Using MQTT in Flowcode to Communicate between Embedded and App Developer Projects

Introduction

This worked example will detail a simple application for exchanging messages between embedded and non-embedded devices using the popular MQTT protocol. Flowcode is used to create three apps as follows:

- an embedded app running on an ESP32 device,
- a PC app (created using PC Developer) running on a Windows PC, and
- a web app (created using Web Developer) running in a web browser.

Flowcode is predominantly used to simplify the development of embedded software projects, but also allows users to develop apps that can interact with these embedded programs using Flowcode App Developer. App Developer has two modes – PC Developer creates apps that run on Windows PCs and Web Developer creates apps that run in a web browser.

MQTT is a light-weight messaging protocol for machine-to-machine (M2M) communication that is commonly used for Internet of Things (IoT) applications. It uses a publish/subscribe model where several clients connect to a single server (sometimes known as a broker).

MQTT clients can "publish" messages to the MQTT server. At their simplest, these messages consist of a topic name and a data payload, which can be a string or an array of bytes. These clients can also "subscribe" to topics, meaning that any published messages using that topic name will automatically be sent on by the server to them.

A central MQTT server is needed to coordinate the collection and distribution of these MQTT messages. There are several options available (free and commercial), and this worked example will use a local PC-based Mosquitto installation. Other options, including free on-line servers, will be also discussed.

More information about MQTT can be found on the Internet, where you will often see the term "broker" and "server" being used interchangeably. In this document, the term "server" will be used as this is the standardised term used within the latest MQTT specification.

The three Flowcode MQTT client apps are designed to run together as a system by exchanging messages with the MQTT server using the same 2 topics – "mtx" and "val". These apps will publish and subscribe messages using these two topics, allowing communication between the embedded device, the PC and any browser hosting the web app. An MQTT server will be needed, but each of the MQTT client apps are optional and can be developed independently (although at least 2 clients are preferable to show the communication between different clients).

Installing and Testing the MQTT Server

This section will provide steps to set up and test a local MQTT server on a PC using the open source Mosquitto server. Alternative MQTT servers can be used instead, and these are discussed later in this document.

Installing the server

At the time or writing, Mosquitto can be downloaded from this site:

https://mosquitto.org/download/

The version used by this worked example is v2.0.16 installed on a Windows PC, but the latest version should be fine too.

Once downloaded and installed, a few edits to the "mosquitto.conf" file are needed. The first is to specify the ports that clients will use to connect to the server. Flowcode-made apps require 2 ports to be available – "mqtt" (used by Embedded and PC Developer apps) and "websockets" (used by Web Developer apps). The following entries (just the highlighted ones) should be added to the "Listeners" section of the config file:

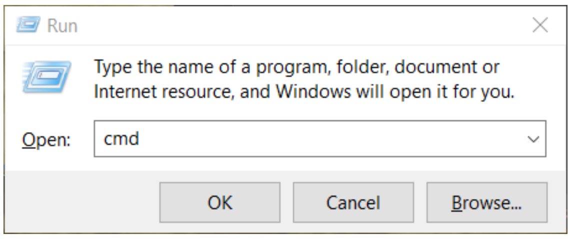
```
# listener port-number [ip address/host name/unix socket path]
#listener
listener 1883
protocol mqtt

listener 8883
protocol websockets
```

The second edit is to enable MQTT clients that are not on the local PC (such as the ESP32 device or a different PC on the network) to connect to the server. This requires finding the following entry under "Security" and replacing it with the highlighted one:

```
#allow_anonymous false
allow_anonymous true
```

Running the server

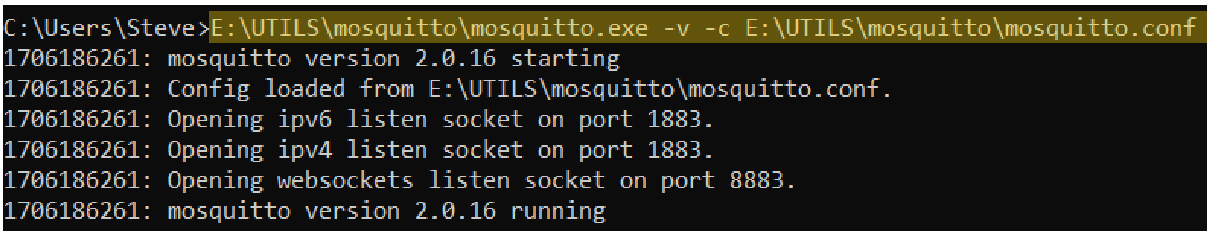
To start the server, first open a command prompt. For example, by pressing Windows-R and entering "cmd" (without quotes) in the "Run" dialog:


Run
Type the name of a program, folder, document or Internet resource, and Windows will open it for you.
Open: cmd
OK
Cancel
Browse...

In this, type the following and press return. Replace the highlighted %PATH_TO_MOSQUITTO% with the actual location of your local Mosquitto installation:

```
%PATH_TO_MOSQUITTO%\mosquitto.exe -v -c %PATH_TO_MOSQUITTO%\mosquitto.conf
```

For example, my installation is in "E:\UTILS\mosquitto" and so I enter the highlighted line below. This starts mosquitto in verbose mode, which forces the server to output information which will be helpful for debugging connection issues. Some of these debugging messages are output showing the version and the ports being used for MQTT messages:



```
C:\Users\Steve>E:\UTILS\mosquitto\mosquitto.exe -v -c E:\UTILS\mosquitto\mosquitto.conf
1706186261: mosquitto version 2.0.16 starting
1706186261: Config loaded from E:\UTILS\mosquitto\mosquitto.conf.
1706186261: Opening ipv6 listen socket on port 1883.
1706186261: Opening ipv4 listen socket on port 1883.
1706186261: Opening websockets listen socket on port 8883.
1706186261: mosquitto version 2.0.16 running
```

To stop the MQTT server, click on this command window and press Ctrl-C, then close that window. But this will not be necessary yet because it needs to be running to be able to test and use it!

Testing the server

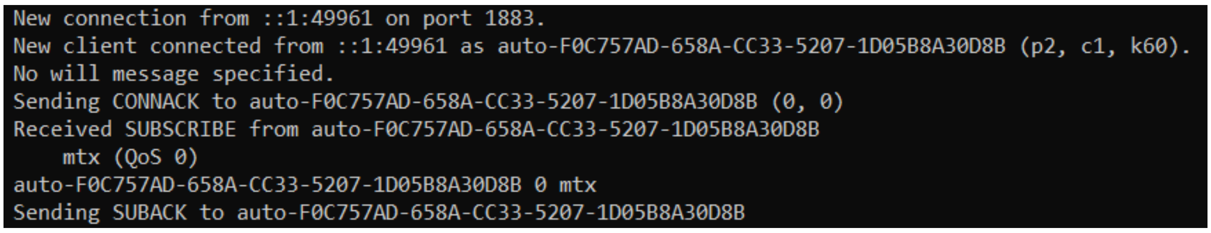
To test the server is working, two more command prompts can be used – one to subscribe to an MQTT topic and another to publish a message to that topic. This will mean we have 3 command prompt windows open:

- 1st = run the server and display debug info
- 2nd = subscribe to a topic and display received messages
- 3rd = publish a test message

To subscribe, enter the following into one of these new command prompt windows:

```
%PATH_TO_MOSQUITTO%\mosquitto_sub.exe -t "mtx" -p 1883
```

This should produce several debugging messages (in the original command window used to run the server) showing that a client has connected and has subscribed to the "mtx" message:

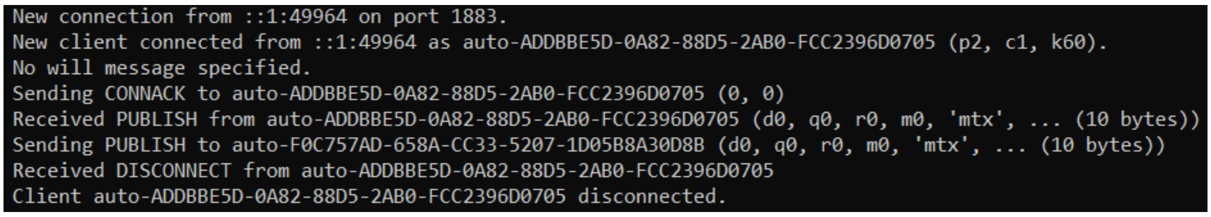


```
New connection from ::1:49961 on port 1883.
New client connected from ::1:49961 as auto-F0C757AD-658A-CC33-5207-1D05B8A30D8B (p2, c1, k60).
No will message specified.
Sending CONNACK to auto-F0C757AD-658A-CC33-5207-1D05B8A30D8B (0, 0)
Received SUBSCRIBE from auto-F0C757AD-658A-CC33-5207-1D05B8A30D8B
    mtx (QoS 0)
auto-F0C757AD-658A-CC33-5207-1D05B8A30D8B 0 mtx
Sending SUBACK to auto-F0C757AD-658A-CC33-5207-1D05B8A30D8B
```

In the 3rd command prompt window, enter the following to publish a message on this same "mtx" topic:

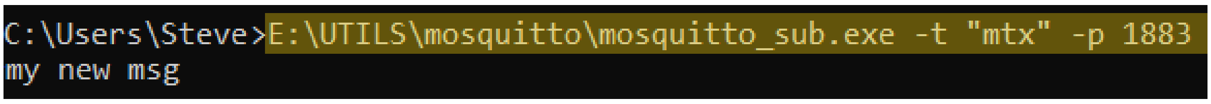
```
%PATH_TO_MOSQUITTO%\mosquitto_pub.exe -t "mtx" -m "my new msg" -p 1883
```

