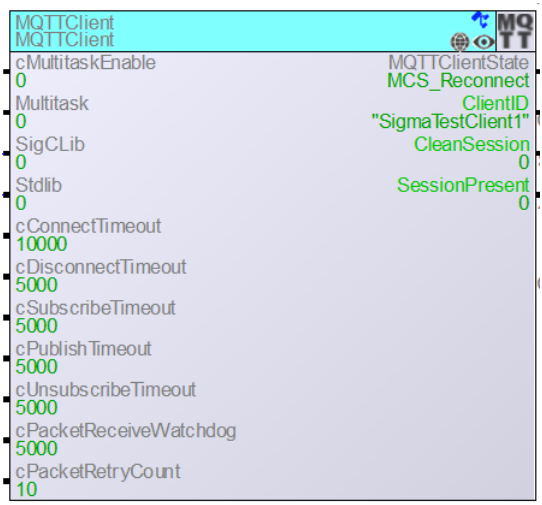
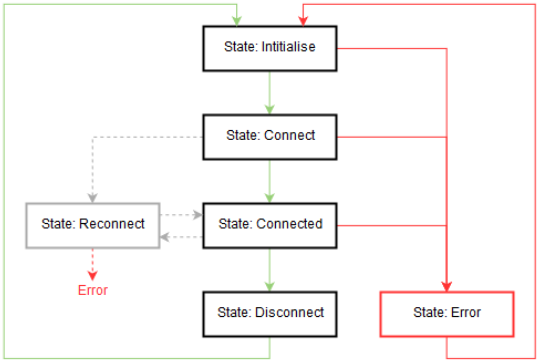
# MQTTClient




Die MQTTClient Klasse funktioniert als MQTT Client und dient als Schnittstelle zwischen der LASAL Class Applikation und dem MQTT Broker. Die MQTTClient Klasse implementiert das MQTT Protokoll V3.1.1.

## Hinweise für den Benutzer

Der MQTTClient Status Fluss ist unterhalb dargestellt und beschrieben:



| **Status** | **Beschreibung** |
|---|---|
| Initialise | Der MQTTClient startet im “Initialise“ Status und wartet darauf vom Benutzer konfiguriert zu werden. |
| | Der Benutzer kann die MQTTClient Klasse mit den folgenden Globalen Funktionen konfigurieren: (Weiter unten beschrieben)<br>mqtt_connect_callback_set()<br>mqtt_connect_with_flags_callback_set()<br>mqtt_disconnect_callback_set()<br>mqtt_publish_callback_set()<br>mqtt_message_callback_set()<br>mqtt_subscribe_callback_set()<br>mqtt_unsubscribe_callback_set()<br>mqtt_log_callback_set()<br>mqtt_username_pw_set()<br>mqtt_will_set()<br>mqtt_will_clear()<br>mqtt_tls_set()<br>mqtt_reconnect_delay_set()<br>mqtt_max_inflight_messages_set()<br>mqtt_reinitialise() |
| | Um den Status weiterzuschalten (siehe das Status Diagramm oberhalb), muss der Benutzer die folgenden Globalen Funktionen aufrufen:<br>mqtt_loop_start()<br>mqtt_connect_async() |
| Connect | In diesem Status versucht der MQTTClient eine Verbindung zum spezifizierten MQTT Broker aufzubauen. |
| | Der Verbindungs-Status kann im Status “Reconnect“, „Disconnect“ oder „Connected“ weitergeschalten werden. Folgende Eingabe Parameter beeinflussen die Verbindungsprozedur:<br>cConnectTimeout (Client)<br>CleanSession (Server) |
| Connected | Der MQTTClient befindet sich im „Connected“ Status. In diesem Status können Nachrichten gesendet, abonniert und empfangen werden etc. |
| | Der Status “Connected” kann zu „Reconnect“, “Disconnect” oder “Error States” weitergeschalten werden. |
| | Während dem “Connected” Status kann der Benutzer Methoden für das Verbindungs-Management / Kommunikation aufrufen, wie:<br>mqtt_reconnect_async()<br>mqtt_disconnect()<br>mqtt_publish()<br>mqtt_subscribe() |
| Reconnect | Wenn der MQTTClient die Verbindung verliert / wird mqtt_reconnect_async() aufgerufen. Die Klasse wechselt in den „Reconnect“ Status und in diesem Status versucht der MQTTClient die Verbindung zum MQTT Broker wiederaufzubauen. |

| | |
|---|---|
| | Die Parameter, die die Wiederverbindungs-Prozedur beeinflussen können durch den Aufruf der Methode mqtt_reconnect_delay_set() geändert werden. |
| | Die Klasse kann vom “Reconnect“ Status in den Status „Connected“, „Disconnect“ oder „Error“ weiterschalten. |
| Disconnect | Im “Disconnect” Status trennt der MQTTClient die Verbindung zum Broker und schaltet in den Status „Initialise“. |
| | Der Eingabe Parameter der den Trennungs-Vorgang beeinflusst ist:<br>cDisconnectTimeout (Client) |
| | Die Klasse kann vom “Disconnect” Status in den Status „Initialise“ wechseln. |
| Error | Der MQTTClient wird in den „Error“ Status gesetzt, wenn ein Fehler auftritt und wird zurückgesetzt um wieder funktionstüchtig zu werden. |
| | Die MQTTClient Klassen setzt sich automatisch zurück in den „Initialise“ Status und versucht wieder eine Verbindung aufzubauen. |

