MANUAL VERSION 2.0.0

FEBRUARY 22, 2026

MQTT CONNECTOR PROFESSIONAL 2

SW MANUAL – URCAP V.2.0.0

Developer guide

4EACH

WWW.4EACH.CZ

CONTENTS

RELEASE NOTES

This version brings a whole new approach which is more user-friendly and brings many benefits to customers:

- **Multi broker connection** support
- **Wide range of broker configuration** parameters
- Topic publishing and subscribing sum up into **Tags.**
- **New program nodes** for publishing and getting the subscribe values.
- Support **various TLS connection options**.
- **Publishing of robot diagnostic data**.

URCAP NODE STRUCTURE

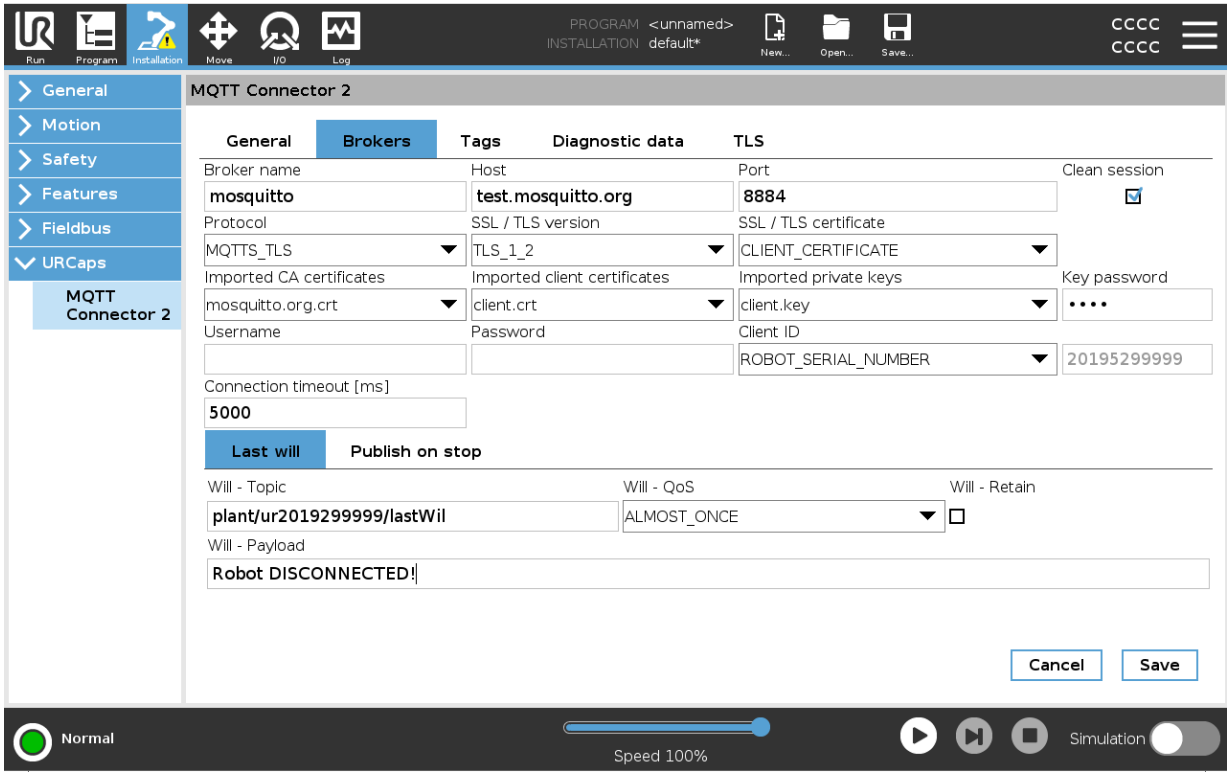
INSTALLATION NODE

The installation node has 4 new tabs:

- In the ***General*** tab are Logging settings and license information. If your license is not valid, the URCAP will not be possible to use.

BROKERS TAB

In the Brokers tab, you configure your brokers. There are many possible options as you can see in the picture below.