Again, this will produce some debugging messages in the first command prompt window showing client connection, reception of the message and sending it on to the subscribed client:



```
New connection from ::1:49964 on port 1883.
New client connected from ::1:49964 as auto-ADDBBE5D-0A82-88D5-2AB0-FCC2396D0705 (p2, c1, k60).
No will message specified.
Sending CONNACK to auto-ADDBBE5D-0A82-88D5-2AB0-FCC2396D0705 (0, 0)
Received PUBLISH from auto-ADDBBE5D-0A82-88D5-2AB0-FCC2396D0705 (d0, q0, r0, m0, 'mtx', ... (10 bytes))
Sending PUBLISH to auto-F0C757AD-658A-CC33-5207-1D05B8A30D8B (d0, q0, r0, m0, 'mtx', ... (10 bytes))
Received DISCONNECT from auto-ADDBBE5D-0A82-88D5-2AB0-FCC2396D0705
Client auto-ADDBBE5D-0A82-88D5-2AB0-FCC2396D0705 disconnected.
```

The message will also appear in the command prompt window used by the subscribing client:

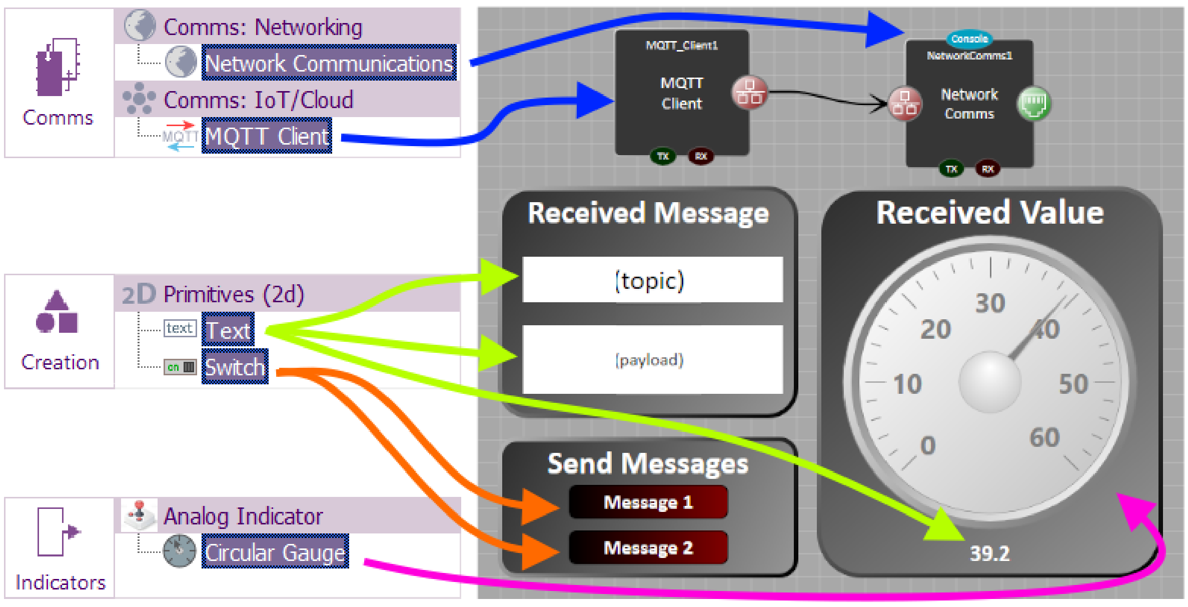


```
C:\Users\Steve>E:\UTILS\mosquitto\mosquitto_sub.exe -t "mtx" -p 1883
my new msg
```

PC Developer App

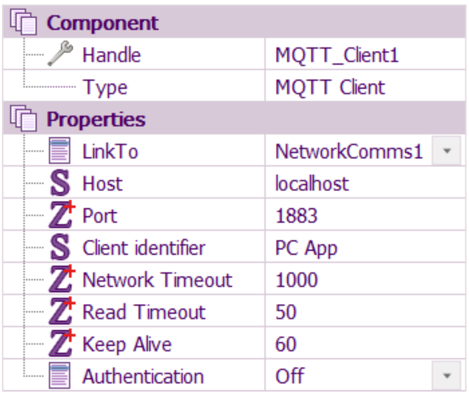
Flowcode PC Developer is used to create a Windows-based app that will exchange messages with the embedded app via the MQTT server. The first step is to add components to the 2d panel to create a user interface (UI) for the app and to provide communications facilities.

The image below shows a panel using added shapes and text to create a nice-looking UI, but only those components shown with the coloured arrows are necessary for this worked example. The panel itself is to the right, the components and their location within the components toolbar in Flowcode is shown on the left, and the name and purpose of each component shown below.



- MQTT_Client1 – holds the MQTT client code.
- NetworkComms1 – code for using the network.
- txtTopic – shows the topic name of a received message.
- txtPayload – shows the payload of a received message.
- txtValue – shows the value received on the “val” topic.
- swMsg1 – sends a message on the “mtx” topic.
- swMsg2 – sends another message on the “mtx” topic.
- gaugeValue - displays the value received on the “val” topic.

Most of these components can keep their default property values after being added to the panel, apart from the MQTT component which needs to use the properties below (assuming a Mosquitto server has been configured to run on the local network as previously outlined).

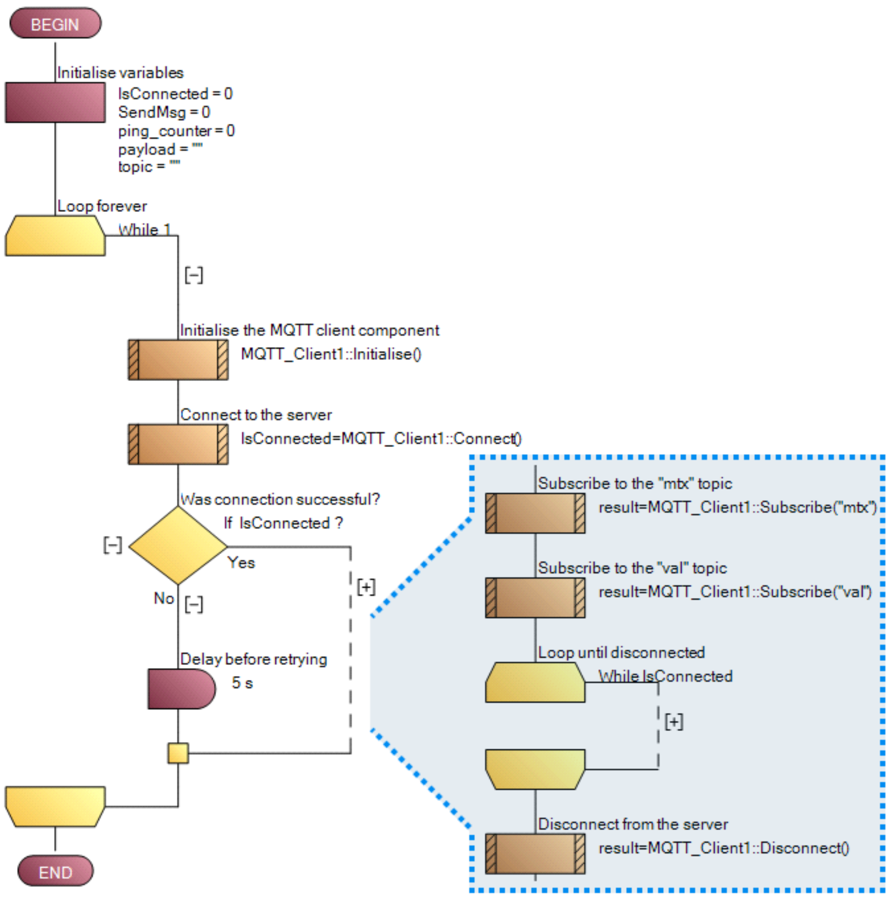


Component	
Handle	MQTT_Client1
Type	MQTT Client
Properties	
LinkTo	NetworkComms1
Host	localhost
Port	1883
Client identifier	PC App
Network Timeout	1000
Read Timeout	50
Keep Alive	60
Authentication	Off

Several global variables will be used within this program as shown in this table:

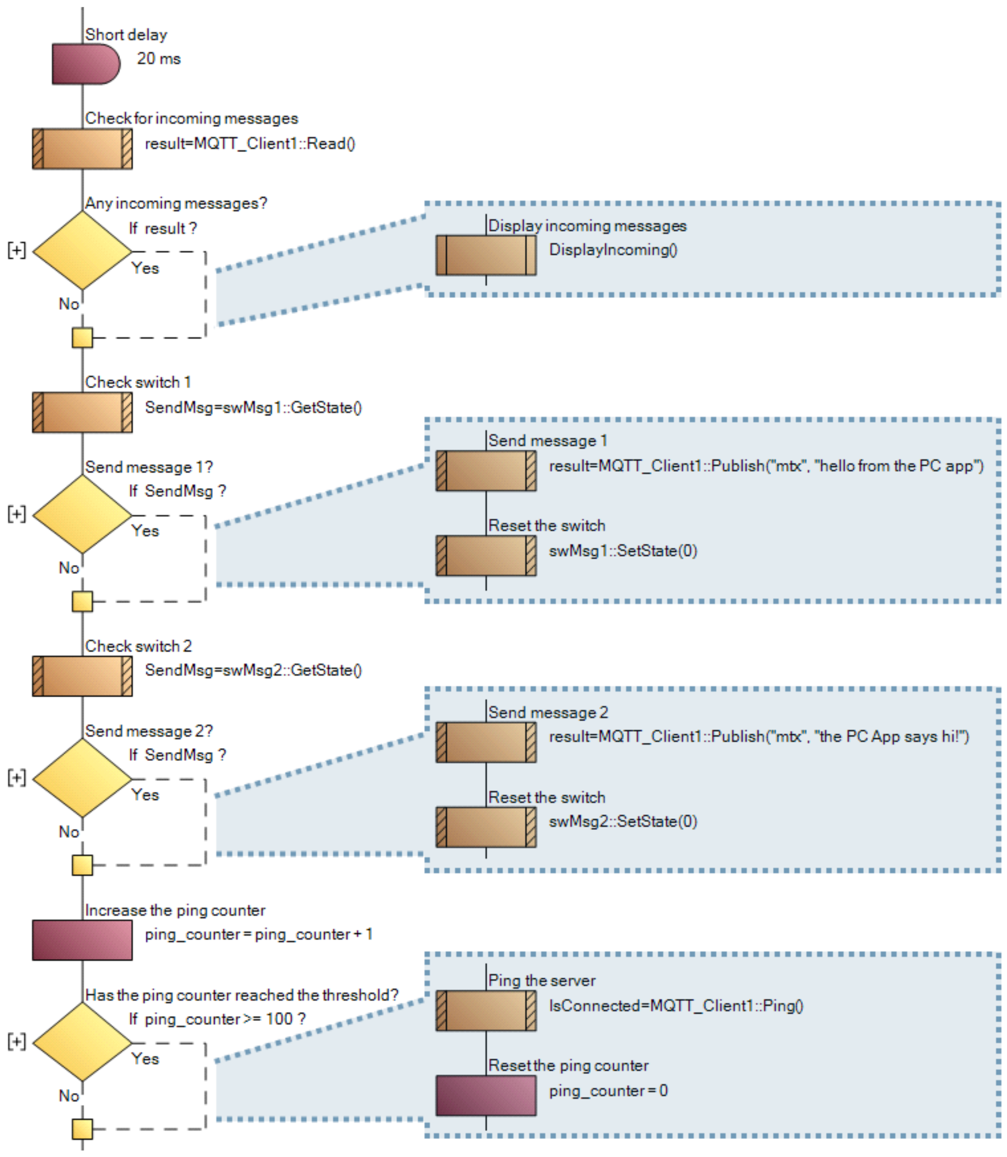
Variable name	Type	Purpose
IsConnected	Byte	Keeps track of MQTT server connection (0 = unconnected)
result	UInt	Used to hold the value of MQTT macro results
ping_counter	UInt	Counter to ensure the server is regularly 'pinged'
SendMsg	Bool	Flag to show if a message should be sent
topic[32]	String	The received message topic name
payload[100]	String	The received message payload (i.e. contents)
Value	Floating point	Stores the value received by the "val" message as a number

There are only 2 macros used in this program – "Main" and "DisplayIncoming" – with the former containing most of the code. Below shows the set-up code in the Main macro:



BEGIN
Initialise variables
IsConnected = 0
SendMsg = 0
ping_counter = 0
payload = ""
topic = ""
Loop forever
While 1
[-]
Initialise the MQTT client component
MQTT_Client1::Initialise()
Connect to the server
IsConnected=MQTT_Client1::Connect()
Was connection successful?
If IsConnected ?
[-]
Yes
No
[-]
[+]
Delay before retrying
5 s
END
Subscribe to the "mtx" topic
result=MQTT_Client1::Subscribe("mtx")
Subscribe to the "val" topic
result=MQTT_Client1::Subscribe("val")
Loop until disconnected
While IsConnected
[+]
Disconnect from the server
result=MQTT_Client1::Disconnect()

The global variables are first initialised and then the program enters an infinite loop which tries to connect to the MQTT server and if this fails, it delays before trying again. On successful connection, two MQTT message topics – "mtx" and "val" are subscribed to and another loop is entered. This inner loop is shown below and is executed continually until the connection to the server is no longer available.


Short delay
20 ms
Check for incoming messages
result=MQTT_Client1::Read()
Any incoming messages?
If result ?
[+]
Yes
No
Display incoming messages
DisplayIncoming()
Check switch 1
SendMsg=swMsg1::GetState()
Send message 1?
If SendMsg ?
[+]
Yes
No
Send message 1
result=MQTT_Client1::Publish("mtx", "hello from the PC app")
Reset the switch
swMsg1::SetState(0)
Check switch 2
SendMsg=swMsg2::GetState()
Send message 2?
If SendMsg ?
[+]
Yes
No
Send message 2
result=MQTT_Client1::Publish("mtx", "the PC App says hi!")
Reset the switch
swMsg2::SetState(0)
Increase the ping counter
ping_counter = ping_counter + 1
Has the ping counter reached the threshold?
If ping_counter >= 100 ?
[+]
Yes
No
Ping the server
IsConnected=MQTT_Client1::Ping()
Reset the ping counter
ping_counter = 0

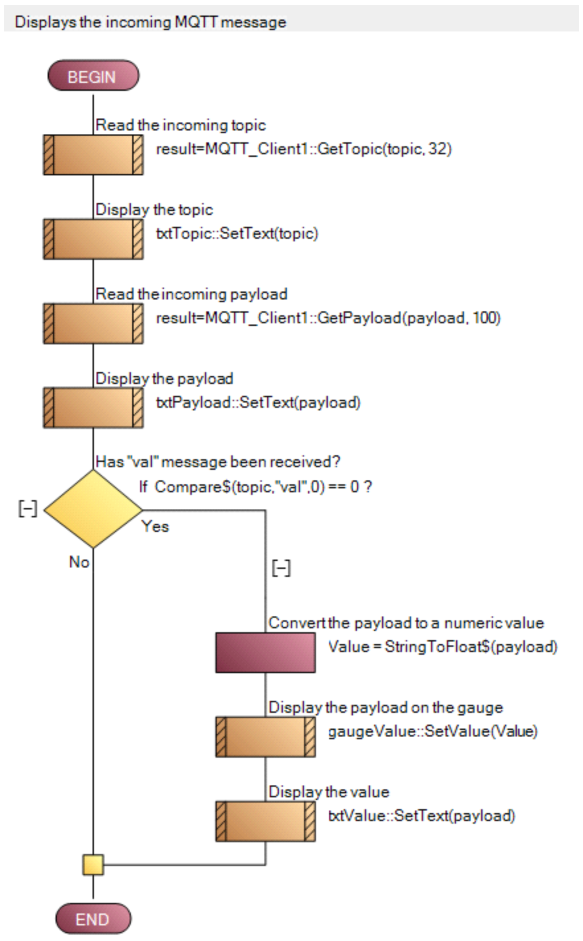
The loop starts with a check for subscribed messages and calls the DisplayIncoming macro if any have been received.

Next, each switch is checked in turn and if any are on then the appropriate outgoing message is published to the MQTT server. Any switch that is on is now turned off and can be pressed again by the user when the message needs to be sent again.

Finally, a ‘ping’ is sent to the MQTT server so that the connection to the server is maintained. This ‘ping’ also checks that the server is still available and connected. If it is not, the global “IsConnected” variable is set to zero which ends the inner loop and forces the program to reconnect with the server.

The DisplayIncoming macro read the topic and payload and sets the appropriate textbox with these values. The program has subscribed to only the “mtx” and “val” topics, so no others should be received.

If the “val” topic is received, the payload string is converted to a floating-point value and displayed on the gauge and in its associated textbox.


Displays the incoming MQTT message

BEGIN

Read the incoming topic
result=MQTT_Client1::GetTopic(topic, 32)

Display the topic
txtTopic::SetText(topic)

Read the incoming payload
result=MQTT_Client1::GetPayload(payload, 100)

Display the payload
txtPayload::SetText(payload)

Has "val" message been received?
If Compare$(topic,"val",0) == 0 ?

Convert the payload to a numeric value
Value = StringToFloat$(payload)

Display the payload on the gauge
gaugeValue::SetValue(Value)

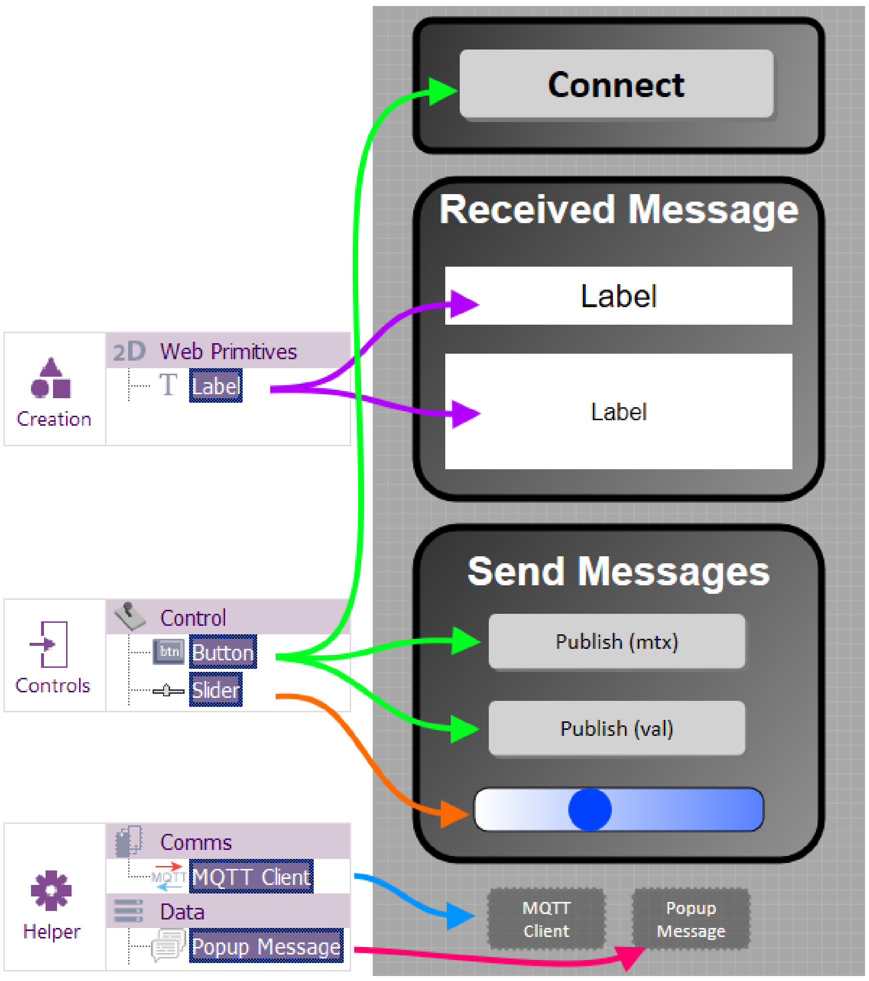
Display the value
txtValue::SetText(payload)

END

This project can now be deployed as a stand-alone PC app file the File…Export menu.

