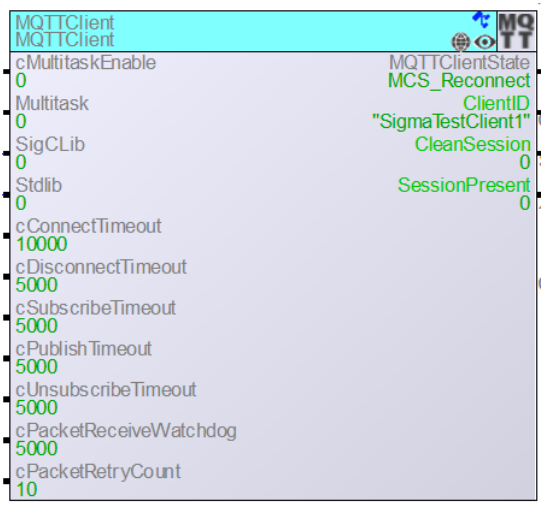
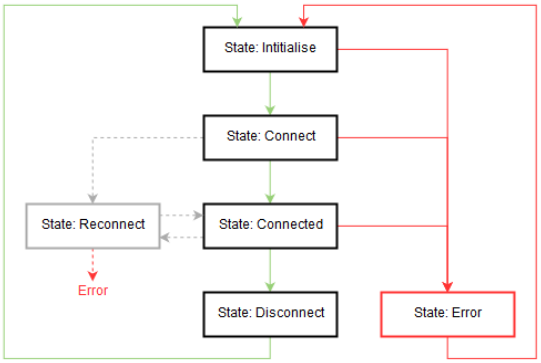
MQTTClient



The MQTTClient class functions as an MQTT Client and serves as the interface between the LASAL Class application and an MQTT Broker. The MQTTClient class implements MQTT Protocol v3.1.1.

Notes for the User

The MQTTClient State Flow is seen & described below:



State	Description
Initialise	The MQTTClient starts in the Initialise state and waits to be configured by the user.
	The user can configure the MQTTClient class by calling the global functions: (Described Below) mqtt_connect_callback_set() mqtt_connect_with_flags_callback_set() mqtt_disconnect_callback_set() mqtt_publish_callback_set() mqtt_message_callback_set() mqtt_subscribe_callback_set() mqtt_unsubscribe_callback_set() mqtt_log_callback_set() mqtt_username_pw_set() mqtt_will_set() mqtt_will_clear() mqtt_tls_set() mqtt_reconnect_delay_set() mqtt_max_inflight_messages_set() mqtt_reinitialise()
	To advance the state (see the state diagram above), the user needs to call the global functions: mqtt_loop_start() mqtt_connect_async()
Connect	In this state the MQTTClient class attempts to build a connection to the specified MQTT Broker.
	The connect state can advance into the Reconnect, Disconnect or Connected States. The input parameters that influence the Connect procedure are: cConnectTimeout (Client) CleanSession (Server)
Connected	The MQTTClient class resides in the Connected state for most of the operation cycle, during this state messages can be published, subscribed and received etc.
	The Connected State can advance to the Reconnect, Disconnect or Error States.
	During the Connected state the user can call connection management / communication methods such as: mqtt_reconnect_async() mqtt_disconnect() mqtt_publish() mqtt_subscribe()
Reconnect	If the MQTTClient loses connection / mqtt_reconnect_async() is called, the class switches to the Reconnect state, in this state the MQTTClient class attempts to rebuild the connection to the MQTT Broker.
	The parameters that influence the reconnect procedure are given by calling the mqtt_reconnect_delay_set() method.

	The Class can advance from the Reconnect state to the Connected, Disconnect or Error States.
Disconnect	In the Disconnect state the MQTTClient disconnects from the Broker and advances to the Intialise state.
	The input parameter that influence the disconnect procedure is: cDisconnectTimeout (Client)
	The Class can advance from the Disconnect state to the Initialise state.
Error	The MQTTClient enters the error state if an error occurs and is reset to become operational again.
	To MQTTClient Class automatically resets to the Initialise state to try rebuilding a connection.

Logging Texts

A definition of the Logging Texts that can be created by this class can be found in the Documentation for the Class MQTTClient_Interface.

