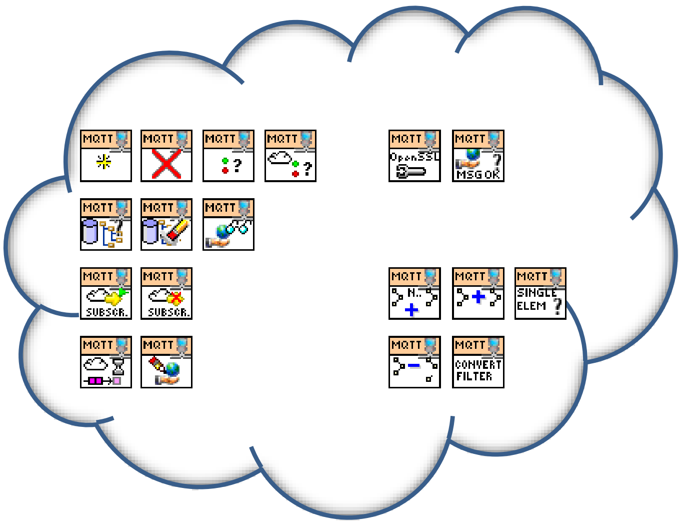
wireflow

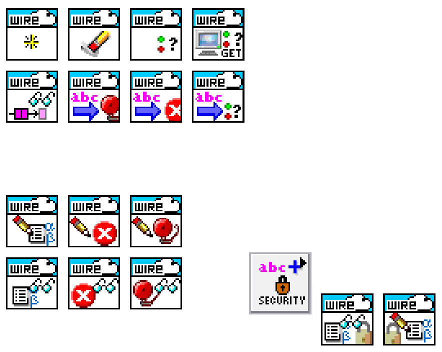
AC0075-001, rev F

WireQueue MQTT User Manual

MQTT



WireQueue



Contents

Support information

Technical support and Product information

www.wireflow.se

WireFlow headquarters

WireFlow AB
Theres Svenssons gata 10
SE-417 55 Göteborg

Important information

Warranty

Copyright

Trademarks

EULA

END-USER LICENSE AGREEMENT FOR

WireFlow WireQueue LabVIEW driver (AC0075)

IMPORTANT PLEASE READ THE TERMS AND CONDITIONS OF THIS LICENSE AGREEMENT CAREFULLY BEFORE CONTINUING WITH THIS PROGRAM DOWNLOAD/INSTALL: WireFlow's End-User License Agreement ("EULA") is a legal agreement between you (either an individual or a single entity) and WireFlow, for the WireFlow software product(s) identified above which may include associated software components, media, printed materials, and "online" or electronic documentation ("SOFTWARE PRODUCT"). By installing, copying, or otherwise using the SOFTWARE PRODUCT, you agree to be bound by the terms of this EULA. This license agreement represents the entire agreement concerning the program between you and WireFlow, (referred to as "licenser"), and it supersedes any prior proposal, representation, or understanding between the parties. If you do not agree to the terms of this EULA, do not download, install or use the SOFTWARE PRODUCT.

The SOFTWARE PRODUCT is protected by copyright laws and international copyright treaties, as well as other intellectual property laws and treaties. The SOFTWARE PRODUCT is licensed, not sold.

1 GRANT OF LICENSE

The SOFTWARE PRODUCT is licensed as follows:

1.1 Installation and Use

WireFlow grants you a personal, non-transferable and non-exclusive right to use the copy of the Software provided with this EULA on your computer running a validly licensed copy of the operating system for which the SOFTWARE PRODUCT was designed.

1.2 Backup Copies

You may also make copies of the SOFTWARE PRODUCT as may be necessary for backup and archival purposes.

1.3 Evaluation Version

For clarity in the case of Trial Licenses, if You do not pay the applicable license fees prior to the conclusion of any applicable Trial Period, you have no right or license, express or implied, to further use the SOFTWARE PRODUCT in any manner thereafter.

2 DESCRIPTION OF OTHER RIGHTS AND LIMITATIONS

2.1 Maintenance of Copyright Notices

You must not remove or alter any copyright notices on any and all copies of the SOFTWARE PRODUCT.

2.2 Distribution

You may not distribute registered copies of the SOFTWARE PRODUCT to third parties. Evaluation versions available for download from WireFlow's websites may be freely distributed.

2.3 Prohibition on Reverse Engineering, Decompilation, and Disassembly

You may not reverse engineer, decompile, or disassemble the SOFTWARE PRODUCT, except and only to the extent that such activity is expressly permitted by applicable law notwithstanding this limitation.

2.4 Rental

You may not rent, lease, or lend the SOFTWARE PRODUCT.

2.5 Support Services

WireFlow may provide you with support services related to the SOFTWARE PRODUCT ("Support Services"). Any supplemental software code provided to you as part of the Support Services shall be considered part of the SOFTWARE PRODUCT and subject to the terms and conditions of this EULA.

2.6 Compliance with Applicable Laws

You must comply with all applicable laws regarding use of the SOFTWARE PRODUCT.

2.7 Export Laws

The export of the SOFTWARE PRODUCT from the country of original purchase may be subject to control or restriction by applicable local law. Licensee is solely responsible for determining the existence and application of any such law to any proposed export and for obtaining any needed authorization. Licensee agrees not to export the SOFTWARE PRODUCT from any country in violation of applicable legal restrictions on such export.

3 TERMINATION

Without prejudice to any other rights, WireFlow may terminate this EULA if you fail to comply with the terms and conditions of this EULA. In such event, you must destroy all copies of the SOFTWARE PRODUCT in your possession.

4 COPYRIGHT

All title, including but not limited to copyrights, in and to the SOFTWARE PRODUCT and any copies thereof are owned by WireFlow or its suppliers. All title and intellectual property rights in and to the content which may be accessed through use of the SOFTWARE PRODUCT is the property of the respective content owner and may be protected by applicable copyright or other intellectual property laws and treaties. This EULA grants you no rights to use such content. All rights not expressly granted are reserved by WireFlow.

4.1 Third party software.

The SOFTWARE PRODUCT may include software under license from third parties (“Third Party Software” and “Third Party License”). Any Third Party Software is licensed to you subject to the terms and conditions of the corresponding Third Party License. Generally, the Third Party License is located in a separate file such as license.txt or a readme file.

