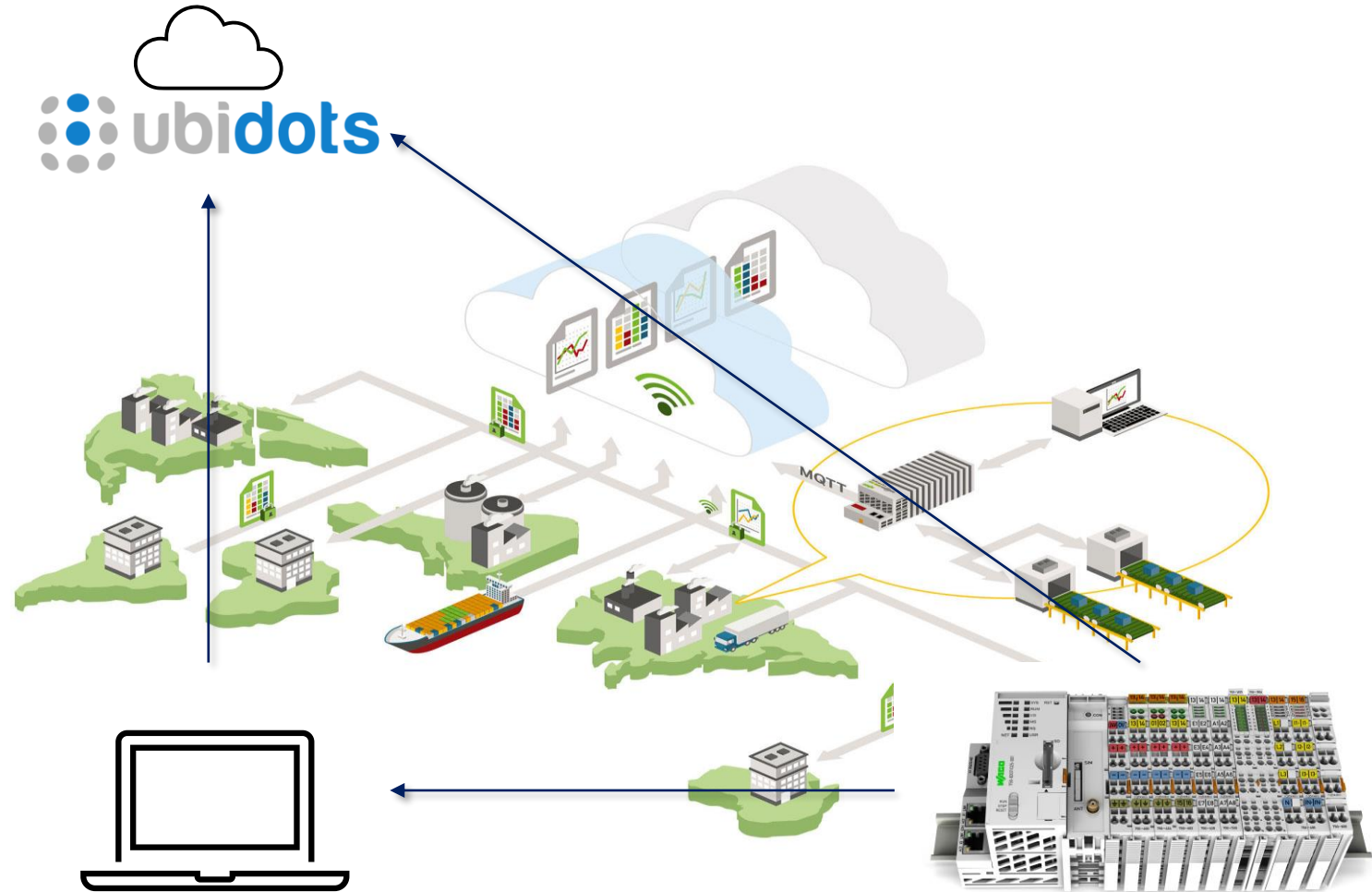


## Connecting the Wago PFC200 to Ubidots Via Codesys

### Glossary:

1. Ubidots Setup
  - a) Login
  - b) Create New Device
  - c) Get API Label, ID, & Token
  - d) Download PEM Certs
2. PFC200 Setup
  - a) Go online
  - b) WBM Cloud Connectivity
3. Codesys Setup
  - a) Create New Project
  - b) Add Device
  - c) Create POU and Task
  - d) Go online
4. The Code (MQTT Publishing)
  - a) The MQTT library
  - b) Example 1: Simple Hard-code
  - c) The JSON library
  - d) Example 2: Parameterized JSON



## Connecting the Wago PFC200 to Ubidots Via Codesys

### Ubidots Setup

1a) Login your Ubidots Account

1b)

Go to the 'Devices' tab on the top of the page, then click 'devices' in the drop-down.