Interfaces

Clients

cMultitaskEnable	If set to 1, the MQTTClient object uses a Lower Priority Multitasking thread to perform its cyclic work. Consider thread safety when using this feature. Data type: DINT
Multitask	Object channel to the _MultiTask OS Interface.
SigCLib	Object channel to the SigCLib OS Interface.
StdLib	Object channel to the _StdLib OS Interface.
cConnectTimeout	The Connect timeout duration can be initialized to this client and is specified in milliseconds. The Timeout is used when waiting for a TCP Connection and when waiting for the MQTT Broker CONNACK response. Data type: UDINT
cDisconnectTimeout	The Disconnect timeout duration can be initialized to this client and is specified in milliseconds. The Timeout is used when waiting for the DISCONNECT Control Packet sent confirmation, if the timeout occurs (sending the packet takes too long) the TCP Connection is closed, a Log message is made, and the Class is advanced as if the Packet sent successfully. Data type: UDINT
cSubscribeTimeout	The Subscribe timeout duration can be initialized to this client and is specified in milliseconds. The Timeout is used when a SUBACK packet is expected in response to a newly sent SUBSCRIBE Control Packet. If the timeout occurs, a Log message is made, and the Subscription is retried until the retry count specified in cPacketRetryCount is reached. After the retry count is reached another Log Message is made and the SUBSCRIBE Control Packet is dropped. Data type: UDINT
cPublishTimeout	The Publish timeout duration can be initialized to this client and is specified in milliseconds. The Timeout is used when a Publish related Control Packet (PUBACK, PUBREC, PUBREL, PUBCOMP) is expected in response to a newly sent Publish (PUBLISH, PUBREC, PUBREL) Control Packet. If the timeout occurs, a Log message is made, and the Packet sending is retried until the retry count specified in cPacketRetryCount is reached. After the retry count is reached another Log Message is made and the Publish related Control Packet is dropped. Data type: UDINT
cUnsubscribeTimeout	The Unsubscribe timeout duration can be initialized to this client and is specified in milliseconds. The Timeout is used when an UNSUBACK packet is expected in response to a newly sent UNSUBSCRIBE Control

	Packet. If the timeout occurs, a Log message is made, and the Packet sending is retried until the retry count specified in cPacketRetryCount is reached. After the retry count is reached another Log Message is made and the UNSUBSCRIBE Control Packet is dropped. Data type: UDINT
cPacketReceiveWatchdog	The Packet Receive Watchdog duration can be initialized to this client and is specified in milliseconds. The watchdog is active when a TCP Packet is being received, if the watchdog timeout occurs, a log is made, and the current packet being received is dropped. It is important to increase the watchdog timeout if the user wants to receive / transmit large messages or operates on a slower / unreliable TCP Connection. Data type: UDINT
cPacketRetryCount	The cPacketRetryCount client is used to specify the amount of times to retry transmitting an MQTT Control Packet. The Retry Count is referenced in the descriptions of the above clients.

Servers

MQTTClient-State

The MQTTClientState server is used as the Object channel to the MQTTClient class and show the current Class state as described in the Section: “Notes For the User”.

Server Value	State (Referenced Above)
MCS_Initialise	Initialise State.
MCS_Connect	It is waited until the client has established a connection to the server.
MCS_Connected	Class is ready for the operation defined in the server TopicType
MCS_Reconnect	A logon to the defined topic is executed.
MCS_Disconnect	The system waits for confirmation of the logon process.
MCS_Error	An error has occurred.

Unit	-	Data type	t_e_MQTTClientStates
Value range	-	Write Protected	TRUE
Default value	-	Retentive	FALSE

ClientID

The MQTTClient ClientID can be set on the ClientID Server, this server is an object channel to a String object contained inside the MQTTClient complex network.

The ClientID must be unique for each client registered on the broker and is set to the CPU Serial Number at initialisation. It is also possible to change the ClientID by calling the mqtt_reinitialise() global method.

Possibilities for configuration:

1. Change the string using this server after calling mqtt_reinitialise() and before calling mqtt_connect_async().
2. Pass a string to the input variable “id” in the method mqtt_reinitialise().