## Logging Texte

Definition der Logging Texte, die von der Klasse erstellt werden können, finden Sie in der Dokumentation zur Klasse MQTTClient_Interface.

# Schnittstellen

## Clients

| | |
|---|---|
| **cMultitaskEnable** | Wenn auf 1 gesetzt, verwendet das MQTTClient-Objekt einen Multitasking-Thread mit niedrigerer Priorität, um seine zyklische Arbeit auszuführen. Berücksichtigen Sie die Thread-Sicherheit bei der Verwendung dieser Funktion.<br>Datentyp: DINT |
| **Multitask** | Objekt Kanal zur _MuliTask OS Schnittstelle. |
| **SigCLib** | Objekt Kanal zur SigCLib OS Schnittstelle. |
| **StdLib** | Objekt Kanal zur _StdLib OS Schnittstelle. |
| **cConnectTimeout** | Das Verbindungs-Timeout kann an diesem Client initialisiert werden [ms].<br>Das Timeout wird während der TCP Verbindung und wenn auf das CONNACK vom MQTT Broker gewartet wird, verwendet.<br>Datentyp: UDINT |
| **cDisconnectTimeout** | Das “Disconnect” Timeout kann an diesem Client initialisiert werden [ms].<br>Das Timeout wird verwendet, wenn auf die DISCONNECT Control Packet gesendet Bestätigung gewartet wird. Wenn das Timeout auftritt (senden des Packet hat zu lange gedauert) wird die TCP Verbindung geschlossen, eine Log Nachricht wird erstellt und die Klasse wird weitergeschalten als ob das Packet erfolgreich gesendet wurde.<br>Datentyp: UDINT |
| **cSubscribeTimeout** | Das “Subscribe” Timeout kann an diesem Client initialisiert werden [ms].<br>Das Timeout wird verwendet, wenn das SUBACK Packet, infolge eines kürzlich gesendeten SUBSCRIBE Control Pakets, erwartet wird. Wenn das Timeout auftritt, wird eine Log Nachricht erstellt und die Subscription wird wiederholt bis der Wiederholversuch Anzahl definiert im Client cPacketRetryCount erreicht ist. Nachdem die Wiederholversuch Anzahl erreicht wurde, wird eine andere Log Nachricht generiert und das SUBSCRIBE Control Paket wird verworfen.<br>Datentyp: UDINT |
| **cPublishTimeout** | Das “Publish” Timeout kann an diesem Client initialisiert werden [ms].<br>Das Timeout wird verwendet, wenn ein Publish bezogenes Control Paket (PUBACK, PUBREC, PUBREL, PUBCOMP), infolge eines kürzlich gesendeten Publish (PUBLISH, PUBREC, PUBREL) Control Packet, erwartet wird. Wenn das Timeout auftritt, wird eine Log Nachricht erstellt und das Packet wird nochmal gesendet, solange bis der Wiederholversuch, definiert im Client cPacketRetryCount, erreicht ist. Nachdem der Wiederholversuch erreicht wurde, wird eine Log Nachricht generiert und das Publish bezogene Control Packet wird verworfen.<br>Datentyp: UDINT |

| | |
|---|---|
| **cUnsubscribeTimeout** | Das “Unsubscribe” Timeout kann an diesem Client initialisiert werden [ms].<br>Das Timeout wird verwendet, wenn ein UNSUBACK Packet, infolge eines kürzlich gesendeten UNSUBSCRIBE Control Packet, erwartet wird.<br>Wenn das Timeout auftritt, wird eine Log Nachricht erstellt und das Packet wird nochmal gesendet, solange bis der Wiederholversuch definiert im Client cPacketRetryCount erreicht wurde. Wurde der maximale Wiederholversuch erreicht, wird eine andere Log Nachricht erzeugt und das UNSUBSCRIBE Control Packet wird verworfen.<br>Datentyp: UDINT |
| **cPacketReceiveWatchdog** | Der Paket Empfangs-Watchdog kann an diesem Cllient initialisiert werden [ms].<br>Der Watchdog ist aktiv, wenn ein TCP Paket empfangen wird. Wenn das Watchdog Timeout auftritt, wird eine Log Nachricht erstellt und das aktuelle Packet das empfangen wurde, wird verworfen. Es ist wichtig das Wachtdog Timeout zu erhöhen, wenn der Benutzer große Nachrichten empfangen / senden möchte oder wenn eine unzuverlässige / langsame TCP Verbindung benutzt wird.<br>Datentyp: UDINT |
| **cPacketRetryCount** | Der cPacketRetryCount Client wird verwendet um die Wiederholversuche für das Senden der MQTT Control Packet zu definieren. Die Verwendung dieses Clients wird in der oberen Beschreibung der Clients erwähnt. |

