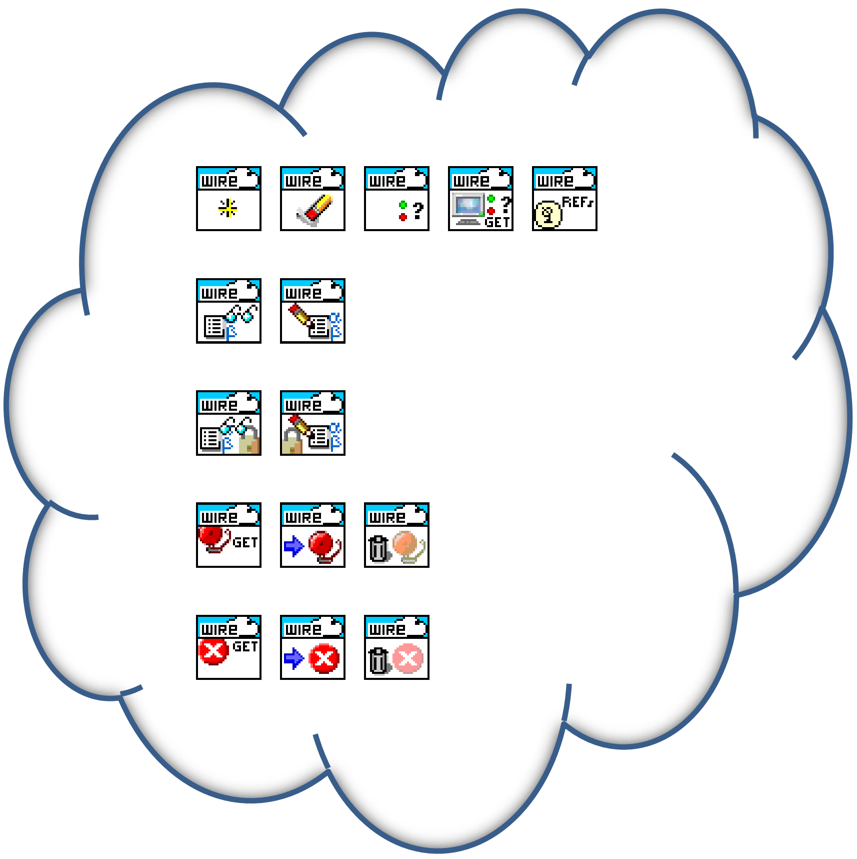
wireflow.se

WireQueue User's Manual

AC0075-001 rev A



WireFlow AB, June 2015

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

Trademarks

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 and LabVIEW RT targets (32-bit) with at least 128MB RAM.

N.B. has not been tested on 64-bit LabVIEW.

Required software

The software runs on LabVIEW 2013 and higher, and requires the OpenSSL libraries to be installed. We do our best to make sure all necessary shared libraries are included when an application is built, but if we fail please see the chapter

Troubleshooting how to resolve missing shared libraries.

In order to use the WireFlow dongles the National Instruments VISA drivers (including USB passport) has to be installed.

N.B. 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 SSL libraries are installed.

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 an user to connect, disconnect and check status of the communication.

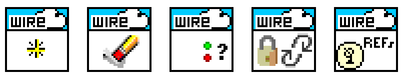


Figure 1. General API methods

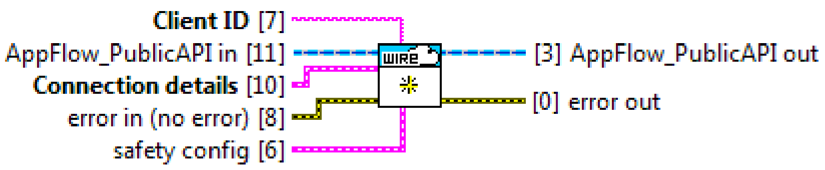
Init.vi

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

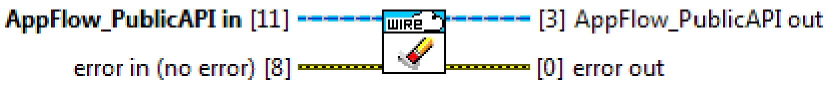
- Client ID = Unique identifier for the current machine/session
- Connection details.server instance = the name of the instance of the WireQueue server

- 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
- 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

N.B. 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.



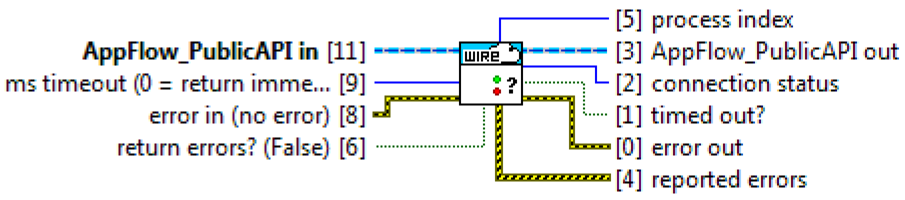
Clear.vi
Stops all processes and clears all buffers and references created in the session.



