Function block library

IIOT_Library_1

for PLCnext Engineer

Documentation for
PHOENIX CONTACT function blocks

Phoenix Contact (Nanjing) Smart Technology and Solutions Co. Ltd.
No.36 Phoenix Road, Jiangning Development Zone, Nanjing,
PC:21110 China

This documentation is available in English only.

Table of Contents

1 Installation hint

If you did not specify a different directory during **library** installation all data in the MSI file will be unpacked to

c:\Users\Public\Documents\Phoenix Contact Libraries\PLCnext Engineer

Please copy the library data to your PLCnext Engineer working library directory.

If you did not specify a different directory during **PLCnext Engineer** installation the default PLCnext Engineer working library directory is

c:\Users\Public\Documents\PLCnext Engineer\Libraries

2 General information

This library is delivered to support direct communication methods to connect to AWS IOT, AZURE IOT HUB and Ali IOT Platfrom. As well as tools for encoding and decoding data structures (such as JSON). The components contained in the package are suitable for communication in the IIoT environment.

This library provides MQTT client functions based on the Eclipse Paho MQTT C++ Client Library version 1.1. Direct references to relevant sections of the Paho library documentation are provided throughout this document.

This library provides JSON functions based on the jsoncpp C++ Library version 1.8.3
Direct references to relevant sections of the jsoncpp library documentation are provided throughout this document.

Note that every function block and method in this library currently includes an input parameter xEN, of type BOOL. This is intended to mimic the EN/ENO feature that will be introduced in a future version of PLCnext Engineer. When the EN/ENO feature becomes available in PLCnext Engineer, the xEN parameters in this library will be deprecated.

3 Change notes

Library version	Library build	PLCnext Engineer version	Change notes	Supported PLCs
1	20201209	2020.6	Initial release	AXCF 2152 (2404267) FW 2020.6

New version number: Functional changes of at least one function block, incompatibilities (e.g. change of library format).

New build number: No functional changes, but changes in the MSI file (e.g. documentation update, additional examples).

4 Function blocks

Function block	Description	Version	Supported articles	License
IIOT_MqttClient_1	Function block for MQTT client operations.	1	-	none
IIOT_JsonCoder_1	JSON Coder	1	-	none
IIOT_JsonDecoder_1	JSON Decoder	1	-	none
IIOT_AzureCertificateInfo_1	Function block to translate original Azure IOT configuration to internal configuration which can be used to feed IIOT_MqttClient_1 FBs	1	-	none
IIOT_AwsCertificateInfo_1	Function block to translate original AWS IOT configuration to internal configuration which can be used to feed IIOT_MqttClient_1 FBs	1	-	none
IIOT_AliCertificateInfo_1	Function block to translate original Ali IOT configuration to internal configuration which can be used to feed IIOT_MqttClient_1 FBs	1	-	none

5 IIOT_MqttClient_1

This function block allows MQTT client instances to be created.

All instances of this function block should be called unconditionally in a cyclic task.

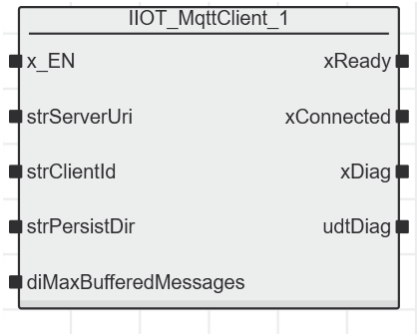
Calls to this function block and its methods take a non-zero amount of time. It is the users responsibility to ensure that the task execution time does not exceed the task watchdog time. It is recommended that any POU using this function block is executed in a task with a watchdog time no lower than 100 milli-seconds, and that a maximum of 10 messages be published and/or consumed every second.

Each MQTT client instance is associated with a single MQTT server. TCP and SSL connections are supported. While a client instance is connected to an MQTT server, messages can be published to and consumed from the server. Message payloads are of the IEC 61131 type ANY.

This function block is based on the async_client class in the Paho library.

The first call to this function block with x_EN=TRUE creates a Paho client object with the specified parameters. After this, all changes to the input parameters are ignored, even when x_EN=FALSE. If it is necessary to connect to MQTT servers/brokers with different input parameters, a new instance of the FB is required.

5.1 Function block call



5.2 Input parameters

Name	Type	Description
x_EN	BOOL	Enables the function block.
strServerUri	STRING	The address of the server to connect to, specified as a URI[1][2].
strClientId	STRING	A client identifier that is unique on the server being connected to[1].
strPersistDir	STRING	Not currently implemented.
diMaxBufferedMessages	DINT	Not currently implemented.

5.3 Output parameters

Name	Type	Description
xReady	BOOL	FALSE: The MQTT client is not ready to receive method calls. TRUE: The MQTT client is ready to receive method calls.
xConnected	BOOL	FALSE: The MQTT client is not connected to the server. TRUE: The MQTT client is connected to the server.
xDiag	BOOL	FALSE: There is no diagnostics information available. TRUE: Diagnostics information is available at udtDiag.
udtDiag	MQTT_UDT_DIAGNOSTICS	Diagnostics information. Only valid while xDiag is TRUE.

5.4 Reference

Each function block instance is initialised by calling the corresponding async_client constructor in the Paho MQTT C++ client library.

The xConnected output is set using the async_client::is_connected function on the Paho client.

5.5 Connect

Connects to an MQTT server using the provided connect options.

DO NOT call this method unconditionally in a cyclic task.

Instead, call this method ONCE to establish a connection.

The method will fail if the xReady output on the IIOT_MqttClient_1 function block instance is FALSE.

This is an asynchronous method call, so even when this method returns TRUE, do not assume that the client is connected to the server unless the xConnected output on the IIOT_MqttClient_1 instance is TRUE.

The progress of the Connect action can be checked using the IsComplete method.

5.5.1 Input parameters

Name	Type	Description
xEN	BOOL	Enables the method.
strUserName	STRING or STRING[]	The user name to use for the connection[1]. Pass an empty string '' if not required.
strPassword	STRING or STRING[]	The password to use for the connection[1]. Pass an empty string '' if not required.
udtOptions	MQTT_UDT_CONNECT_OPTIONS	A set of connection parameters.

5.5.2 Return value

Type	Description
DINT	-1 : Error. Check the Output.log file for more information. All other values : The ID of the asynchronous action.

5.5.3 Reference

This method is based on the async_client::connect function in the Paho MQTT C++ client library.

5.6 Publish

Publishes a message to a topic on the server.

DO NOT call this method unconditionally in a cyclic task.

Instead, call this method ONCE to publish a single message.

The method will fail if the xReady output on the IIOT_MqttClient_1 function block instance is FALSE.

This is an asynchronous method call, so even when this method returns TRUE, the message has not necessarily been published successfully.

A list of pending deliveries can be obtained using the GetPendingDeliveryIDs method.

All bytes from the anyPayload variable are published as the message payload. This means that, when a STRING variable is passed as anyPayload, the message payload will include the four header bytes that appear at the start of every IEC 61131 STRING variable. If the intention is to publish the complete IEC 61131 string, then udiLength should account for the four header bytes and the null terminator. If the intention is to pass the string data without the header or terminator bytes, then the STRING_TO_BUF function should be used to extract the required characters from the STRING variable.

5.6.1 Input parameters

Name	Type	Description
xEN	BOOL	Enables the method.
strTopic	STRING or STRING[]	The topic to deliver the message to.
anyPayload	ANY	The bytes to use as the message payload.
udiLength	UDINT	The number of bytes to publish. This must be less than or equal to the size of the variable passed to the anyPayload input.
diQos	DINT	The Quality of Service to deliver the message at[1].
xRetained	BOOL	Whether or not this message should be retained by the server[2].