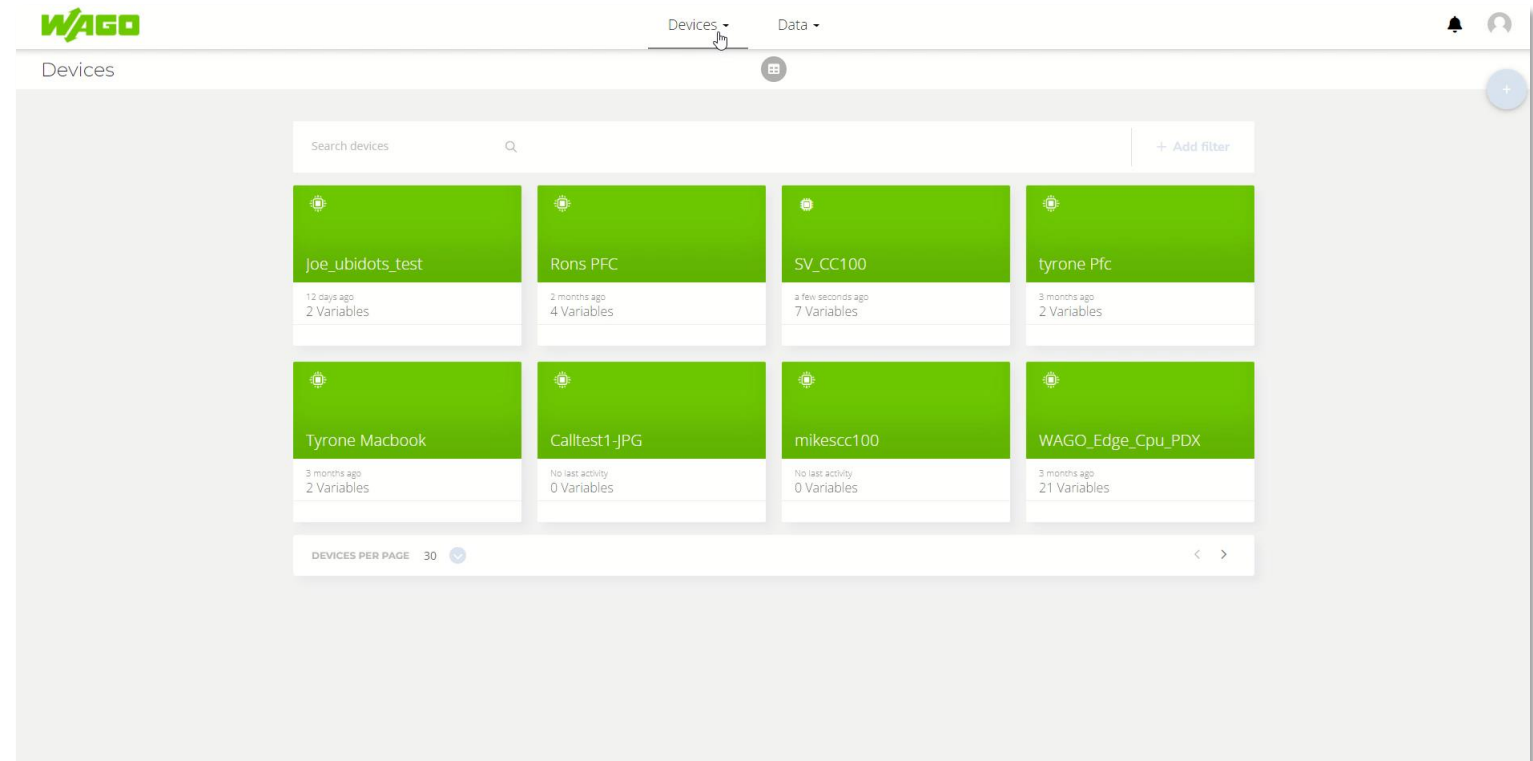
Click the '+' symbol on the top right corner of the screen.

Select 'Blank device'

Create a device name and then you should see the new device pop up.



[Video: Add device in Ubidots](#)



## Connecting the Wago PFC200 to Ubidots Via Codesys

### Ubidots Setup

1c)

Click on the newly created device and record the API Label, ID, and Token of the device.

You'll need it for later.

1d)

You'll need the Ubidots PEM certificate for TLS encryption to secure messages going to the cloud.

Found here:

<https://docs.ubidots.com/v1.6/reference/broker-urls>

Click on the link PEM cert download link then save the file as 'roots.crt'

The screenshot shows the Ubidots device page for a device named 'joes\_test'. The page has a green header with the device name. Below the header, there are several fields: 'Description' (with a 'Change description' link), 'API Label' (containing 'joes\_test'), 'ID' (containing '63c...'), 'Token' (containing a masked token), and 'Tags' (with an 'Add new tag' button). The 'API Label', 'ID', and 'Token' fields are highlighted with colored boxes (blue, purple, and green respectively).