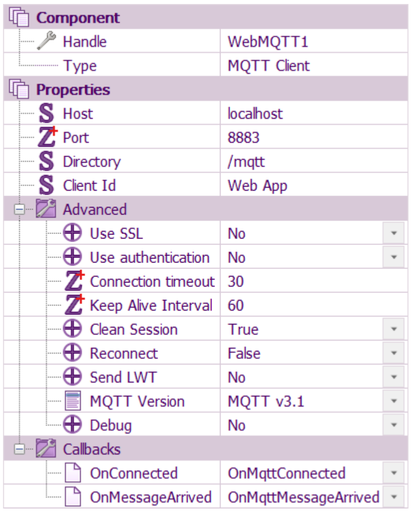
Web Developer app

As with the other apps, the UI of the web app will first be developed. Again, the panel has been constructed with added shapes in the image below to make it look nice, but the important components have been highlighted. The location of these components in the Components toolbar are shown to the left of the panel image and their names and function listed below.

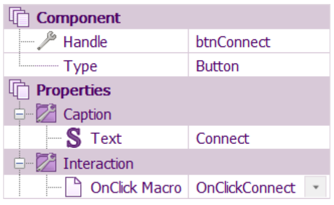


- btnConnect – button to initiate connection to the MQTT client.
- txtTopic – shows the topic name of a received message.
- txtMessage – shows the payload of a received message.
- btnSendMtx – sends a text message on the “mtx” topic.
- btnSendVal – sends a numeric value on the “val” topic.
- Slider – sets the value to send when btnSendVal is clicked.
- WebMQTT1 – holds the MQTT client code.
- WebPopup1 – code for displaying messages and getting user input.

The properties for the MQTT client component are shown below, assuming a local MQTT server is being used as described previously. Take special note that the Port property requires a different value to the Embedded and PC Developer apps. This is because, unlike the others, the Web Developer app uses the websockets protocol to communicate with the server.



Component	
Handle	WebMQTT1
Type	MQTT Client
Properties	
Host	localhost
Port	8883
Directory	/mqtt
Client Id	Web App
Advanced	
Use SSL	No
Use authentication	No
Connection timeout	30
Keep Alive Interval	60
Clean Session	True
Reconnect	False
Send LWT	No
MQTT Version	MQTT v3.1
Debug	No
Callbacks	
OnConnected	OnMqttConnected
OnMessageArrived	OnMqttMessageArrived



Component	
Handle	btnConnect
Type	Button
Properties	
Caption	
Text	Connect
Interaction	
OnClick Macro	OnClickConnect

Also note that 2 Callback macros need to be defined. These are for the “OnConnected” and “OnMessageArrived” notifications. Also shown above is some of the properties for the btnConnect button which is set to call a macro when a user clicks that button.

The use of macros for callbacks and user events (like button clicks) is commonly used in Web Developer apps. To create these macros, click the drop-down menu next to the property and select “<Add new>”. This will show a dialog box with the recommended parameters for the macro.

The macros associated with these and the other buttons are listed below:

Component	Property	Macro	Function
WebMQTT1	OnConnected	OnMqttConnected	Called when an MQTT connection to the server has been successfully established. Once received, the app can subscribe to topics.
	OnMessageArrived	OnMqttMessageArrived	Called when an MQTT message has arrived at this client.
btnConnect	OnClick Macro	OnClickConnect	The user will click this button to establish a connection with the MQTT server.
btnSendMtx	OnClick Macro	SendMtxClick	The user clicks this button to send an MQTT text message.
btnSendVal	OnClick Macro	SendValClick	The user clicks this button to send the value of the slider as an MQTT message.

The 5 macros listed above are the only user macros created for this Web Developer project. There is a “Main” macro, but this will contain no code (not even an infinite loop). This is different to traditional Embedded and PC Developer apps and is because Web Developer projects are constructed using an “event driven” approach.

Before discussing each individual macro, it is worth considering the overall flow of the program structure because this will also be different to the Embedded and PC Developer apps. In this simple Web Developer project, the flow of the program will be based on the various events that occur – i.e. when the user clicks a button, when a connection to the MQTT server is established and when an MQTT message arrives.

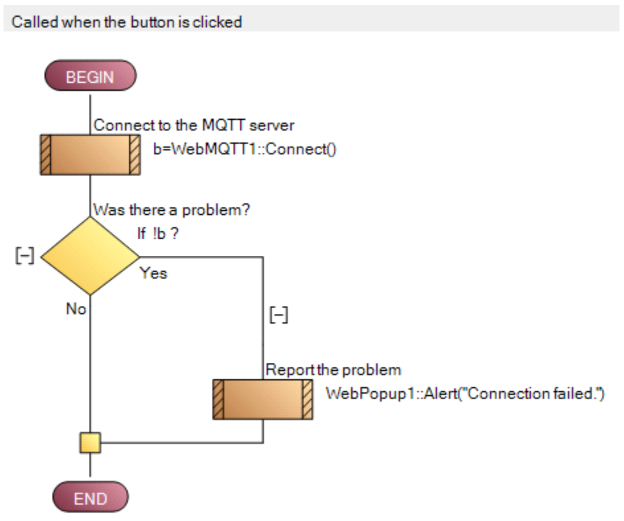
For example, the flow of the web app program when establishing a connection with the MQTT server is listed below:

1) Once the web app loads, none of its code will currently be executing.
2) When the user clicks the "Connect" button, the OnClickConnect macro is executed.
3) This macro initiates connection to the MQTT server.
4) The macro completes and at this point, no code will be running.
5) Once the MQTT server responds to confirm connection the OnMqttConnected macro will be executed.
6) This macro subscribes the web app to the MQTT topics.
7) Again, the macro completes and no code will be running.

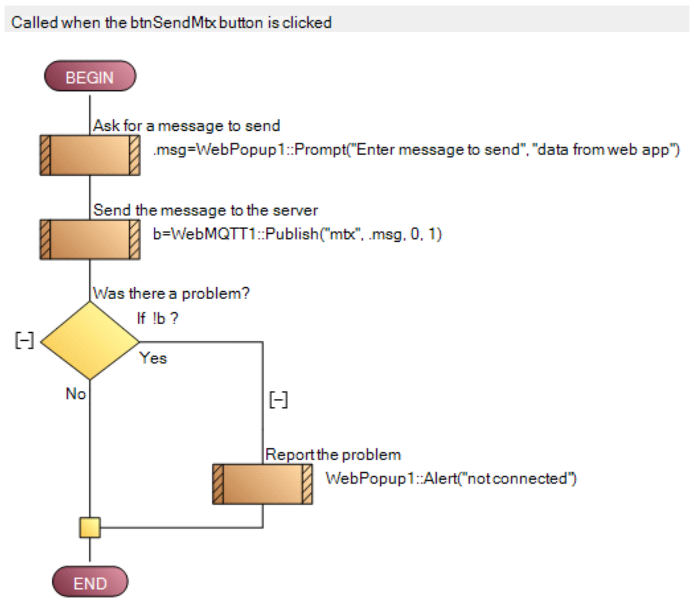
At this point, the MQTT server link has been established and the web app is subscribed to the required topics. The user could click a button to send a message, or a message could be sent by the server and received by this app. Each of these events will result in the associated macro being called.

All five macros in this web app are simple and contain just a few command icons. They generally use local variables, except a global Boolean variable ("b") is used to check the macro responses from the MQTT client component.

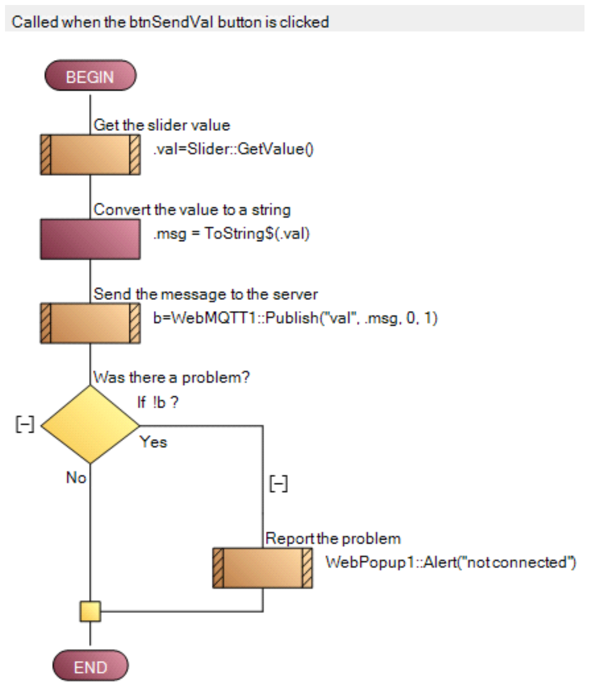
Here is the OnClickConnect macro, which initiates the connection to the MQTT server. The response is checked, and a message is displayed if it fails for some reason (for example, the server may already be connected).