1. There are three levels of Quality of Service:
   - 0 : Fire and forget - the message may not be delivered.
   - 1 : At least once - the message will be delivered, but may be delivered more than once in some circumstances.
   - 2 : Once and one only - the message will be delivered exactly once.

2. xRetained = TRUE indicates that the MQTT server should retain a copy of the message. The message will then be transmitted to new subscribers to a topic that matches the message topic. xRetained = FALSE indicates that this message should not be retained by the MQTT server.

5.6.2 Return value

Type	Description
DINT	-1 : Error. Check the Output.log file for more information. All other values : The delivery ID of the message.

5.6.3 Reference

This method is based on the async_client::publish function in the Paho MQTT C++ client library.

5.7 GetPendingDeliveryIDs

Returns the delivery IDs for any outstanding publish operations.

The method will fail if the xReady output on the IIOT_MqttClient_1 function block instance is FALSE.

5.7.1 Input parameters

Name	Type	Description
xEN	BOOL	Enables the method.

5.7.2 InOut parameters

Name	Type	Description
anyDelivery-IDs	ANY	An array of delivery IDs for all incomplete publish actions. The connected variable should be an array of DINTs.

5.7.3 Return value

Type	Description
DINT	-1 : Error. Check the Output.log file for more information. All other values : The number of delivery IDs returned.

5.7.4 Reference

This method is based on the async_client::get_pending_delivery_tokens function in the Paho MQTT C++ client library.

5.8 StartConsuming

This initialises the client to receive messages through a queue.

This method must be called before TryConsumeMessage is called for the first time.

DO NOT call this method unconditionally in a cyclic task.

Instead, call this method ONCE to start consuming messages.

The method will fail if the xReady output on the IIOT_MqttClient_1 function block instance is FALSE.

5.8.1 Input parameters

Name	Type	Description
xEN	BOOL	Enables the method.

5.8.2 Return value

Type	Description
BOOL	FALSE: The start consume request failed. TRUE: The client is ready to consume messages.

5.8.3 Reference

This method is based on the async_client::start_consuming function in the Paho MQTT C++ client library.

5.9 Subscribe

Subscribe to a topic, which may include wildcards.

This method can be called multiple times, if necessary, to subscribe to multiple topics.

DO NOT call this method unconditionally in a cyclic task.

Instead, call this method ONCE to subscribe to a topic.

Messages received on all subscribed topics will be added to the message queue created by the

StartConsuming method.

The method will fail if the xReady output on the IIOT_MqttClient_1 function block instance is FALSE.

This is an asynchronous method call, so even when this method returns TRUE, the topic has not necessarily been subscribed successfully.

The progress of the Subscribe action can be checked using the IsComplete method.

5.9.1 Input parameters

Name	Type	De-scription
xEN	BOOL	Enables the method.
strTop-icFilter	STRING or STRING[]	The topic to subscribe to, which can include wildcards.
diQos	DINT	The maximum quality of service at which to subscribe. Messages published at a lower quality of service will be received at the published QoS. Messages published at a higher quality of service will be received using the QoS specified on the subscribe.

5.9.2 Return value

Type	Description
DINT	-1 : Error. Check the Output.log file for more information. All other values : The ID of the asynchronous action.

5.9.3 Reference

This method is based on the async_client::subscribe function in the Paho MQTT C++ client library.

5.10 TryConsumeMessage

Try to read the next message from the queue.

This method must only be called after the client has been initialised to receive messages using the

StartConsuming method.

The method will fail if the xReady output on the IIOT_MqttClient_1 function block instance is FALSE.

If the length of the variable passed to anyPayload is shorter than the number of bytes in the payload of the consumed message, then the method will fail, none of the message payload bytes will be copied to anyPayload, and the message payload will be lost. However in this case, the strTopic and udtInfo values for the consumed message will be returned, which may help with diagnosis.

Note that the type of the incoming message payload cannot be checked by this method, and so payload bytes are copied blindly to anyPayload. In applications where the type of incoming message payloads may vary, it is recommended that the variable passed to anyPayload is a byte array with length equal to (or greater than) the maximum expected payload length. Once the type of the payload has been determined (for example, by examining the value of strTopic), then one of the PLCnext Engineer function blocks BUF_TO_* can be used to copy the message payload to a variable of the correct type.

5.10.1 Input parameters

Name	Type	Description
xEN	BOOL	Enables the method.

5.10.2 InOut parameters

Name	Type	Description
strTopic	STRING or STRING[]	The topic that the consumed message was published on.
anyPayload	ANY	The bytes from the consumed message payload.
udtInfo	MQTT_UDT_MESSAGE_INFO	Infomation about the consumed message.

5.10.3 Return value

Type	Description
BOOL	FALSE: No message was available, or the request failed. TRUE: A message was read.

5.10.4 Reference

This method is based on the async_client::try_consume_message function in the Paho

MQTT C++ client library.

5.11 Unsubscribe

Requests the server unsubscribe the client from a topic.

This method can be called multiple times, if necessary, to unsubscribe from multiple topics.

DO NOT call this method unconditionally in a cyclic task.

Instead, call this method ONCE to unsubscribe from a topic.

The method will fail if the xReady output on the IIOT_MqttClient_1 function block instance is FALSE.

This is an asynchronous method call, so even when this method returns TRUE, the topic has not necessarily been unsubscribed successfully.

The progress of the Unsubscribe action can be checked using the IsComplete method.

5.11.1 Input parameters

Name	Type	Description
xEN	BOOL	Enables the method.
strTopicFilter	STRING or STRING[]	The topic to unsubscribe from. It must match a topic filter specified on an earlier subscribe.

5.11.2 Return value

Type	Description
DINT	-1 : Error. Check the Output.log file for more information. All other values : The ID of the asynchronous action.

5.11.3 Reference

This method is based on the async_client::unsubscribe function in the Paho MQTT C++ client library.

5.12 StopConsuming

This deletes the internal message queue that was created by the StartConsuming call, and discards any unread messages.

DO NOT call this method unconditionally in a cyclic task.

Instead, call this method ONCE to stop consuming messages.

The method will fail if the xReady output on the IIOT_MqttClient_1 function block instance is FALSE.

5.12.1 Input parameters

Name	Type	Description
xEN	BOOL	Enables the method.

5.12.2 Return value

Type	Description
BOOL	FALSE: The stop consume request failed. TRUE: The client has stopped consuming messages.

5.12.3 Reference

This method is based on the async_client::stop_consuming function in the Paho MQTT C++ client library.

5.13 Disconnect

Disconnects from the server.

DO NOT call this method unconditionally in a cyclic task.

Instead, call this method ONCE to disconnect from the server.

The method will fail if the xReady output on the IIOT_MqttClient_1 function block instance is FALSE.

This is an asynchronous method call, so even when this method returns TRUE, do not assume that the client has disconnected from the server unless the xConnected output on the IIOT_MqttClient_1 instance is FALSE.

The progress of the Disconnect action can be checked using the IsComplete method.

5.13.1 Input parameters

Name	Type	Description
xEN	BOOL	Enables the method.
diTimeout	DINT	The amount of time in milliseconds to allow for existing work to finish before disconnecting. A value of zero or less means the client will not quiesce.

5.13.2 Return value

Type	Description
DINT	-1 : Error. Check the Output.log file for more information. All other values : The ID of the asynchronous action.

5.13.3 Reference

This method is based on the async_client::disconnect function in the Paho MQTT C++ client library.

5.14 Reconnect

Reconnects the client using the parameters from the previous call to Connect. The client must have previously called Connect for this to work.

DO NOT call this method unconditionally in a cyclic task.

Instead, call this method ONCE to re-establish a connection.

The method will fail if the xReady output on the IIOT_MqttClient_1 function block instance is FALSE.

This is an asynchronous method call, so even when this method returns TRUE, do not assume that the client is connected to the server unless the xConnected output on the IIOT_MqttClient_1 instance is TRUE.

