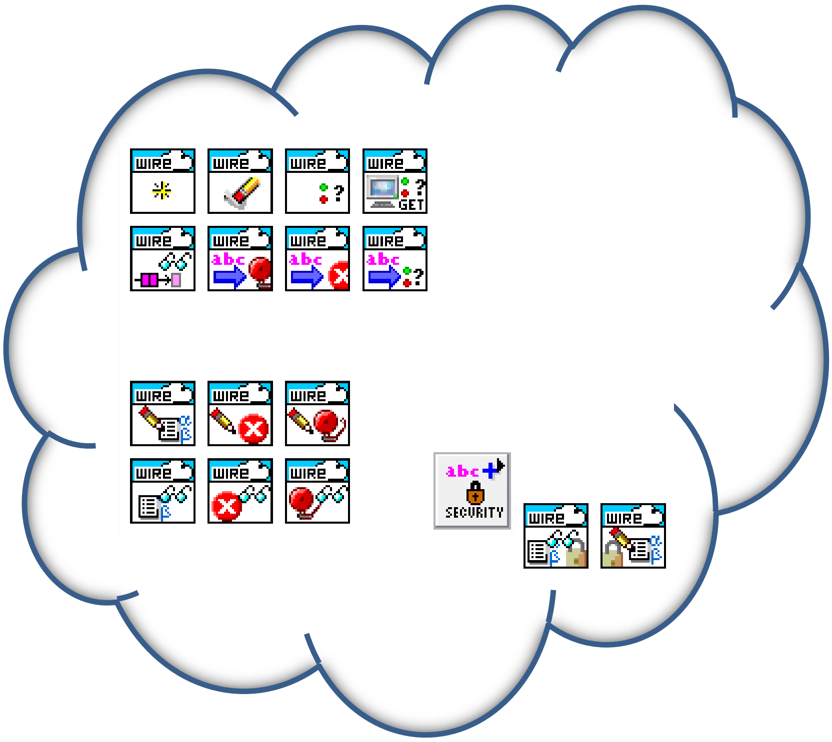
WireQueue User’s Manual

AC0075-001 rev D



WireFlow AB, April 2018

Contents

Support information

Technical support and Product information

www.wireflow.se

WireFlow headquarters

WireFlow AB
Theres Svenssons gata 10
SE-417 55 Göteborg
Sweden

Please see appendix “Technical support and Services” for more information.

Important information

Copyright

The Software is © WireFlow 2015
The LabVIEW API uses the OpenSSL libraries that are copyright the OpenSSL project.

Trademarks

LabVIEW is trademark of National Instruments

Warning regarding use of WireFlow products

The software is not tested to be used in dangerous locations, or safety critical applications. Even if the software runs on almost all LabVIEW targets, it is the end users responsibility if it still is used in a safety critical application.

Supported platforms

Hardware

The LabVIEW API is running on Windows, Linux, MacOSX and LabVIEW RT targets (32-bit) with at least 128MB RAM.

NOTE: has not been tested on 64-bit LabVIEW.

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 LabVIEW API requires the user to sign up for an account at WireQueue.com (examples can still be run without a subscription though)

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.

NOTE: It is only possible to run one WireQueue session per LabVIEW context.

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.

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)

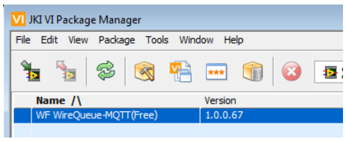


Figure 1. WireQueue in VI Package manager

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

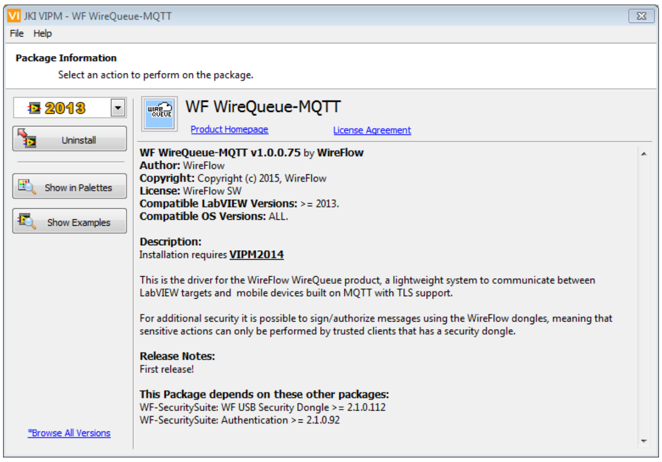


Figure 2. WireQueue information window in VIPM

Open the BasicMonitorClient- and the BasicPublishClient LabVIEW projects and run the BasicMonitorClient- and BasicPublishClient VIs…