In the MQTT Connector Professional, there is various possibility of secure connection using TLS. The highest available is TLS 1.2. If the user chooses HIGHEST_POSSIBLE, it is up to the client and broker to agree on the highest possible TLS version they both are capable of. The certificate version has 3 options:

- CA_SIGNED_SERVER_CERTIFICATE (0 certificates requested)
- CA_CERTIFICATE_ONLY (CA certificate requested)
- CLIENT_CERTIFICATE (CA certificate + client certificate + private key + password (optional))

Client ID is the identification of the MQTT broker. A broker can handle only one client with unique identification, if there is another MQTT client with the same client ID connected to the broker, other clients will be disconnected. MQTT Connector Professional allows to define a client ID with the following settings:

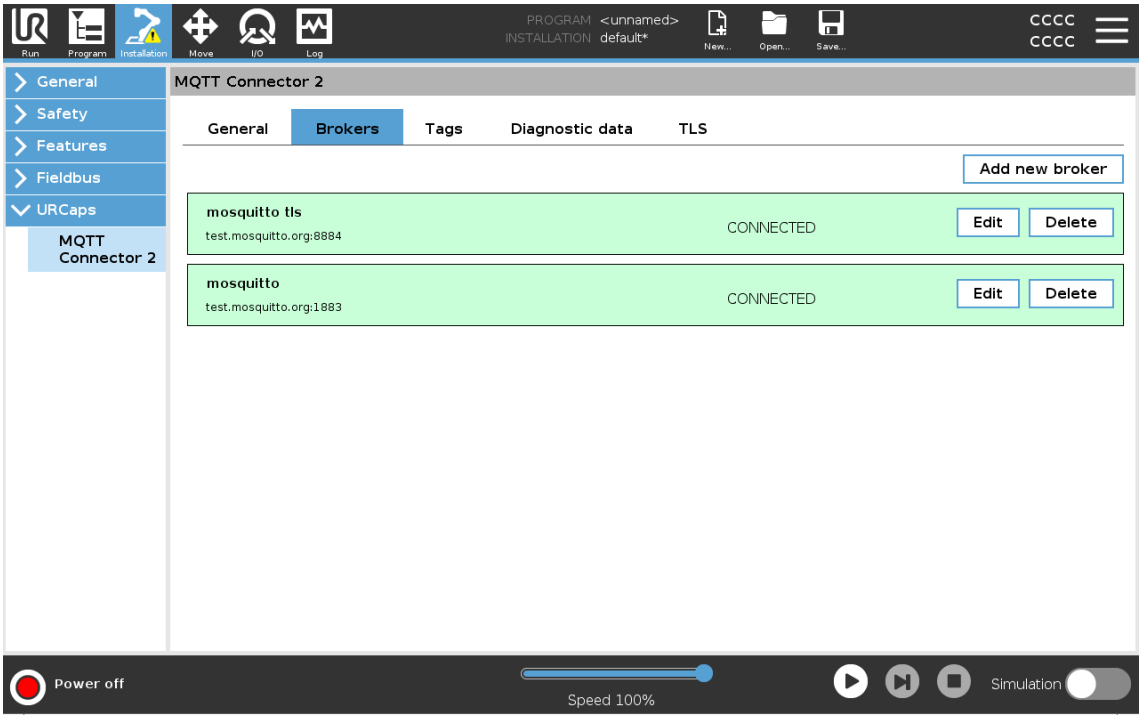
- **Automatically generated**: A new Client ID is generated during the MQTT_INITIALIZE and used for all the communication with the broker.
- **Robot serial number:** The robot serial number is used as a client ID.
- **User defined:** The user text in the field will be used as a client ID

Note: If you transfer the installation file from another robot, please set your ClientID again. The robot serial number is not refreshed automatically.

Connection timeout is a period when the client tries to connect to the broker. This state is shown as “CONNECTING…”.