The progress of the Reconnect action can be checked using the IsComplete method with the Connect option.

5.14.1 Input parameters

Name	Type	Description
xEN	BOOL	Enables the method.

5.14.2 Return value

Type	Description
DINT	-1 : Error. Check the Output.log file for more information. All other values : The ID of the asynchronous action.

5.14.3 Reference

This method is based on the async_client::reconnect function in the Paho MQTT C++ client library.

5.15 IsComplete

Checks whether or not an asynchronous action has finished.

Once an action has finished, this method will return TRUE until the action is triggered again.

The method will fail if the xReady output on the IIOT_MqttClient_1 function block instance is FALSE.

5.15.1 Input parameters

Name	Type	Description
xEN	BOOL	Enables the method.
enAction	MQTT_EN_ASYNC_ACTION	The asynchronous action to check.

5.15.2 Return value

Type	Description
BOOL	FALSE: The action is not complete. TRUE: The action is complete.

5.15.3 Reference

This method is based on the token::is_complete function in the Paho MQTT C++ client library.

5.16 Diagnostics

wDiagCode	Source of diagnostics information
0xC901	Connect
0xC902	Publish
0xC903	GetPendingDeliveryIDs
0xC904	StartConsuming
0xC905	Subscribe
0xC906	TryConsumeMessage
0xC907	Unsubscribe
0xC908	StopConsuming
0xC909	Disconnect
0xC90A	Reconnect
0xC90B	IsComplete

wAddDiagCode	Description
0x0100	MQTT Client is not ready.
0x02xx	One of the parameters is NULL. LSB is the parameter number, starting from 0.
0x03xx	One of the parameters is too small. LSB is the parameter number, starting from 0.
0x04xx	One of the parameters is too large. LSB is the parameter number, starting from 0.
0x0500	Security exception from Paho.
0x0600	General exception from Paho.
0x0700	General exception, not from Paho.
0x0800	The asynchronous method did not complete successfully.

Additional information on method failures or other unexpected behaviour may be available in the Output.log file in the PLC directory /opt/plcnext/logs.

Instructions on how to read and copy files from the PLC are available in the PLCnext Technology Info Centre.

5.17 Examples

For examples showing how to use the MQTT Client, the following PLCnext Engineer project is installed with this library:

- MQTT_1_EXA_MQTT_Client.pcwex

For these examples, the following hardware is used:

- AXC F 1152 (1151412)

5.17.1 Common setup tasks

To use the examples in the project, open the project in PLCnext Engineer and complete the following tasks:

1. In the Project Settings window, set the fields in the IP range section to suit your network. In particular, if you want to use the default Mosquitto test broker, the Default gateway must be set to the IP address of your internet router .

2. Check that the controller model and firware version in the project matches the controller you are using.

3. In the controller Settings window, set the fields in the Ethernet [Profinet] section to suit your controller.

4. In the Project Online Devices window, check that the PLC is online and that the Status field is OK (indicated with a check mark).

5. Add the MQTT Client library to the project using the “Add User Library…” context menu in the Libraries area of the Component view.

6. Download the project to the PLC and check that the PLC starts OK.

You are now ready to try the example programs.

Before using any of the example programs, check the values of the following variables, and change them if required:

Name	Description	Default value
SERVER_ADDRESS	The address of the MQTT broker. Must be accessible from the PLC. (check using ping if necessary).	tcp://5.196.95.208:1883
CLIENT_ID	The ID of the MQTT client. Must be unique on the MQTT broker.	PLCnext Subscriber or PLCnext Publisher
TOPIC	The topic to publish/subscribe to on the broker.	plcnext

Note that every program in this example project contains two code sheets, one written in Ladder and the other in Structured Text. These two code sheets implement exactly the same functions, but only one will be executed at a time. You can control which sheet is executed using the Language variable in the respective program.

5.17.2 Publish

This is an example of how to send messages as an MQTT publisher. The example demonstrates:

- Connecting to an MQTT server/broker Publishing messages
- Automatic reconnects Off-line buffering
- Default file-based persistence

To use the Publish example, open the program named “Publish”, and complete the following tasks:

1. Check the values of the SERVER_ADDRESS, CLIENT_ID and TOPIC variables, and change them if required.
2. In the PLCnext Tasks and Events window, create an instance of the Publish program in the Cycle100 task.
3. Write and Start the project. PLCnext Engineer will go online to the PLC and enter Debug mode.
4. Add the Run variable to the Watch window.
5. Set the value of Run to TRUE.
6. Using an MQTT subscriber, check that the test messages are being published.
7. Set the value of Run to FALSE.

The program implements a simple state sequencer that progresses through the following steps:

- Idle - waiting for the Run signal.
- Connecting - waiting for the client to connect to the server.
- Running - publishing a simple message every 5 seconds, and waiting for the Run signal to disappear.
- Disconnecting - waiting for the client to disconnect from the server.

This example is based on the async_publish sample in the Paho MQTT C++ library.

5.17.3 Subscribe

This is an example of how to receive messages as an MQTT subscriber. The example demonstrates:

- Connecting to an MQTT server/broker Subscribing to a topic
- Receiving messages Attempting manual reconnects.

- Using a “clean session” and manually re-subscribing to topics on reconnect.

To use the Subscribe example, open the program named “Subscribe”, and complete the following tasks:

1. Check the values of the SERVER_ADDRESS, CLIENT_ID and TOPIC variables, and change them if required.

2. In the PLCnext Tasks and Events window, create an instance of the Subscribe program in the Cycle100 task.

3. Write and Start the project. PLCnext Engineer will go online to the PLC and enter Debug mode.

4. Add the Run variable to the Watch window.

5. Set the value of Run to TRUE.

6. Using an MQTT publisher, check that messages are being received on the subscribed topic.

7. Set the value of Run to FALSE.

The program implements a simple state sequencer that progresses through the following steps:

- Idle - waiting for the Run signal.

- Connecting - waiting for the client to connect to the server. Subscribing - waiting for the subscription to complete.

- Running - receiving messages, and waiting for the Run signal to disappear. Unsubscribing - waiting for the unsubscription to complete.

- Disconnecting - waiting for the client to disconnect from the server.

This example is based on the async_subscribe sample in the Paho MQTT C++ library.

5.17.4 Using SSL

The previous examples can be adjusted to use an SSL connection using the following steps:

1. Download the server certificate for the Mosquitto test broker (in PEM format) from test.mosquitto.org.

2. Copy the server certificate to the PLC. Take note of the full path to the server certificate, e.g. /opt/plcnext/mosquitto.org.crt

3. Change the value of the SERVER_ADDRESS variable to the following: ssl://5.196.95.208:8883

4. In the code, before calling the Connect method, set the following values:

- connOpts.xUseSsl := TRUE
- connOpts.udtSsl.strTrustStore := /opt/plcnext/mosquitto.org.crt (* the full path to the server certificate on the PLC *)

5. Write and start the project.

The client will now use SSL to connect and communicate with the MQTT broker.

6 IIOT_JsonCoder_1

This function block allows IIOT_JsonCoder_1 instances to be created.
All instances of this function block should be called unconditionally in a cyclic task.

Calls to this function block and its methods take a non-zero amount of time. It is the users responsibility to ensure that the task execution time does not exceed the task watchdog time.

The first call to this function block with x_EN=TRUE creates a internal empty JSON root object. After this, toggling the x_EN parameter will clear the internal JSON root object.

6.1 Input parameters

Name	Type	Description
x_EN	BOOL	Enables the function block.

6.2 Output parameters

Name	Type	Description
xReady	BOOL	FALSE: The IIOT_JsonCoder_1 is not ready to execute methods. TRUE: The IIOT_JsonCoder_1 is ready to execute methods.
udtStatusFB	JSON_UDT_STATUS	Diagnostics information. Only valid if xExecuted is TRUE (located inside the udtStatusFB structure).