The screenshot shows the Ubidots Configuration page. The page has a white background with a 'Configuration' header. Below the header, there are several settings: 'Enabled' (checked), 'Cloud platform' (MQTT AnyCloud), 'Hostname' (industrial.api.ubidots.com), 'Port number' (8883), 'Client ID' (63c...), 'Clean session' (checked), 'TLS' (checked), 'Last Will' (unchecked), 'User' (BBFF-1...), 'Password' (empty), and 'CA file' (/etc/ssl/certs/roots.crt). The 'Client ID' and 'User' fields are highlighted with colored boxes (purple and green respectively).

### MQTT

Security	Ubidots Account	Endpoint	Port
No TLS	Industrial	industrial.api.ubidots.com	1883
TLS	Industrial	industrial.api.ubidots.com	8883

You can download the Ubidots PEM certificate for TLS [here](#)

## Connecting the Wago PFC200 to Ubidots Via Codesys

[Video: PFC Config in WBM](#)

### PFC200 Setup

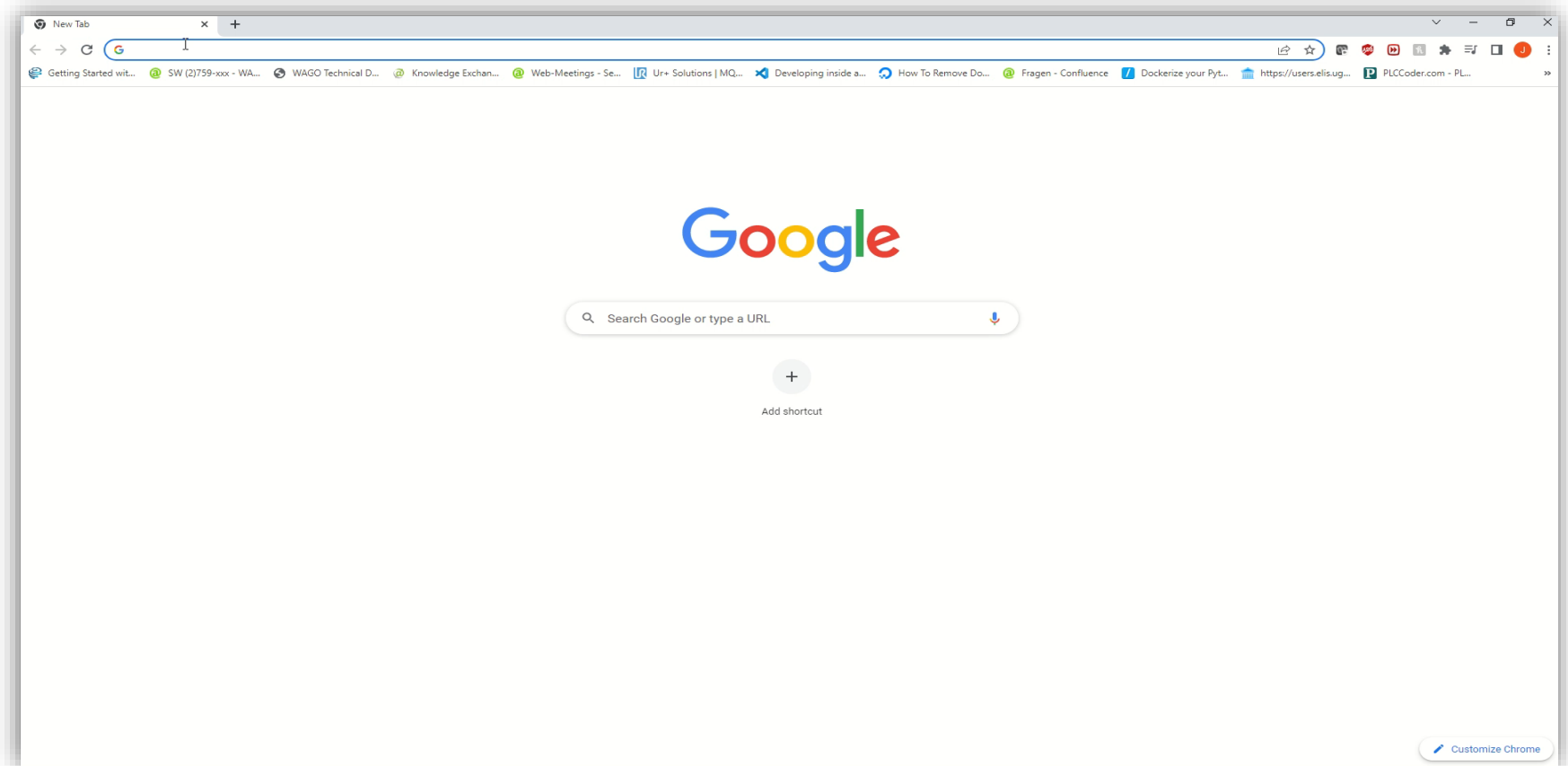
2a)

i. Log onto the WBM of your PFC200 by using any web browser and type in the IP address of your device.

ii. Type in default username/password = admin/wago if prompted.

iii. Ensure the FW of the device is FW23 or above. Then check that the IP address, gateway, and subnet are all set for an outbound connection.

iv. Check that your clock on the PFC200 is formatted to the correct time/timezone



Note: My router is set to IP address 192.168.1.1, thus I've used it as the gateway address

Note: If you need to change the FW use the below link as a guide:

<https://www.youtube.com/watch?v=6CxdrnHlIMo>

<https://github.com/WAGO/pfc-firmware>

## Connecting the Wago PFC200 to Ubidots Via Codesys

[Video: Confirm online connection via SSH](#)

### PFC200 Setup

2a)

v. Verify that you are connected online by using any ssh program to access the PFC terminal.