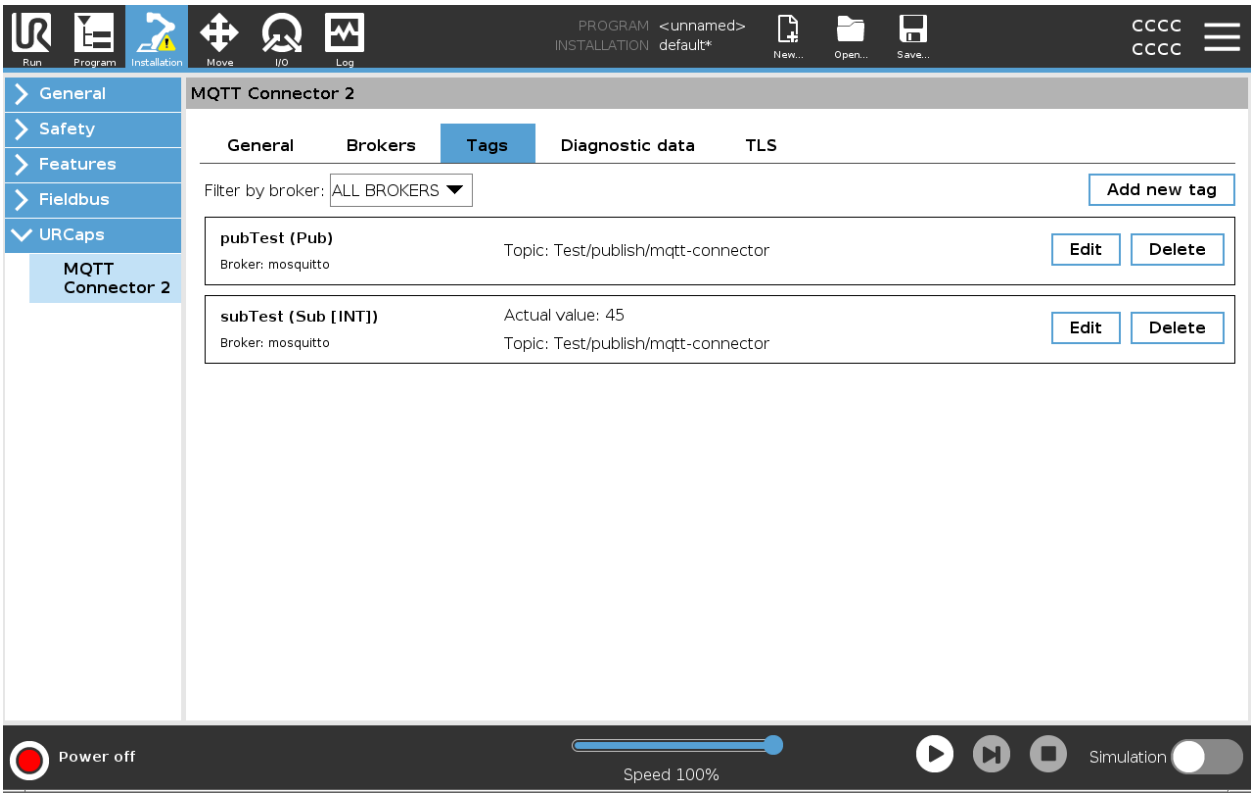
The last section is the last-will and publish on stop configuration.

When the broker is configured and saved, the info panel will show the connection status. If there is any problem, you can see the error message and it is always possible to edit the configuration. When you decide to delete the broker, all connected features (such as Tags or Diagnostic data configuration) will also be deleted. Before it happens, you will be warned by a pop-up window about this event.



TAGS TAB

In the ***Tags*** tab, you can define a topic you wish to use in the program to publish or subscribe to and its related parameters. Each configuration is wrapped in the tag. You can reference this configuration in the program nodes. Moreover, if you define a subscribe tag, you will be able to see the actual value from the broker.



THE DIAGNOSTIC DATA TAB

This tab brings very valuable feature. The Universal Robots provides an interface to get the robot diagnostic data. The MQTT Connector 2 gives the customer an option to publish this robot's diagnostic data to the requested broker within the requested period.

The customer selects which data he would like to publish. Every selected data has its own topic.

If the topic prefix contains this string **@thisRobot**, the URCap replaces this string with robots serial number.

The topic is created by *SelectedPrefix/DataStructNameFromUR/DataNameFromUR.*

- topic names [gui name: topic name]:
  - ADDITIONAL_INFO: additional_info
    - TP Button State: tpButtonState
    - Freedrive Button Pressed: freeDriveButtonPressed
    - Freedrive Button Enabled: freeDriveButtonEnabled
    - IO Enabled Freedriver: IoEnabledFreeDriver
    - Reserved: reserved
  - CALIBRATION_DATA: calibration_data
    - Fx: fx
    - Fy: fy
    - Fz: fz
    - Frx: frx
    - Fry: fry
    - Frz: frz
  - CARTESIAN_INFO: cartesian_info
    - X: x
    - Y: y
    - Z: z
    - Rx: rx
    - Ry: ry
    - Rz: rz
    - TCP Offset X: tcpOffsetX
    - TCP Offset Y: tcpOffsetY
    - TCP Offset Z: tcpOffsetZ
    - TCP Offset Rx: tcpOffsetRx
    - TCP Offset Ry: tcpOffsetRy
    - TCP Offset Rz: tcpOffsetRz
  - FORCE_MODE_DATA: force_mode_data
    - Fx: fx
    - Fy: fy
    - Fz: fz
    - Frx: frx
    - Fry: fry
    - Frz: frz
    - Robot Dexterity: robotDexterity