Called when the button is clicked
BEGIN
Connect to the MQTT server
b=WebMQTT1::Connect()
Was there a problem?
If !b ?
[-]
Yes
No
[-]
Report the problem
WebPopup1::Alert("Connection failed.")
END

The following 2 macros show the button click macros for the “mtx” message and the “val message. The first one asks the user for the message text using the Popup Message component and the second uses the current value of the slider as the message payload.

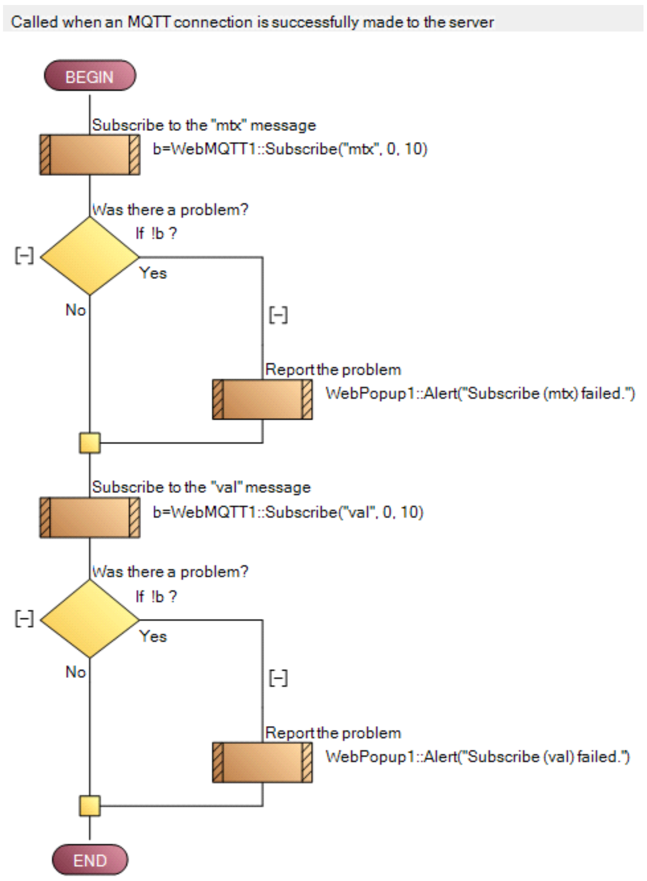

Called when the btnSendMtx button is clicked
BEGIN
Ask for a message to send
Send the message to the server
Was there a problem?
If !b ?
[-]
Yes
No
[-]
Report the problem
END

```
.msg=WebPopup1::Prompt("Enter message to send", "data from web app")
b=WebMQTT1::Publish("mtx", .msg, 0, 1)
WebPopup1::Alert("not connected")
```

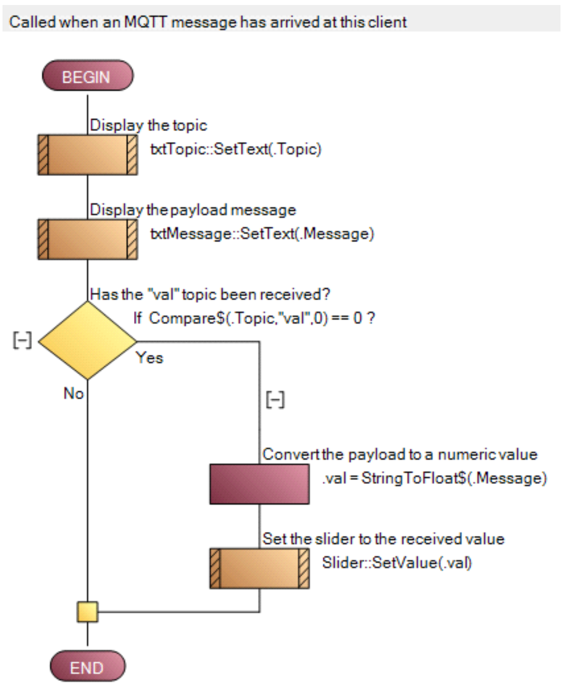

Called when the btnSendVal button is clicked
BEGIN
Get the slider value
Convert the value to a string
Send the message to the server
Was there a problem?
If !b ?
[-]
Yes
No
[-]
Report the problem
END

```
.val=Slider::GetValue()
.msg = ToString$(.val)
b=WebMQTT1::Publish("val", .msg, 0, 1)
WebPopup1::Alert("not connected")
```

And finally, the macros associated with the MQTT component callbacks are shown below. The first is called when a link has been established to the server and contains code that subscribes to the message topics. The other is called when an MQTT message arrives – displaying the text of the topic and payload in the web app and adjusting the slider value accordingly.


Called when an MQTT connection is successfully made to the server
BEGIN
Subscribe to the "mtx" message
Was there a problem?
[-]
Yes
No
[-]
Report the problem
Subscribe to the "val" message
Was there a problem?
[-]
Yes
No
[-]
Report the problem
END

```
b=WebMQTT1::Subscribe("mtx", 0, 10)
If !b ?
WebPopup1::Alert("Subscribe (mtx) failed.")
b=WebMQTT1::Subscribe("val", 0, 10)
If !b ?
WebPopup1::Alert("Subscribe (val) failed.")
```


Called when an MQTT message has arrived at this client
BEGIN
Display the topic
Display the payload message
Has the "val" topic been received?
[-]
Yes
No
[-]
Convert the payload to a numeric value
Set the slider to the received value
END

```
txtTopic::SetText(.Topic)
txtMessage::SetText(.Message)
If Compare$(.Topic,"val",0) == 0 ?
.val = StringToFloat$(.Message)
Slider::SetValue(.val)
```

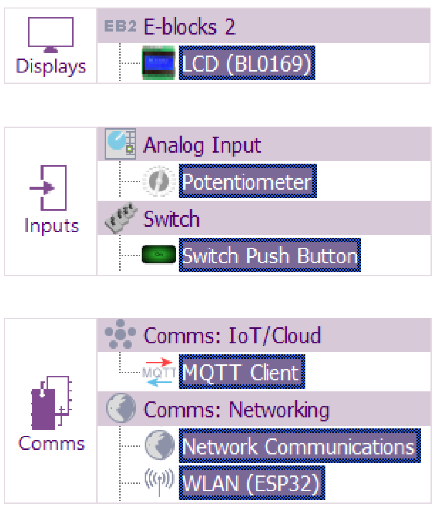
The web app can be created using the Deploy option in Flowcode's File…Export menu or the "Create web page" item in the Build menu. The resulting HTML file can be opened in a web browser on the local PC, a mobile phone, or similar device.

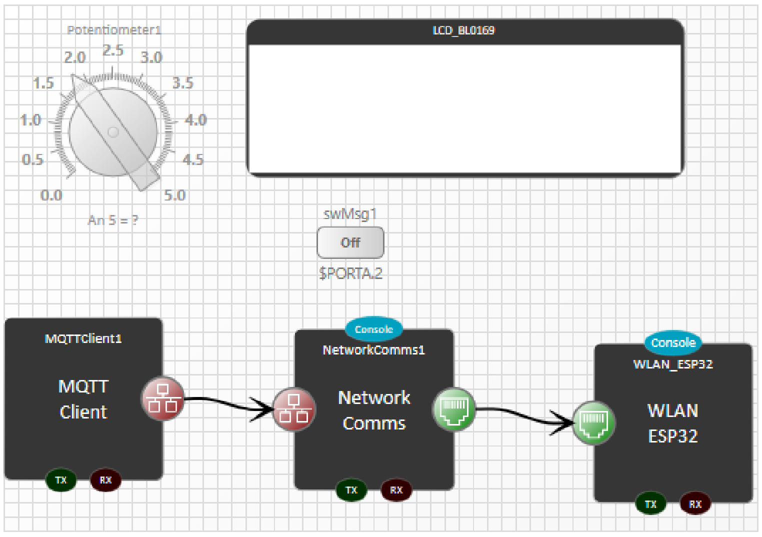
Embedded App

This is the most complicated of the three Flowcode apps and is based on an ESP32 board connected to a single button switch, a potentiometer, and an LCD.

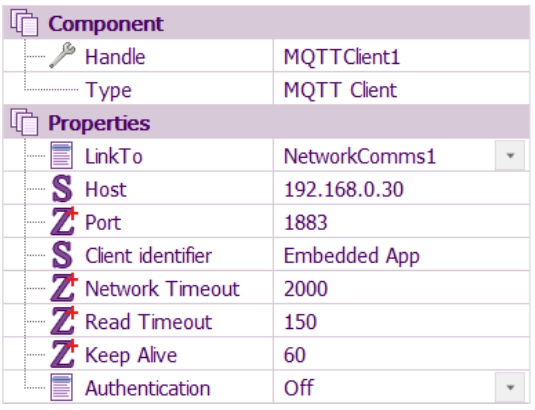
Component panel

The 2D panel holds six components – three are required to enable web-based communication and the other three are used for user I/O on the hardware. The panel is shown below, with the location of the components in the toolbar shown on the left.

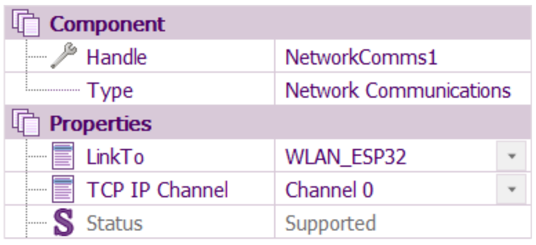




The properties of the three I/O components should be set to reflect the connections in the actual hardware. The WLAN and NetworkComms component use default properties, although the latter needs to be linked to the WLAN component using the 'LinkTo' property. Similarly, the MQTT Client component needs to be linked to the NetworkComms component. It also needs to be configured to connect to the MQTT server. These settings are shown below.