6.3 Add_element

The Add_element method adds a new JSON member (name/value pair) to a JSON object created and routed by IIOT_JsonCoder_1 function block. The JSON name must be provided as STRING and the JSON value as IEC 61131-3 elementary. The multiple instances of this method allow to build a full tree of JSON elements, with the exception of the JSON arrays and JSON objects.
The following IEC 61131-3 elementary types are supported:

Type	Description
BOOL	A simple type representing Boolean values of true or false.
BYTE	An integral type representing unsigned 8-bit integers with values between 0 and 255.
SINT	An integral type representing signed 8-bit integers with values between -128 and 127.
USINT	An integral type representing unsigned 8-bit integers with values between 0 and 255.
INT	An integral type representing signed 16-bit integers with values between -32768 and 32767.
UINT	An integral type representing unsigned 16-bit integers with values between 0 and 65535.
DINT	An integral type representing signed 32-bit integers with values between -2147483648 and 2147483647.
UDINT	An integral type representing unsigned 32-bit integers with values between 0 and 4294967295.
LINT	An integral type representing signed 64-bit integers with values between -9223372036854775808 and 9223372036854775807.
ULINT	An integral type representing unsigned 64-bit integers with values between 0 and 18446744073709551615.
REAL	A floating point type representing values ranging from approximately 1.5 x 10 -45 to 3.4 x 10 38 with a precision of 7 digits.
LREAL	A floating point type representing values ranging from approximately 5.0 x 10 -324 to 1.7 x 10 308 with a precision of 15-16 digits.
STRING	A general type representing a string corresponding to the IEC1131-3 programming model.

The method can be executed if the xReady output on the IIOT_JsonCoder_1 function block instance is TRUE.
Positive edge on xEN input parameter starts the method execution once.

NOTE that this method is not thread-safe. Ensure that multiple threads/tasks are not accessing a JSON object instance concurrently at the same time.
Access to the JSON object instance must be serialized (using multiple instances of this method in one program) or synchronized between programs executing in different ESM-Tasks. Otherwise an inconsistent state can result.

DO NOT call this method multiple to add the same JSON member to a JSON object.
The JSON member will be not added to the JSON object if a duplicate JSON member is already included into the JSON object.

6.3.1 Input parameters

Name	Type	Description
xEN	BOOL	Enables the method.
strJsonKey	STRING	The JSON member name.
anyJsonValue	ANY	The JSON value of IEC 61131-3 elementary datatype.

6.3.2 Output parameter

Name	Type	Description
udtStatusMethod	JSON_UDT_STATUS	Diagnostics information.

6.3.3 Return value

Type	Description
BOOL	FALSE: The method is not executed, the parameters are incomplete/invalid. TRUE: The method is executed, the parameters are complete/valid.

6.3.4 Reference

This method is based on the public member functions of Json::Value Class in the jsoncpp C++ Library.

6.4 Print_to_buf

The Print_to_buf method store the full tree of JSON object rooted by IIOT_JsonCoder_1 function block into the provided byte array buffer.
After the method call, the byte array buffer contains JSON data coded in UTF8.
The method can be executed if the xReady output on the IIOT_JsonCoder_1 function block instance is TRUE. Positive edge on xEN input parameter starts the method execution once.

NOTE that this method is not thread-safe. Ensure that multiple threads/tasks are not accessing a Json object instance concurrently at the same time.
Access to the Json object instance must be serialized (using multiple instances of this method in one program) or synchronized between programs executing in different ESM-Tasks. Otherwise an inconsistent state can result.

6.4.1 Input parameters

Name	Type	Description
xEN	BOOL	Enables the method.

6.4.2 Output parameter

Name	Type	Description
anyJsonBuffer	ANY	The byte buffer for storing the JSON object.
anyBufCnt	ANY	Copied count of bytes as a 32-bit signed integer.
udtStatusMethod	JSON_UDT_STATUS	Diagnostics information.

6.4.3 Return value

Type	Description
BOOL	FALSE: The method is not executed, the parameters are incomplete/invalid. TRUE: The method is executed, the parameters are complete/valid.

6.5 Print_to_file

The Print_to_file method store the full tree of Json object rooted by IIOT_JsonCoder_1 function block into the specified JSON file.
The method can be executed if the xReady output on the IIOT_JsonCoder_1 function block instance is TRUE. Positive edge on xEN input parameter starts the method execution once.

NOTE that this method is not thread-safe. Ensure that multiple threads/tasks are not accessing a Json object instance concurrently at the same time.
Access to the Json object instance must be serialized (using multiple instances of this method in one program) or synchronized between programs executing in different ESM-Tasks. Otherwise an inconsistent state can result.

6.5.1 Input parameters

Name	Type	Description
xEN	BOOL	Enables the method.
strJsonPath	STRING	The path of JSON file name with the end suffix ".json" .

6.5.2 Output parameter

Name	Type	Description
udtStatusMethod	JSON_UDT_STATUS	Diagnostics information.

6.5.3 Return value

Type	Description
BOOL	FALSE: The method is not executed, the parameters are incomplete/invalid. TRUE: The method is executed, the parameters are complete/valid.

6.6 Diagnostics

wDiagCode	Source of diagnostics information
0x0000	Function block is deactivated
0x8000	Function block is in regular operation
0xC340	FILE_OPEN error
0xC341	FILE_CLOSE error
0xC342	FILE_READ error
0xC343	FILE_WRITE error
0xC520	Error of an internal used FB. This FB is instantiated in the current FB
0xC110	Invalid input or inout parameter

wAddDiagCode	Description
0x0000	Not additional information exist.
0x0001	File is Empty.
0x0002	File is not Valid.
0x0003	File size Error.
0x0004	Invalid Json Key.
0x0005	Buffer size Error.
0x0006	Type is not Elementary.

Additional information on method failures or other unexpected behaviour may be available in the Output.log file in the PLC directory /opt/plcnext/logs.

Instructions on how to read and copy files from the PLC are available in the PLCnext Technology Info Centre.

7 IIOT_JsonDecoder_1

This function block allows IIOT_JsonDecoder_1 instances to be created.

All instances of this function block should be called unconditionally in a cyclic task.

Calls to this function block and its methods take a non-zero amount of time. It is the users responsibility to ensure that the task execution time does not exceed the task watchdog time.

The first call to this function block with x_EN=TRUE, starts the JSON parser, which reads in the JSON root object provided via byte buffer or via JSON file. After first call to the FB with x_EN=TRUE, subsequent toggling of the x_EN parameter replaces the internal root object with the JSON provided at the anyJsonData input.

The byte buffer content can be a path to JSON file (.json) or a JSON string.

If the buffer content is a path to JSON file, the input "x_IsPath" must be activated. IIOT_JsonDecoder_1 method "Get_value" requires as a parameter the complete JSON key path.

If you are not sure whether JSON key path has the JSON value, you can activate the "x_GetTree" input. In this case, the IIOT_JsonDecoder_1 store the complete JSON tree with JSON key paths and associated values into the file "/opt/plcnext/JsonValueTree.json".

7.1 Input parameters

Name	Type	Description
x_EN	BOOL	Enables the function block.
x_IsPath	BOOL	FALSE: The content of byte buffer "anyJsonData" is a JSON string. TRUE: The content of byte buffer "anyJsonData" is a path to JSON file.
x_GetTree	BOOL	FALSE: The JSON tree will be not stored on the plcnext target. TRUE: The JSON tree will be stored on the plcnext target under "/opt/plcnext/JsonValueTree.json".

7.2 Output parameters

Name	Type	Description
xReady	BOOL	FALSE: The IIOT_JsonDecoder_1 is not ready to execute methods. TRUE: The IIOT_JsonDecoder_1 is ready to execute methods.
udtStatusFB	JSON_UDT_STATUS	Diagnostics information. Only valid if xExecute is TRUE (located inside the udtStatusFB structure).