- JOINTS_DATA: joints_data (*Every selected joint has its parameters. The parameters are the same for each joint. The user can select the joints, and their parameters will be published in separate sub-topics.*)
  - Base: base
    - q_actual
    - q_target
    - qd_actual
    - i_actual
    - v_actual
    - t_motor
    - t_micro
    - jointMode
  - Shoulder: shoulder
  - Elbow: elbow
  - Wrist1: wrist1
  - Wrist2: wrist2
  - Wrist3: wrist3
- ROBOT_MODE_DATA: robot_mode_data
  - Package Length: packageLen
  - Package Type: packageType
  - Timestamp: timestamp
  - Real Robot Connected: isRealRobotConnected
  - Real Robot Enabled: isRealRobotEnabled
  - Robot Power On: isRobotPoweOn
  - Emergency Stopped: isEmergencyStopped
  - Protective Stopped: isProtectiveStopped
  - Program Running: isProgramRunning
  - Program Paused: isProgramPaused
  - Robot Mode: robotMode
  - Control Mode: controlMode
  - Target Speed Fraction: targetSpeedFraction
  - Speed Scaling: speedScaling
  - Target Speed Fraction Limit: targetSpeedFractionLimit
  - Reserved: reserved
- MASTERBOARD_DATA: masterboard_data
  - Package Length: packageLength
  - Package Type: packageTypes
  - Digital Input Bits: digitalInputBits
  - Digital Output Bits: digitalOutputBits
  - Analog Input Range 0: analogInputRange0
  - Analog Input Range 1: analogInputRange1
  - Analog Input 0: analogInput0
  - Analog Input 1: analogInput1
  - Analog Output Domain 0: analogOutputDomain0

        - Analog Output Domain 1: analogOutputDomain1
        - Analog Output 0: analogOutPut0
        - Analog Output 1: analogOutPut1
        - Masterboard Temperature: masterboardTemperature
        - Robot Voltage 48V: robotVoltage48V
        - Robot Current: robotCurrent
        - Master IO Current: masterIoCurrent
        - Safety Mode: safetyMode
        - In Reduced Mode: inReduceMode
        - Euromap 67 Installed: euromap67Installed
        - Euromap Input Bits: euromapInputBits
        - Euromap Output Bits: euromapOutputBits
        - Euromap Voltage: euromapVoltage
        - Euromap Current: euromapCurrent
        - Dummy: dummy
        - Operational Mode Selector Input: operationalModeSelectorInput
        - Three-Position Enabling Device Input
    - SINGULARITY_INFO: singularity_info
        - Singularity Severity: singularitySeverity
        - Singularity Type: singularityType
    - TOOL_DATA: tool_data
        - Analog Input Range 0: analogInput0
        - Analog Input Range 1: analogInput1
        - Analog Input 0: analogInputRange0
        - Analog Input 1: analogInputRange1
        - Tool Voltage 48V: toolVoltage48V
        - Tool Output Voltage: toolOutputVoltage:
        - Tool Current: toolCurrent
        - Tool Temperature: toolTemperature
        - Tool Mode: toolMode
    - TOOL_COMMUNICATION_INFO: tool_communication_info
        - Baud Rate: baudRate
        - Parity: parity
        - RX Idle Characters : rxIdleChars
        - Stop Bits: stopBits
        - Tool Communication Enabled: toolCommIsEnabled
        - TX Idle Characters: txIdleChars
    - TOOL_MODE_INFO: tool_mode_info
        - Digital Output Mode for Output 0: digOutModeOutput0
        - Digital Output Mode for Output 1: digOutModeOutput1
        - Output Mode: outputMode

For more information about the data types and their values you can find here: *ClientInterfaces_Primary.pdf*

TLS TAB

MQTT Connector professional offers secure connection using TLS. In the broker configuration, it is possible to set the TLS configuration. In this tab, you can import and see the imported certificates and private keys. These files are then shown in the broker configuration panel.