## Servers

**MQTTClient-State**

Der MQTTClientState Server wird als Objekt Kanal zur MQTTClient Klasse verwendet und zeigt den aktuellen Status, wie beschrieben im Bereich „Hinweise für den Benutzer“.

| Server Wert | Status (siehe Oben) |
|---|---|
| MCS_Initialise | Initialisierungs-Status. |
| MCS_Connect | Es wird darauf gewartet ,dass der Client eine Verbindung zum Server aufbaut. |
| MCS_Connected | Der MQTTClient befindet sich im „Connected“ Status. In diesem Status können Nachrichten gesendet, abonniert und empfangen werden etc. |
| MCS_Reconnect | In diesem Status versucht der MQTTClient die Verbindung zum MQTT Broker wiederaufzubauen. |
| MCS_Disconnect | Im “Disconnect” Status trennt der MQTTClient die Verbindung zum Broker und schaltet in den Status „Initialise“. |
| MCS_Error | Der MQTTClient wird in den „Error“ Status gesetzt, wenn ein Fehler auftritt und wird zurück-gesetzt um wieder funktionstüchtig zu werden. |

| | | | |
|---|---|---|---|
| Einheit | - | Datentyp | t_e_MQTTClientStates |
| Wertebereich | - | Write Protected | TRUE |
| Defaultwert | - | Retentive | FALSE |

**ClientID**

Die MQTTClient ClientID kann an diesem Server gesetzt werden. Dieser Server ist ein Objektkanal auf ein String Objekt, welches im komplexen Netzwerk der Klasse platziert ist.

Die ClientID muss für jeden Client, der beim Broker registriert ist, einzigartig sein. Während der Initialisierung wird die ClientID auf die Serialnummer der CPU gesetzt. Es ist auch möglich die ClientID durch den Aufruf der globalen Methode mqtt_reinitialise() zu ändern.

Möglichkeiten der Konfiguration:

1. Änderung des Strings an diesem Server nachdem mqtt_reinitialise() und bevor mqtt_connect_async() aufgerufen wird.
2. Übergeben eines Strings an den Übergabeparameter „id“ der Methode mqtt_reinitialise().

| | | | |
|---|---|---|---|
| Einheit | - | Datentyp | Object Channel: String |
| Wertebereich | Max Len: 23 Characters | Write Protected | FALSE |
| Defaultwert | CPU Serial Number (Gesetzt in der Init) | Retentive | FALSE |

**CleanSession**

Der Session Typ (Clean Session / Non Clean Session) wird hier definiert. Die Funktionalität des Session Management ist nach der MQTT Spezifikation implementiert.

(CleanSession = 0) – Session Daten werden mit der Client ID, bei einem Verbindungsabbruch, im Broker gespeichert. Somit ist es nicht mehr notwendig nochmal die vorherigen Topics zu abonnieren, wenn die Verbindung wiederaufgebaut ist.

(CleanSession = 1) – Session Daten werden bei einem Verbindungsabbruch nicht gespeichert. Nachdem die Verbindung wiederaufgebaut ist, wird eine neue Session erstellt.

| | | | |
|---|---|---|---|
| Einheit | - | Datentyp | DINT |
| Wertebereich | 0 / 1 | Write Protected | FALSE |
| Defaultwert | 0 | Retentive | SRAM |

**SessionPresent**

In diesem Server wird angezeigt ob eine Session vorhanden ist. Dieser Server wird nur im Connected Status aktualisiert und wird bestimmt nachdem Typ der Session, der am Server CleanSession eingestellt ist.

| | | | |
|---|---|---|---|
| Einheit | ms | Datentyp | UDINT |
| Wertebereich | - | Write Protected | FALSE |
| Defaultwert | 10 s | Retentive | SRAM |

# Globale Methoden

**mqtt_reinitialise**

Wird aufgerufen um den MQTT Client neu zu initialisieren. Es werden alle aktuellen Verbindungen getrennt und man kann angeben ob die aktuelle Session beendet werden soll. Während der Re-Initialisierung verwendet der MQTT Client die neuen Parameter, die gesetzt wurden.