Component	
Handle	MQTTClient1
Type	MQTT Client
Properties	
LinkTo	NetworkComms1
Host	192.168.0.30
Port	1883
Client identifier	Embedded App
Network Timeout	2000
Read Timeout	150
Keep Alive	60
Authentication	Off



Component	
Handle	NetworkComms1
Type	Network Communications
Properties	
LinkTo	WLAN_ESP32
TCP IP Channel	Channel 0
Status	Supported

The 'Host' property will be the IP address of the computer on the local network that is running the MQTT server or, if applicable, the web address of the remote MQTT server being used.

The panel components have the following functions:

- MQTTClient1 – contains the MQTT client code.
- NetworkComms1 – code for enabling TCP/IP comms on the network.
- WLAN_ESP32 – code for communicating via the local Wi-Fi.
- LCD_BL0169 – displays received messages as well as status and error reports.
- swMsg1 – represents the switch connected to the ESP32 for initiating sending of a message.
- Potentiometer1 – reads the value of the potentiometer connected to the ESP32.

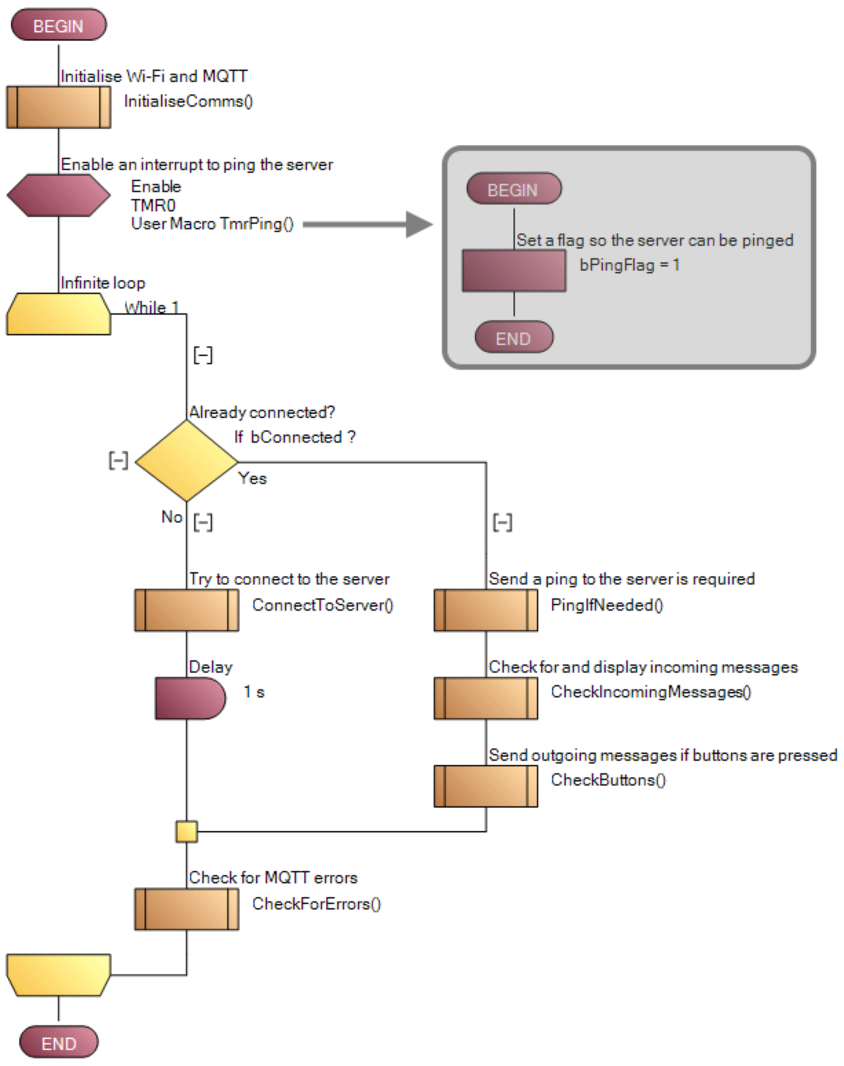
This embedded app has several user macros that are described below. It also uses the following global variables:

Variable name	Type	Purpose
bConnected	Bool	Keeps track of MQTT server connection (0 = unconnected)
bPingFlag	Bool	Timer interrupt sets this to ensure the server remains connected
bWifiOk	Bool	Set when the app has successfully connected to the W0Fi
iError	Byte	Holds the value of an error if one is detected
iResponse	Byte	General use for holding the response of an MQTT operation
sData[200]	String	Used to create messages to display and to send as an MQTT message

Macro: Main

For clarity, this program has been split into several user macros which are called during the execution of the Main macro.

After first initialising the Wi-Fi and MQTT comms, this macro enables a timer interrupt (set to a period of 30 seconds) which calls a small macro that simply sets the bPingFlag variable. An infinite loop is then started which connects to the MQTT server or responds to incoming messages and user input. Also in this loop, any error conditions are checked for.



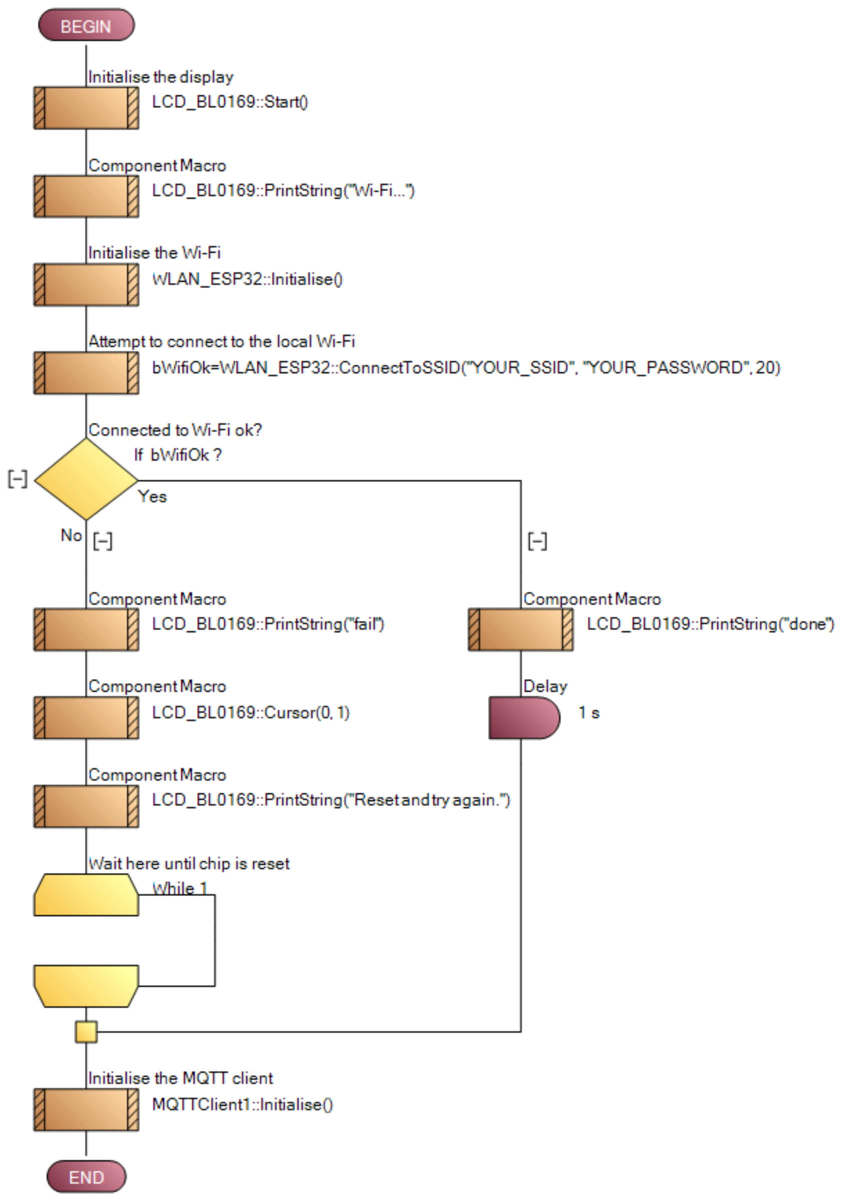
BEGIN
Initialise Wi-Fi and MQTT
InitialiseComms()
Enable an interrupt to ping the server
Enable
TMR0
User Macro TmrPing()
BEGIN
Set a flag so the server can be pinged
bPingFlag = 1
END
Infinite loop
While 1
Already connected?
If bConnected ?
Yes
No
Try to connect to the server
ConnectToServer()
Delay
1 s
Send a ping to the server is required
PingIfNeeded()
Check for and display incoming messages
CheckIncomingMessages()
Send outgoing messages if buttons are pressed
CheckButtons()
Check for MQTT errors
CheckForErrors()
END

Macro: InitialiseComms

After the WLAN component is initialised, an attempt is made to connect to the local Wi-Fi network. The name and password for the network must be entered correctly in the call to ConnectToSSID before the program is compiled and downloaded to the ESP32.

If the Wi-Fi connection fails, the program will display an error message on the display and then enter an infinite loop. This probably indicates that the network name and/or password are incorrect.