7.3 InOut parameters

Name	Type	Description
anyJsonData	ANY	Array of Byte for providing the JSON data as JSON file with the end suffix ".json" or JSON string.

7.4 Get_value

The Get_value method gets a JSON value from JSON element of JSON object rooted by IIOT_JsonDecoder_1 function block. The multiple instances of this method allow to parse a full tree of Json object. The method can be executed if the xReady output on the IIOT_JsonDecoder_1 function block instance is TRUE. Positive edge on xEN input parameter starts the method execution once.

The method Get_value requires as a parameter the complete JSON key path name of JSON element, provided as a byte array of characters.

The following structure of JSON key path name is required:
- The byte array must be null-terminated (x0).
- Each JSON object (also nested) will be separate via "." (Dot) character. Since JSON data is contained in at least one JSON object, the key path name must start with a period ".".

The following structure of JSON key path name is required:

- The byte array must be null-terminated (x0).
- Each JSON object (also nested) will be separate via "." (Dot) character. Since JSON data is contained in at least one JSON object, the key path name must start with a period ".".
- If the key name includes an array, the array will be zero-based and array values are accessed using the element number in square brackets "[]".

If you are not sure whether JSON key path name has the requested JSON value, you can activate the x_GetTree input of IIOT_JsonDecoder_1 function block before executing. In this case you will find the complete JSON tree with JSON key path names and associated values in the file /opt/plcnext/JsonValueTree.json on the plcnext target.

As result the method Get_value store the JSON value and JSON type in provided variable udtValue of structure JSON_UDT_VALUE.

NOTE that this method is not thread-safe. Ensure that multiple threads/tasks are not accessing a JSON object instance concurrently at the same time.

Access to the JSON object instance must be serialized (using multiple instances of this method in one program) or synchronized between programs executing in different ESM-Tasks. Otherwise an inconsistent state can result.

7.4.1 Input parameters

Name	Type	Description
xEN	BOOL	Enables the method.
anyJsonKey	ANY	The complete JSON path name of requested JSON element.

7.4.2 Output parameter

Name	Type	Description
udtValue	JSON_UDT_VALUE	JSON value as IEC 61131-3 elementary and associated JSON data type.
udtStatusMethod	JSON_UDT_STATUS	Diagnostics information.

7.4.3 Return value

Type	Description
BOOL	FALSE: The method is not executed, the parameters are incomplete/invalid. TRUE: The method is executed, the parameters are complete/valid.

7.4.4 Reference

This method is based on the public member functions of Json::Value Class in the jsoncpp C++ Library.

7.5 Print_to_buf

The Print_to_buf method stores the full tree of JSON object rooted by IIOT_JsonDecoder_1 function block into the provided byte array buffer.

After method call, the byte array buffer contents JSON data coded in UTF8.

The method can be executed if the xReady output on the IIOT_JsonDecoder_1 function block instance is TRUE. Positive edge on xEN input parameter starts the method execution once.

NOTE that this method is not thread-safe. Ensure that multiple threads/tasks are not accessing a Json object instance concurrently at the same time.
Access to the Json object instance must be serialized (using multiple instances of this method in one program) or synchronized between programs executing in different ESM-Tasks. Otherwise an inconsistent state can result.

7.5.1 Input parameters

Name	Type	Description
xEN	BOOL	Enables the method.

7.5.2 Output parameter

Name	Type	Description
anyJsonBuffer	ANY	The byte buffer for storing the JSON object.
anyBufCnt	ANY	Copied count of bytes as a 32-bit signed integer.
udtStatusMethod	JSON_UDT_STATUS	Diagnostics information.

7.5.3 Return value

Type	Description
BOOL	FALSE: The method is not executed, the parameters are incomplete/invalid. TRUE: The method is executed, the parameters are complete/valid.

7.6 Print_to_file

The Print_to_file method stores the full tree of Json object rooted by IIOT_JsonDecoder_1 function block into the specified JSON file.

The method can be executed if the xReady output on the IIOT_JsonDecoder_1 function block instance is TRUE. Positive edge on xEN input parameter starts the method execution once.

NOTE that this method is not thread-safe. Ensure that multiple threads/tasks are not accessing a Json object instance concurrently at the same time.
Access to the Json object instance must be serialized (using multiple instances of this method in one program) or synchronized between programs executing in different ESM-Tasks. Otherwise an inconsistent state can result.

7.6.1 Input parameters

Name	Type	Description
xEN	BOOL	Enables the method.
strJsonPath	STRING	The path of JSON file name with the end suffix ".json" .

7.6.2 Output parameter

Name	Type	Description
udtStatusMethod	JSON_UDT_STATUS	Diagnostics information.

7.6.3 Return value

Type	Description
BOOL	FALSE: The method is not executed, the parameters are incomplete/invalid. TRUE: The method is executed, the parameters are complete/valid.

7.7 Diagnostics

wDiagCode	Source of diagnostics information
0x0000	Function block is deactivated
0x8000	Function block is in regular operation
0xC340	FILE_OPEN error
0xC341	FILE_CLOSE error
0xC342	FILE_READ error
0xC343	FILE_WRITE error
0xC520	Error of an internal used FB. This FB is instantiated in the current FB
0xC110	Invalid input or inout parameter

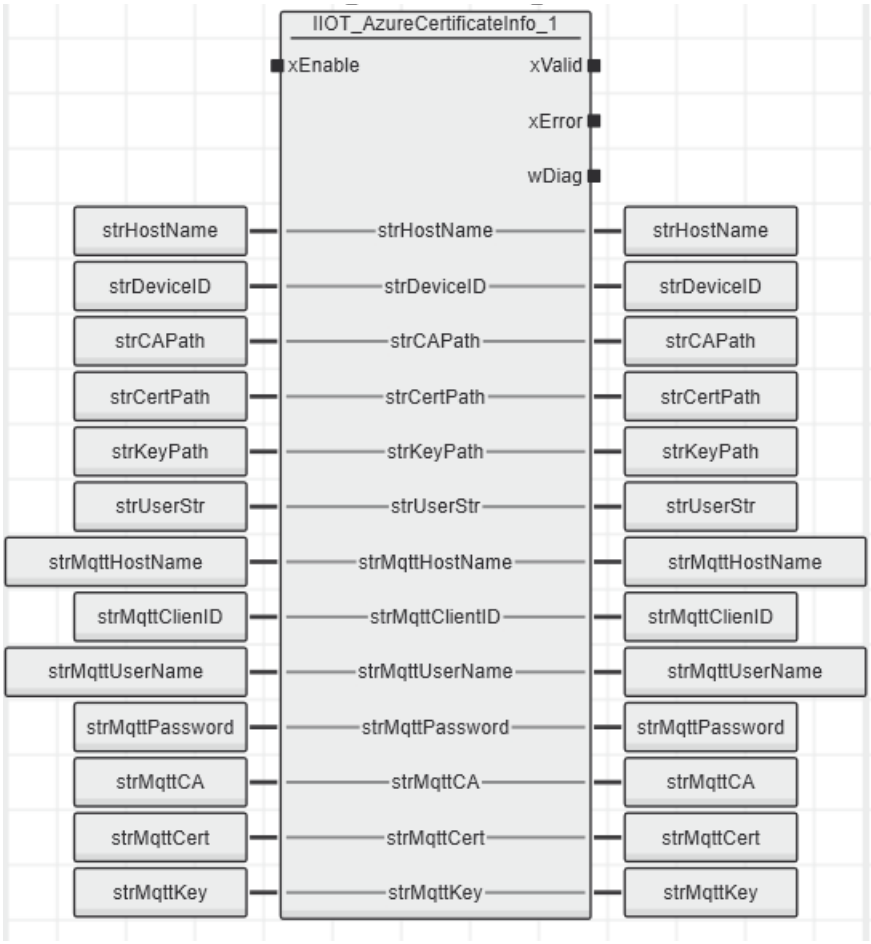
wAddDiagCode	Description
0x0000	Not additional information exist.
0x0001	File is Empty.
0x0002	File is not Valid.
0x0003	File size Error.
0x0004	Invalid Json Key.
0x0005	Buffer size Error.
0x0006	Type is not Elementary.