5 NO WARRANTIES

WireFlow expressly disclaims any warranty for the SOFTWARE PRODUCT. The SOFTWARE PRODUCT is provided 'As Is' without any express or implied warranty of any kind, including but not limited to any warranties of merchantability, noninfringement, or fitness of a particular purpose. WireFlow does not warrant or assume responsibility for the accuracy or completeness of any information, text, graphics, links or other items contained within the SOFTWARE PRODUCT. WireFlow makes no warranties respecting any harm that may be caused by the transmission of a computer virus, worm, time bomb, logic bomb, or other such computer program. WireFlow further expressly disclaims any warranty or representation to Authorized Users or to any third party.

6 HIGH RISK ACTIVITIES

The SOFTWARE PRODUCT is not fault-tolerant and is not designed, manufactured or intended for use or resale as on-line control equipment in hazardous environments requiring fail-safe performance, such as in the operation of nuclear facilities, aircraft navigation or communication systems, air traffic control, direct life support machines, or weapons systems, in which the failure of the SOFTWARE PRODUCT could lead directly to death, personal injury, or severe physical or environmental damage ("High Risk Activities"). WireFlow and its suppliers specifically disclaim any express or implied warranty of fitness for High Risk Activities.

7 LIMITATION OF LIABILITY

In no event shall WireFlow be liable for any damages (including, without limitation, lost profits, business interruption, or lost information) rising out of 'Authorized Users' use of or inability to use the SOFTWARE PRODUCT, even if WireFlow has been advised of the possibility of such damages. In no event will WireFlow be liable for loss of data or for indirect, special, incidental, consequential (including lost profit), or other damages based in contract, tort or otherwise. WireFlow shall have no liability with respect to the content of the SOFTWARE PRODUCT or any part thereof, including but not limited to errors or omissions contained therein, libel, infringements of rights of publicity, privacy, trademark rights, business interruption, personal injury, loss of privacy, moral rights or the disclosure of confidential information.

8 CONTACT

All questions about this EULA shall be directed to: info@wireflow.se.
WireFlow AB
Theres Svenssons gata 10
SE-417 55 Göteborg
Sweden

Pre-requisites

Supported Platforms

Hardware

The LabVIEW API is running on Windows and LabVIEW RT targets with at least 128MB RAM.

Required software

The software runs on LabVIEW 2013 and higher, and requires the NI-HTTPS libraries with SSL support to be installed. All shared libraries shall be included in each build, if not, please see the chapter “Missing libraries” for detailed information on how to resolve missing shared libraries.

The software is installed with VI Package Manager version 2014 or higher.

The WireQueue specific examples are using the WireQueue_API, and connect to the wirequeue.com testing broker using demo credentials.

WireFlow dongle support

In order to extend the functionality of WireQueue by using the WireFlow dongles for more secure messaging the National Instruments VISA drivers (including USB passport) has to be installed.

RealTime targets

In order to run the software on the LabVIEW RT targets, please use MAX to make sure the NI HTTP with SSL support libraries are installed.

MQTT_API

The MQTT_API enables communication with a MQTT server either using standard TCP/IP or via TLS encrypted communication using OpenSSL. The driver starts a background process that listens on incoming topic updates from the MQTT server, and also handles the reconnections in the background. In the case of reconnect, the background process also handles re-subscription of topics

All received- and all published topics are saved to a local database (using Variant attributes), giving access to topics throughout the application. Each topic stored in the database is time-stamped, with the local time, when received or when published.

The background process also allows a third party implementation to filter and process topics before they are added to the local database.

General

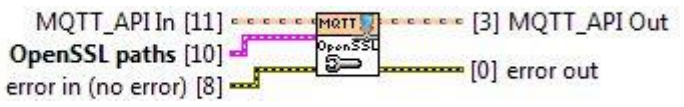
The general API methods handle connections and configurations of the MQTT session, including the OpenSSL paths and certificates

MQTT_ConfigureOpenSSL.vi

Configure the TCP/TLS communication, in terms of the OpenSSL paths to be used.

If this VI is not run before the MQTT connection is initialized, the connection will use the default paths for OpenSSL.

NOTE: this VI must be placed before the MQTT connection is initialized



MQTT_ConfigureEvents.vi

This method configures and creates the Message events and/or message queues to be used in the system.

By default the system only uses the internal data storage to read subscribed Topics, this means that only the latest values can be read and that the application will be polling instead of event driven.

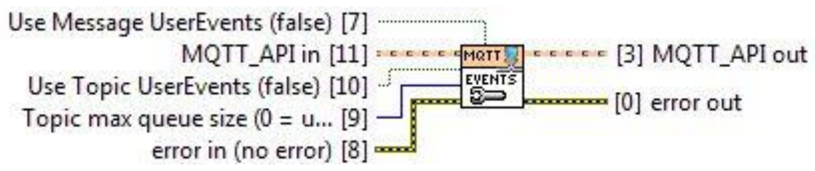
There are three types of event driven mechanisms that can be used in the system

- Topic UserEvents = Enable this to use UserEvents to receive and handle topics in an event structure.
- MQTT UserEvents = Enable this to use UserEvents to receive and handle MQTT messages (not PUBLISH) in an event structure, e.g. to perform actions at connection etc.

- Topic MsgQueue = Enable this to use a LabVIEW queue to receive topics with the MQTT_WaitForNext method

In order to use the UserEvents, first get UserEvent refs with the MQTT_GetMessageEventRefs method then register for the events and read them in an event structure.

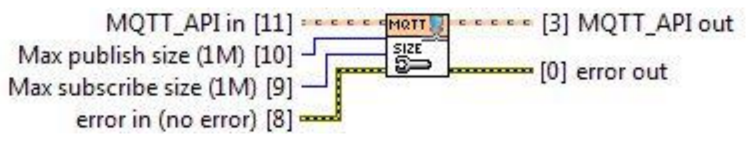
NOTE: this VI must be placed before the MQTT connection is initialized.


Use Message UserEvents (false) [7]
MQTT_API in [11]
Use Topic UserEvents (false) [10]
Topic max queue size (0 = u... [9]
error in (no error) [8]
MQTT
EVENTS
[3] MQTT_API out
[0] error out

MQTT_ConfigureTopicSize.vi

This method configures the maximum size of publish and subscribe messages.
If a call to the publish method exceeds the Max publish size and error is generated.
If a message of size greater than Max subscribe size is received, it is silently dropped.