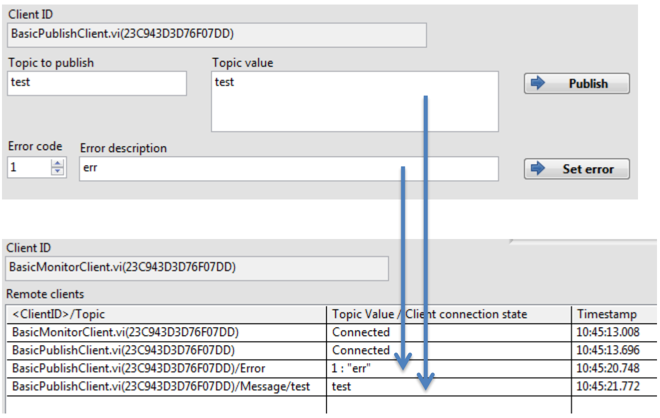


Figure 3. Basic Monitor- and Publish examples in action

The examples should connect automatically to a WireQueue demo server, and topics updated in the publish example should be shown in the listbox of the monitor VI (the monitor VI might also show topics posted by other users).

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

NOTE: The demo server is restarted periodically, and can go up- and down without notice.

Software

This chapter describes the software: API parts, User Interfaces, Examples etc. that is included in the software package.

API

The WireQueue 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 runs on most LabVIEW targets, and 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.

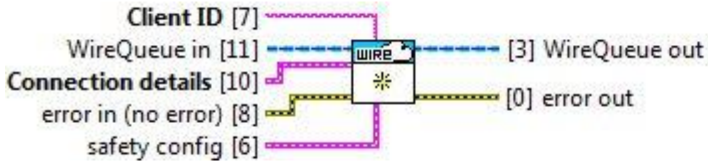
Figure 4. General API methods

Init.vi

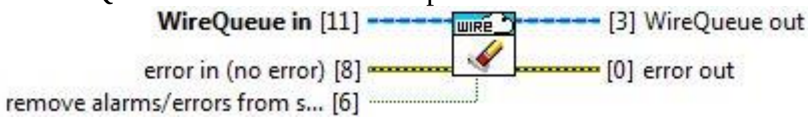
Initializes the buffers and references needed for the WireQueue session, also starts the BG process that does all the actual network communication.

- 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.



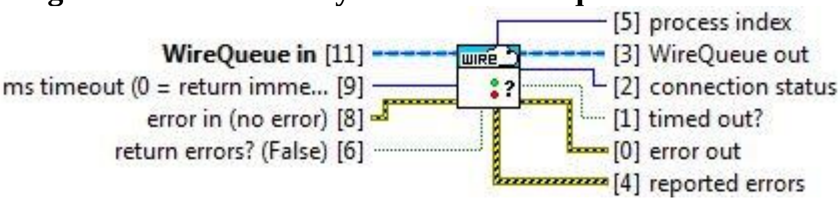
Clear.vi
Stops all processes and clears all buffers and references created in the session.
Optionally clears WireQueue Alarm and Error topics.



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

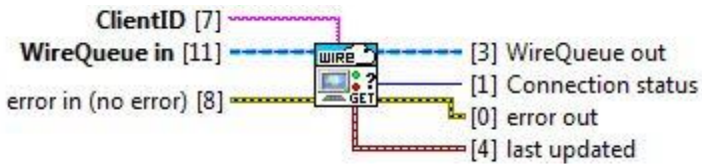
NOTE: Returning errors automatically flush the error queue.



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.



Topic monitoring

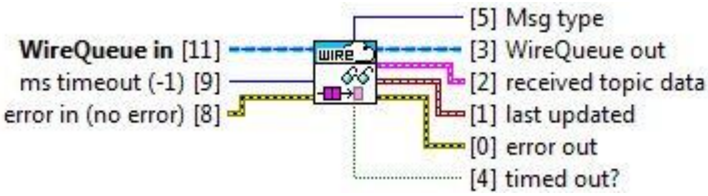
Topic monitoring allows monitoring without knowing the topic names in advance

Figure 5. 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.

