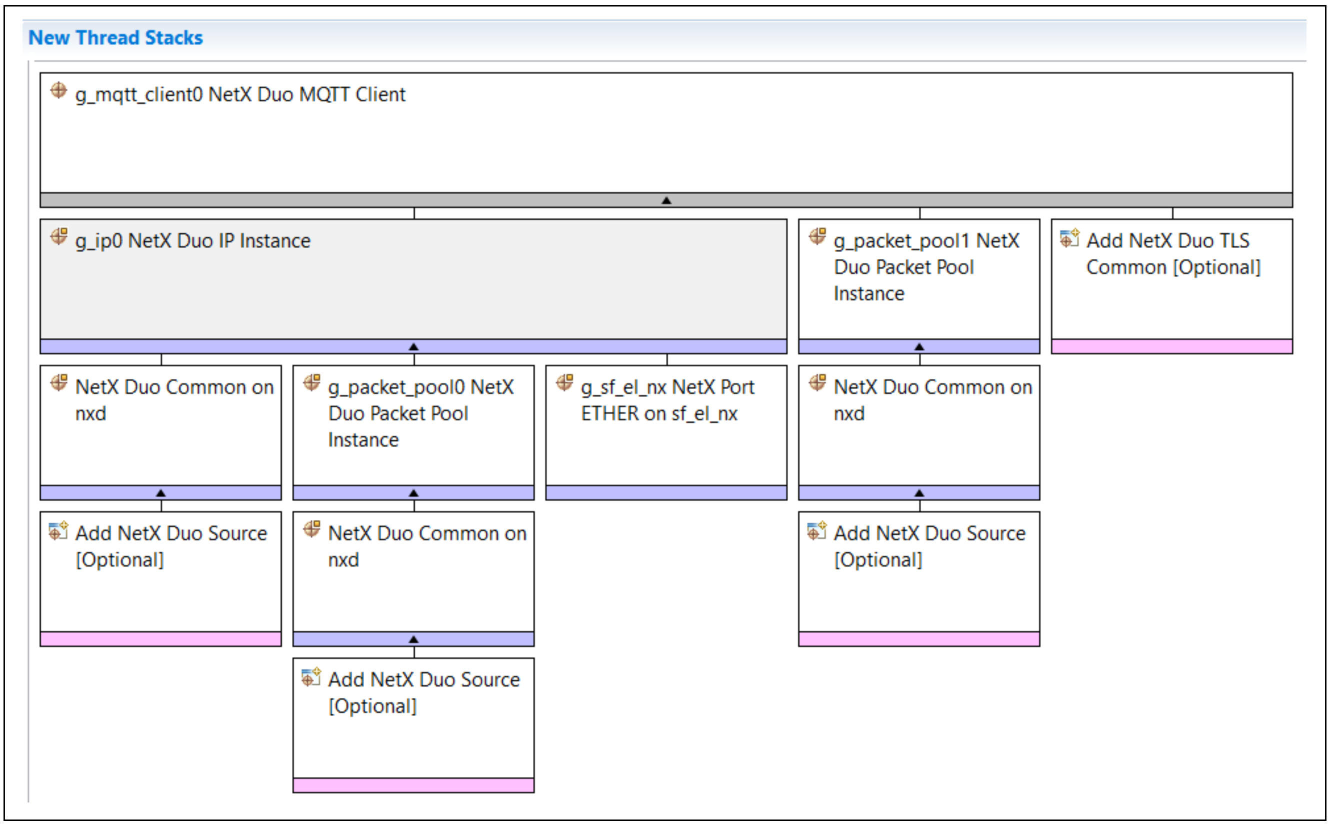

New Thread Stacks
g_mqtt_client0 NetX Duo MQTT Client
g_ip0 NetX Duo IP Instance
g_packet_pool1 NetX Duo Packet Pool Instance
Add NetX Duo TLS Common [Optional]
NetX Duo Common on nxd
g_packet_pool0 NetX Duo Packet Pool Instance
g_sf_el_nx NetX Port ETHER on sf_el_nx
NetX Duo Common on nxd
Add NetX Duo Source [Optional]
NetX Duo Common on nxd
Add NetX Duo Source [Optional]
Add NetX Duo Source [Optional]

Figure 6 MQTT Client block with the NetX Port Ether driver added

Configure the other properties of the MQTT Client. Please refer to the **Error! Reference source not found.** section for details about each property.

Generate the code by clicking on the Generate Project Content button. This creates the auto generated code under the configured thread. If the **Auto Initialization** property is set to Enable, this auto generated code includes the MQTT Client initialization function specified in the **Name of generated initialization function** property. This function internally calls the callback specified in the Client ID Callback property to obtain Client ID string and string length, and then calls the nxd_mqtt_client_create API to create an MQTT client. If **Auto Initialization** is set to Disable, the application must call nxd_mqtt_client_create directly.