NOTE: this VI must be placed before the MQTT_Init.vi is called
NOTE2: If size is set to a negative number the default size will be used
NOTE3: the total receive buffer is set to 10 times the size of the Max subscribe size or 5M, whatever is greater



MQTT_Init.vi

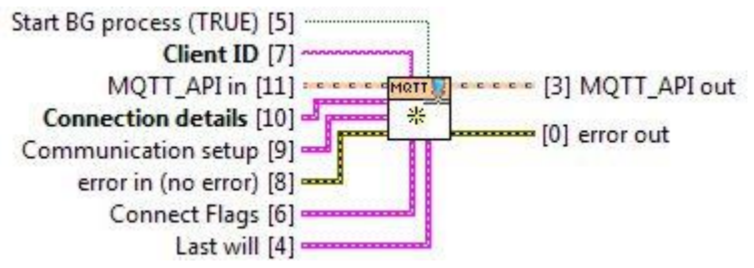
Initializes a MQTT session using the specified configuration parameters.

The CONNECT flags “User name”, “Password” and “Will flag” is automatically handled when the corresponding text fields are specified.

- Client ID = a unique identifier for the current application.
- Connection Details
  - server address = web address of the server
  - server port = port of the MQTT server
  - User name = account name for the connection
  - password = password for the connection
  - Keep Alive time = time from last message from the server until a PING message is sent. If no response is received within 2 * Keep alive time the connection is closed.
  - reconnect period = if communication is disconnected, the background process will automatically try to reconnect at this interval
- Communication setup

- Communication type = specifies if we should use TCP or TCP/TLS (recommended!)
- TLS level = specifies at which level the TLS certificate should be validated:
    - Full check (recommended)
    - allow valid certificate with wrong server name
    - allow any self signed certificate.
- TCP_NO-DELAY = If True the Nagle algorithm is disabled, and small packets are not buffered, but sent immediately.

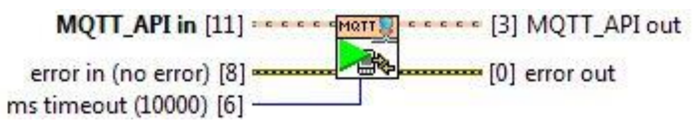
- Connect Flags = defines the MQTT connection flags, i.e. what parts is active in the connect message
    - Will Retain
    - Will QoS
    - Clean Session
- Last Will = defines a topic/value that will be set if the client is disconnected incorrectly, e.g. in the case of network failure.
    - topic
    - value
- Start BG process = indicates if the background process should be started, if set to FALSE, make sure to start the process using the MQTT_StartBackgroundProcess method



MQTT_StartBackgroundProcess.vi

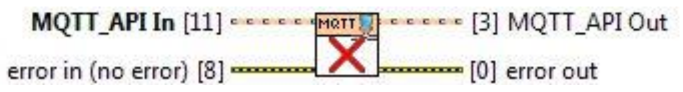
Starts the MQTT background process unless started at the init session.

This can be used to create advanced applications that should act on received MQTT messages, e.g. CONNACK etc. See the WireQueue API Init.vi method for an example



MQTT_Clear.vi

Clears the MQTT session, and disconnects from the server gracefully.

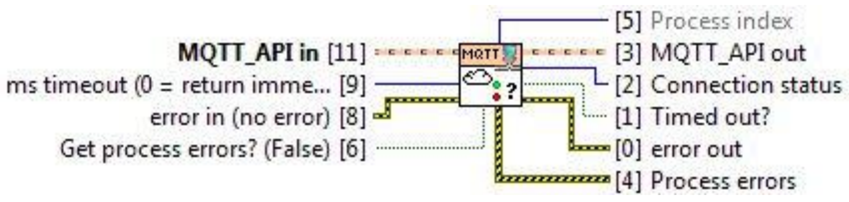


MQTT_GetStatus.vi

Returns the status of the background process, and optionally also returns the last 100 errors in the BG process.

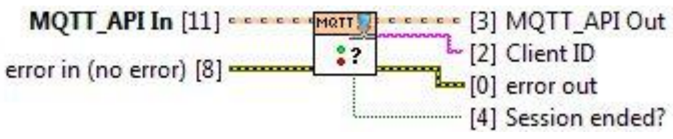
- Unknown = the BG process is in an unknown state (should only be reported at startup or exit)
- Idle = connection has been lost, waiting to reconnect
- Connecting = connect sequence has been initiated, waiting for server to respond
- Connected = the background process is sucessfully connected to the MQTT server.

NOTE. if ms timeout is set to a value greater than 0 the method will wait for an updated status message.



MQTT_GetSessionStatus.vi

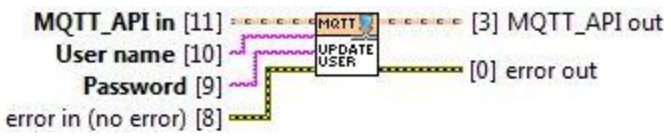
Returns the status of the current session, i.e. if the session is valid and the ClientID used to intialize.



MQTT_UpdateConnectionCredentials.vi

Updates the "User name" and "Password" used at MQTT Connect.

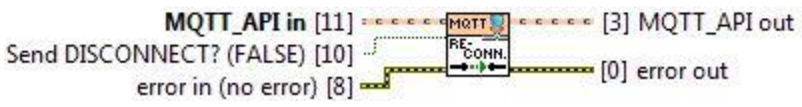
This can be used to update a time limited token so that the background process can correctly reconnect to the broker.



MQTT_ForceReconnect.vi

Forces a reconnection to the broker by closing the network connection to the broker.

Optionally sends a DISCONNECT message to the broker to signal that this is an intended shutdown.



Server override methods

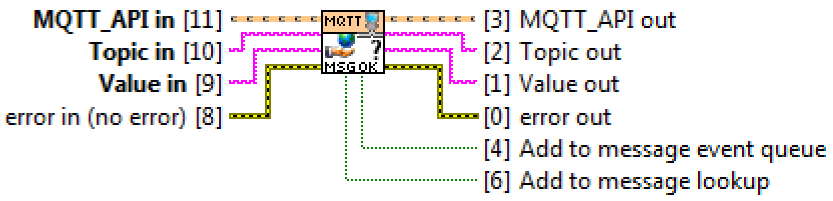
This group of methods can be overridden to implement low level filtering of topics as they arrive from the MQTT broker.

MQTT_CheckReceivedMessage.vi

Allows the implementation to filter on incoming messages before they are added to internal lookups or message queues, e.g. remove topics over a certain size.

- Topic in = name of the received topic
- Value in = value of the received topic
- Topic out = name of the topic to be used in the system
    - This can for example be the topic name stripped of a MAC
- Value out = value of the topic to be used in the system
    - Can for example be a decrypted string
- Add to message event queue
    - If TRUE the Topic/Value outputs will be added to the message queue
- Add to message lookup
    - If TRUE the Topic/Value outputs will be added to the message lookup

This VI can be overridden by a child class to expand filtering, stop use of message queue handling or lookup etc.