| Variable: | Beschreibung: |
|---|---|
| ► id – (^CHAR) | Zeiger auf die Client ID welche beim Verbindungsaufbau verwendet wird. Ist der Pointer NIL, wird eine selbst generierte ID zugeteilt und clean_session muss auf True gesetzt werden. |
| ► clean_session – (BOOL) | Wenn dieser Wert auf True gestellt wird, löscht der MQT Client alle „Subscriptions“, wenn die Verbindung zum Broker getrennt wird. Wenn die id NIL ist, muss dieser Wert auf True gestellt werden. |
| ◄ retCode – (INT) | MQTT_ERR_SUCCESS – erfolgreich initialisiert.<br>MQTT_ERR_NOMEM – Nicht genüg Speicher übrig um neu zu initialisieren.<br>MQLTT_ERR_INVAL – Fehler in der Konfiguration / Ungültige Übergabeparameter. |

**mqtt_connect_callback_set**

Mit dieser Methode kann ein Callback installiert werden, welcher aufgerufen wird, wenn eine Verbindung aufgebaut wurde. (Wenn CONNACK empfangen wurde)

| Variable: | Beschreibung: |
|---|---|
| ► pThis – (^VOID) | This-Zeiger des aufrufenden Objekts. Dieser wird beim Callback mit übergeben. |
| ► on_connect – (^VOID) | Zeiger auf die Callback Methode, welche aufgerufen werden soll. |

Callback Prototyp:

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

END_FUNCTION
```

Callback Schnittstelle:

| Variable: | Beschreibung: |
|---|---|
| ► pThis – (^VOID) | This-Zeiger des aufrufenden Objekts. Dieser wird beim Callback mit übergeben. |

| | |
|---|---|
| | ► retCode – (INT) — Rückgabewert welcher den Status der Connect Operation anzeigt.<br>0 - Erfolgreich.<br>1 - Protocol Mismatch.<br>2 - Identifier Rejected.<br>3 - Broker Unavailable.<br>255 Reserved. |
| **mqtt_connect_with_flags_callback_set** | Mit dieser Methode kann ein Callback installiert, welcher aufgerufen wird, wenn die Verbindung aufgebaute wurde. (Wenn CONNACK empfangen wurde) |

| Variable: | Beschreibung: |
|---|---|
| ► pThis – (^VOID) | This-Zeiger des aufrufenden Objekts. Dieser wird beim Callback mit übergeben. |
| ► on_connect_with_flags – (^VOID) | Zeiger auf die Callback Methode, welche aufgerufen werden soll. |

Callback Prototyp:

```
FUNCTION MQTTClient_Interface::Callback_OnConnectWithFlags
  VAR_INPUT
    pThis    : ^void;
    retCode  : INT;
    flags    : BYTE;
  END_VAR
```

Callback Schnittstelle:

| Variable: | Beschreibung: |
|---|---|
| ► pThis – (^VOID) | This-Zeiger des aufrufenden Objekts. Dieser wird beim Callback mit übergeben. |
| ► retCode – (INT) | Rückgabewert welcher den Status der Connect Operation anzeigt.<br>0 - Erfolgreich.<br>1 - Protocol Mismatch.<br>2 - Identifier Rejected.<br>3 - Broker Unavailable.<br>255 Reserved. |
| ► flags – (BYTE) | Zeigt die Flags des empfangenen CONNACK Control Packet. Bits:<br>0: Session vorhanden Flag<br>1-6: Reserved. |

**mqtt_disconnect_callback_set**

Mit dieser Methode kann ein Callback installiert werden, welcher aufgerufen wird, wenn der Server das DISCONNECT Kommando empfangen und die Verbindung zum Client getrennt hat.

| Variable: | Beschreibung: |
|---|---|

| | |
|---|---|
| ► pThis – (^VOID) | This-Zeiger des aufrufenden Objekts. Dieser wird beim Callback mit übergeben. |
| ► on_disconnect – (^VOID) | Zeiger auf die Callback Methode, welche aufgerufen werden soll. |

Callback Prototyp:

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

END_FUNCTION
```

Callback Schnittstelle:

| Variable: | Beschreibung: |
|---|---|
| ► pThis – (^VOID) | This-Zeiger des aufrufenden Objekts. Dieser wird beim Callback mit übergeben. |
| ► retCode – (INT) | Rückgabewert welcher den Status der diconnect Operation anzeigt.<br>0 – Trennung der Verbindung, nach Benutzeraufforderung.<br><> 0 – Unerwarteter Verbindungsabbruch. |

**mqtt_publish_callback_set**

Mit dieser Methode kann ein Callback installiert werden, welcher aufgerufen wird, wenn eine Nachricht gesendet wurde.