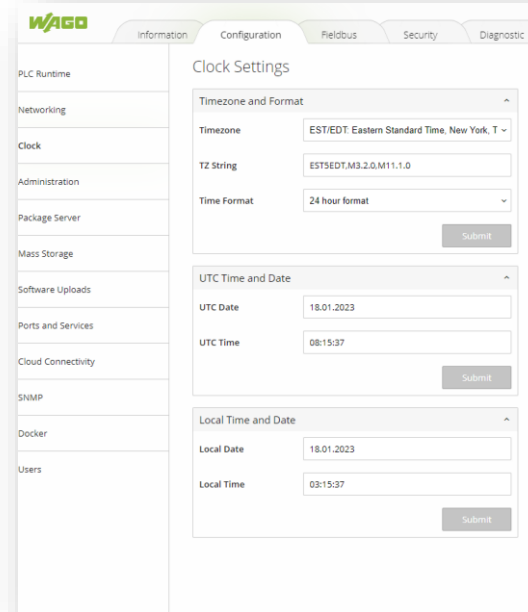
default credentials: root/wago

ping www.google.com

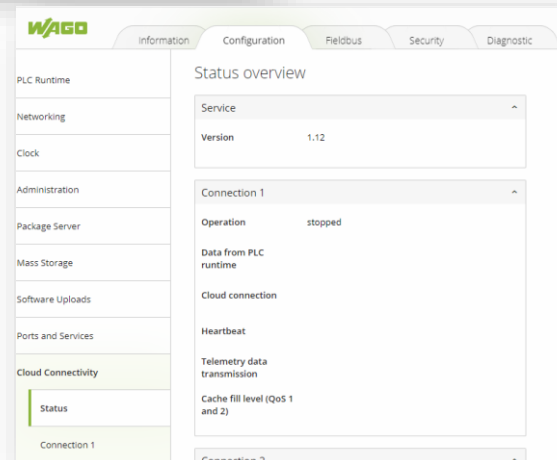
If you receive an immediate response with ping statistics, then you are connected.

2b)

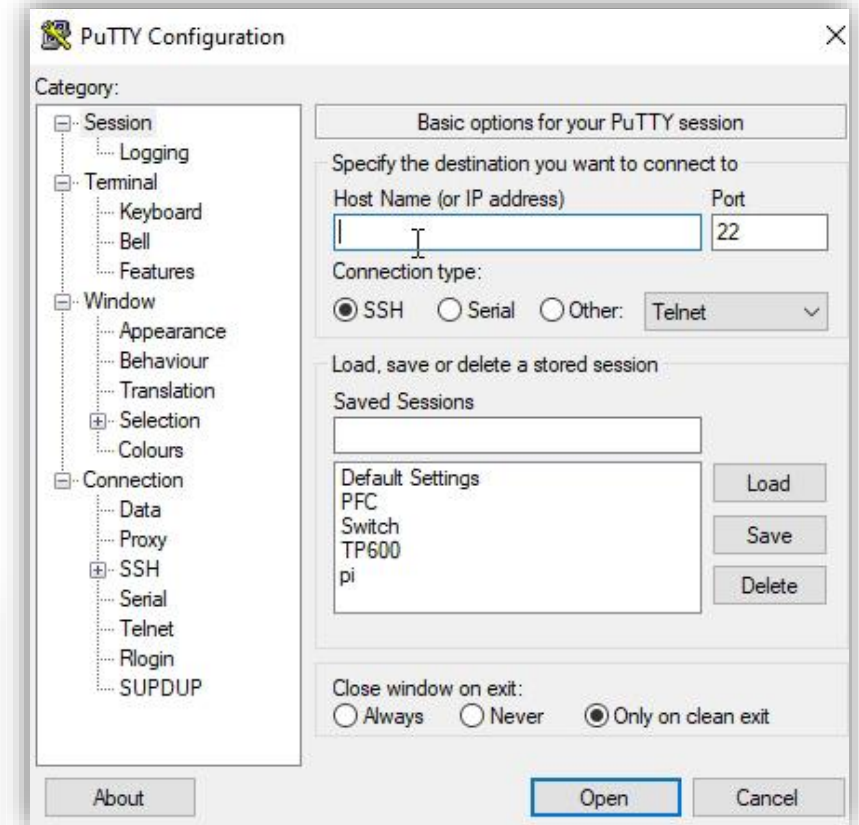
i. Go to the 'Cloud connectivity' tab under 'Configuration'. Click on 'Connection 1'. Here is where we input the Ubidots credentials.