Unit	-	Data type	Object Channel: String
Value range	Max Len: 23 Characters	Write Protected	FALSE
Default value	CPU Serial Number (Set at Init)	Retentive	FALSE

<table>
<tr><td>CleanSession</td><td>The Session Type (Clean Session / Non Clean Session) is specified here, the operation of Session management according to the MQTT Specification is implemented.
(CleanSession = 0) – Session Data is saved against the ClientID at disconnect and it is not necessary to re-subscribe after reconnecting.
(CleanSession = 1) – Session Data is not saved at disconnect and a new session is created after each reconnect.
<table><tr><td>Unit</td><td>-</td><td>Data type</td><td>DINT</td></tr><tr><td>Value range</td><td>0 / 1</td><td>Write Protected</td><td>FALSE</td></tr><tr><td>Default value</td><td>0</td><td>Retentive</td><td>SRAM</td></tr></table></td></tr>
<tr><td>SessionPresent</td><td>Whether a Session is present is indicated on the SessionPresent server. The Session Present server is only updated in the Connected state and is determined by the type of session set on the CleanSession Server.
<table><tr><td>Unit</td><td>ms</td><td>Data type</td><td>UDINT</td></tr><tr><td>Value range</td><td>-</td><td>Write Protected</td><td>FALSE</td></tr><tr><td>Default value</td><td>10 s</td><td>Retentive</td><td>SRAM</td></tr></table></td></tr>
</table>

Global Methods

mqtt_reinitialise

Called to reinitialise the MQTT Client if it has already been initialised, this closes all connections and offers the option to end the current session, on re-initialisation the MQTT Client will load using all the new parameters set.

Variable:	Description:
► id – (^CHAR)	Client ID to use when buidling the Client, if NULL a self Generated ID will be assigned and clean_session must be set to true.
► clean_session – (BOOL)	If set to true the MQTT Client will clear all the Subscriptions after disconnecting from the Broker. If id is NULL then this must be true.
◄ retCode – (INT)	MQTT_ERR_SUCCESS - Initialized successfully. MQTT_ERR_NOMEM - Not enough memory to initialise. MQLTT_ERR_INVAL - Error with Configuration / Input Variables.

mqtt_connect_callback_set

Call this method to set the callback to be called when the connection has been established (Upon receipt of CONNACK).

Variable:	Description:
► pThis – (^VOID)	This pointer of the calling object, this pointer will be passed back to the callback.
► on_connect – (^VOID)	Pointer to the callback method to call.

Callback Prototype:

```
FUNCTION GLOBAL MyClass::OnConnected
  VAR_INPUT
    pThis    : ^void;
    retCode    : INT;
  END_VAR

END_FUNCTION
```

Callback Interface:

Variable:	Description:
► pThis – (^VOID)	This pointer of the calling object, this pointer will be passed back to the callback.
► retCode – (INT)	Return value indicating the status of the connect operation: 0 - Success.

<table>
<tr><td></td><td>
<table>
<tr><td></td><td>1 - Protocol Mismatch.
2 - Identifier Rejected.
3 - Broker Unavailable.
255 - Reserved.</td></tr>
</table>
</td></tr>
<tr><td>mqtt_connect_with_flags_callback_set</td><td>
Call this method to set the callback to be called when the connection has been established (Upon receipt of CONNACK).
<table>
<tr><td>Variable:</td><td>Description:</td></tr>
<tr><td>► pThis – (^VOID)</td><td>This pointer of the calling object, this pointer will be passed back to the callback.</td></tr>
<tr><td>► on_connect_with_flags – (^VOID)</td><td>Pointer to the callback method to call.</td></tr>
</table>
Callback Prototype:
<pre>
FUNCTION MQTTClient_Interface::Callback_OnConnectWithFlags
  VAR_INPUT
    pThis   : ^void;
    retCode : INT;
    flags   : BYTE;
  END_VAR
</pre>
Callback Interface:
<table>
<tr><td>Variable:</td><td>Description:</td></tr>
<tr><td>► pThis – (^VOID)</td><td>This pointer of the calling object, this pointer will be passed back to the callback.</td></tr>
<tr><td>► retCode – (INT)</td><td>Return value indicating the status of the connect operation:
0 - Success.
1 - Protocol Mismatch.
2 - Identifier Rejected.
3 - Broker Unavailable.
255 Reserved.</td></tr>
<tr><td>► flags – (BYTE)</td><td>Indicates the flags received in the CONNACK Control Packet, bits:
0: Session Present Flag.
1-6: Reserved.</td></tr>
</table>
</td></tr>
<tr><td>mqtt_disconnect_callback_set</td><td>
Call this method to set the callback to be called when the server has received the DISCONNECT command and has disconnected the client.
<table>
<tr><td>Variable:</td><td>Description:</td></tr>
<tr><td>► pThis – (^VOID)</td><td>This pointer of the calling object, this pointer will be passed back to the callback.</td></tr>
</table>
</td></tr>
</table>

	► on_disconnect – (^VOID) — Pointer to the callback method to call.