PROGRAM NODES

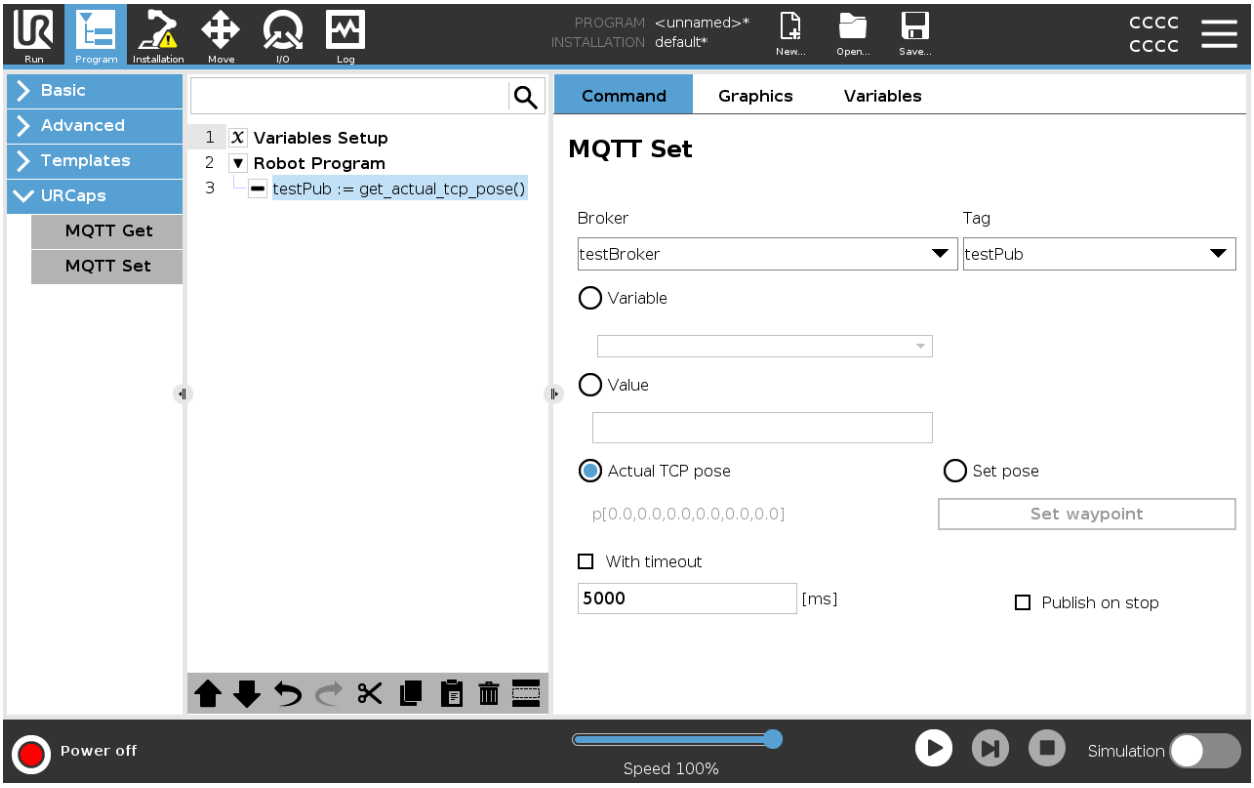
There are 2 program nodes available for the users.

MQTT SET

This program node is for data publishing purposes. The node references the Installation tags. The user chooses the tag and selects the value he wants to publish. There are 4 options:

- Variable — selected variable value will be published.
- Value — Set the value to publish.
- Actual TCP pose — uses a script function to publish an actual TCP pose.
- Set pose —pose set by user

With every MQTT Set node, you can choose the timeout for checking the publishing success. Also, you can choose if this value should be used for “publish on stop” purposes.



MQTT GET

GET program node retrieves a subscribed topic value. So as the MQTT Set node it references the Installation tags configuration. The user selects the subscribe type tag which he would like to get actual value from. In the tag selection bar, the user can see the tag data type. There is an option for the user to choose whether the subscribed value should be assigned to the variable, or it should be used as wait for the value option.

Remember if the parsing of the subscribed value is unsuccessful, the default value configured in the installation is passed instead. You can always see the actual subscribed value in in the Installation node in the *Tags* tab section. Also remember, that the size of the array is always fixed and pre-defined by the default value. If the value from the broker has a different size of the array, it will not be parsed.

UR SCRIPT FUNCTIONS

Apart from the program nodes, there are still various mqtt functions available for the users. All functions are identical as in version 1.8.0.

Every function has added a last optional parameter, the broker's name. It says which broker from installation the user refers to. If the last parameter is not used, it refers to the GENERAL BROKER. This feature is here only for backward compatibility. If there is a new program created, it is strongly recommended to refer to a defined broker.

MQTT_INITIALIZE

initializes connection details using credentials.

It takes the following arguments:

- Hostname – the hostname or IP address of the remote broker [string]
- Port - the network port of the server host to connect to [int]
- User – username to access to remote broker [string]
- Password - password to access to remote broker [string]
- Broker – selected broker name (OPTIONAL) [string]

Note that some brokers will accept anonymous users, even though they provide username and password.

MQTT_INITIALIZE_ANONYMOUS

initializes connection details using anonymous access (without username and password).

It takes the following arguments:

- Hostname – the hostname or IP address of the remote broker [string]
- Port - the network port of the server host to connect to [int]
- Broker – selected broker name (OPTIONAL) [string]

MQTT_SET_MAX_QUEUE

Sets maximum outgoing messages queue length. This is important in case of connection loss, because every queued message is sent after connection is restored. This can cause temporary network overload. Parameter with value 0 sets unlimited queue length. When the queue is full, any further outgoing messages would be dropped.

- Length - maximum outgoing messages queue length [int]

- Broker – selected broker name (OPTIONAL) [string]

MQTT_SET_LAST_WILL

Set a Will to be sent to the broker. If the client (ungracefully) disconnects without calling disconnect(), the broker will publish the message on its behalf.