If using Auto initialization, the application thread entry function needs to define the Client ID callback specified in the **Client ID Callback** property of the **MQTT Client** block. As mentioned previously, this callback is called by the MQTT Client initialization function prior to calling the `nxd_mqtt_client_create` API. This is because the `nxd_mqtt_client_create` function takes as its input the Client ID and length of the ID. The Client ID callback returns a unique client ID, typically the device MAC address, and length of the Client ID. Please refer to Setting a Unique Client ID in section 3.1.1.

Build the application and if there are no errors, run it. Now the thread entry function can execute. Below is an example MQTT Client session (without security).

MQTT Client application example

1. Wait for the network link to be enabled by calling the `nx_ip_status_check` (or if your system has multiple network interfaces, call `nx_ip_interface_status`_check) with the `NX_IP_LINK_ENABLED` option.
2. Create an event flag group using the `tx_event_flags_create` API.
3. Connect to the MQTT server (broker) using the `nxd_mqtt_client_connect` API.
4. Set a receive notification callback using the `nxd_mqtt_client_receive_notify_set` API. The receive callback sets a flag when notified by the underlying NetX Duo socket services that it has received a packet on this connection.
5. Subscribe to a topic on the MQTT Server using the `nxd_mqtt_client_subscribe` API.
6. Publish a message to the topic using the `nxd_mqtt_client_publish` API.
7. Wait to receive messages by calling the `tx_event_flags_get` API.
8. Receive the message using the `nxd_mqtt_client_message_get` API. Note that, unlike when receiving packets from a socket, the MQTT Client need not be concerned about releasing packets. The MQTT Client thread task handles packet and message block allocate and release.
9. Unsubscribe from the topic by calling the `nxd_mqtt_client_unsubscribe` API, specifying the topic.
10. Disconnect from the topic by calling the `nxd_mqtt_client_disconnect` API.