The image shows the 'Clock Settings' configuration page in the WAGO PFC200 web interface. The left sidebar contains a menu with options: PLC Runtime, Networking, Clock, Administration, Package Server, Mass Storage, Software Uploads, Ports and Services, Cloud Connectivity, SNMP, Docker, and Users. The main content area is titled 'Clock Settings' and has three sections: 'Timezone and Format' with fields for Timezone (EST/EDT: Eastern Standard Time, New York, T), TZ String (ESTEDT.M3.2.0.M11.1.0), and Time Format (24 hour format); 'UTC Time and Date' with fields for UTC Date (18.01.2023) and UTC Time (08:15:37); and 'Local Time and Date' with fields for Local Date (18.01.2023) and Local Time (03:15:37). Each section has a 'Submit' button.



The image shows the 'Status overview' page in the WAGO PFC200 web interface. The left sidebar is the same as in the previous image. The main content area is titled 'Status overview' and contains a 'Service' section with a 'Version' field showing '1.12'. Below this is a 'Connection 1' section with a 'Operation' field showing 'stopped'. Further down, there are sections for 'Data from PLC runtime', 'Cloud connection', 'Heartbeat', and 'Telemetry data transmission'. At the bottom, there is a 'Cache fill level (QoS 1 and 2)' field.



The image shows the PuTTY Configuration dialog box. The 'Category' list on the left includes Session, Logging, Terminal, Keyboard, Bell, Features, Window, Appearance, Behaviour, Translation, Selection, Colours, Connection, Data, Proxy, SSH, Serial, Telnet, Rlogin, and SUPDUP. The 'Basic options for your PuTTY session' section on the right has fields for 'Host Name (or IP address)' and 'Port' (22). The 'Connection type' section has radio buttons for SSH (selected), Serial, and Other, with a dropdown menu set to 'Telnet'. The 'Load, save or delete a stored session' section has a 'Saved Sessions' list with 'Default Settings', 'PFC', 'Switch', 'TP600', and 'pi'. The 'Close window on exit' section has radio buttons for Always, Never, and Only on clean exit (selected). The 'Open' button is highlighted with a blue border.

## Connecting the Wago PFC200 to Ubidots Via Codesys

### PFC200 Setup

2b)

ii. Once on the 'Connection 1' page, select 'MQTT AnyCloud'.

Input the Hostname  
'industrial.api.ubidots.com'

iii. Click the 'TLS' box then use port number 8883.

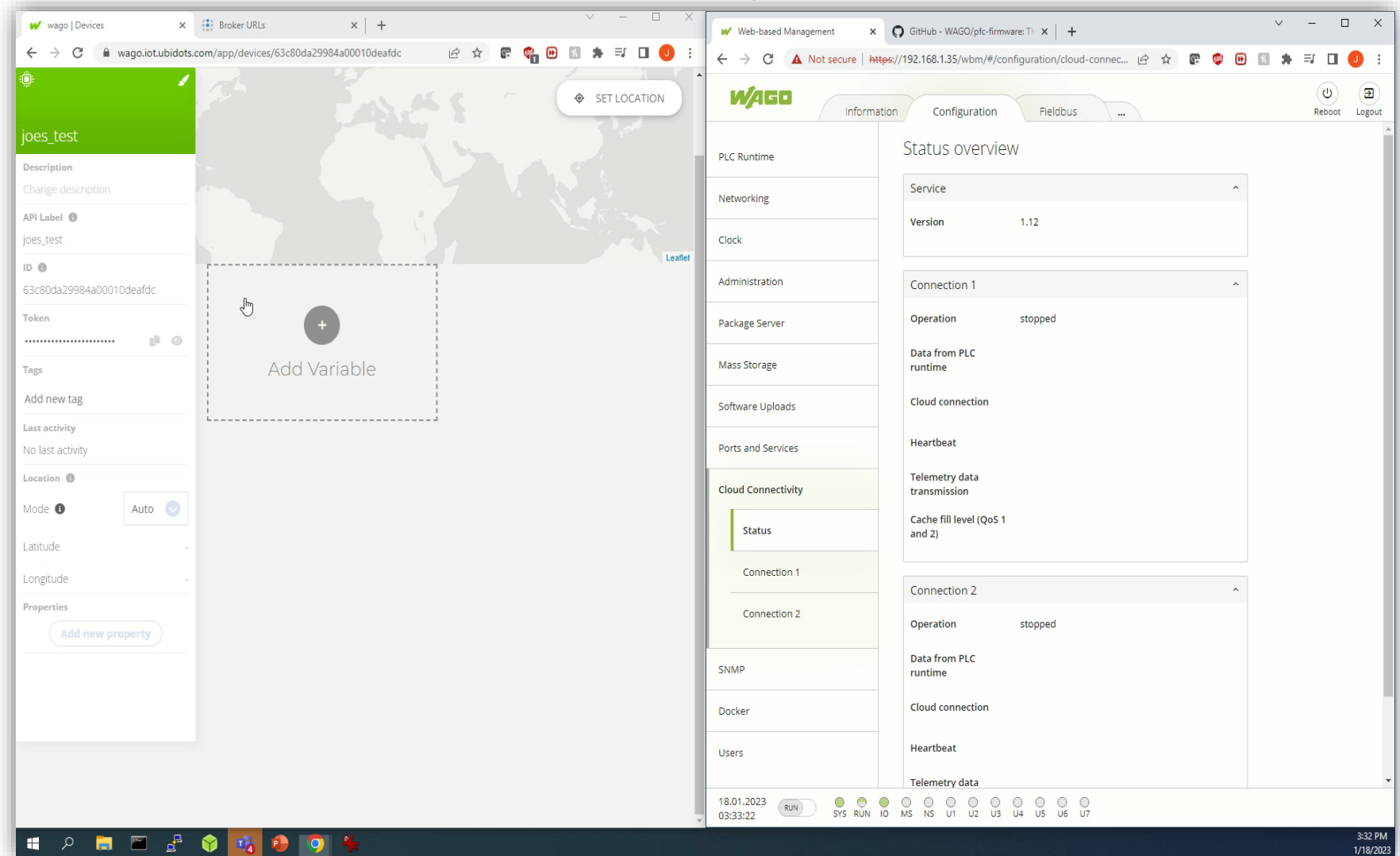
iv. Input the Ubidots ID into the Client ID field of the Wago WBM.