- Topic - the topic that the will message should be published on [string]
- Message - the message to send as a will [string]
- Qos - the quality of service level to use for the will [int]
- Retained - if set to True, the will message will be set as the “last known good”/retained message for the topic [bool]
- Broker – selected broker name (OPTIONAL) [string]

Note, that this function must be called after the initialize function and before connect function. This function is deprecated. Set the last will in the broker configuration.

MQTT_SET_PUBLISH_ON_STOP

Set a message to be sent to the broker when UR program stops.

- Topic - the topic that the will message should be published on [string]
- Message - the message to send as a will [string]
- Qos - the quality of service level to use for the will [int]
- Retained - if set to True, the will message will be set as the “last known good”/retained message for the topic [bool]
- Broker – selected broker name (OPTIONAL) [string]

The timeout parameter used in previous versions of the product is deprecated. The message is always sent until 10 seconds after the program is stopped.

Note, that this function differs from last will. This function sends specified messages when the robot program changes the state from running to stopped. This can occur by user interaction; emergency stop or other factors. Messages to be send are stored locally, therefore if ungraceful disconnect occurs, no messages can be sent. Message buffer is limited to 100 entries.

TIP: If the user wants to know whether the program is running, it is possible to use the Diagnostic data. In the ROBOT_MODE_DATA struct is the Program Running attribute which can be published.

MQTT_CONNECT

- Broker – selected broker name (OPTIONAL) [string]

Connects the client to a broker, using a default timeout period of **5** seconds.

MQTT_CONNECT_TLS

Connects the client to a broker, using default timeout period of **5** seconds and CA signed server certificate. The connection is established using TLS protocol v 1.2. The certificate must be imported (please see the chapter Installation tab) and the parameter contains the filename with .crt extension without the file path (as displayed in the list of imported certificates on the installation tab).

MQTT_CONNECT_TIMEOUT

Connects the client to a broker, using custom timeout period defined by parameter.

- Timeout - Ensures maximum blocking duration of function call. Time is defined in milliseconds (internally truncated to 100ms steps). [int]
- Broker – selected broker name (OPTIONAL) [string]

MQTT_SUBSCRIBE

Subscribe the client to one topic.

- Topic - subscription topic to subscribe to [string]
- Broker – selected broker name (OPTIONAL) [string]

MQTT_UNSUBSCRIBE

Unsubscribe the client from one topic.

- Topic - subscription topic to unsubscribe from [string]
- Broker – selected broker name (OPTIONAL) [string]

MQTT_UNSUBSCRIBE_ALL

Unsubscribe the client from all topics.

- Broker – selected broker name (OPTIONAL) [string]

MQTT_GET_MESSAGE

Function used to lookup incoming messages buffer. Returns latest received value from given topic.

- Topic – Specifies topic to search for [string]

Note, that this function has different communication interface than other methods. It doesn't return OK status code, but actual message payload. If error is present, error codes are returned.

MQTT_GET_PARSED_MESSAGE

Function used to lookup incoming messages buffer and convert to the URScript native type. Returns latest received typed value from given topic or the default value in case of any issue.

- Topic – Specifies topic to search for [string]
- DefaultValue – variable or constant for default value
- Broker – selected broker name (OPTIONAL) [string]

Note, that this function has a different communication interface than other methods. It does not return the OK status code but the current typed message payload. This function recognizes the type of the default value parameter and tries to parse the incoming MQTT payload.

If an error is present or the payload cannot be parsed, the default value is returned. If the log level is set to "Debug" the specific error code is written to the Universal Robots Log.

Here is the list of MQTT payload examples for each supported UR Script data type:

UR Script type	MQTT payload example
Boolean	True
Integer	78
Float	3.14159265358
Array of Boolean	[True, False, True, False]
Array of Integer	[1921439, 1921440, 1921441, 1921442]
Array of Float	[1.11, 2.22, 3.33]
Pose	[-0.189632, -0.609684, 0.260971, -0.001221, 3.116277, 0.038892]

MQTT PAYLOAD NOTES:

- Do not use comma (,) character as a decimal separator (ie. ~~3,1415~~)
- Try to avoid the GET_PARSED_MESSAGE for STRING variables, use GET_MESSAGE instead (it works, but all recognizable datatypes will be considered as error and payload cannot contain keywords like "NG-CODE", "nan", or "ERROR")
- MQTT payload should never contain the following characters (character sequences): `'|', '<~', '~>'`
- If the size of the array in the payload does not fit with the size of the default value, the function returns NG-CODE-075 (type inequality).

MQTT_GET_JSON_MESSAGE

The function is used to look up variables in the JSON single-level structure. Each JSON structure element can contain one of the data types supported by the MQTT_GET_PARSED_MESSAGE.