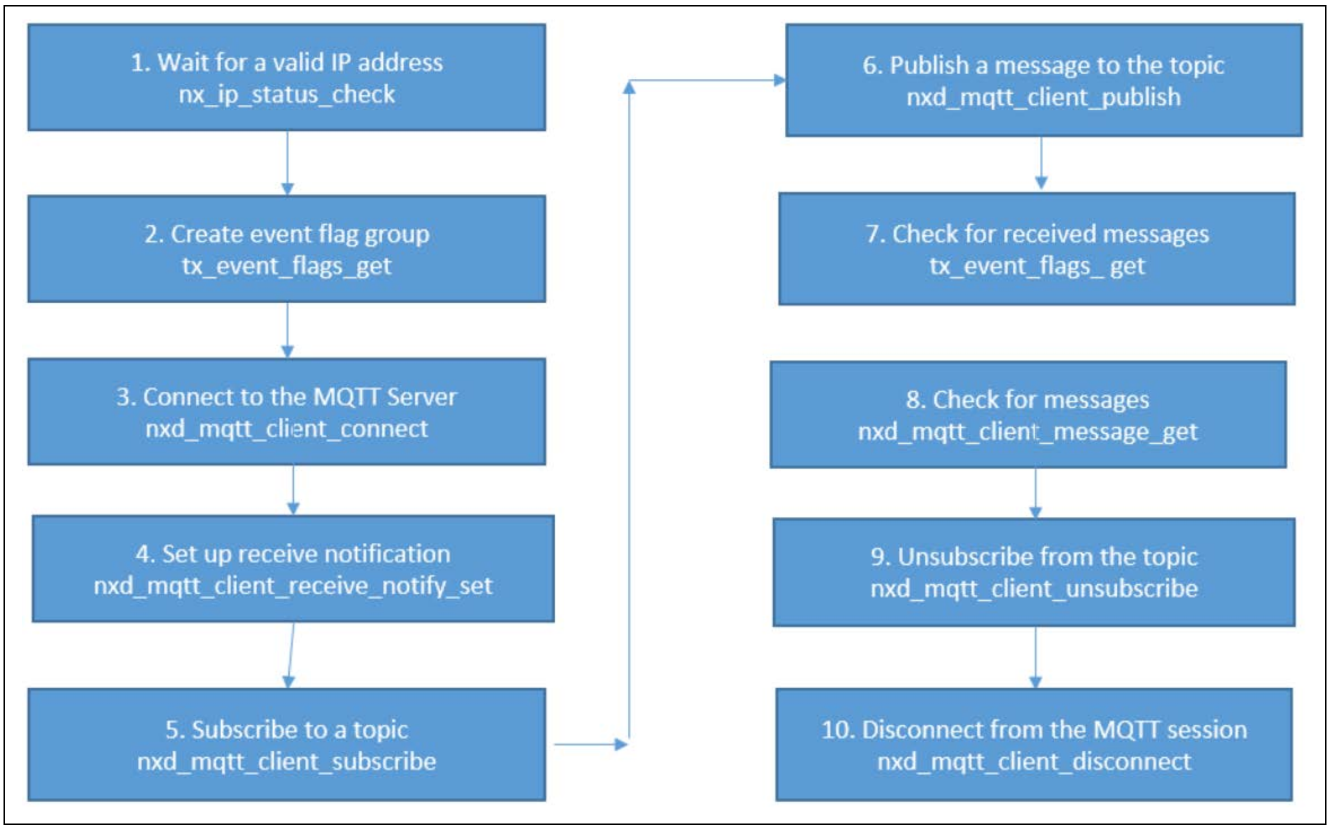

1. Wait for a valid IP address
nx_ip_status_check
2. Create event flag group
tx_event_flags_get
3. Connect to the MQTT Server
nxd_mqtt_client_connect
4. Set up receive notification
nxd_mqtt_client_receive_notify_set
5. Subscribe to a topic
nxd_mqtt_client_subscribe
6. Publish a message to the topic
nxd_mqtt_client_publish
7. Check for received messages
tx_event_flags_ get
8. Check for messages
nxd_mqtt_client_message_get
9. Unsubscribe from the topic
nxd_mqtt_client_unsubscribe
10. Disconnect from the MQTT session
nxd_mqtt_client_disconnect

Figure 3 Flow Diagram of a Typical NetX Duo MQTT Client Module Application

7. The NetX Duo MQTT Client Module Application Project

The application project associated with this module guide demonstrates the steps needed in a full design. The project can be found using the link provided in the References section at the end of this document. You may want to import and open the application project within the ISDE and view the configuration settings for the NetX and NetX Duo MQTT Client module. You can also read over the code in `mqtt_thread0_entry.c` which illustrates the NetX Duo MQTT Client Module APIs in a complete design.

The application project demonstrates the typical use of the NetX Duo MQTT Client Module APIs. The IP protocol is responsible for creating a valid IP address. Internally, NetX Duo handles all network transmission and reception of data for the MQTT Client. The text output of the project presents the current state of the application, and the current IP address. This is enabled by defining `SEMI_HOSTING` in the thread entry function, `mqtt_client_thread_entry.c`.

The following table identifies the target versions for the associated software and hardware used by the Application Project:

Table 11 Software and Hardware Resources Used by the Application Project

Resource	Revision	Description
e^2 studio	5.4.023	Integrated Solution Development Environment
SSP	1.3.0	Synergy Software Platform
IAR EW for Synergy	7.71.3	IAR Embedded Workbench® for Renesas Synergy™
SSC	5.4.023	Synergy Standalone Configurator
SK-S7G2	v3.0 to v3.1	Starter Kit

The `mqtt_client_thread_entry.c` file is the entry function for the main program-control section. You can open the project files within the ISDE and follow along with the description provided to help identify key uses of APIs.

`mqtt_client_thread_entry.c` lists the header files, `mqtt_client_thread.h` and `MQTT_Client_MG_AP.h`. It defines the client ID callback function `mqtt_client_id_callback`, for creating the MQTT Client ID when auto initialization is enabled. It also includes code for allowing semi-hosting to display results using `printf()`. To enable debug output in the Renesas Virtual Debug Console, the `SEMI_HOSTING` macro must be defined in the `MQTT_Client_MG_AP.h` file.

Note: This description assumes you are familiar with using `printf()` with the Debug Console in the Synergy Software Package. If you are unfamiliar with this, refer to the "How do I Use Printf() with the Debug Console in the Synergy Software Package" Knowledge Base article, available as described in the References section at the end of this document. Alternatively, you can see results via the watch variables in the debug mode.

Before calling the `run_mqtt_client_session` function, the `mqtt_client_thread_entry` function waits for the network link to be enabled. If `USE_IPV6` is defined, the MQTT Client will use IPv6. In this case, it calls the `enable_for_ipv6` function to enable the necessary services in NetX Duo for IPv6 services. NetX Duo by default supports IPv6 so no change to the NetX Duo library is required. Otherwise, the MQTT Client uses IPv4 by default and defines the IPv4 MQTT server address.

At this time, the MQTT server must be started and waiting for connection requests. The server IP address is defined in `mqtt_client_thread_entry.c` by the `LOCAL_SERVER_ADDRESS` symbol. If your MQTT server is at IP address 192,2,2,201 it should be defined as follows:

```
#define LOCAL_SERVER_ADDRESS  IP_ADDRESS(192,2,2,201
```

If you are running MQTT on an IPv6 network, the IPv6 address of the MQTT server is defined in a utility function in `mqtt_client_thread_entry.c` called `enable_for ipv6()`. (`LOCAL_SERVER_ADDRESS` is not used for defining the sever IPv6 address.) It might look like the following (depending on your network):

```
server_ip.nxd_ip_address.v6[3] = 0x201;
server_ip.nxd_ip_address.v6[2] = 0x0;
server_ip.nxd_ip_address.v6[1] = 0x0;
server_ip.nxd_ip_address.v6[0] = 0x20010db8;
server_ip.nxd_ip_version = NX_IP_VERSION_V6;
```

More details for how to set up and run an MQTT server are available in section 9.