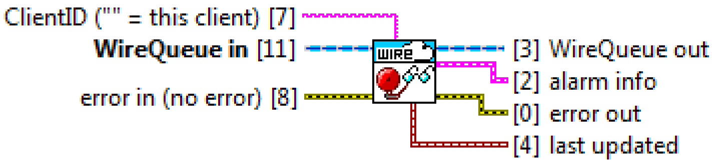
Figure 6. Normal topic access methods

Alarm_Get.vi

Reads the last alarm for the specified ClientID.

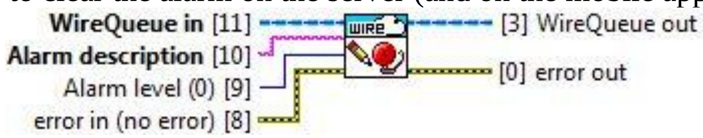
- ClientID = remote client ID that posted the alarm
- alarm.descr = text posted by the remote client describing the alarm
- alarm.level = numeric value indicating the level of the alarm. The mobile app uses 0 as no alarm.
- alarm.timestamp = timestamp string from the client indicating when the message 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).

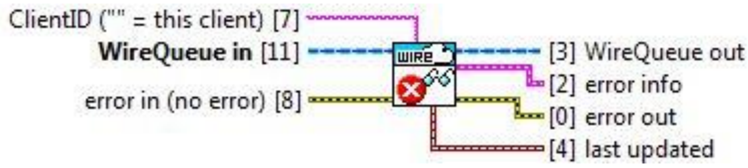


Error_Get.vi

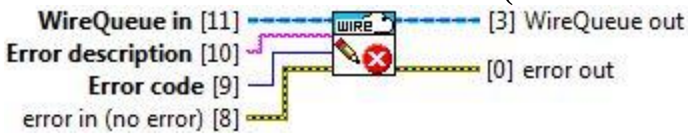
Reads the current error for the a specified ClientID.

- 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

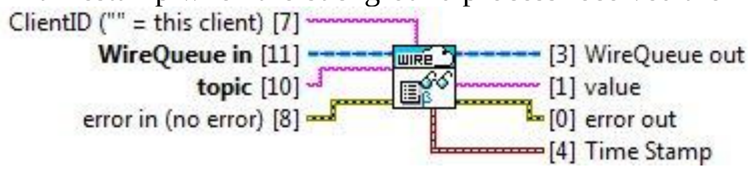


Error_Set.vi
Publishes an error for the local ClientID, using the specified code and description.
Send an error with code = 0 to clear the error on the server (and on the mobile app)



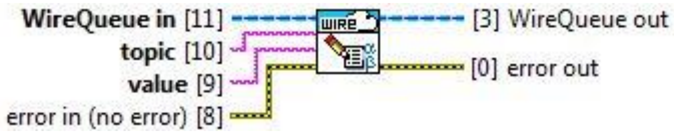
MessageRead.vi
Reads a topic from the local lookup buffer

- 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.

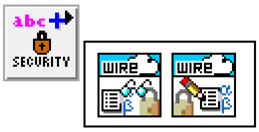


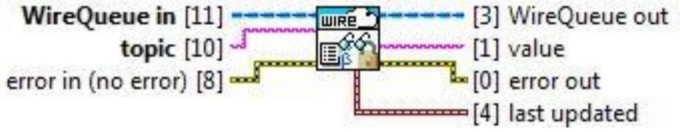
Figure 7. Security topic access methods

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.

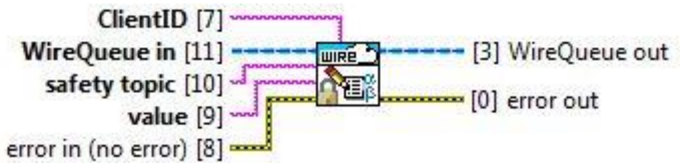


Security_MessageWrite.vi

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)



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 security messaging, logging as well as monitoring of all clients.
By default the examples are pre-configured to connect to a 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
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 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.)

Technical support and Professional services

Please send all support questions to
support@wireflow.se