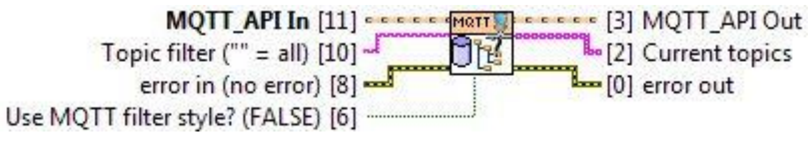
Local topic access

This group of API methods handles the topic access in the local database.

MQTT_ListTopics.vi

Return a list of all received or published topics in the local lookup that match a specific topic filter.

Filter is given in the LV match-pattern notation or in MQTT filter notation (experimental).

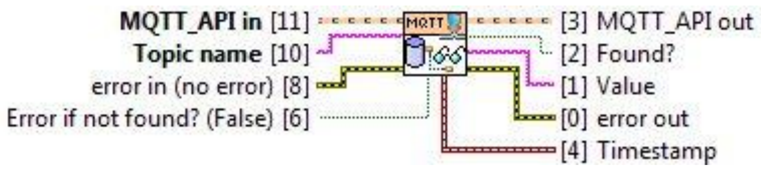


MQTT_ReadTopic.vi

Reads a local copy of a MQTT topic.

- Topic name = name of the topic to be read
- Error if not found= if TRUE and the topic is not found an error with code 6507 will be returned.
- Value = the value returned for the specified topic
- Found? = if False the specified topic was not found
- Timestamp = timestamp when the topic was received by the back ground process

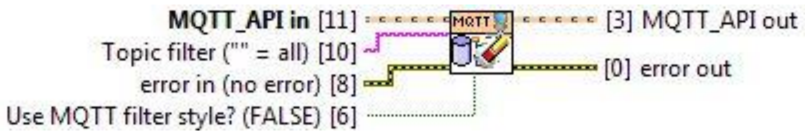
A Topic is built as a number of segments separated by '/', e.g. Device/Subsystem/TopicName



MQTT_ClearLocalTopics.vi

Remove the named topics from the local database, e.g. if a remote device has been removed.

If filter is empty string all topics are removed, otherwise filter is given in the LV matchpattern notation or in MQTT filter notation (experimental).



Server access methods

To use MQTT a central broker is needed, and this group of access methods deals with the communication between the broker and this client.

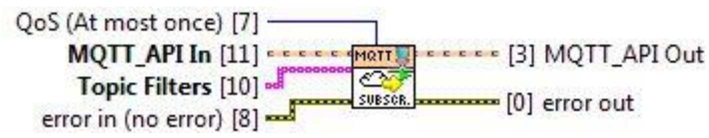
MQTT_Subscribe.vi

Start subscribing on a number of topics, using the specified QoS. To subscribe to topics with different QoS, call this VI multiple times with different settings.

- QoS = specifies the MQTT quality of service to be used with this subscription
- Topic Filters = Each element in the input array defines one subscription topic, with or without wildcards.

Note.

1. **Topics are case sensitive, e.g. Main/MyTopic and main/mytopic is two different topics**
2. **Use '+' as a single level wildcard, e.g. the filter Main/+/Status, matches any topic starting with Main and ending with Status**
3. **Use '#' to match any subtopic, e.g. the filter Main/Subsystem1/# matches all topics under the SubSystem Topic.**



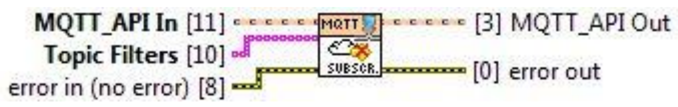
MQTT_Unsubscribe.vi

Unsubscribes from the specified MQTT topic filters. The topic filter must exactly match the subscribed topics to be unsubscribed.

- Topic Filters = Each element in the input array defines one subscription topic, with or without wildcards.

Note.

1. **Topics are case sensitive, e.g. Main/MyTopic and main/mytopic is two different topics**
2. **Use '+' as a single level wildcard, e.g. the filter Main/+/Status, matches any topic starting with Main and ending with Status**
3. **Use '#' to match any subtopic, e.g. the filter Main/Subsystem1/# matches all topics under the SubSystem Topic.**



MQTT_Publish.vi

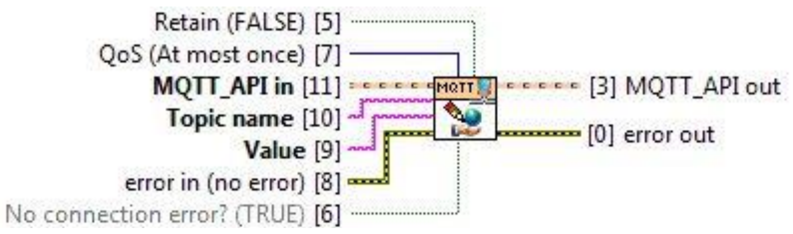
Publish a topic to the MQTT server with specified QoS and retain settings.

- QoS = The MQTT quality of service to be used with this publish topic
- Retain = If True the server will store a copy of the last message on the server
- Topic name = name of the topic to update
- Value = value of the topic

Note 1. To remove a retained message from the server, publish the topic with empty string value, and with Retain=True

Note 2. A Topic is built as a number of segments separated by '/', e.g.

Device/Subsystem/TopicName

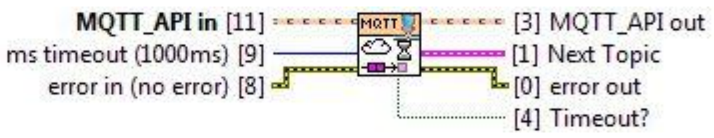


MQTT_WaitForNext.vi

Returns the next received topic from the MQTT server.

- Next Topic
  - Time Stamp = time when the background process received the message
  - Topic = name of the topic sent from the broker
  - Value = value of the topic
- Timeout? = True if no message was found in the queue

Note. The queue is limited in size (set by MQTT_ConfigureEvents) to prevent out of memory issues.

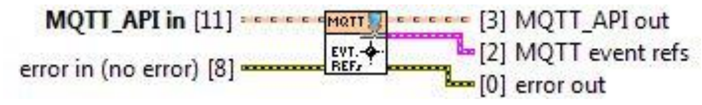


MQTT_GetMessageEventRefs

Returns UserEvent refs that can be used to create event driven applications.