The `mqtt_client_thread_entry` function can now call the `run_mqtt_client_session` function, defined in the `MQTT_Client_MG_AP.c` file. In this function call, the MQTT client specifies the MQTT server IP and listening port. It also specifies a keepalive timer on a 60 second t to be activated, a QoS level of 1 and the clean session feature disabled. It also specifies the topic it will subscribe to and a message it would like to publish to this topic.

`run_mqtt_client_session` conducts the actual MQTT session: it connects to the MQTT Server and subscribes to and publishes messages. That function returns the number of errors encountered in the MQTT session if any. The `mqtt_client_thread_entry` function prints out the success or number of errors in completing the MQTT session. The details are below.

`run_mqtt_client_session` begins by creating a ThreadX event group. After it connects with the MQTT server by calling the `nxd_mqtt_client_connect` API, it registers a callback function, `my_receive_notify_callback`, with the MQTT Client for the MQTT Client thread task to notify the application when a message is received. This callback function is called in the context of the MQTT Client thread so it should do minimal processing and not make blocking calls. In the project, it sets a flag to notify `run_mqtt_client_session` about the received message. This spares the `run_mqtt_client_session` from having to 'poll' the MQTT Client task for received messages.

Now the Client subscribes to the topic specified in the `run_mqtt_client_session` by calling the `nxd_mqtt_client_publish` API. It waits for the next message with the `tx_event_flags_get` API. It checks the events that are detected for including an `MQTT_MESSAAGE_EVENT`. If so, it retrieves the message from the MQTT Client receive queue using the `nxd_mqtt_client_message_get` API.

The Client can stay connected to the Server indefinitely, publishing and receiving messages, or until the server disconnects. In this case the Client only publishes a single message. Then it exits the session by unsubscribing (`nxd_mqtt_client_unsubscribe` API) and disconnecting (`nxd_mqtt_client_disconnect` API) from the MQTT server. All messages on the receive queue are released (returned to memory pool) and the TCP socket connection is closed. When it calls the `nxd_mqtt_client_delete` API, all messages on the transmit queue are released as well.

The table below indicates the settings of the properties configured in this application project to support the required operations and the physical properties of the target board and MCU. The **Bolded** entries are changed from the default. *Italicized* entries are required settings.

Table 12 Configuration Default Settings for the NetX Duo MQTT Client Module

ISDE Property	Value	Description
NX Secure	***Disable***	NX secure selection
Topic Name Max Length	12	Topic name max length selection
Message Max Length	32	Message max length selection
Keepalive Timer Rate(s)	1	Keepalive timer rate(s) selection
Ping Timeout Delay(s)	1	Ping timeout delay(s) selection
Name	g_mqtt_client0	Module name
Name of generated initialization function	mqtt_client_init0	Name of generated initialization function selection
Auto Initialization	Enable, Disable Default: Enable	Auto initialization selection
Client ID Callback	mqtt_client_id_callback	Client ID callback selection
Client ID Max Length	12	Client ID max length selection
Client Thread Stack Size	4096	Client thread stack size selection
Number of Messages to be stored in memory	**6**	Number of messages to be stored in memory selection
Client thread priority	2	Client thread priority selection
Name of generated initialization function	Default: mqtt_client_init0	Function that calls the Client ID callback and create the MQTT Client instance.
Auto initialization	Enable, Disable Default: Enable	Automatically generates the MQTT Client e.g. enables the function specified in Name of generated initialization function to be called.

Note: The example settings and defaults are for a project using the Synergy S7G2 MCU Group. Other MCUs may require different configuration settings, particularly where memory is concerned

You can also open the application project and view these settings in the **Properties** window as a hands-on exercise.

8. Customizing the NetX Duo MQTT Client Module for a Target Application

As mentioned previously, an MQTT session can be enabled for security. This is not discussed in detail in this document but there are module guides on TLS security available in Renesas Gallery.

The MQTT Client application can specify a disconnect callback if the MQTT Server initiates a disconnect with the client. This is done using the `nxd_mqtt_client_disconnect_notify_set`. Like the receive notify callback, this callback is called in the context of the MQTT Client thread and therefore must be kept to a minimum and no blocking calls made. Typically, it can set a flag for the application will receive when it checks its event queue in the project for this module guide.

The MQTT Client application can enable a keep alive feature in the nxd_mqtt_client_connect API. This creates a timer which on expiration signals the MQTT Client to send a keepalive message to the MQTT Server if no messages have been received during the Keepalive timeout.

- When this timer is created, it has a timer entry function that keeps track of whether the keepalive timeout has expired. If it has, it sends a simple message to the Server which should get a response. In this way, the Client (and the Server) both know the other end of the connection is still live.