| Variable: | Beschreibung: |
|---|---|
| ► pThis – (^VOID) | This-Zeiger des aufrufenden Objekts. Dieser wird beim Callback mit übergeben. |
| ► on_publish – (^VOID) | Zeiger auf die Callback Methode, welche aufgerufen werden soll. |

Callback Prototyp:

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

END_FUNCTION
```

Callback Schnittstelle:

| Variable: | Beschreibung: |
|---|---|

| | |
|---|---|
| | ► pThis – (^VOID) — This-Zeiger des aufrufenden Objekts. Dieser wird beim Callback mit übergeben.<br>► mid – (INT) — Nachrichten ID der Publish Nachricht. |
| **mqtt_message_callback_set** | Mit dieser Methode kann ein Callback installiert werden, welcher aufgerufen wird, wenn eine Nachricht empfangen wurde.<br><br>Variable: / Beschreibung:<br>► pThis – (^VOID) — This-Zeiger des aufrufenden Objekts. Dieser wird beim Callback mit übergeben.<br>► on_message – (^VOID) — Zeiger auf die Callback Methode, welche aufgerufen werden soll.<br><br>Callback Prototyp:<br>`FUNCTION GLOBAL MyClass::OnMesssage`<br>`  VAR_INPUT`<br>`    pThis    : ^void;`<br>`    pMessage  : ^MQTTClient::t_s_MQTTMessage;`<br>`  END_VAR`<br><br>`END_FUNCTION`<br><br>Callback Schnittstelle:<br>Variable: / Beschreibung:<br>► pThis – (^VOID) — This-Zeiger des aufrufenden Objekts. Dieser wird beim Callback mit übergeben.<br>► pMessage – (^t_s_MQTT-Message) — Zeiger auf die empfangene Nachricht. Der Zeiger ist nur während dem Aufruf des Callbacks gültig. Werden die Daten im Nachhinein verwendet, müssen diese kopiert werden.<br>`t_s_MQTTMessage : STRUCT`<br>`  mid : INT;`<br>`  topic : ^CHAR;`<br>`  payload : ^void;`<br>`  payloadLen : UDINT;`<br>`  qos : INT;`<br>`  retainMsg : BOOL;`<br>`END_STRUCT;` |
| **mqtt_subscribe_callback_set** | Mit dieser Methode kann ein Callback installiert werden, welcher aufgerufen, wenn der Server auf eine Subscribe Anforderung antwortet. |

| Variable: | Beschreibung: |
|---|---|
| ► pThis – (^VOID) | This-Zeiger des aufrufenden Objekts. Dieser wird beim Callback mit übergeben. |
| ► on_subscribe – (^VOID) | Zeiger auf die Callback Methode, welche aufgerufen werden soll. |

Callback Prototyp:

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

END_FUNCTION
```

Callback Schnittstelle:

| Variable: | Beschreibung: |
|---|---|
| ► pThis – (^VOID) | This-Zeiger des aufrufenden Objekts. Dieser wird beim Callback mit übergeben. |
| ► mid – (INT) | Nachrichten ID der Subscribe Nachricht |
| ► qos_count – (INT) | Anzahl der Subscriptions. Das ist die Anzahl der QoS Integers welche im granted_qos verwendet werden. |
| ► granted_qos – (^INT) | Array von Integers welche die erteilte QoS für jede Subscription kennzeichnet.<br>1. 0, 1, 2 – erteilte QoS.<br>2. 128 - Subscription fehlgeschlagen. |

**mqtt_unsubscribe_callback_set**

Mit dieser Methode kann ein Callback installiert werden, welcher aufgerufen wird, wenn der Server zu einem Unsubscribe Anforderung antwortet.

| Variable: | Beschreibung: |
|---|---|
| ► pThis – (^VOID) | This-Zeiger des aufrufenden Objekts. Dieser wird beim Callback mit übergeben. |
| ► on_unsubscribe – (^VOID) | Zeiger auf die Callback Methode, welche aufgerufen werden soll. |

Callback Prototyp:

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

END_FUNCTION
```

Callback Schnittstelle:

| Variable: | Beschreibung: |
|---|---|
| ► pThis – (^VOID) | This-Zeiger des aufrufenden Objekts. Dieser wird beim Callback mit übergeben. |
| ► mid – (INT) | Nachrichten ID der Unsubscribe Nachricht |

**mqtt_log_call-back_set**

Mit dieser Methode kann ein Callback installiert werden, welcher aufgerufen wird, wenn ein Log vom MQTTClient erstellt wird.

| Variable: | Beschreibung: |
|---|---|
| ► pThis – (^VOID) | This-Zeiger des aufrufenden Objekts. Dieser wird beim Callback mit übergeben. |
| ► on_log – (^VOID) | Zeiger auf die Callback Methode, welche aufgerufen werden soll. |

Callback Prototyp:

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

END_FUNCTION
```