Callback Prototype:

```
FUNCTION GLOBAL MyClass::OnDisconnect
  VAR_INPUT
    pThis    : ^void;
    retCode  : INT;
  END_VAR

END_FUNCTION
```

Callback Interface:

Variable:	Description:
► pThis – (^VOID)	This pointer of the calling object, this pointer will be passed back to the callback.
► retCode – (INT)	Return value indicating the status of the disconnect operation: 0 – Disconnected upon user request. Not 0 – Unexpected Disconnect.

mqtt_publish_callback_set

Call this method to set the callback to be called when a message has been published.

Variable:	Description:
► pThis – (^VOID)	This pointer of the calling object, this pointer will be passed back to the callback.
► on_publish – (^VOID)	Pointer to the callback method to call.

Callback Prototype:

```
FUNCTION GLOBAL MyClass::OnPublish
  VAR_INPUT
    pThis    : ^void;
    mid      : INT;
  END_VAR

END_FUNCTION
```

Callback Interface:

Variable:	Description:
► pThis – (^VOID)	This pointer of the calling object, this pointer will be passed back to the callback.

	► mid – (INT) — Message ID of the subscribe message.
mqtt_message_callback_set	Call this method to set the callback to be called when a message has been received.

Variable:	Description:
► pThis – (^VOID)	This pointer of the calling object, this pointer will be passed back to the callback.
► on_message – (^VOID)	Pointer to the callback method to call.

Callback Prototype:

```
FUNCTION GLOBAL MyClass::OnMesssage
  VAR_INPUT
    pThis    : ^void;
    pMessage  : ^MQTTClient::t_s_MQTTMessage;
  END_VAR

END_FUNCTION
```

Callback Interface:

Variable:	Description:
► pThis – (^VOID)	This pointer of the calling object, this pointer will be passed back to the callback.

► pMessage – (^t_s_MQTTMessage)	Pointer to the message received, the pointer will be freed after the callback method completes, if the user requires the data lateron the user needs to copy the data during the callback.
	
	`t_s_MQTTMessage : STRUCT`
	`  mid : INT;`
	`  topic : ^CHAR;`
	`  payload : ^void;`
	`  payloadLen : UDINT;`
	`  qos : INT;`
	`  retainMsg : BOOL;`
	`END_STRUCT;`

mqtt_subscribe_callback_set	Call this method to set the callback to be called when the server responds to the subscribe request.

Variable:	Description:

► pThis – (^VOID)	This pointer of the calling object, this pointer will be passed back to the callback.
► on_subscribe – (^VOID)	Pointer to the callback method to call.

Callback Prototype:

```
FUNCTION GLOBAL MyClass::OnSubscribe
  VAR_INPUT
    pThis          : ^void;
    mid            : INT;
    qos_count      : INT;
    granted_qos    : ^INT;
  END_VAR

END_FUNCTION
```

Callback Interface:

Variable:	Description:
► pThis – (^VOID)	This pointer of the calling object, this pointer will be passed back to the callback.
► mid – (INT)	Message ID of the subscribe message.
► qos_count – (INT)	Number of subscriptions, this is the number of QoS Integers that will be found at granted_qos.
► granted_qos – (^INT)	Array of integers denoting the QoS granted to each subscription. 1. 0, 1, 2 - QoS granted. 2. 128 - Subscription denied / failed.

mqtt_unsubscribe_callback_set

Call this method to set the callback to be called when the broker responds to an unsubscribe request.

Variable:	Description:
► pThis – (^VOID)	This pointer of the calling object, this pointer will be passed back to the callback.
► on_unsubscribe – (^VOID)	Pointer to the callback method to call.

Callback Prototype:

```
FUNCTION GLOBAL MyClass::OnUnsubscribe
  VAR_INPUT
    pThis   : ^void;
    mid     : INT;
  END_VAR

END_FUNCTION
```

Callback Interface:

Variable:	Description:
► pThis – (^VOID)	This pointer of the calling object, this pointer will be passed back to the callback.
► mid – (INT)	The message id of the unsubscribe message.

mqtt_log_callback_set

Call this method to set the callback to be called when a log is made from the MQTTClient.

Variable:	Description:
► pThis – (^VOID)	This pointer of the calling object, this pointer will be passed back to the callback.
► on_log – (^VOID)	Pointer to the callback method to call.