Each MQTT session with QoS 1 (or QoS 2 which is not supported here) can able be enabled or disabled for the `clean_session` feature. This is specified in the `nxd_mqtt_client_connect` API.

- If set, this removes all messages on the MQTT Server's transmit queue if there are any. That way the Client does not have to receive "stale" messages. If it is not set, then the Server will retransmit them. Also, the Client will retransmit all unacknowledged messages on its own transmit queue. This is useful if the network connection abruptly broke off before the Client could receive or send all its messages.

The MQTT Server and Client may have a preset username password logon. If so, the MQTT Client application can call the `nxd_mqtt_client_set_login` before connecting with the Server to set the username and password. Note this is not part of the TLS Security processing. When the MQTT Client connects, the MQTT Client will send a message with the username and password to the Server and wait for the Server to acknowledge it. Note that the username and password are sent open text, not encrypted.

9. Running the NetX Duo MQTT Client Module Application Project

To run the NetX Duo MQTT Client Module application project and to see it executed on a target kit, you can simply import it into your ISDE, compile, and run debug.

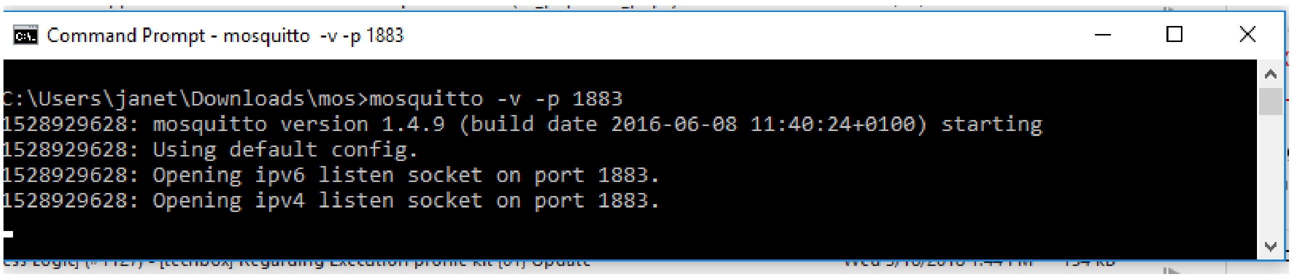
Note: The following steps are described in sufficient detail for someone experienced with the basic flow through the Synergy development process. If these steps are not familiar, refer to the first few chapters of the *SSP User's Manual* for a description of how to accomplish these steps.

1. Refer to the Synergy Project Import Guide (r11an0023eu0116-synergy-ssp-import-guide.pdf, included in this package) for instructions on importing the project into e^2 studio or the IAR EW for Synergy ISDE and building/running the application.
2. Connect to the host PC via a micro USB cable to J19 on SK-S7G2 Kit.
3. Connect to the host PC running a MQTT server via an Ethernet cable to J11 on SK-S7G2 Kit.
4. Compile the application.
5. Start the MQTT Server utility on the host PC. The MQTT server used for this project is available from Microsoft for Windows 10 https://mosquitto.org/download/. The defaults will work for the MQTT Client in this project.
   a. To run this utility in verbose output mode and for listening on a specific port, type this command [if you installed the utility to the `c:\mos` directory]:

      `C:\mos\mosquitto -v` [for verbose output] `-p`
      [specify a port number, no brackets, make sure your firewall permits TCP connections on this port]
   b. Type in this command in the command shell window after CDing into the directory where Mosquitto is installed:

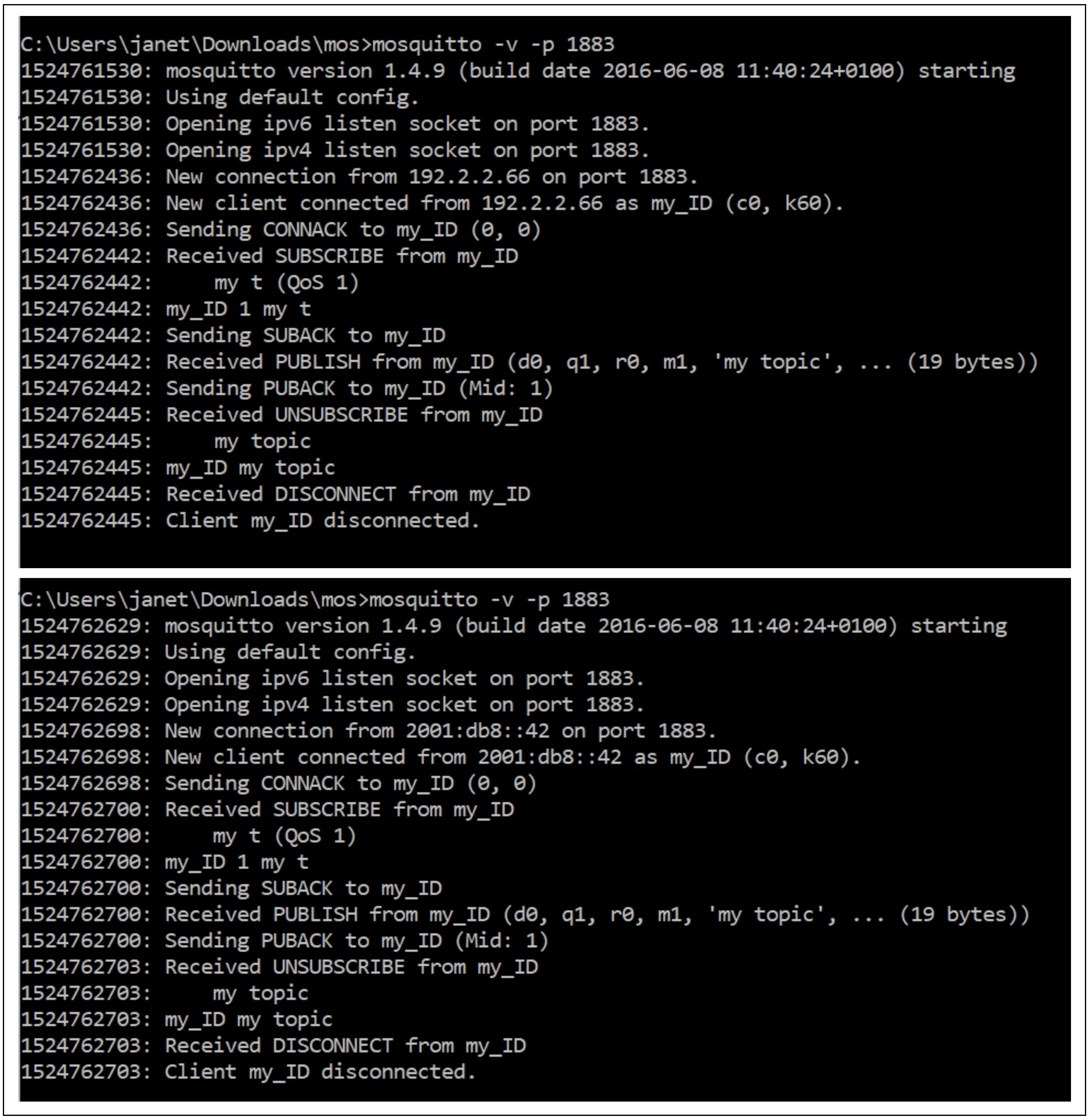
      ```
      mosquitto -v -p 1883
      ```

   The standardized listening port for MQTT servers is 1883. For MQTT Servers over TLS it is 8883. (TLS is not applied in this module guide project.) However, the MQTT server may listen on any port if there is a conflict or other reason for 1883 not being available.



```
C:\Users\janet\Downloads\mos>mosquitto -v -p 1883
1528929628: mosquitto version 1.4.9 (build date 2016-06-08 11:40:24+0100) starting
1528929628: Using default config.
1528929628: Opening ipv6 listen socket on port 1883.
1528929628: Opening ipv4 listen socket on port 1883.
```

   Below is the output from this utility when used with this module guide project:
6. Start the MQTT Client on your device. While in Debug mode in e^2 studio, click **Run** > **Resume** or click on the **Play** icon twice.
   a. Verify from the output in the command window that the MQTT Client has connected with the Server and the output is similar to the following:

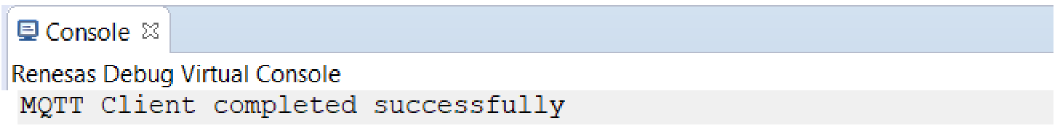


```
C:\Users\janet\Downloads\mos>mosquitto -v -p 1883
1524761530: mosquitto version 1.4.9 (build date 2016-06-08 11:40:24+0100) starting
1524761530: Using default config.
1524761530: Opening ipv6 listen socket on port 1883.
1524761530: Opening ipv4 listen socket on port 1883.
1524762436: New connection from 192.2.2.66 on port 1883.
1524762436: New client connected from 192.2.2.66 as my_ID (c0, k60).
1524762436: Sending CONNACK to my_ID (0, 0)
1524762442: Received SUBSCRIBE from my_ID
1524762442:     my t (QoS 1)
1524762442: my_ID 1 my t
1524762442: Sending SUBACK to my_ID
1524762442: Received PUBLISH from my_ID (d0, q1, r0, m1, 'my topic', ... (19 bytes))
1524762442: Sending PUBACK to my_ID (Mid: 1)
1524762445: Received UNSUBSCRIBE from my_ID
1524762445:     my topic
1524762445: my_ID my topic
1524762445: Received DISCONNECT from my_ID
1524762445: Client my_ID disconnected.
```

```
C:\Users\janet\Downloads\mos>mosquitto -v -p 1883
1524762629: mosquitto version 1.4.9 (build date 2016-06-08 11:40:24+0100) starting
1524762629: Using default config.
1524762629: Opening ipv6 listen socket on port 1883.
1524762629: Opening ipv4 listen socket on port 1883.
1524762698: New connection from 2001:db8::42 on port 1883.
1524762698: New client connected from 2001:db8::42 as my_ID (c0, k60).
1524762698: Sending CONNACK to my_ID (0, 0)
1524762700: Received SUBSCRIBE from my_ID
1524762700:     my t (QoS 1)
1524762700: my_ID 1 my t
1524762700: Sending SUBACK to my_ID
1524762700: Received PUBLISH from my_ID (d0, q1, r0, m1, 'my topic', ... (19 bytes))
1524762700: Sending PUBACK to my_ID (Mid: 1)
1524762703: Received UNSUBSCRIBE from my_ID
1524762703:     my topic
1524762703: my_ID my topic
1524762703: Received DISCONNECT from my_ID
1524762703: Client my_ID disconnected.
```