Callback Schnittstelle:

| Variable: | Beschreibung: |
|---|---|
| ► pThis – (^VOID) | This-Zeiger des aufrufenden Objekts. Dieser wird beim Callback mit übergeben. |

| | |
|---|---|
| | ► level – (INT): Der Level der Log Nachricht:<br>1. MQTT_LOG_INFO<br>2. MQTT_LOG_NOTICE<br>3. MQTT_LOG_WARNING<br>4. MQTT_LOG_ERR<br>5. MQTT_LOG_DEBUG |
| | ► pStr – (^CHAR): Zeiger auf den Log Nachrichten String |
| **mqtt_username_pw_set** | Kann aufgerufen werden um den Benutzernamen und das Passwort zu konfigurieren, welche beim Verbindungsaufbau zum MQTT Broker verwendet werden. Wird diese Methode nicht aufgerufen, so wird kein Benutzername oder Passwort, beim Aufruf von mqtt_connect(), mitgeschickt. Wenn der Benutzername NIL ist, wird das Passwort nicht gespeichert. Minimale MQTT Broker Spezifikation ist V3.1.1. |
| | Variable: / Beschreibung:<br>► username (^CHAR): Zeiger auf den Null terminierten Benutzernamen.<br>► password (^CHAR): Zeiger auf das Null terminierte Passwort.<br>► udPassLen (UDINT): Länge des Passworts.<br>◄ retCode (INT): 0… Benutzername und Passwort wurden erfolgreich gesetzt.<br>3…ungültige Übergabeparameter |
| **mqtt_will_set** | Muss vor dem Aufruf von mqtt_connect, aufgerufen werden, um einen „Letzten Willen“ beim Verbindungsaufbau zum MQTT Server mitzuschicken. Wird diese Methode nicht aufgerufen, so wird kein “Letzter Wille“ beim Verbinden mitgeschickt. |
| | Variable: / Beschreibung:<br>► topic (^CHAR): Zeiger auf den Topic Namen, auf welches der „Letzte Wille“ gesendet wird.<br>► payloadLen (UDINT): Payload Länge in Bytes. (0 -> 268 435 455 (256MB)).<br>► payload (^VOID): Zeiger auf die Payload Daten.<br>► qos (INT): QoS welcher beim Senden des „Letzen Willens“ verwendet werden soll. Gültige Werte sind 0,1 oder 2.<br>► retain_will (BOOL): Wenn dieses Flag gesetzt wird, so speichert der MQTT Server diese Publish Nachricht und schickt sie an alle neuen Clients, die auf dieses Topic subscriben.<br>◄ retCode (INT): 0… Letzer Will akzeptiert. Wird bei mqtt_connect() mitgeschickt.<br>3…ungültige Übergabeparameter |

<table>
<tr><td>mqtt_will_clear</td><td>Mit dieser Methode kann ein zuvor gesetzter „Letzter Wille“ gelöscht werden. (mqtt_will_set()).
<table>
<tr><td>Variable:</td><td>Beschreibung:</td></tr>
<tr><td>◄ retCode (INT)</td><td>0…erfolgreich gelöscht<br>3…kein Letzter Wille gesetzt</td></tr>
</table></td></tr>
<tr><td>mqtt_tls_set</td><td>Mit dieser Methode können die TLC Verbindungsparameter gesetzt werden. Wenn diese Methode aufgerufen wird, so wird bei mqtt_connect() eine Verbindung mit TLS erstellt.
<table>
<tr><td>Variable:</td><td>Beschreibung:</td></tr>
<tr><td>► cafile (^CHAR)</td><td>Null terminierter String des Pfades der PEM kodierten CA Zertifikat Datei, welche die vertrauten „Root Server“ enthält. Wird kein Pfad angegeben, so werden die default Zertifika-te unter dem Pfad C:\LSLSYS\SSL\ verwendet.</td></tr>
<tr><td>► certfile (^CHAR)</td><td>Null terminierter String des Pfades einer optionalen PEM kodierten Datei, welche die Client Zertifikate (Zertifikat Kette) enthält. Diese Datei muss im Verzeichnis C:\LSLSYS\SSL\ liegen. Der Pfad darf eine Länge von 255 Zeichen nicht überschreiten.<br>Wird ein Pfad übergeben so muss auch eine gültige „Key-„ Datei übergeben werden. (<keyfile>)</td></tr>
<tr><td>► keyfile (^CHAR)</td><td>Null terminierter String des Pfades einer PEM Kodierten Datei, die die „private Keys“ des Clients enthält. Diese Datei muss im Verzeichnis C:\LSLSYS\SSL\ liegen. Der Pfad darf eine Länge von 255 Zeichen nicht überschreiten.</td></tr>
<tr><td>► pw (^CHAR)</td><td>Optionales null terminiertes Passwort das zur Verschlüsselung der „Key-„ Datei verwendet wurde. Das Passwort darf eine Länge von 1023 Zeichen nicht überschreiten.</td></tr>
<tr><td>◄ retCode (INT)</td><td>0…Setzen der Parameter erfolgreich<br>3…ungültige Übergabeparameter</td></tr>
</table></td></tr>
<tr><td>mqtt_reconnect_delay_set</td><td>Mit dieser Methode kann die „Reconnect“ Verzögerung eingestellt werden, welches beim „Reconnect“ verwendet wird.
<table>
<tr><td>Variable:</td><td>Beschreibung:</td></tr>
<tr><td>► reconnect_delay (UINT)</td><td>Angabe des minimalen Verbindungsversuchsintervalls in Sekunden</td></tr>
<tr><td>► reconnect_delay_max (UINT)</td><td>Angabe des maximalen Verbindungsversuchsintervalls in Sekunden</td></tr>
</table></td></tr>
</table>