v. Input the Ubidots Token into the User field in the Wago WBM.

vi. Select 'Native MQTT' as the Data protocol in the WBM.

vii. Enable FTP via the 'Ports and Services' tab, then FTP into the Wago controller. Once this is done, insert the root.crt file into the CA file path 'etc/ssl/certs/roots.crt'.

[Video: Cloud setup in WBM](#)



The screenshot displays the WAGO Web-based Management (WBM) interface. The left sidebar shows the 'joes\_test' device configuration page with fields for Description, API Label, ID, Token, Tags, Location, Mode, Latitude, Longitude, and Properties. The main area shows a map with a 'SET LOCATION' button and an 'Add Variable' button. The right sidebar shows the 'Status overview' page with tabs for Information, Configuration, and Fieldbus. The 'Status overview' page displays the service status (stopped), data from PLC runtime, cloud connection status (stopped), heartbeat, and telemetry data transmission. The bottom status bar shows the date and time (18.01.2023 03:33:22) and a 'RUN' button.

## Connecting the Wago PFC200 to Ubidots Via Codesys

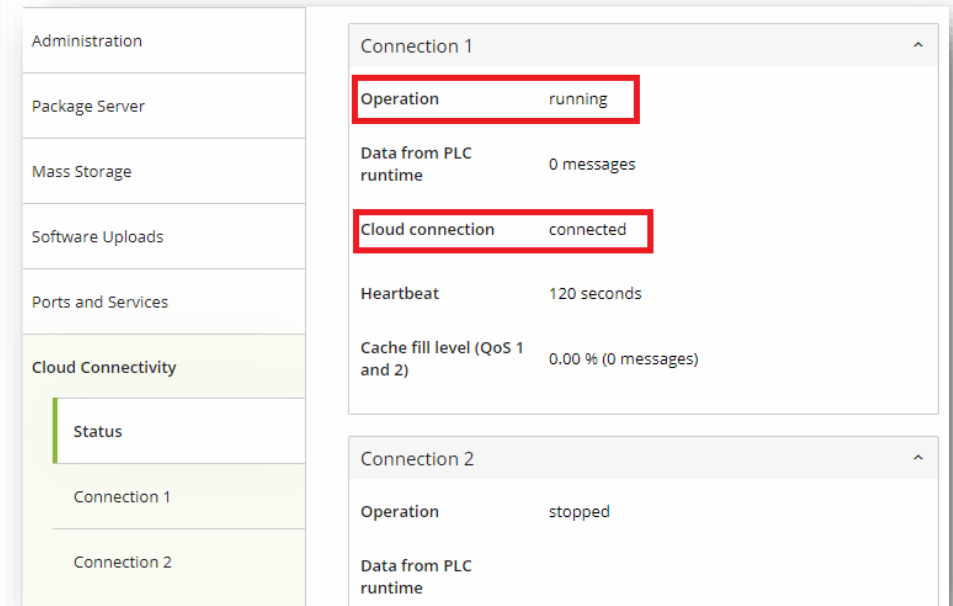
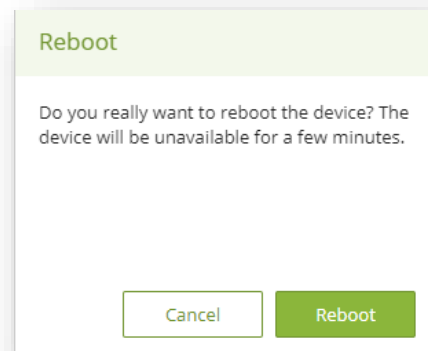
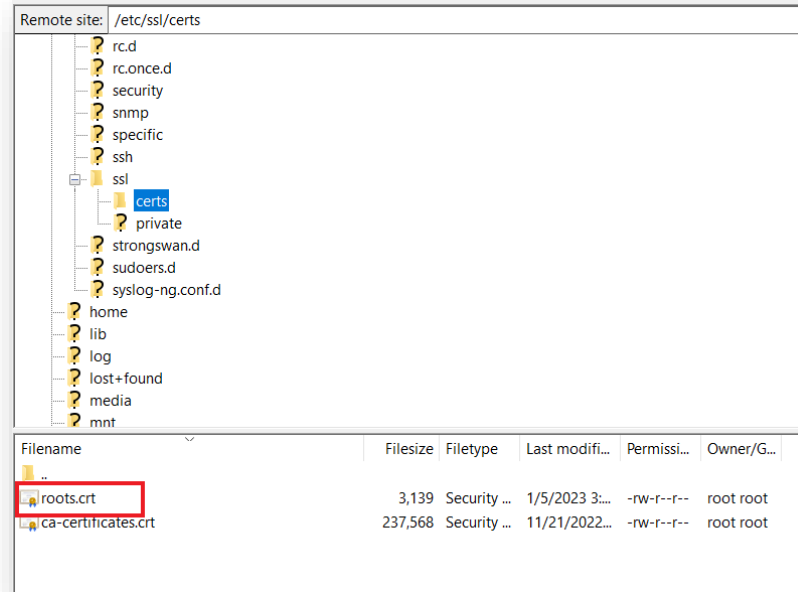
### PFC200 Setup