Callback Prototype:

```
FUNCTION GLOBAL MyClass::OnLog
  VAR_INPUT
    pThis    : ^void;
    level    : INT;
    pStr     : ^CHAR;
  END_VAR

END_FUNCTION
```

Callback Interface:

Variable:	Description:
► pThis – (^VOID)	This pointer of the calling object, this pointer will be passed back to the callback.
► level – (INT)	The level of the log message: 1. MQTT_LOG_INFO 2. MQTT_LOG_NOTICE

	3. MQTT_LOG_WARNING 4. MQTT_LOG_ERR 5. MQTT_LOG_DEBUG
► pStr – (^CHAR)	Pointer to the log message string.

mqtt_username_pw_set

Call to configure the Username and Password to use to connect the MQTT Broker, if this is not called no Username or Password will be sent on mqtt_connect(). If the username input variable is NULL the password is not stored. Minimum MQTT Broker Specification is V3.1.1.

Variable:	Description:
► username (^CHAR)	Pointer to the Null terminated username to use.
► password (^CHAR)	Pointer to the Null terminated password to use.
► udPassLen (UDINT)	Length of the Password.
◄ retCode (INT)	0… Successfully set the username and password 3…invalid input parameters

mqtt_will_set

Must be called before mqtt_connect, this will set the will that is passed to the MQTT server upon connecting, if not called, no will is set on connection.

Variable:	Description:
► topic (^CHAR)	Pointer to the topic name on which to publish the will.
► payloadLen (UDINT)	Payload length in bytes. (0 -> 268 435 455 (256MB)).
► payload (^VOID)	Pointer to the payload data.
► qos (INT)	QoS to use for the will that is published, valid values are 0, 1 or 2.
► retain_will (BOOL)	If the flag is set to true, the server will store this publication and send it to any new clients that connect to the specified topic.
◄ retCode (INT)	0… Will accepted and will be sent on mqtt_connect() 3…invalid input parameters

mqtt_will_clear

Remove a previously set will (mqtt_will_set()).

Variable:	Description:
◄ retCode (INT)	0…successfully cleared 3…no will set

mqtt_tls_set

Call this method to setup the TLS connection variables, if this method has been called the connection will be established using TLS when mqtt_connect() is called.

Variable:	Description:
► cafile (^CHAR)	Null terminated string indicating the file path to the PEM encoded CA Certificate file containing trusted root servers. If the option is set to NIL then the Certificates installed in the library (Placed in C:\LSLSYS\SSL\) will be used.
► certfile (^CHAR)	Null terminated string indicating the file path to an optional PEM encoded file containing the client certificate / certificate chain. This file must be in the folder: C:\LSLSYS\SSL\ and if the filename is not Null then a valid filename must be passed to keyfile. The filename cannot exceed a length of 255 characters.
► keyfile (^CHAR)	Null terminated string indicating the path to the PEM encoded private key for the client. This file must be in the folder C:\LSLSYS\SSL\ and the filename length must not exceed 255 characters.
► pw (^CHAR)	Optional Null terminated string containing the password with which the keyfile is encrypted. The password cannot be longer than 1023 bytes.
◄ retCode (INT)	0…setting parameters successful 3…invalid input parameters

mqtt_reconnect_delay_set

Call this method to set the reconnect delay if a reconnect needs to be performed due to unexpected disconnect.

Variable:	Description:
► reconnect_delay (UINT)	Shortest interval to wait for reconnect (senconds).
► reconnect_delay_max (UINT)	Longest interval to wait for each reconnect (seconds).
► reconnect_exponential_delay (BOOL)	Set to TRUE to use an exponential delay: FALSE: min = 3 secs, max = 30 secs, Intervals = 3,6,9,12,15,18,21,24,27,30,30,30… TRUE: min = 3 secs, max = 30 secs, Intervals = 3,6,12,24,30,30,30…
◄ retCode (INT)	0…setting parameters successful 3…invalid input parameters

mqtt_max_in-flight_messages_set

Call this method to set the maximum number of inflight messages, an inflight message denotes a message that has been sent but awaits a response, if the number of maximum number of inflight messages has been reached no new messages can be accepted.

Variable:	Description:
► max_inflight_messages (INT)	Number of maximum inflight messages, the default is 20.
◄ retCode (INT)	0…successfully set the parameter 1…Allocate memory failed 3…input parameter invalid

mqtt_loop_start

Call this to start the task for the MQTT Client main task.