GetStatus.vi
Returns the current status of the system:

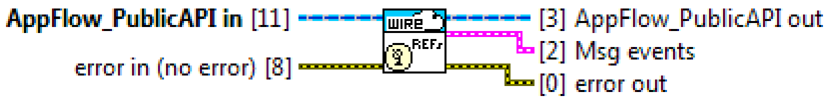
- Connection status = the current state of the background process
- process index = the number of iterations that the BG process has taken
- reported errors = last 100 reported error from the BG process

N.B. Getting status also means that we flush the Error queue.

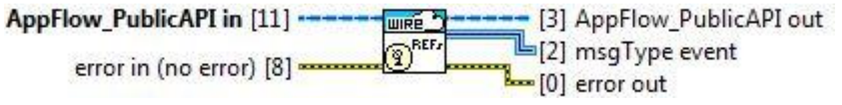

AppFlow_PublicAPI in [11]
ms timeout (0 = return imme... [9]
error in (no error) [8]
return errors? (False) [6]
[5] process index
[3] AppFlow_PublicAPI out
[2] connection status
[1] timed out?
[0] error out
[4] reported errors

GetUserEventRefs.vi
Returns user events to monitor changes in subscribed messages.
The messages are separated in:

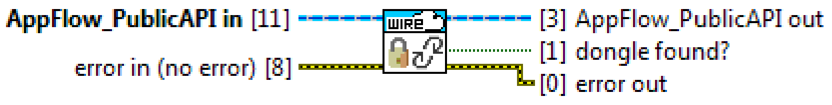
- Connects = every time a client is connected or disconnected we can get an event with the client connection status
- Alarms = remote client alarm messages
- Errors = remote client error messages
- Message = remote clients normal messages
- Safety = messages authorized with a MAC using the WireFlow dongles.



GetMessageEventRef.vi
Returns user event to monitor changes in subscribed messages.
The data returned indicates which message queue that was recently updated.
Use the corresponding *_GetNext method to read the actual value.

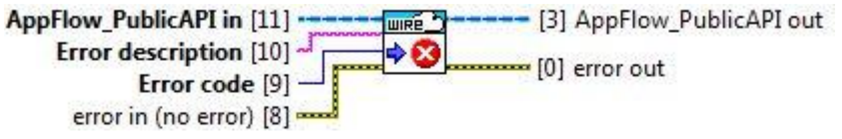


ForceSafetyChallengeUpdate.vi
Force an update of the safety challenge values, e.g. if a new dongle has been inserted, or if the dongle was inserted after the session started.
If dongle is found, "dongle found?" boolean will be TRUE.

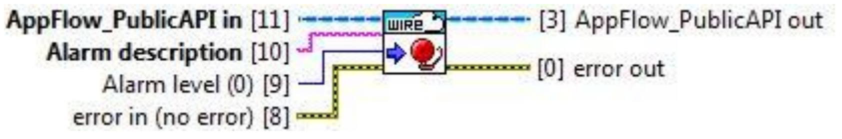


Writing Messages
Depending on the type of message to be updated, we have to use different methods.

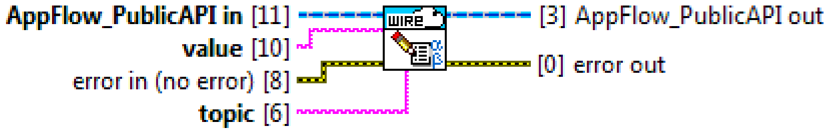
Error_Set.vi
Sets an error message for the current session.
Send an error with code = 0 to clear the error on the server (and on the mobile app)



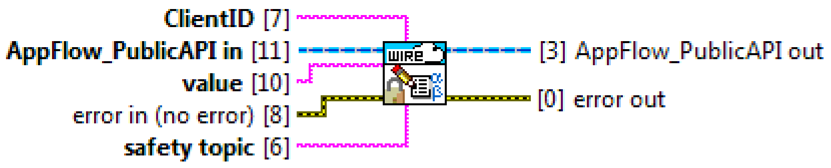
Alarm_Set.vi
Sets an alarm for the local ClientID, using the level and description specified.
Set alarm level to 0 to clear the alarm on the server (and on the mobile app).



MessageWrite.vi
Writes a normal topic to the server.



Security_MessageWrite.vi
Writes a security message to the server, using the WireFlow dongle specified at init for authentication.

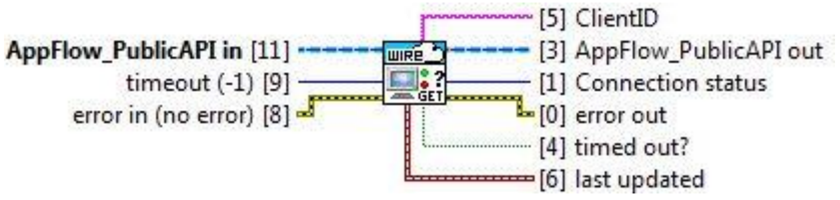


Reading Messages
Depending on the type of message to be updated, we have to use different methods to get the next message from the network.