Additional information on method failures or other unexpected behaviour may be available in the Output.log file in the PLC directory /opt/plcnext/logs.

Instructions on how to read and copy files from the PLC are available in the PLCnext Technology Info Centre.

8 IIOT_AzureCertificateInfo_1

8.1 Function block call



8.2 Input parameters

Name	Type	Description
xEnable	BOOL	Rising edge: Enable the function block. FALSE: Disable the function block.

8.3 Output parameters

Name	Type	Description
xValid	BOOL	TRUE: Configuration is valid. FALSE: Invalid configuration.
xError	BOOL	TRUE: Configuration failure detected. For details refer to wDiag.
wDiag	WORD	Diagnosis code. Refer to diagnostic table.

8.4 Inout parameters

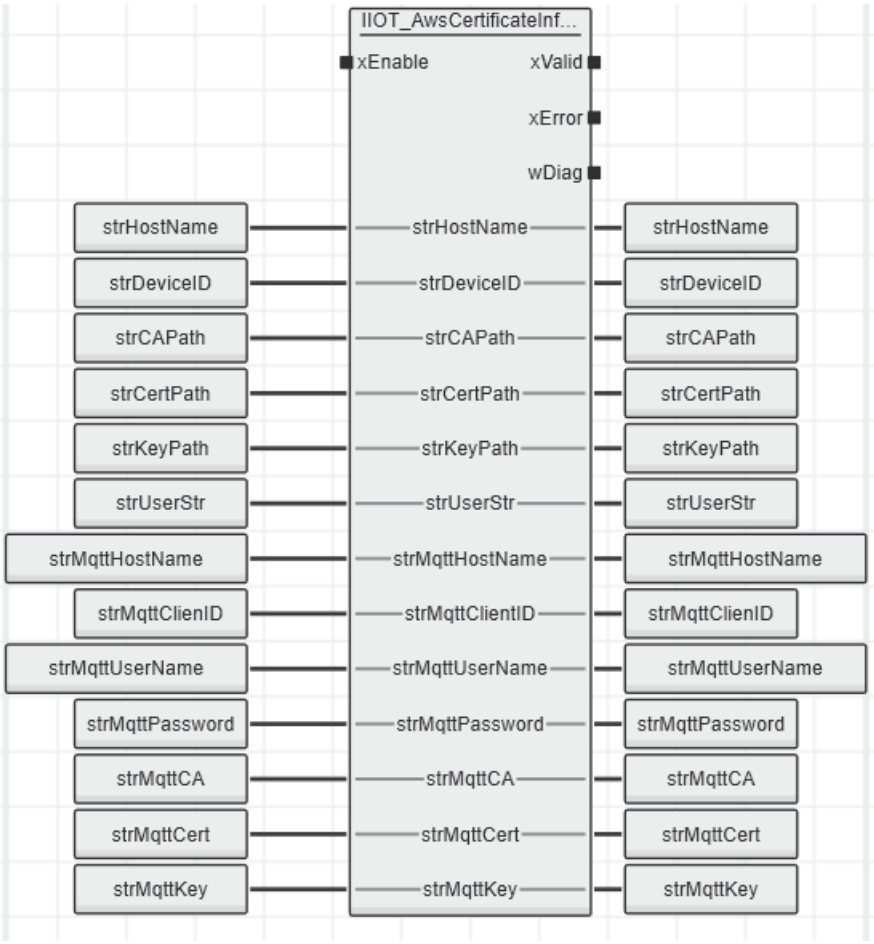
Name	Type	Description
strHostName	STRING or STRING[]	Used as input Hostname of the azure iot hub instance
strDeviceID	STRING or STRING[]	Used as input Device ID of the device created in iot hub
strCAPath	STRING or STRING[]	Used as input Path of the CA file stored in plcnext device. e.g /opt/plcnext/certs/your_ca.pem
strCertPath	STRING or STRING[]	Used as input Path of the device certificate file stored in plcnext device. e.g /opt/plcnext/certs/your_certificate.pem
strKeyPath	STRING or STRING[]	Used as input Path of the device certificate corresponding key file which stored in plcnext device. e.g /opt/plcnext/certs/your_key.pem
strUserStr	STRING or STRING[]	Used as input This input is reserved for future adaption. Currently just feed an empty string to it.
strMqttHostName	STRING or STRING[]	Used as output This output is generated for IIOT_MqttClient_1. strServ-erUri
strMqttClientID	STRING or STRING[]	Used as output This output is generated for IIOT_MqttClient_1. strClientId
strMqttUserName	STRING or STRING[]	Used as output This output is generated for IIOT_MqttClient_1.Connect. strUserName
strMqttPassword	STRING or STRING[]	Used as output This output is generated for IIOT_MqttClient_1.Connect. strPassword Due to the reason that current implementation only sup-port the encryption based on X.509 certificates, and for this method, the mqtt password is not necessary, we can just create a string variable will null default value, and feed it to this function block.
strMqttCA	STRING or STRING[]	Used as output This output is generated for IIOT_MqttClient_1.Connect. udtOptions.udtSsl.strTrustStore
strMqttCert	STRING or STRING[]	Used as output This output is generated for IIOT_MqttClient_1.Connect. udtOptions.udtSsl.strKeyStore
strMqttKey	STRING or STRING[]	Used as output This output is generated for IIOT_MqttClient_1.Connect. udtOptions.udtSsl. strPrivateKey

8.5 Diagnosis

wDiagCode	Source of diagnostics information
0x0001	Some input variables or inoutput variables which will be used as input is null string
0x0002	Certificate concerned files do not exist in the input path.

9 IIOT_AwsCertificateInfo_1

9.1 Function block call



9.2 Input parameters

Name	Type	Description
xEnable	BOOL	Rising edge: Enable the function block. FALSE: Disable the function block.

9.3 Output parameters

Name	Type	Description
xValid	BOOL	TRUE: Configuration is valid. FALSE: Invalid configuration.
xError	BOOL	TRUE: Configuration failure detected. For details refer to wDiag.
wDiag	WORD	Diagnosis code. Refer to diagnostic table.