| | | |
|---|---|---|
| | ► reconnect_exponential_delay (BOOL) | Angabe ob das Verbindungsversuchsintervall exponentiell steigen soll. (True)<br><br>FALSE: min = 3 secs, max = 30 secs, Intervalle = 3,6,9,12,15,18,21,24,27,30,30,30…<br><br>TRUE: min = 3 secs, max = 30 secs, Intervalle= 3,6,12,24,30,30,30… |
| | ◄ retCode (INT) | 0…setzen der Parameter erfolgreich<br>3…ungültige Übergabeparameter |
| **mqtt_max_inflight_messages_set** | Mit dieser Methode kann die maximale Anzahl der einkommenden Nachrichten gesetzt werden. Das sind Nachrichten, die bereits gesendet wurden, aber noch auf eine Rückantwort warten. Wenn die maximale Anzahl erreicht wurde, können keine weiteren Nachrichten akzeptiert werden. | |
| | Variable: | Beschreibung: |
| | ► max_inflight_messages (INT) | Maximale Anzahl der einkommenden Nachrichten. Default 20 |
| | ◄ retCode (INT) | 0…Parameter erfolgreich gesetzt<br>1…Speicher-Allokierung fehlgeschlagen<br>3…Ungültige Übergabeparameter |
| **mqtt_loop_start** | Diese Methode muss aufgerufen werden um den MQTT Client Haupttask zu starten. | |
| | Variable: | Beschreibung: |
| | ◄ retCode (INT) | 0…MQTT Client Task erfolgreich gestartet.<br>3…Status des Clients ist nicht korrekt. |
| **mqtt_loop_stop** | Mit dieser Methode kann der MQTT Client Haupttask, welcher mit mqtt_loop_start() gestartet wurde, gestoppt werden. Mit den Übergabeparameter force kann entschieden werden ob der MQTT Client Task auch dann gestoppt werden soll, wenn der MQTT Client zum Broker verbunden ist. | |
| | Variable: | Beschreibung: |
| | ► force (BOOL) | Wenn gesetzt, wird der Client Task auch dann gestoppt, wenn der Client verbunden ist. |
| | ◄ retCode (INT) | 0…der MQTT Client Task wurde erfolgreich gestoppt.<br>3…Übergabeparameter ungültig oder der MQTT Client ist verbunden zum MQTT Broker. Es muss vorher mqtt_disconnect() aufgerufen oder force auf TRUE gesetzt werden. |
| **mqtt_connect_async** | Wird diese Methode aufgerufen, so wird eine Verbindung, spezifiziert mit den Konfigurations-Methoden und den Übergabeparametern, zum MQTT Broker | |