Returns the latest received typed value from the given topic, JSON element name, and the default value in case of any issue.

- Topic – Specifies topic to search for [string]
- Element name – Specifies the name of the JSON element to search for [string]
- DefaultValue –constant for default value
- Broker – selected broker name (OPTIONAL) [string]

Example of payload in the test/json_topic topic:

{"processBusy": true, "counterData": 9, "stationName": "Joe Black", "processProgress": 0.9, "roboPose": [-0.326488, -0.700507, 0.252493, -0.001122, 3.119379, 0.035188]}

Example of MQTT_GET_JSON_MESSAGE call

1) mqtt_get_json_message("test/json_topic","processBusy", False) returns True
2) mqtt_get_json_message("test/json_topic","counterData", 0) returns 9
3) mqtt_get_json_message("test/json_topic"," roboPose", [0,0,0,0,0,0]) returns [-0.326488, -0.700507, 0.252493, -0.001122, 3.119379, 0.035188]

MQTT_PUBLISH

This function causes a message to be sent to the broker and subsequently from the broker to any clients subscribing to matching topics, using a default timeout period of 3 seconds. It takes the following arguments:

- Topic - the topic that the will message should be published on [string]
- Message - the message to send as a will [string]
- Qos - the quality of service level to use for the will [int]
- Retained - if set to True, the will message will be set as the “last known good”/retained message for the topic [bool]
- Broker – selected broker name (OPTIONAL) [string]

MQTT_PUBLISH_TIMEOUT

This function causes a message to be sent to the broker and subsequently from the broker to any clients subscribing to matching topics, using a custom timeout period defined by the parameter. It takes the following arguments:

- Topic - the topic that the will message should be published on [string]
- Message - the message to send as a will [string]
- Qos - the quality of service level to use for the will [int]
- Retained - if set to True, the will message will be set as the “last known good”/retained message for the topic [bool]
- Timeout - Ensures maximum blocking duration of the function call. Time is defined in milliseconds, and changes in 100ms increments. [int]
- Broker – selected broker name (OPTIONAL) [string]

JSON DATA PUBLISHING

Sometimes it is useful to group some data together in one mqtt message. MQTT Connector allows data grouping in the one-level structure [name, value]. The structure must be cleared, filled with data, and after, published to one MQTT topic. The JSON serialization will be performed automatically by the MQTT Connector according to all added data types.

Here is a [name, value] structure example and the corresponding mqtt payload:

Name	Value	UR Script type
robotPose		Pose
counterData		Integer
processProgress		Float
stationName		String
processBusy		Boolean

Payload:

{"processBusy": true, "counterData": 9, "stationName": "Joe Black", "processProgress": 0.9, "roboPose": [-0.326488, -0.700507, 0.252493, -0.001122, 3.119379, 0.035188]}

MQTT_JSON_MESSAGE_CLEAR

This function Initializes the one-level (name, value) structure for publishing a group of data in JSON format.

- Broker – selected broker name (OPTIONAL) [string]

MQTT_JSON_MESSAGE_ADD

- Name - the name of the structure element (should be unique in the structure)
- Value – value or the UR Script variable with the value
- Broker – selected broker name (OPTIONAL) [string]

MQTT_PUBLISH_JSON

This function publishes the current content of the single-level structure.

- Topic - the topic that the will message should be published on [string]
- Qos - the quality of service level to use for the will [int]
- Retained - if set to True, the will message will be set as the “last known good”/retained message for the topic [bool]
- Broker – selected broker name (OPTIONAL) [string]

MQTT_PUBLISH_JSON_TIMEOUT

This function publishes the current content of the single-level structure with a defined timeout.

- Topic - the topic that the will message should be published on [string]
- Qos - the quality-of-service level to use for the will [int]
- Retained - if set to True, the will message will be set as the “last known good”/retained message for the topic [bool]
- Timeout - Ensures maximum blocking duration of the function call. Time is defined in milliseconds, and changes in 100ms increments. [int]
- Broker – selected broker name (OPTIONAL) [string]

MQTT_DISCONNECT

Disconnects from the broker cleanly. Using disconnect () does not result in a will message being sent by the broker. Disconnect does not wait for all queued messages to be sent.

DEBUGGING PRACTICE AND ERROR CODES

To ease programming, error troubleshooting, and general ease of use, MQTT Connector uses error codes as return values for most of its functions. This can be helpful in situations where client-broker connectivity is not stable, or during initial setup.

The new function GET_PARSED_MESSAGE always returns the default value in case of any issue. The error code of this function can be found in standard log of UR, if the log-level is set to “Debug” on the installation tab (advanced options). Changing of the log-level must be done before the UR program is started.