Variable:	Description:
◄ retCode (INT)	0…successfully startet the MQTT client task. 3…Client state is not correct.

mqtt_loop_stop

Stop the MQTT Client main task that was started using mqtt_loop_start(). With the force variable the user can decide whether the MQTT Client task should be stopped even if the MQTT Client is connected to the Broker.

Variable:	Description:
► force (BOOL)	If set the Client Task will be stopped even if the Client is connected.
◄ retCode (INT)	0…successfully stopped the MQTT client task. 3…Input Variables invalid or MQTT Client is connected to the Broker, call mqtt_disconnect() before stopping the thread or set force to TRUE

mqtt_connect_async

Call this method to connect to an MQTT Broker specified by the configuration methods and input variables, if called before mqtt_loop_start() the connection will be made after mqtt_loop_start() is called.

Variable:	Description:
► host (^UINT)	Hostname or IP address to connect to.
► port (INT)	Network port to connect to, the MQTT Specification V3.1.1 suggests 1883 for unencrypted and 8883 for connections using TLS.
► keepalive (INT)	Interval at which the server should send a PINGREQ to the client to confirm that the connection is still alive, specified in seconds.
◄ retCode (INT)	0…connection procedure started successful 3…invalid input parameters

mqtt_reconnect_async

Call this method to reconnect to a previously connected MQTT Broker. This should only be called after mqtt_connect_async(), the same parameters are reused.

Variable:	Description:
◄ retCode (INT)	0…restart connection successful 3…MQTT Client state is not correct

mqtt_disconnect

Call this method to disconnect from a currently connected MQTT Broker.

Variable:	Description:
◄ retCode (INT)	0…disconnect successful 4…MQTT client is not connected to the Server

mqtt_publish

Call this method to publish to a specified topic.

Variable:	Description:
► mid (^INT)	Pointer to the MID of the publish message. If QoS > 0 the MQTT Broker returns this value in response, so that the specified publish response can be tracked.
► topic (^UINT)	Pointer to the topic name. No wildcards are allowed in the publish topic name.
► payloadLen (UDINT)	Length of the payload to be published in bytes. (0-256MB)
► payload (^VOID)	Pointer to the data to be published
► qos (INT)	Requested QoS for this publish
► retain_pub (BOOL)	If the flag is set to true, the server will store this publication and send it to any new clients that connect to the specified topic.
◄ retCode (INT)	0…publish successful 1…no memory left to add the message to the buffer 3…invalid input parameters 4…MQTT client is not connected to the Server

mqtt_subscribe

Call this method to subscribe to a topic.

Variable:	Description:
► mid (^INT)	Pointer to the MID of the subscribe message. If QoS > 0 the MQTT Broker returns this value in response, so that the specified publish response can be tracked.

<table>
<tr><td></td><td>
<table>
<tr><td>► sub (^UINT)</td><td>Pointer to the topic name. Wildcards are acceptable as in the MQTT V3.1.1 Specification.</td></tr>
<tr><td>► qos (INT)</td><td>Requested Quality of Service</td></tr>
<tr><td>◄ retCode (INT)</td><td>0…subscribe successful
1…no memory left to add the message to the buffer
3…invalid input parameters
4…MQTT client is not connected to the Server</td></tr>
</table>
</td></tr>
<tr><td>mqtt_unsubscribe</td><td>Call this method to unsubscribe from a topic.
<table>
<tr><td><u>Variable:</u></td><td><u>Description:</u></td></tr>
<tr><td>► mid (^INT)</td><td>Pointer to the MID of the unsubscribe message.
If QoS > 0 the MQTT Broker returns this value in response, so that the specified publish response can be tracked.</td></tr>
<tr><td>► unsub (^UINT)</td><td>Pointer to the topic name.</td></tr>
<tr><td>◄ retCode (INT)</td><td>0…Unsubscribe successful
1…no memory left to add the message to the buffer
3…invalid input parameters
4…MQTT client is not connected to the Server</td></tr>
</table>
</td></tr>
<tr><td>mqtt_lib_version</td><td>Can be called to get the MQTT Library Version.
<table>
<tr><td><u>Variable:</u></td><td><u>Description:</u></td></tr>
<tr><td>major (^INT)</td><td>If not NULL, the major version of the MQTT Library is written here.</td></tr>
<tr><td>minor (^INT)</td><td>If not NULL, the minor version of the MQTT Library is written here.</td></tr>
<tr><td>revision (^INT)</td><td>If not NULL, the revision of the MQTT Library is written here.</td></tr>
</table>
</td></tr>
</table>