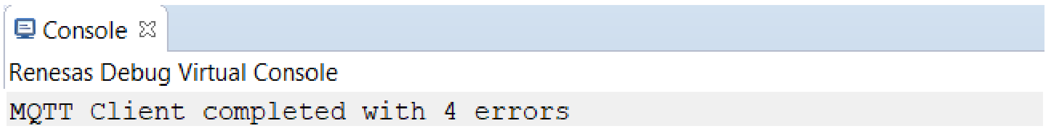
Figure 4 Example Output from the MQTT Server Mosquitto in an IPv4 (upper frame) and IPv6 (lower frame MQTT session with the NetX Duo MQTT Client

7. At the conclusion of the MQTT session, the `mqtt_client_thread_entry` function will print out the success (or number of errors) of the MQTT session. Go to the Renesas Virtual Debug Console to see the output in e^2 studio. In IAR, in the **View** menu, choose the **Terminal I/O** option. If the MQTT session is successful, the output will be:



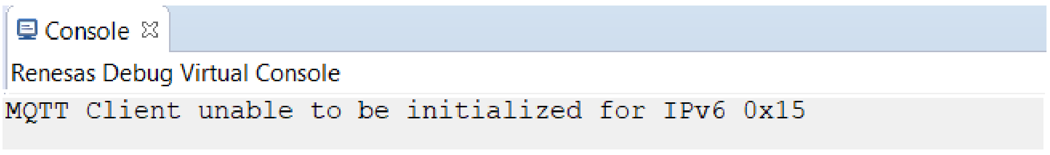
```
MQTT Client completed successfully
```

If the MQTT session encountered errors, the output will report the number of errors:



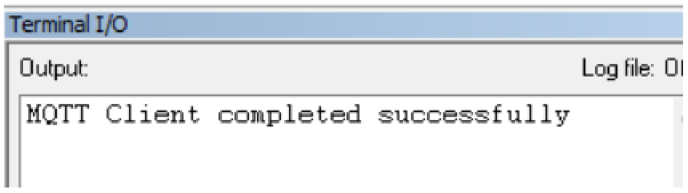
```
MQTT Client completed with 4 errors
```

If the `mqtt_client_thread_entry` function was not able to initialize the session for IPv6, the actual error status will be reported:



```
MQTT Client unable to be initialized for IPv6 0x15
```

Figure 4 Example Output in e2 studio from NetX Duo MQTT Client Module Application Project



```
MQTT Client completed successfully
```

Figure 5 Example Output in IAR from NetX Duo MQTT Client Module Application Project

10. NetX Duo MQTT Client Module Conclusion

This Module Guide has provided all the background information needed to select, add, configure, and use the module in an example project. Many of these steps were time consuming and error-prone activities in previous generations of embedded systems. The Renesas Synergy™ Platform makes these steps much less time consuming and removes the common errors like conflicting configuration settings or incorrect selection of lower-level drivers. The use of high-level APIs (as demonstrated in the application project) illustrates additional development-time savings by allowing work to begin at a high level and avoiding the time required in older development environments to use, or, in some cases, create, lower-level drivers.

11. NetX Duo MQTT Client Module Next Steps

After you have mastered a simple NetX Duo MQTT Client Module project, you may want to review a more complex example. You may wish to connect to multiple MQTT brokers or publish to multiple topics on the same MQTT server. The MQTT application can set up a will message before connecting with the MQTT Server in the event of an ungraceful disconnect with the server. Since MQTT is often used in scenarios where unreliable networks are not uncommon, 'will' messages let the MQTT client request the broker to notify the other clients about an unplanned disconnect. Use the 'will' message with the `nxd_mqtt_client_will_message_set` API to set up the will message and topic.