Generally recommended usage

- Compare return values against constants or predefined variables using IF statement. Return values should be compared against error codes table.
- Store return values in variables. Even though value is never compared against error codes, it can be helpful to view variable values in Polyscope graphical interface.

ERROR CODE TABLE

URScript function	*Description*	Status	Error code
ALL FUNCTIONS	NOT AUTHORIZED	NG	NG-CODE-001
authorize	SUCCESS	OK	OK-CODE-002
	UNKNOWN ERROR	NG	NG-CODE-003
multi-broker	BROKER WITH THIS NAME HAS NOT BEEN INITIALIZED	NG	NG-CODE-404
ALL FUNCTIONS	*deprecated*	*NG*	*NG-CODE-004*
	DAEMON NOT READY – INTERNAL ERROR***	NG	NG-CODE-005
	CLIENT NOT INITIALIZED – call initialize	NG	NG-CODE-006
initialize	SUCCESS	OK	OK-CODE-010
	FAILED	NG	NG-CODE-011
	FALSE PARAMS	NG	NG-CODE-012
connect	MQTT CONNECT OK	OK	OK-CODE-020
	MQTT SOCKET ERROR (previously: NO ROUTE TO HOST)	NG	NG-CODE-021
	FAILED - timeout	NG	NG-CODE-022
	UNKNOWN ERROR	NG	NG-CODE-023
connect_tls	CA CERTIFICATE NOT FOUND	NG	NG-CODE-024
	CLIENT CERTIFICATE NOT FOUND	NG	NG-CODE-025
	PRIVATE KEY NOT FOUND	NG	NG-CODE-026
	SSL ERROR	NG	NG-CODE-027
	Broker already connected	NG	NG-CODE-028
	+ all codes of connect function		
disconnect	MQTT DISCONNECT OK	OK	OK-CODE-030
	UNKNOWN ERROR	NG	NG-CODE-031
subscribe	SUBSCRIBE OK	OK	OK-CODE-040
	SUBSCRIBE FAILED - already exists	NG	NG-CODE-041
	UNKNOWN ERROR	NG	NG-CODE-042
unsubscribe	UNSUBSCRIBE OK	OK	OK-CODE-050
	UNSUBSCRIBE FAILED - no such subscription	NG	NG-CODE-051
	UNKNOWN ERROR	NG	NG-CODE-052
unsubscribe_all	UNSUBSCRIBE ALL OK	OK	OK-CODE-060
	UNKNOWN ERROR	NG	NG-CODE-061
get_message	*MESSAGE PAYLOAD STRING*	OK	
	NOT YET RECEIVED ANY MESSAGE	NG	NG-CODE-071
	NO SUCH SUBSCRIPTION	NG	NG-CODE-072
	FALSE PARAMS	NG	NG-CODE-073
	UNKNOWN ERROR	NG	NG-CODE-074
get_parsed_message**	*PARSED PAYLOAD VALUE OR DEFAULT*	OK	
	DEFAULT VALUE TYPE AND PAYLOAD TYPE IS NOT EQUAL	NG	NG-CODE-075
	all other NG codes of get_message function		
get_json_message	*PARSED PAYLOAD VALUE OR DEFAULT*	OK	
	all other NG codes of get_parsed_message function		
	The payload is not in the JSON format		NG-CODE-076
	Element not found in the JSON struct		NG-CODE-077
publish, publish_json	SUCCESS	OK	OK-CODE-080
	FAILED	NG	NG-CODE-081
	MESSAGE SEND NOK - broker disconnected	NG	NG-CODE-082
	FALSE PARAMS	NG	NG-CODE-083
	UNKNOWN ERROR	NG	NG-CODE-084
json_message_clear	SUCCESS	OK	OK-JSONLIST-CLEAR
json_message_add	SUCCESS	OK	OK-JSONLIST-ADDED
	UNKNOWN ERROR	NG	NG-JSONLIST-NOT-ADDED
set_max_queued_messages	SUCCESS	OK	OK-CODE-090
	FAILED	NG	NG-CODE-091
set_last_will	SUCCESS	OK	OK-CODE-100

URScript function	*Description*	Status	Error code
	FAILED	NG	NG-CODE-101
set_publish_on_stop	SUCCESS	OK	OK-CODE-110
	FALSE PARAMS	NG	NG-CODE-111
	BUFFER IS FULL	NG	NG-CODE-112
	UNKNOWN ERROR	NG	NG-CODE-113

** These error codes are visible in the UR Log, if the feature “Debug MQTT parsing using the UR Log” is enabled in the Installation tab (Advanced options)
*** This error is thrown if the daemon is not started. In very few cases this error can be invoked during the runtime (this issue is known as NO-CONTROLLER robot error). If the get_message/get_parsed_message is called again, the function will return the correct result without necessity of stopping the robot program.