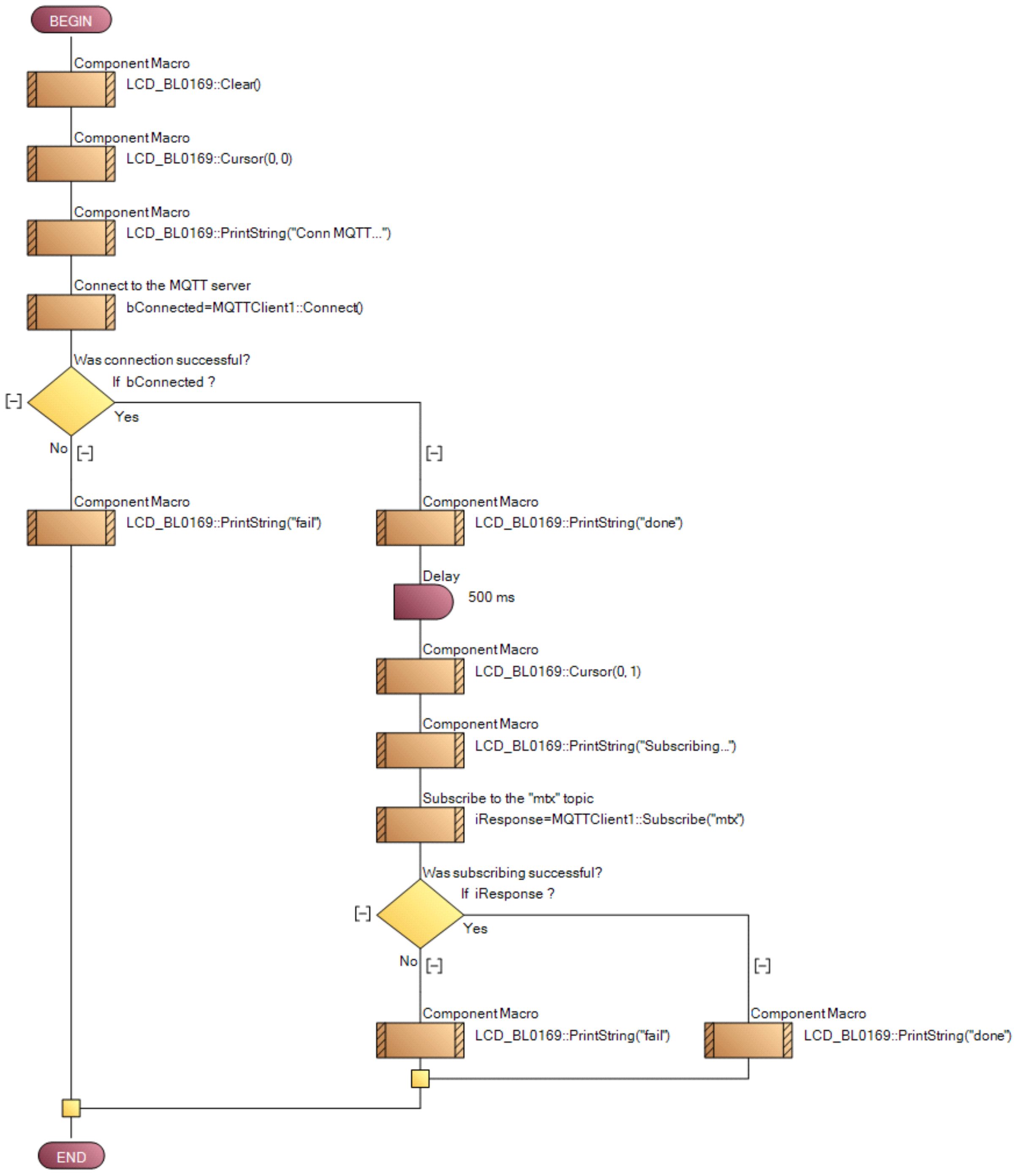
If the Wi-Fi connection is established, the MQTT Client component is initialised and the macro returns.


BEGIN
Initialise the display
LCD_BL0169::Start()
Component Macro
LCD_BL0169::PrintString("Wi-Fi...")
Initialise the Wi-Fi
WLAN_ESP32::Initialise()
Attempt to connect to the local Wi-Fi
bWifiOk=WLAN_ESP32::ConnectToSSID("YOUR_SSID", "YOUR_PASSWORD", 20)
Connected to Wi-Fi ok?
If bWifiOk ?
Yes
No
Component Macro
LCD_BL0169::PrintString("fail")
Component Macro
LCD_BL0169::Cursor(0, 1)
Component Macro
LCD_BL0169::PrintString("Reset and try again.")
Wait here until chip is reset
While 1
Component Macro
LCD_BL0169::PrintString("done")
Delay
1 s
Initialise the MQTT client
MQTTClient1::Initialise()
END

Macro: ConnectToServer

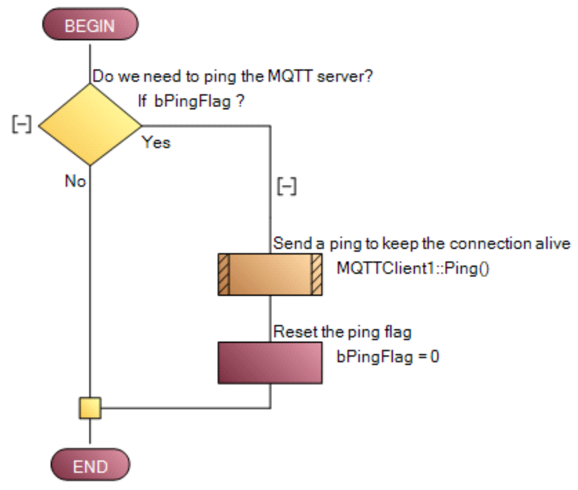
When the infinite loop of the Main macro is first executed, the bConnected flag will be set to zero, which indicates that the device is not yet connected to the MQTT server, and the following macro will be executed.

This macro attempts to connect to the MQTT server using the settings in the MQTT Client properties. If connection is successful, the program will subscribe to the "mtx" topic. Messages are displayed on the LCD to show the progress of this process.


BEGIN
Component Macro
LCD_BL0169::Clear()
Component Macro
LCD_BL0169::Cursor(0, 0)
Component Macro
LCD_BL0169::PrintString("Conn MQTT...")
Connect to the MQTT server
bConnected=MQTTClient1::Connect()
Was connection successful?
If bConnected ?
Yes
No
Component Macro
LCD_BL0169::PrintString("fail")
Component Macro
LCD_BL0169::PrintString("done")
Delay
500 ms
Component Macro
LCD_BL0169::Cursor(0, 1)
Component Macro
LCD_BL0169::PrintString("Subscribing...")
Subscribe to the "mtx" topic
iResponse=MQTTClient1::Subscribe("mtx")
Was subscribing successful?
If iResponse ?
Yes
No
Component Macro
LCD_BL0169::PrintString("fail")
Component Macro
LCD_BL0169::PrintString("done")
END

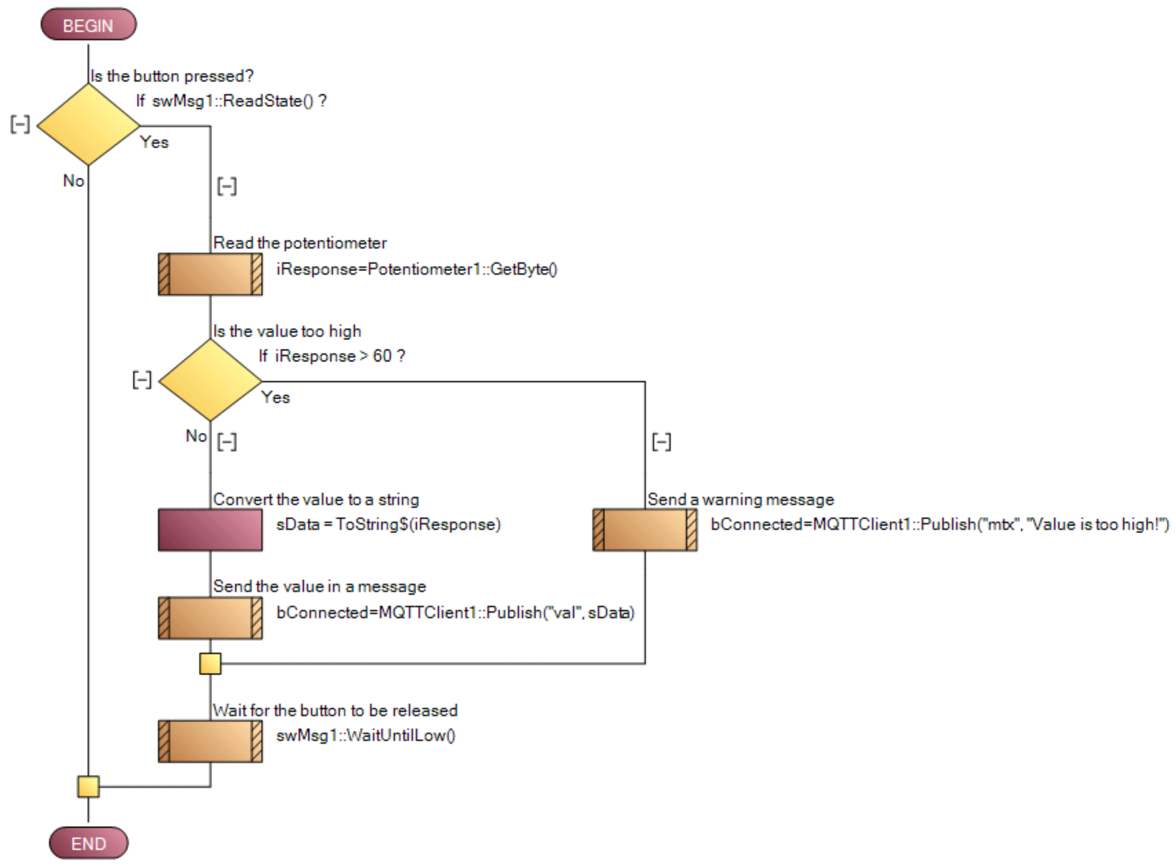
Macro: PingIfNeeded

If the MQTT server connection is valid, the Main macro will call three macros. The first checks to see if the bPingFlag variable has been set and, if so, a ‘ping’ message is sent to the MQTT server to ensure the connection remains established. This flag is set to “1” every 30 seconds using an interrupt and the associated macro.