User Guides for these modules can be found using the instructions provided in the References section at the end of this document.

12. NetX Duo MQTT Client Module Reference Information

SSP User Manual: Available in html format in the SSP distribution package and as a pdf from the Renesas website.

Links to all the most up-to-date NetX Duo MQTT Client Module reference materials and resources are available on the Synergy Knowledge Base: https://en-us.knowledgebase.renesas.com/English_Content/Renesas_Synergy%E2%84%A2_Platform/Renesas_Synergy_Knowledge_Base/NetX_MQTT_Client_Module_Guide_Resources.

Website and Support

Visit the following vanity URLs to learn about key elements of the Synergy Platform, download components and related documentation, and get support.

Synergy Software	renesassynergy.com/software
Synergy Software Package	renesassynergy.com/ssp
Software add-ons	renesassynergy.com/addons
Software glossary	renesassynergy.com/softwareglossary
Development tools	renesassynergy.com/tools
Synergy Hardware	renesassynergy.com/hardware
Microcontrollers	renesassynergy.com/mcus
MCU glossary	renesassynergy.com/mcuglossary
Parametric search	renesassynergy.com/parametric
Kits	renesassynergy.com/kits
Synergy Solutions Gallery	renesassynergy.com/solutionsgallery
Partner projects	renesassynergy.com/partnerprojects
Application projects	renesassynergy.com/applicationprojects
Self-service support resources:	
Documentation	renesassynergy.com/docs
Knowledgebase	renesassynergy.com/knowledgebase
Forums	renesassynergy.com/forum
Training	renesassynergy.com/training
Videos	renesassynergy.com/videos
Chat and web ticket	renesassynergy.com/support

Revision History

		Description	
Rev.	Date	Page	Summary
1.00	Oct 22, 2018	—	First release document

RENESAS

SALES OFFICES Renesas Electronics Corporation http://www.renesas.com

Refer to "http://www.renesas.com/" for the latest and detailed information.

Renesas Electronics Corporation
TOYOSU FORESIA, 3-2-24 Toyosu, Koto-ku, Tokyo 135-0061, Japan

Renesas Electronics America Inc.
1001 Murphy Ranch Road, Milpitas, CA 95035, U.S.A.
Tel: +1-408-432-8888, Fax: +1-408-434-5351

Renesas Electronics Canada Limited
9251 Yonge Street, Suite 8309 Richmond Hill, Ontario Canada L4C 9T3
Tel: +1-905-237-2004

Renesas Electronics Europe Limited
Dukes Meadow, Millboard Road, Bourne End, Buckinghamshire, SL8 5FH, U.K
Tel: +44-1628-651-700

Renesas Electronics Europe GmbH
Arcadiastrasse 10, 40472 Düsseldorf, Germany
Tel: +49-211-6503-0, Fax: +49-211-6503-1327

Renesas Electronics (China) Co., Ltd.
Room 1709 Quantum Plaza, No.27 ZhichunLu, Haidian District, Beijing, 100191 P. R. China
Tel: +86-10-8235-1155, Fax: +86-10-8235-7679

Renesas Electronics (Shanghai) Co., Ltd.
Unit 301, Tower A, Central Towers, 555 Langao Road, Putuo District, Shanghai, 200333 P. R. China
Tel: +86-21-2226-0888, Fax: +86-21-2226-0999

Renesas Electronics Hong Kong Limited
Unit 1601-1611, 16/F., Tower 2, Grand Century Place, 193 Prince Edward Road West, Mongkok, Kowloon, Hong Kong
Tel: +852-2265-6688, Fax: +852 2886-9022

Renesas Electronics Taiwan Co., Ltd.
13F, No. 363, Fu Shing North Road, Taipei 10543, Taiwan
Tel: +886-2-8175-9600, Fax: +886 2-8175-9670

Renesas Electronics Singapore Pte. Ltd.
80 Bendemeer Road, Unit #06-02 Hyflux Innovation Centre, Singapore 339949
Tel: +65-6213-0200, Fax: +65-6213-0300

Renesas Electronics Malaysia Sdn.Bhd.
Unit 1207, Block B, Menara Amcorp, Amcorp Trade Centre, No. 18, Jln Persiaran Barat, 46050 Petaling Jaya, Selangor Darul Ehsan, Malaysia
Tel: +60-3-7955-9390, Fax: +60-3-7955-9510

Renesas Electronics India Pvt. Ltd.
No.777C, 100 Feet Road, HAL 2nd Stage, Indiranagar, Bangalore 560 038, India
Tel: +91-80-67208700, Fax: +91-80-67208777

Renesas Electronics Korea Co., Ltd.
17F, KAMCO Yangjae Tower, 262, Gangnam-daero, Gangnam-gu, Seoul, 06265 Korea
Tel: +82-2-558-3737, Fax: +82-2-558-5338

Colophon 7.2