- MQTT Event refs
  - Message events = All message types, except PUBLISH, are sent in this UserEvent together with their raw payload. E.g. CONNACK, PINGRESP etc.
  - Topic Events = All received PUBLISH messages will be decoded into Topic/Value pairs and sent into this User event

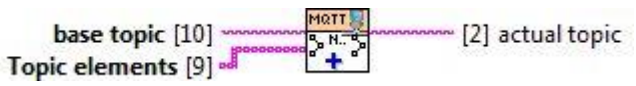
NOTE. To use these references one has to register for the events, and use an Event structure to receive the data. The events must also be activated in the MQTT_ConfigureEvents method.



Helper functions

ArrayBuildTopic.vi

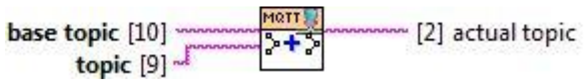
Combines topic segments into one topic, i.e. if the base topic is <base>, and the topic elements are <subA>, <subB>and <subC> the resulting topic will be <base>/<subA>/<subB>/<subC>



BuildTopic.vi

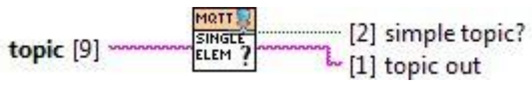
Adds the subtopic to the base topic to create the resulting topic.

actual topic = base topic/topic



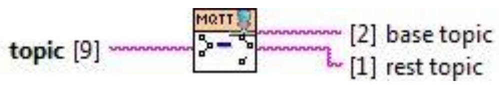
CheckSimpleTopic.vi

Checks if the topic is a single element topic, i.e. it only has one segment



StripTopic.vi

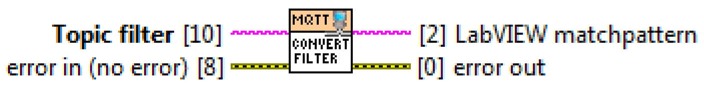
Strips the first element from the input topic, and returns the first element as base topic, and the rest as rest topic.



CreateMatchPatternFromTopicFilter.vi

Converts a MQTT topic filter into LabVIEW matchpattern format.

Use this to create LabVIEW matchpattern to find matching topics, e.g. the MQTT filter basetopic/+/subs/# translates to ^basetopic/[~\/]+/subs/.*$



WireQueue API

The WireQueue API is an extension to the MQTT API to demonstrate how the WireFlow MQTT driver can be expanded to be more advanced.

WireQueue defines a specific set of MQTT Topics to be used as base topics.

- Message
    - Standard messaging between the client and the WireQueue broker
- Alarm
    - Specific Base-topic to handle Alarms
- Error
    - Specific Base-topic to handle Errors
- Security
    - Specific Base-topic to transfer messages that are signed by the WireFlow Security dongles for extra security.

This API consists of a number of VIs that allows the user to publish messages to other clients, as well as subscribing on messages from remote clients. Subscription is handled automatically when clients connect to the server, and is determined by the Access Control List.

For additional security it is possible to publish a message to a remote client with a message authentication code (MAC) added using the WireFlow dongles, this means that critical messages can be protected so that only clients with the correct dongle can write messages (e.g. security features of an application, restarting etc.). The actual value in the dongle doesn't have to be known, as long as it is the same in the dongles at each end of the communication.

The LabVIEW driver is running over TCP using TLS to secure the authentication and communication.

General

The general API methods allow a user to connect, disconnect and check status of the communication.

Figure 1. General API methods

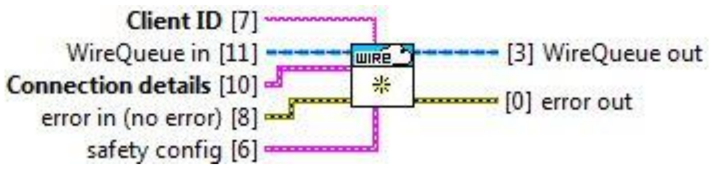
Init.vi

Initializes the buffers and references needed for the WireQueue session, also starts the BG process that does all the actual network communication.

- Client ID = Unique identifier for the current machine/session
- Connection details.server instance = the name of the instance of the WireQueue server

- Connection details.server port = TCP port that is used by the server
- Connection details.User name = user name to authenticate with the server
- Connection details.password = password for the specified user
- Connection details.keep Alive time = time from last message from the server until a PING message is sent. If no response is received within 2 * Keep alive time the connection is closed.
- Connection details.reconnect period = if communication is disconnected, the background process will automatically try to reconnect at this interval
- safety config.WF dongle reference = NI-VISA reference specifying the WireFlow dongle used for safety messaging
- safety config.Key ID = the dongle Key to be used when authenticating a safety message

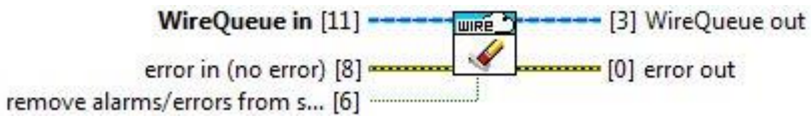
NOTE: The Client ID has to be unique in the system. If two Clients use the same ID they will start kicking each other out at each reconnect.



Clear.vi

Stops all processes and clears all buffers and references created in the session.

Optionally clears WireQueue Alarm and Error topics.

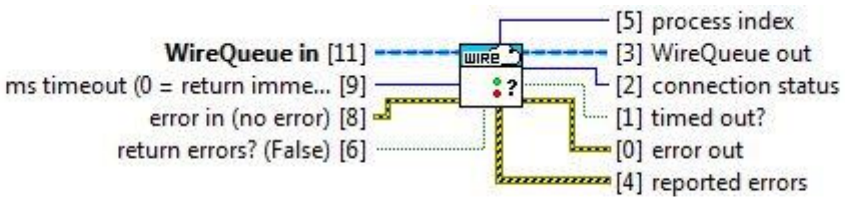


GetStatus.vi

Returns the current status of the system:

- Connection status = the current state of the background process
- process index = the number of iterations that the BG process has taken
- reported errors = last 100 reported error from the BG process
- return errors? = if true the last 100 process errors are returned

NOTE: Returning errors automatically flush the error queue.

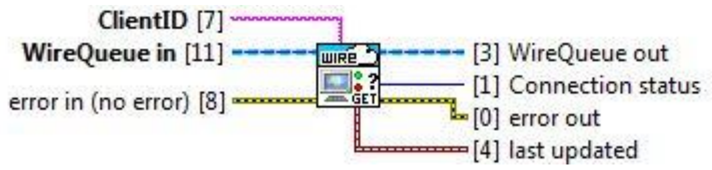


Client_GetStatus.vi

Reads the connection status for a named client.