9.4 InOut parameters

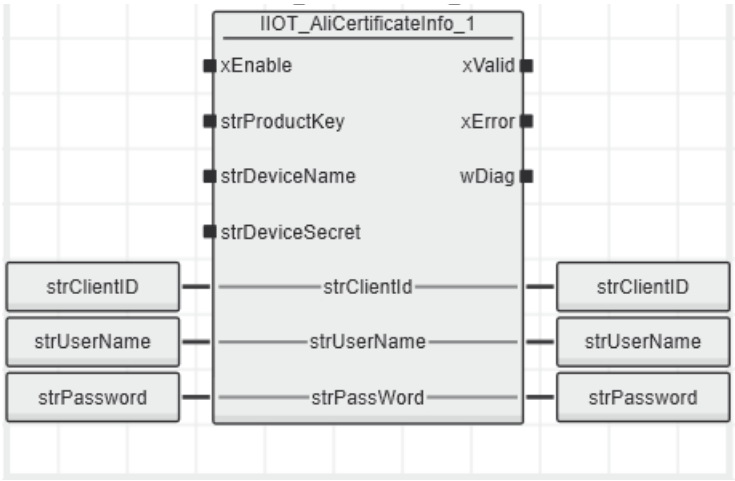
Name	Type	Description
strHostName	STRING or STRING[]	Used as input Hostname of the aws iot hub instance
strDeviceID	STRING or STRING[]	Used as input Device ID of the device created in iot hub
strCAPath	STRING or STRING[]	Used as input Path of the CA file stored in plcnext device. e.g /opt/plcnext/certs/your_ca.pem
strCertPath	STRING or STRING[]	Used as input Path of the device certificate file stored in plcnext device. e.g /opt/plcnext/certs/your_certificate.pem
strKeyPath	STRING or STRING[]	Used as input Path of the device certificate corresponding key file which stored in plcnext device. e.g /opt/plcnext/certs/your_key.pem
strUserStr	STRING or STRING[]	Used as input This input is reserved for future adaption. Currently just feed an empty string to it.
strMqttHostName	STRING or STRING[]	Used as output This output is generated for IIOT_MqttClient_1. strServ-erUri
strMqttClientID	STRING or STRING[]	Used as output This output is generated for IIOT_MqttClient_1. strClientId
strMqttUserName	STRING or STRING[]	Used as output This output is generated for IIOT_MqttClient_1.Connect. strUserName
strMqttPassword	STRING or STRING[]	Used as output This output is generated for IIOT_MqttClient_1.Connect. strPassword Due to the reason that current implementation only sup-port the encryption based on X.509 certificates, and for this method, the mqtt password is not necessary, we can just create a string variable will null default value, and feed it to this function block.
strMqttCA	STRING or STRING[]	Used as output This output is generated for IIOT_MqttClient_1.Connect. udtOptions.udtSsl.strTrustStore
strMqttCert	STRING or STRING[]	Used as output This output is generated for IIOT_MqttClient_1.Connect. udtOptions.udtSsl.strKeyStore
strMqttKey	STRING or STRING[]	Used as output This output is generated for IIOT_MqttClient_1.Connect. udtOptions.udtSsl. strPrivateKey

9.5 Diagnosis

wDiagCode	Source of diagnostics information
0x0001	Some input variables or inoutput variables which will be used as input is null string
0x0002	Certificate concerned files do not exist in the input path.

10 IIOT_AliCertificateInfo_1

10.1 Function block call



10.2 Input parameters

Name	Type	Description
xEnable	BOOL	Rising edge: Enable the function block. FALSE: Disable the function block.
strProductKey	STRING	One of the verification information of the device under the Ali Cloud platform: Product Key
strDeviceName	STRING	One of the verification information of the device under the Ali Cloud platform: Device Name
strDeviceSecret	STRING	One of the verification information of the device under the Ali Cloud platform: Device Secret

10.3 Output parameters

Name	Type	Description
xValid	BOOL	TRUE: Configuration is valid. FALSE: Invalid configuration.
xError	BOOL	TRUE: Configuration failure detected. For details refer to wDiag .
wDiag	WORD	Diagnosis code. Refer to diagnostic table.

10.4 InOut parameters

Name	Type	Description
strClientId	Any	Used as output This output is generated for IIOT_MqttClient_1. strClientId

Name	Type	Description
strUserName	Any	Used as output This output is generated for IIOT_MqttClient_1.Connect.strUserName
strPassWord	Any	Used as output This output is generated for IIOT_MqttClient_1.Connect.strPassword

10.5 Diagnosis

wDiagCode	Source of diagnostics information
0x0001	Some input variables which will be used as input is null string or too long.

11 Known issues

- If a DNS name is used instead of an IP address as the strServerUri value on the IIOT_MqttClient_1 function block, then if at any point the DNS name cannot be resolved (e.g. if the network cable is disconnected), then the Paho library will block and the Task watchdog will be triggered. This issue can be tracked in the Paho MQTT C project on Github.

- The Automatic Reconnect feature does not work reliably. This results from a known issue with the PLCnext Runtime, which is scheduled to be fixed in a future firmware release. It is currently recommended that this feature not be used in production applications.

- In some circumstances, the following error message can appear in the Output.log file when the PLCnext Runtime is shut down:

  ```
  ERROR - MQTT exception occurred in Finalize:
  Code=0, Message=Disconnected
  ```

  This error message should be ignored.

12 Appendix

12.1 MQTT_UDT_CONNECT_OPTIONS

Holds the set of options that control how the client connects to a server. MQTT_UDT_CONNECT_OPTIONS is a STRUCT containing the following elements:

Name	Type	Default value	Description
xUseSslOptions	BOOL	FALSE	If the connection uses SSL, set this to TRUE.
udtSsl	MQTT_UDT_SSL_OPTIONS		The SSL options to use for the connection.
xUseWillOptions	BOOL	FALSE	If the application makes use of the Last Will and Testament feature, set this to TRUE.
udtWill	MQTT_UDT_WILL_OPTIONS		The Last Will and Testament options to use for the connection.
diConnectTimeout	DINT	30	The time interval in seconds to allow a connect to complete.
diKeepAliveInterval	DINT	60	The keep alive interval in seconds[1].
diMaxInflight	DINT	65535	The maximum number of messages that can be in-flight simultaneously.
diMqttVersion	DINT	0	The MQTT version to use for the connection[2].
diMinRetryInterval	DINT	1	The minimum retry interval in seconds. Doubled on each failed retry[3].
diMaxRetryInterval	DINT	60	The maximum retry interval in seconds. The doubling stops here on failed retries[3].
xAutomaticReconnect	BOOL	FALSE	Enable or disable automatic reconnects[4]. PLEASE REFER TO THE KNOWN ISSUE RELATED TO THIS FEATURE
xCleanSession	BOOL	TRUE	Whether the server should forget state for the client across reconnects[4].

1. The “keep alive” interval, measured in seconds, defines the maximum time that should pass without communication between the client and the server. The client will ensure that at least one message travels across the network within each keep alive period. In the absence of a data-related message during the time period, the client sends a very small MQTT “ping” message, which the server will acknowledge. The keep alive interval enables the client to detect when the server is no longer available without having to wait for the long TCP/IP timeout.
2. MQTT version options:
   - 0 = default: start with 3.1.1, and if that fails, fall back to 3.1
   - = only try version 3.1
   - = only try version 3.1.1
3. If the connection to the server is unexpectedly lost and xAutomaticReconnect = TRUE, then automatic reconnection attempts will be made at time intervals defined by the parameters

diMinRetryInterval and diMaxRetryInterval. Once the time interval reaches or exceeds diMaxRetryInterval, reconnection attempts will continue indefinitely at this rate. Users are also free to make their own reconnection attempts at any time, using the Connect method.

4. The “clean session” setting controls the behaviour of both the client and the server at connection and disconnection time. The client and server both maintain session state information. This information is used to ensure “at least once” and “exactly once” delivery, and “exactly once” receipt of messages. Session state also includes subscriptions created by an MQTT client. You can choose to maintain or discard state information between sessions. When xCleanSession is TRUE, the state information is discarded at connect and disconnect. wSetting xCleanSession to FALSE keeps the state information. When you connect an MQTT client application with Connect, the client identifies the connection using the client identifier and the address of the server. The server checks whether session information for this client has been saved from a previous connection to the server. If a previous session still exists, and xCleanSession=TRUE, then the previous session information at the client and server is cleared. If xCleanSession=FALSE, the previous session is resumed. If no previous session exists, a new session is started.

Default values are the same as those defined in the Paho MQTT C Client Library.