2b)

viii. Hit 'Submit', then reboot the controller via the 'Reboot' button the top right corner of the WBM.

ix. After waiting for the reboot, revisit the connection status on the WBM. You should now see a running connection.

We are now ready to create a Codesys project to publish data to Ubidots.



## Connecting the Wago PFC200 to Ubidots Via Codesys

### Codesys Setup

3a)

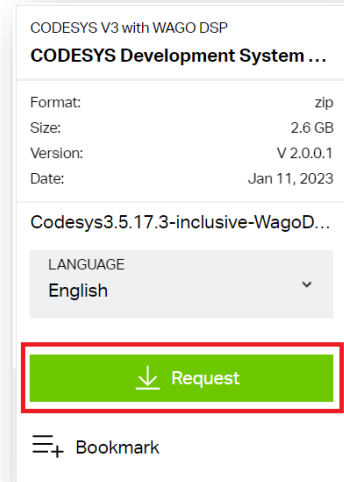
- i. Go to Wago website to download and install the newest version of Codesys 3.5 w/ Device support package here:

[https://www.wago.com/us/d/s/wreg\\_codesys\\_v3\\_dsp\\_c](https://www.wago.com/us/d/s/wreg_codesys_v3_dsp_c)

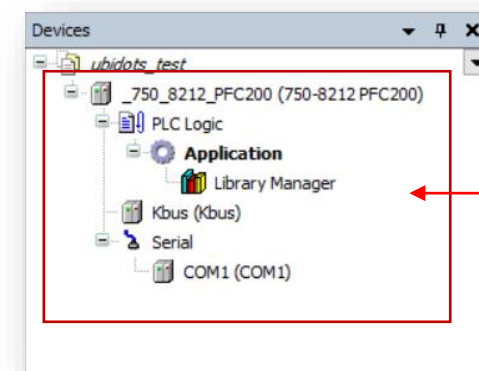
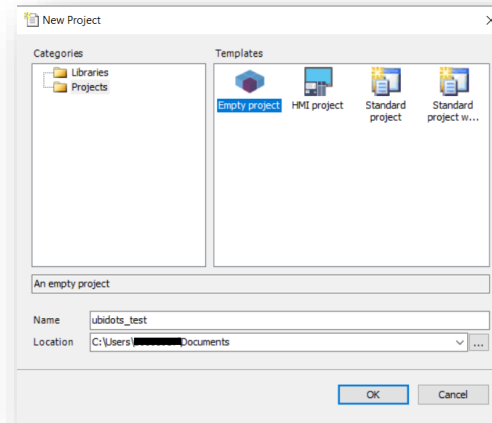
- ii. Create a 'New project', select empty project, and then name the project.

3b)

- i. Add your Wago device. I'll be using the 750-8212 for this project (shown in video).