- ClientID = remote client ID that posted connection status
- Connection status = indicates the last known status of the remote client.
- last update = timestamp when the background process received the status message

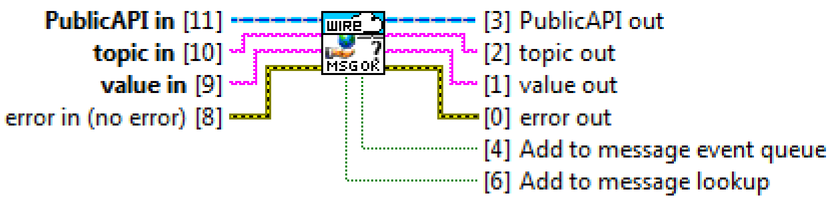
NOTE: When connection status is "Link Lost" the client exited abnormally, i.e. didn't close correctly.



MQTT_CheckReceivedMessage.vi

This VI overrides the standard implementation and skips messages if the security MAC is not correct (i.e. the signing of the message is incorrect).

For more info see MQTT_CheckReceivedMessage.vi under the MQTT_API section



Topic monitoring

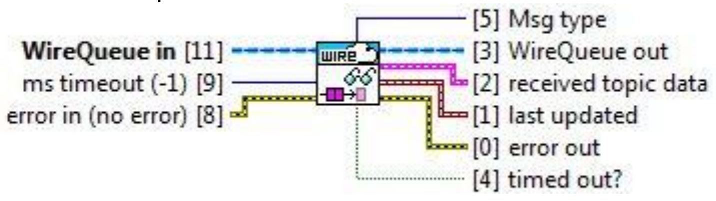
Topic monitoring allows monitoring without knowing the topic names in advance

Figure 2. Topic monitoring methods

GetNextReceivedTopic.vi

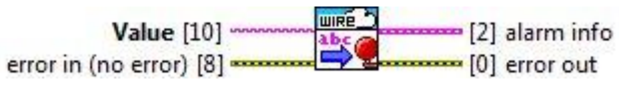
Waits and reads the next message.

- ms timeout = specifies the time to wait for a message to be available in the queue.
- Msg type = type of the returned message (Alarm, error, connection, security or normal)
- received topic data
  - ClientID = the remote client that posted the message
  - topic = name of the message.
  - value = text posted by the remote client as the message value
- last updated = timestamp when the background process received the message.
- timed out? = True if no message was found in the queue within the specified timeout period.



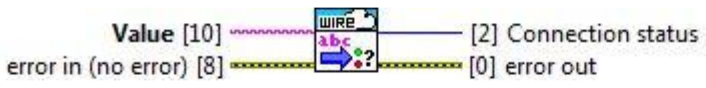
Decode_AlarmTopic.vi

Decodes an Alarm topic string value to an alarm info cluster.



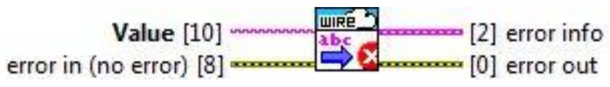
Decode_ConnectionStatusTopic.vi

Decodes a connection status topic string value to a connection status enum.



Decode_ErrorTopic.vi

Decodes an Error topic string value to an error info cluster.



Normal topic access

WireQueue configures a number of topic categories (Message, Alarm and Error) and accessing these “normal” topics are done using these methods.

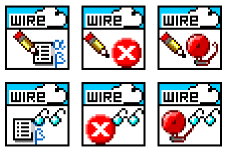


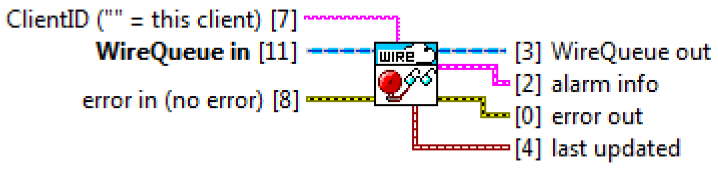
Figure 3. Normal topic access methods

Alarm_Get.vi

Reads the last alarm for the specified ClientID.

- ClientID = remote client ID that posted the alarm
- alarm.descr = text posted by the remote client describing the alarm
- alarm.level = numeric value indicating the level of the alarm. The mobile app uses 0 as no alarm.
- alarm.timestamp = timestamp string from the client indicating when the message was posted in local time.
- last update = timestamp when the background process received the alarm

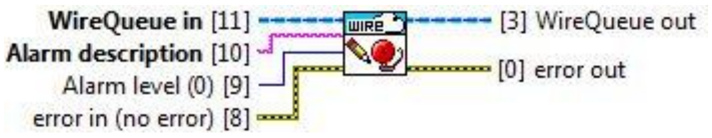
NOTE: When level is 0 and description is empty, the alarm is resolved and removed by the remote client



Alarm_Set.vi

Publishes an alarm for the local ClientID, using the specified level and description.

Set alarm level to 0 to clear the alarm on the server (and on the mobile app).



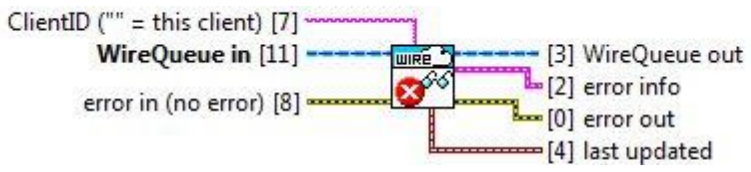
Error_Get.vi

Reads the current error for the a specified ClientID.

- ClientID = remote client ID posting the error
- error.descr = text posted by the remote client describing the error
- error.code = numeric value indicating the error code. The mobile app uses 0 as no error.

- error.timestamp = timestamp string from the client indicating when the message was posted in local time.
- last update = timestamp when the background process received the error.

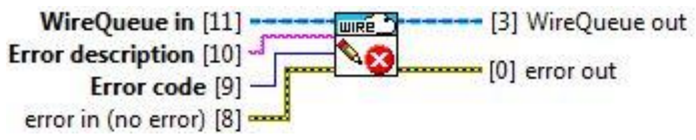
NOTE: When code is 0 and description is empty, the error is resolved and removed by the remote client



Error_Set.vi

Publishes an error for the local ClientID, using the specified code and description.