This type is based on the connect_options class in the Paho MQTT C++ client library. (additional source: Eclipse Paho MQTT C Client Library documentation)

12.2 MQTT_UDT_SSL_OPTIONS

MQTT_UDT_SSL_OPTIONS defines the settings to establish an SSL/TLS connection using the OpenSSL library. It covers the following scenarios:

- Server authentication: The client needs the digital certificate of the server. It is included in a store containting trusted material (also known as “trust store”).
- Mutual authentication: Both client and server are authenticated during the SSL handshake. In addition to the digital certificate of the server in a trust store, the client will need its own digital certificate and the private key used to sign its digital certificate stored in a “key store”.
- Anonymous connection: Both client and server do not get authenticated and no credentials are needed to establish an SSL connection. Note that this scenario is not fully secure since it is subject to man-in-the- middle attacks.

To enable SSL for a specific client, the xUseSslOptions field is set to TRUE in the MQTT_UDT_CONNECT_OPTIONS structure used in the Connect call that connects the client to the server. The xUseSslOptions field can be set to FALSE an SSL connection is not required.

MQTT_UDT_SSL_OPTIONS is a STRUCT containing the following elements:

Name	Type	Default value	Description
strTrustStore	STRING	’’	The filename containing the public digital certificates trusted by the client. Pass an empty string ’’ if not required.
strKeyStore	STRING	’’	The filename containing the public certificate chain of the client. Pass an empty string ’’ if not required.
strPrivateKey	STRING	’’	The filename containing the client’s private key. Pass an empty string ’’ if not required.
strPrivateKeyPassword	STRING	’’	The password to load the client’s privateKey (if encrypted). Pass an empty string ’’ if not required.
strEnabledCipherSuites	STRING	’’	The list of cipher suites that the client will present to the server during the SSL handshake[1]. Pass an empty string ’’ if not required.
xEnableServerCertAuth	BOOL	FALSE	Enable verification of the server certificate.

1. The list of cipher suites that the client will present to the server during the SSL handshake. For a full explanation of the cipher list format, please see the OpenSSL on-line documentation. If this setting is ommitted, its default value will be “ALL”, that is, all the cipher suites - excluding those offering no encryption - will be considered.

This type is based on the ssl_options class in the Paho MQTT C++ client library. (additional source: Eclipse Paho MQTT C Client Library documentation)

12.3 MQTT_UDT_WILL_OPTIONS

MQTT_UDT_WILL_OPTIONS defines the MQTT “Last Will and Testament” (LWT) settings for the client.

In the event that a client unexpectedly loses its connection to the server, the server publishes the LWT message to the LWT topic on behalf of the client. This allows other clients (subscribed to the LWT topic) to be made aware that the client has disconnected.

To enable the LWT function for a specific client, the xUseWillOptions field is set to TRUE in the MQTT_UDT_CONNECT_OPTIONS structure used in the Connect call that connects the client to the server. The xUseWillOptions field can be set to FALSE if the LWT function is not required.

MQTT_UDT_WILL_OPTIONS is a STRUCT containing the following elements:

Name	Type	Default value	Descrip-tion
strTopic	STRING	‘’	The LWT message is published to this topic.
strPay-load	STRING	‘’	The message that is published to the Will Topic.
diQos	DINT	0	The message Quality of Service[1].
xRetained	BOOL	FALSE	Tell the broker to keep the LWT message after send to subscribers.

1. There are three levels of Quality of Service:
   - Fire and forget - the message may not be delivered.
   - At least once - the message will be delivered, but may be delivered more than once in some circumstances.
   - Once and one only - the message will be delivered exactly once.

This type is based on the will_options class in the Paho MQTT C++ client library. (additional source: Eclipse Paho MQTT C Client Library documentation)

12.4 MQTT_UDT_MESSAGE_INFO

Contains information about an MQTT message.

MQTT_UDT_MESSAGE_INFO is a STRUCT containing the following elements:

Name	Type	Default value	Description
udiLength	UDINT	0	The number of bytes in the message payload.
diQos	DINT	0	The message Quality of Service.
xDuplicate	BOOL	FALSE	Whether this message might be a duplicate of one which has already been received[1].
xRetained	BOOL	FALSE	Whether the message should be, or was, retained by the broker.[2].

1. xDuplicate is only meaningful when receiving messages with Quality of Service = 1. When true, the client application should take appropriate action to deal with the potentially duplicated message.
2. The xRetained flag serves two purposes depending on whether the message it is associated with is being published or received:
   a) xRetained = TRUE
      - For messages being published, a TRUE setting indicates that the MQTT server should retain a copy of the message. The message will then be transmitted to new subscribers to a topic that matches the message topic.
      - For subscribers registering a new subscription, the flag being TRUE indicates that the received message is not a new one, but one that has been retained by the MQTT server.

   b) xRetained = FALSE

      - For publishers, this indicates that this message should not be retained by the MQTT server.
      - For subscribers, a FALSE setting indicates this is a normal message, received as a result of it being published to the server.

This type is based on the message class in the Paho MQTT C++ client library.

12.5 MQTT_UDT_DIAGNOSTICS

Contains diagnostics information.

MQTT_UDT_DIAGNOSTICS is a STRUCT containing the following elements:

Name	Type	Default value	Description
wDiagCode	WORD	0x00	A code representing the method that generated the diagnostics information.
wAddDiagCode	WORD	0x00	A code representing the diagnostics information.
strDiagMsg	STRING	''	A human-readable string containing diagnostics information.

12.6 MQTT_EN_ASYNC_ACTION

A selection of asynchronous MQTT actions.

MQTT_EN_ASYNC_ACTION is an enumeration containing the following options:

Name	Value
Connect	0
Subscribe	1
Unsubscribe	2
Disconnect	3

12.7 JSON_EN_DATA_TYPE

JSON_UDT_VALUE is an enumeration describing a type of JSON value:

Name	Value	Description
ERR_KEY_VALUE	0	The ERR_KEY_VALUE indicates an incompatible or exceeded value.
STRING_VALUE	1	The element is stored as STRING value.
LINT_VALUE	2	The element is stored as LINT value.
ULINT_VALUE	3	The element is stored as ULINT value.
LREAL_VALUE	4	The element is stored as LREAL value.
BOOL_VALUE	5	The element is stored as BOOL value.
JSON_NULL	6	The element is stored as NULL value.
NO_JSON_TYPE	7	The element is not founded or stored as not valid Json type.

12.8 JSON_UDT_VALUE

JSON_UDT_VALUE is a STRUCT containing the value and type of json element:

Name	Type	Default value	Description
strValue	STRING	''	A sealed class type representing Unicode character strings.
xValue	BOOL	FALSE	A simple type representing Boolean values of true or false.
liValue	Int64	0	An integral type representing signed 64-bit integers with values between -9223372036854775808 and 9223372036854775807.
uliValue	UInt64	0	An integral type representing unsigned 64-bit integers with values between 0 and 18446744073709551615.
lrValue	Double	0.0	A floating point type representing values ranging from approximately 5.0 x 10 -324 to 1.7 x 10 308 with a precision of 15-16 digits.
enType	JSON_EN_DATA_TYPE	NO_JSON_TYPE	An enumeration describing a type of JSON value.

12.9 JSON_UDT_STATUS

Contains diagnostics information.

JSON_UDT_STATUS is a STRUCT containing the following elements:

Name	Type	Default value	Description
xExecuted	BOOL	FALSE	TRUE: Command is executed.
XDone	BOOL	FALSE	TRUE: Execution is done.
XError	BOOL	FALSE	TRUE: An error has occurred. The DiagCode and AddDiagCode parameters can be used for precise error analysis.
WDiagCode	WORD	0x00	A code representing the method that generated the diagnostics information.
WAddDiagCode	WORD	0x00	A code representing the diagnostics information.
StrDiagMsg	STRING	’’	A human-readable string containing diagnostics information.

13 Support

For technical support please contact your local PHOENIX CONTACT agency

at https://www.phoenixcontact.com

Owner:

Phoenix Contact (Nanjing) Smart Technology and Solutions Co. Ltd.
No.36 Phoenix Road, Jiangning Development Zone, Nanjing,
PC:21110 China