<table>
<tr><td></td><td>aufgebaut. Wenn diese Methode vor mqtt_loop_start() aufgerufen wird, so wird die Verbindung erst nachdem Aufruf von mqtt_loop_start() aufgebaut.
<table>
<tr><th>Variable:</th><th>Beschreibung:</th></tr>
<tr><td>► host (^UINT)</td><td>Hostname oder IP Adresse auf welcher verbunden werden soll.</td></tr>
<tr><td>► port (INT)</td><td>Netzwerk Port auf den Verbunden werden soll. Die MQTT Spezifikation V3.1.1. schreibt 1883 für unverschlüsselte und 8883 für verschlüsselte (TLS) Verbindungen vor.</td></tr>
<tr><td>► keepalive (INT)</td><td>Intervall, in dem der Server ein PINGREQ zum Client senden soll, um zu bestätigen das die Verbindung noch steht. Spezifiziert in Sekunden.</td></tr>
<tr><td>◄ retCode (INT)</td><td>0…Verbindungsprozedur erfolgreich gestartet<br>3…ungültige Übergabeparameter</td></tr>
</table></td></tr>
<tr><td>mqtt_reconnect_async</td><td>Mit dieser Methode kann die Verbindung zum MQTT Broker wiederaufgebaut werden. Sollte nur nachdem Aufruf von mqtt_connect_async() aufgerufen werden. Dieselben Parameter werden wiederverwendet.
<table>
<tr><th>Variable:</th><th>Beschreibung:</th></tr>
<tr><td>◄ retCode (INT)</td><td>0…Neustart der Verbindung erfolgreich<br>3…MQTT Client Status ist nicht korrekt</td></tr>
</table></td></tr>
<tr><td>mqtt_disconnect</td><td>Mit dieser Methode kann die Verbindung zum MQTT Broker getrennt werden.
<table>
<tr><th>Variable:</th><th>Beschreibung:</th></tr>
<tr><td>◄ retCode (INT)</td><td>0…Verbindung erfolgreich getrennt<br>4…MQTT client ist nicht zum Server verbunden.</td></tr>
</table></td></tr>
<tr><td>mqtt_publish</td><td>Mit dieser Methode kann zu einem definierten Topic „ge-published“ werden.
<table>
<tr><th>Variable:</th><th>Beschreibung:</th></tr>
<tr><td>► mid (^INT)</td><td>Zeiger auf die MID der Publish Nachricht.<br>Wenn QoS > 0 liefert der MQTT Broker diesen Wert zurück, um die spezifizierte Publish Rückantwort nachzuverfolgen.</td></tr>
<tr><td>► topic (^UINT)</td><td>Zeiger auf den Topic Namen. Es sind keine wildcards im Topic Namen erlaubt.</td></tr>
<tr><td>► payloadLen (UDINT)</td><td>Länge der Payload Daten die gesendet werden sollen in Bytes. (0-256MB)</td></tr>
<tr><td>► payload (^VOID)</td><td>Zeiger auf die Daten die gesendet werden sollen.</td></tr>
</table></td></tr>
</table>

| | | |
|---|---|---|
| | ► qos (INT) | Angeforderter QoS für dieses Publish. |
| | ► retain_pub (BOOL) | Wenn dieses Flag gesetzt wird, so speichert der MQTT Server diese Publish Nachricht und schickt sie an alle neuen Clients, die dieses Topic abonnieren. |
| | ◄ retCode (INT) | 0…Publish erfolgreich<br>1…kein Speicher vorhanden um die Nachricht in den Puffer einzutragen<br>3…ungültige Übergabeparameter<br>4…MQTT Client ist nicht zum Server verbunden. |
| **mqtt_subscribe** | Mit dieser Methode kann ein Topic abonniert werden. | |
| | Variable: | Beschreibung: |
| | ► mid (^INT) | Zeiger auf die MID der Subscribe Nachricht.<br>Wenn QoS > 0 liefert der MQTT Broker diesen Wert zurück, um die spezifizierte Subscribe Rückantwort nachzuverfolgen. |
| | ► sub (^UINT) | Zeiger auf den Topic Namen. Wildcards sind nach der MQTT V3.1.1. Spezifikation erlaubt. |
| | ► qos (INT) | Angefordeter Quality of Service. |
| | ◄ retCode (INT) | 0…Subscribe erfolgreich<br>1…kein Speicher mehr vorhanden um die Nachricht in den Puffer einzutragen.<br>3…ungültige Übergabeparameter<br>4…MQTT Client ist nicht zum Server verbunden |
| **mqtt_unsubscribe** | Mit dieser Methode kann man sich von einem Topic abmelden. | |
| | Variable: | Beschreibung: |
| | ► mid (^INT) | Zeiger auf die MID der Unsubscribe Nachricht.<br>Wenn QoS > 0 liefert der MQTT Broker diesen Wert zurück, um die spezifizierte Unsubscribe Rückantwort nachzuverfolgen. |
| | ► unsub (^UINT) | Zeiger auf den Topic Namen. |
| | ◄ retCode (INT) | 0…Unsubscribe erfolgreich<br>1… kein Speicher mehr vorhanden um die Nachricht in den Puffer einzutragen.<br>3…ungültige Übergabeparameter<br>4…MQTT Client ist nicht zum Server verbunden |

**mqtt_lib_version**

Mit dieser Methode kann die MQTT Library Version ausgelesen werden.

| Variable: | Beschreibung: |
|---|---|
| major (^INT) | Wenn nicht NIL, wird hier die Major Version der MQTT Library zurückgeliefert. |
| minor (^INT) | Wenn nicht NIL, wird hier die Minor Version der MQTT Library zurückgeliefert. |
| revision (^INT) | Wenn nicht NIL, wird hier die Revision der MQTT Library zurückgeliefert. |