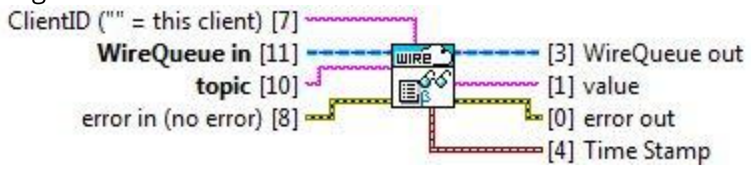
Send an error with code = 0 to clear the error on the server (and on the mobile app)



MessageRead.vi

Reads a topic from the local lookup buffer

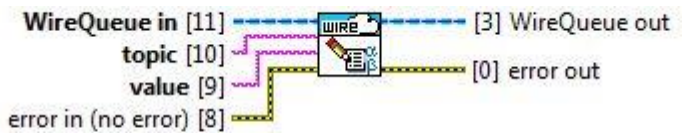
- topic = name of the message.
- Message lookup data
- value = text posted by the remote client as the message value
- last update = timestamp when the background process received the message.



MessageWrite.vi

Publishes a topic to the server.

NOTE: The topic value and name will be encoded in UTF-8 for safe transfer, but since LabVIEW doesn't support unicode this is a plain ASCII to UTF8 conversion. This means that the smart phone apps might not be able to correctly display strings containing characters outside standard ASCII (i.e. ASCII values greater than 0x7F)



Security topic access

These methods handles messages that are automatically signed by the WireQueue background process, so that only correctly signed messages are accepted and read.

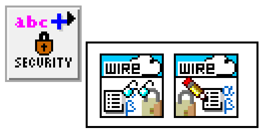


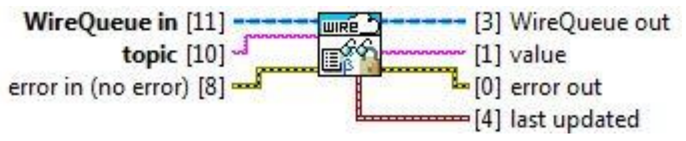
Figure 4. Security topic access methods

Security_MessageRead.vi

Reads the named security message, i.e. a message signed by a security token.

- topic = name of the safety message that is to be read.
- value = text posted by a remote client as the message value
- last update = timestamp when the background process received the message

NOTE: Only messages that are correctly signed will be available in the message queue.

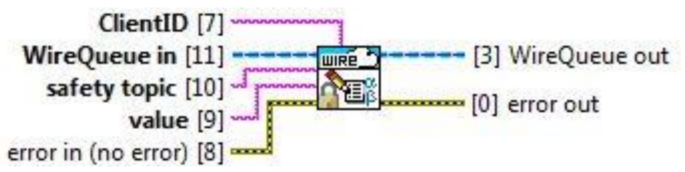


Security_MessageWrite.vi

Writes a security message to the specified ClientID (i.e. a message signed by a security token), using the WireFlow dongle specified at init for authorization.

NOTE: Requires the specified ClientID to have a matching dongle, and that it has posted a security challenge to be used to authorize the message

NOTE: The topic value and name will be encoded in UTF-8 for safe transfer, but since LabVIEW doesn't support unicode this is a plain ASCII to UTF8 conversion. This means that the smart phone apps might not be able to correctly display strings containing characters outside standard ASCII (i.e. ASCII values greater than 0x7F)



Quick start

Install WireQueue using VIPM (VI Package manager is free and can be downloaded at http://jki.net/vipm)

Once VIPM is installed, install WireQueue by double-clicking the item in the list of packages (enter WireQueue in the filter box for fast find)

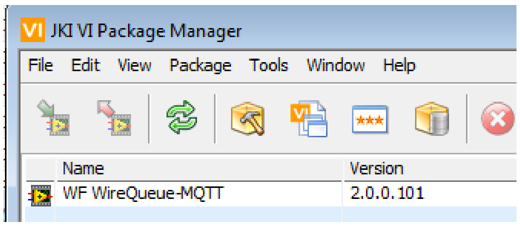
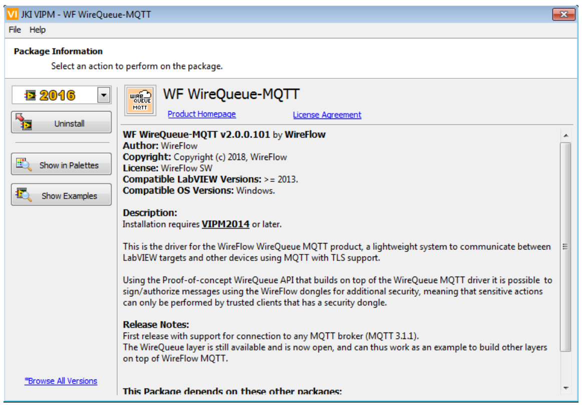


Figure 5. WireQueue in VI Package manager

After completion of installation of the WireQueue package go to Show Examples.



Package Information

Select an action to perform on the package.

WF WireQueue-MQTT

WF WireQueue-MQTT v2.0.0.101 by **WireFlow**
Author: WireFlow
Copyright: Copyright (c) 2018, WireFlow
License: WireFlow SW
Compatible LabVIEW Versions: >= 2013.
Compatible OS Versions: Windows.

Description:
Installation requires **VIPM2014** or later.

This is the driver for the WireFlow WireQueue MQTT product, a lightweight system to communicate between LabVIEW targets and other devices using MQTT with TLS support.