Macro: CheckButtons

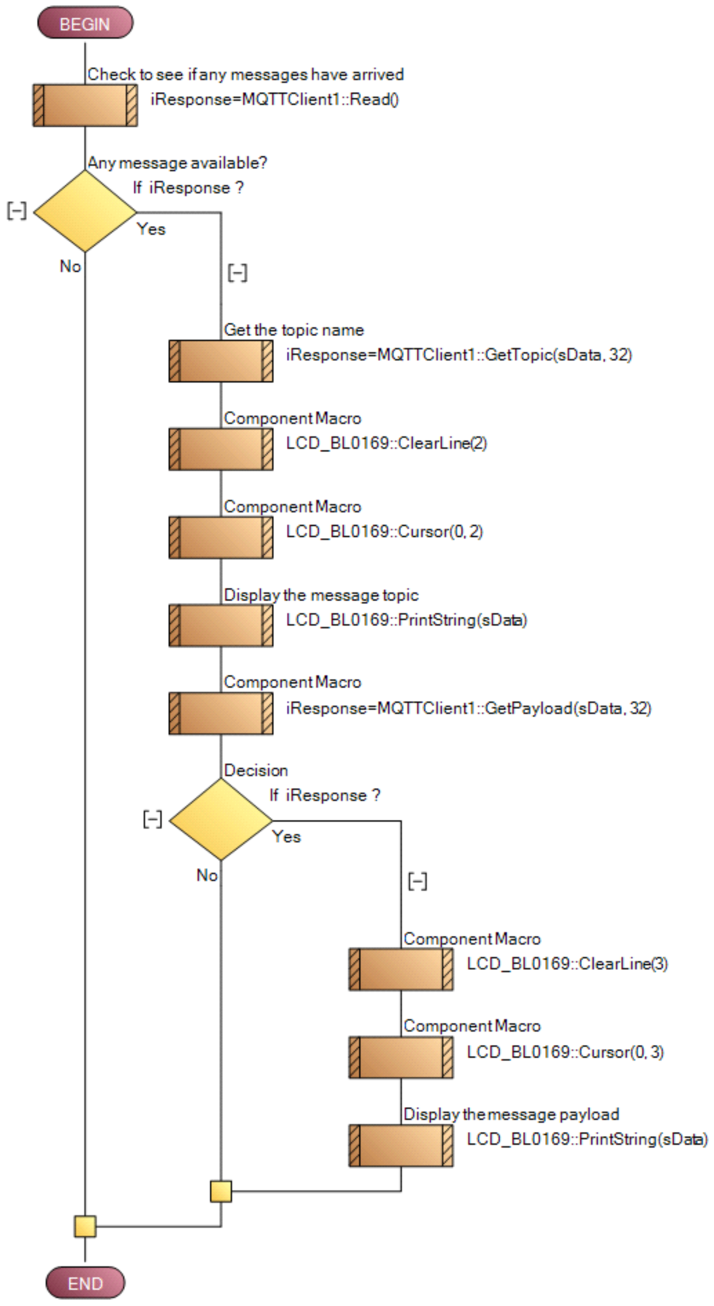
Another macro called within the Main loop checks the state of the button and if pressed it publishes an MQTT message. This message depends on the potentiometer reading – values above 60 cause an error message to be sent on the “mtx” topic and lower values send the value on the “val” topic.


BEGIN
Is the button pressed?
If swMsg1::ReadState() ?
[-]
Yes
No
[-]
Read the potentiometer
iResponse=Potentiometer1::GetByte()
Is the value too high
If iResponse > 60 ?
[-]
Yes
No
[-]
[-]
Convert the value to a string
sDataa = ToString$(iResponse)
Send a warning message
bConnected=MQTTClient1::Publish("mtx", "Value is too high!")
Send the value in a message
bConnected=MQTTClient1::Publish("val", sData)
Wait for the button to be released
swMsg1::WaitUntilLow()
END

Macro: CheckIncomingMessages

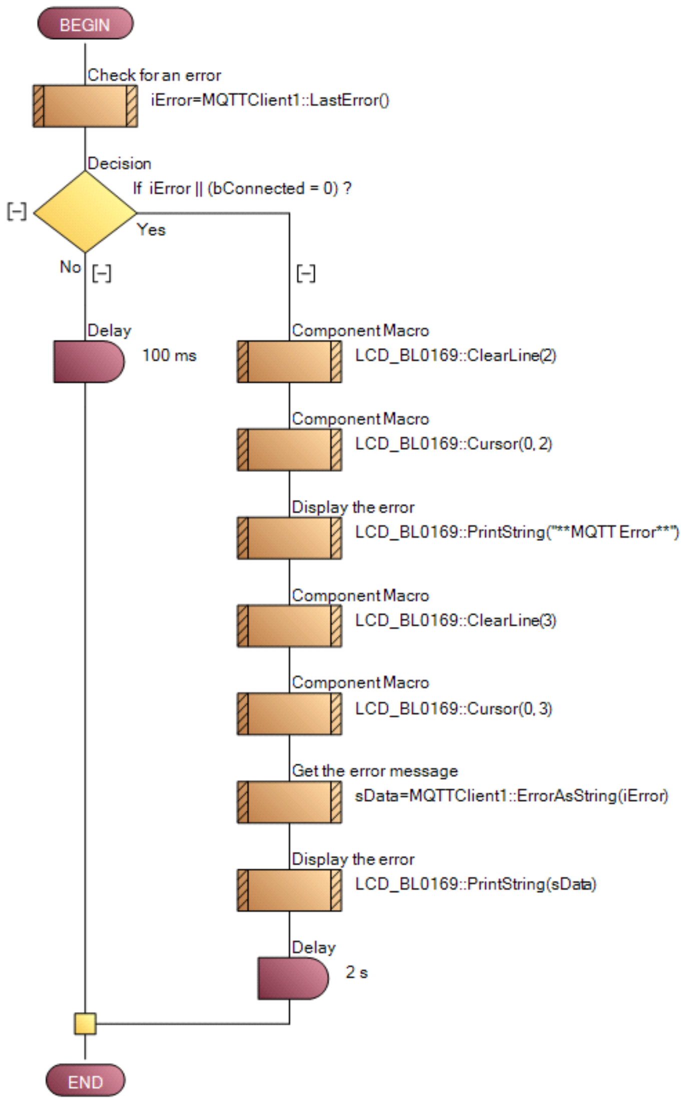
This macro is called before the previous macro and checks to see if an MQTT message has been received. If one is available, its topic name and payload are read and then output to the display.

This program subscribes to the “mtx” message and so only that message will be received by this app.



Macro: CheckForErrors

The LCD is also used to display any errors that have occurred. This macro is called at the end of the main loop and checks the MQTT Client component to see if any errors have occurred. If they have, or if the app has disconnected from the server, the error is displayed.



Compiling the project

Before compiling this project, the COM port used to connect to and reprogram the ESP32 device needs to be set. To do this, open the Project Options screen (in Flowcode's Build menu) and set the correct port in the Programmer Port drop-down.

It is also important to set the correct Wi-Fi network name and password in the call to WLAN_ESP32::ConnectToSSID in the InitialiseComms macro.

To compile the project and send the app to the ESP32 device, select "Compile to target" from the Build menu.

MQTT Alternatives

The MQTT server used in this worked example is Mosquitto v2.0.16 on a Windows PC, but other solutions exist and can be used in preferred.

Mosquitto itself can be installed on several different platforms – local Mac or Linux PCs, single-board computers such as the Raspberry Pi, or a web server. In this case, the installation and configuration settings will dictate the required MQTT Client component properties within Flowcode.

Online MQTT servers also exist that are free to use publicly for test purposes. For example, here are details for two popular free servers:

broker.hivemq.com

- Host: broker.hivemq.com
- Ports:
  - 1883: TCP Port
  - 8883: TLS TCP Port
  - 8000: WebSocket Port
  - 8884: TLS WebSocket Port

test.mosquitto.org

- Host: test.mosquitto.org
- Ports:
  - 1883 : MQTT, unencrypted, unauthenticated
  - 1884 : MQTT, unencrypted, authenticated
  - 8883 : MQTT, encrypted, unauthenticated
  - 8884 : MQTT, encrypted, client certificate required
  - 8885 : MQTT, encrypted, authenticated
  - 8886 : MQTT, encrypted, unauthenticated
  - 8887 : MQTT, encrypted, server certificate deliberately expired
  - 8080 : MQTT over WebSockets, unencrypted, unauthenticated
  - 8081 : MQTT over WebSockets, encrypted, unauthenticated
  - 8090 : MQTT over WebSockets, unencrypted, authenticated
  - 8091 : MQTT over WebSockets, encrypted, authenticated

Note that few servers should be used with caution as they are designed for testing purposes only and so message delivery is not guaranteed or could be limited. The server might be closed for maintenance or access could be blocked (if abused). And as they are publicly available, messages can be received and send by others using the same topic name.

For commercial projects, a paid-for server should be used instead. Several companies offer commercial MQTT servers, with various levels of service, features, and cost.

Conclusion

This document shows how easy it is to use Flowcode to create a set of apps that communicate with each other using the popular MQTT protocol. These apps can run on a range of network-connected devices such as ESP32s and other embedded devices, Windows PCs, and mobile phones.

It has discussed the three types of projects that can be created using Flowcode – Embedded, PC Developer and Web Developer – and shown how to use the MQTT Client components in each to interface with an MQTT server and communicate with each other by subscribing to and publishing MQTT topics.

It has also shown how to set up a local Mosquitto MQTT server and provided suggestions for alternative MQTT servers that already exist on the Internet.

The concepts used provide a firm foundation in using MQTT with Flowcode and can be extended to create many diverse types of application that need to communicate simple messages across a local or global network.