Client_GetConnectMessage.vi
Waits for and reads next connection message, i.e. clients connecting or disconnecting from the server.

- 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 alarm

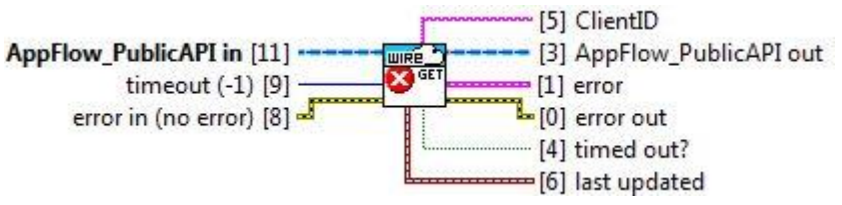
N.B. When connection status is "Link Lost" the client exited abnormally, i.e. didn't close correctly.



Error_GetNext.vi
Waits and reads the next error.

- 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.

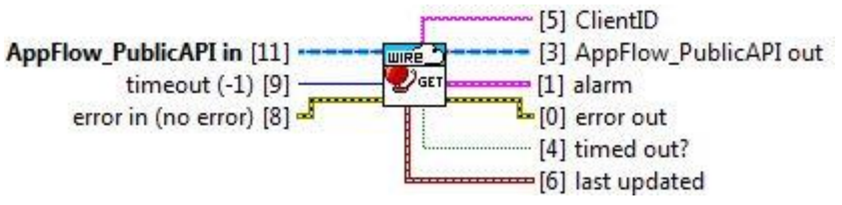
N.B. When code is 0 and description is empty, the error is resolved and removed by the remote client



Alarm_GetNext.vi
Waits for and reads next alarm.

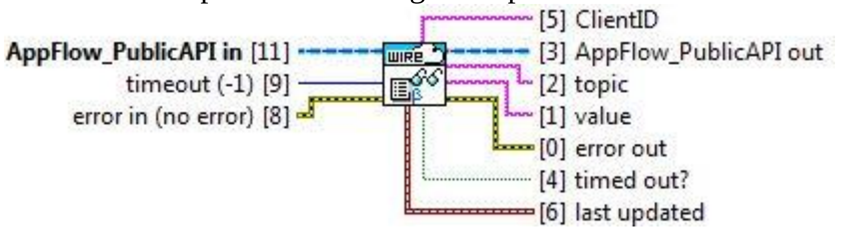
- ClientID = remote client ID that posted an alarm
- Alarm name = name of the alarm.
- alarm.descr = text posted by the remote client describing the alarm
- alarm.level = numeric value indicating the level of the alarm. The mobile app uses 0 as no alarm.
- alarm.timestamp = timestamp string from the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

N.B. When level is 0 and decription is empty, the alarm is resolved and removed by the remote client



Message_GetNext.vi
Waits and reads the next message.

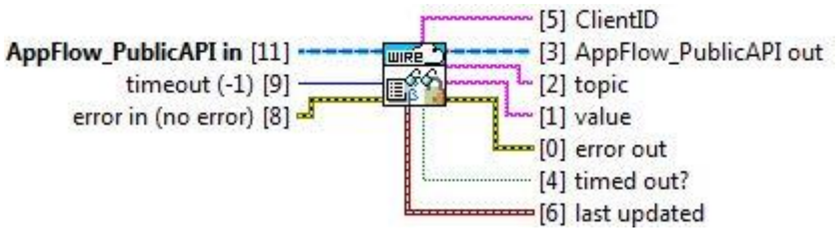
- ClientID = remote client ID that posted the message
- topic = name of the message.
- value = text posted by the remote client as the message value
- last update = timestamp when the background process received the message.



Security_GetNext.vi
Waits for and reads the next security message, i.e. a message signed by a security token.

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

N.B. Only messages that are correctly signed will be available in the message queue.

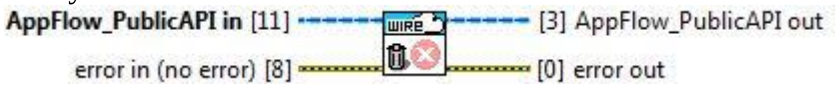


Removing Alarms/Errors

Alarms and Errors are normally posted with flags that make them persistent on the server, but it is possible to remove/hide them from new clients.

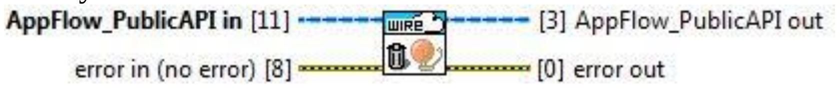
Error_Remove.vi

Removes a published error from the server, meaning that new connections to the server will not see this error anymore.



Alarm_Remove.vi

Removes a published alarm from the server, meaning that new connections to the server will not see this alarm anymore.



Examples

The driver comes with a number of examples that can be found using the LabVIEW Example Finder, just search for WireQueue.
The examples cover security messaging, logging as well as monitoring of all clients.

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)
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 should list 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.

Technical support and Professional services

This chapter should list support information and other services applicable to the product.
Please send all support questions to
support@wireflow.se