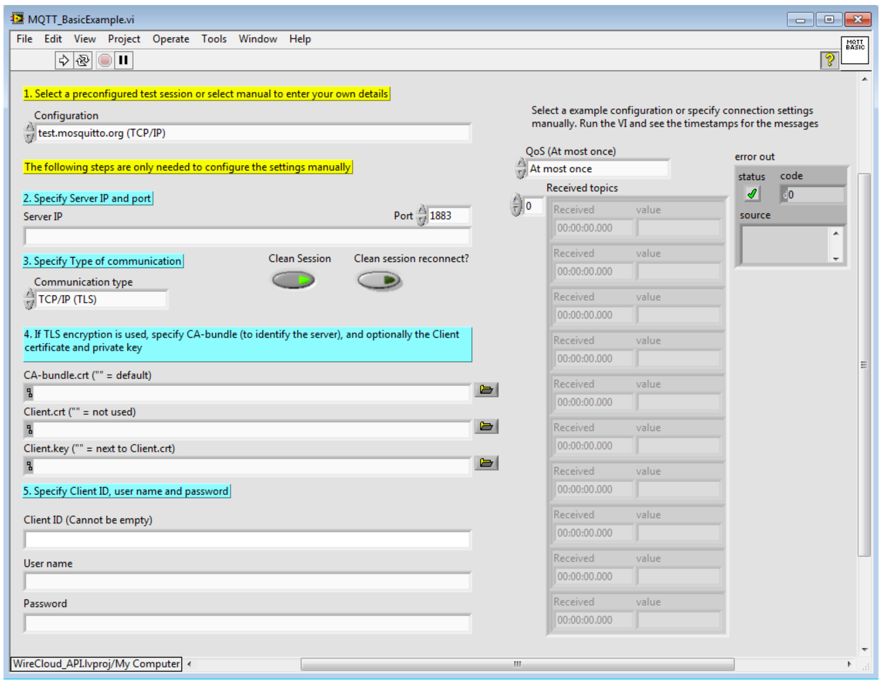
Using the Proof-of-concept WireQueue API that builds on top of the WireQueue MQTT driver it is possible to sign/authorize messages using the WireFlow dongles for additional security, meaning that sensitive actions can only be performed by trusted clients that has a security dongle.

Release Notes:
First release with support for connection to any MQTT broker (MQTT 3.1.1).
The WireQueue layer is still available and is now open, and can thus work as an example to build other layers on top of WireFlow MQTT.

This Package depends on these other packages:

Figure 6. WireQueue information window in VIPM

Open the MQTT_BasicExample.vi example.



1. Select a preconfigured test session or select manual to enter your own details

Configuration: test.mosquitto.org (TCP/IP)

The following steps are only needed to configure the settings manually

2. Specify Server IP and port

Server IP:

Port: 1883

3. Specify Type of communication

Communication type: TCP/IP (TLS)

Clean Session

Clean session reconnect?

4. If TLS encryption is used, specify CA-bundle (to identify the server), and optionally the Client certificate and private key

CA-bundle.crt ("" = default)

Client.crt ("" = not used)

Client.key ("" = next to Client.crt)

5. Specify Client ID, user name and password

Client ID (Cannot be empty)

User name

Password

Select a example configuration or specify connection settings manually. Run the VI and see the timestamps for the messages

QoS (At most once): At most once

Received topics: 0

Received	value
00:00:00.000	
00:00:00.000	
00:00:00.000	
00:00:00.000	
00:00:00.000	
00:00:00.000	
00:00:00.000	
00:00:00.000	
00:00:00.000	
00:00:00.000	

error out

status: ✓ code: 0

source

Figure 7. MQTT_BasicExample

This example connects to the test.mosquitto.org test broker, and can connect with TCP/IP, TCP/IP with TLS or TCP/IP with TLS and Client certificate authentication.

Select the type of communication and then run the example. The example subscribes to its own topics and post and read 10 topics.

Examples

The driver comes with a number of examples that can be found using the LabVIEW Example Finder, just search for WireQueue. You can also directly after installing the VIPM package show the included examples.

The examples cover basic MQTT messaging as well as WireQueue security messaging, logging as well as monitoring of all clients.

To monitor the data from a smart phone (iPhone or Android) please install WireQueue on the smartphone and login to the same server instance

The WireQueue examples are pre-configured to connect to a WireQueue demo server that is restarted periodically.

NOTE: To run the Security examples you need a WireFlow dongle.

Error codes

The software uses the following error codes

Error code	Description
6501	Server connection failed
6502	Invalid Alarm subsystem
6503	Invalid Error subsystem
6504	Only one WireQueue instance per LabVIEW context is allowed!
6505	Server connection was lost (no ping response)
6506	Operation not possible: No server connection
6507	Specified Topic was not found in local buffer
6508	Security message write failed: remote challenge is missing
6509	Local clock is probably not set
6510	Invalid topic filter when converting to LabVIEW matchpattern
6511	Invalid connection configurations
6512	Bad OpenSSL configuration
6513	Invalid configuration order
6514	Background process is already running
6550	SSL connection error
6551	SSL Connection has been closed
6552	SSL operation failed, more data to write or read
6553	SSL Connection is not completed
6554	SSL certificate lookup failed
6555	SSL system called failed
6556	SSL library error

Troubleshooting

This chapter lists the most common problems that a user might encounter

Connection fails

If there is an error at init the API will automatically clean up all connections and exit, but if the background process successfully connects once and then has to perform a reconnect a number of errors can occur. The connection can fail for a number of reasons, and the table below lists the actions by the software as well as the resolution that can be taken by the developer.

Fail reason	Software action	Resolution
Wrong IP address	Enters reconnection and waits for the IP to become available	Check the status, and verify that a correct IP address has been entered
Wrong IP port	See "Wrong IP address"	See "Wrong IP address"
Bad user name/password	Disconnects the session and closes all references and buffers with an error	Fix username and password
Not authorized	Disconnects the session and closes all references and buffers with an error	Ask an administrator to check that the specified ClientID is allowed access
Server unavailable	TCP connected ok on the port but there is no cloud server serving on that port The session is disconnected and closed with an error.	Verify the IP address/port and/or ask an administrator to verify that the service is up and running on that port.
SSL connection failed	Disconnects the session and closes all references and buffers with an error	Check that the OpenSSL config is correctly setup (see the "Missing libraries" section), either using default or specific configuration. Also check that the time is correctly set on the computer.

Missing libraries

The API runs out of the box with the OpenSSL libraries that are installed together with LabVIEW. In some cases it might be necessary to specify other versions, or other locations for the files. This is normally done with the MQTT_ConfigureOpenSSL.vi, but it can also be done by creating a single file named WireQueueSSL.cfg that should be placed in the \data folder of the application. The configuration specifies three OpenSSL files (two shared libraries and one certificate bundle):

```
[OpenSSL_Paths]

; this section defines the paths to the libraries used for Open SSL access.

; Each path can be given as an absolute path, or as a path relative to the folder containing this file.

; NOTE: paths are platform dependent, and different OS's handle type case differently


ssleay = "C:\Program Files (x86)\National Instruments\Shared\nissl\NIlibeay32.dll"

libeay = "C:\Program Files (x86)\National Instruments\Shared\nissl\NIssleay32.dll"

CA-bundle.crt = "C:\Program Files (x86)\National Instruments\Shared\nicurl\ca-bundle.crt"
```

The paths can be given absolute or relative to the "data"-folder.

During development the config file can be put in a "data" folder next to the lvproj that contains the source code (if the code is opened outside of a project, the default OpenSSL configuration will be used.)