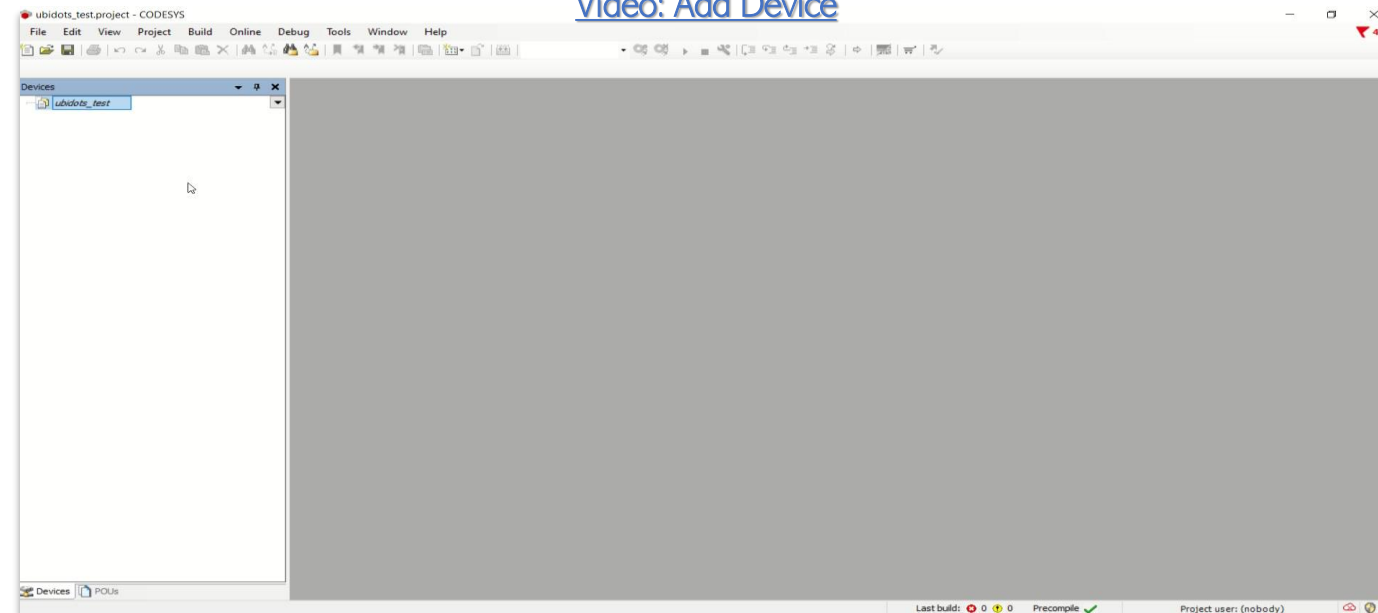
Note: At the time of this How-To the project is created with Codesys 3.5 SP17 Patch 3



Note:

You'll see this added to the device screen once you've added your Wago Device

### Video: Add Device







## Connecting the Wago PFC200 to Ubidots Via Codesys

### Codesys Setup

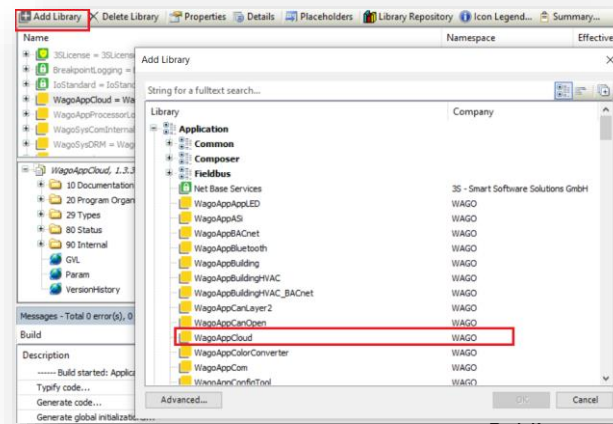
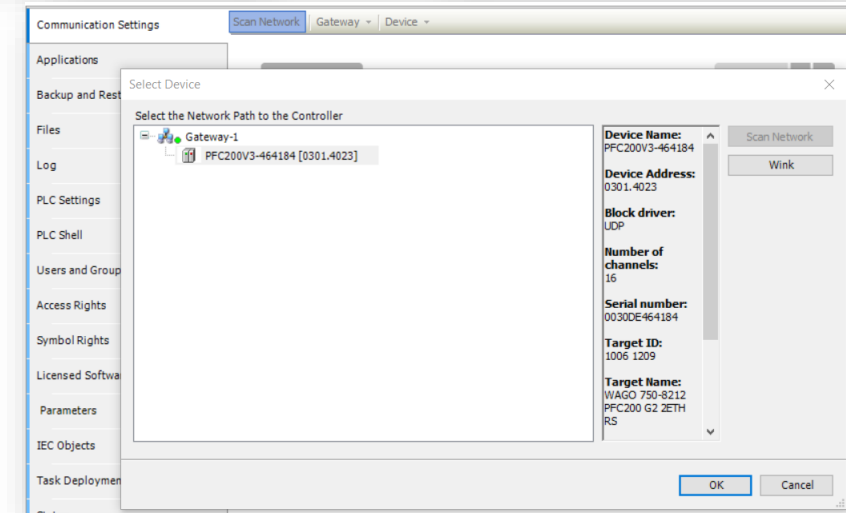
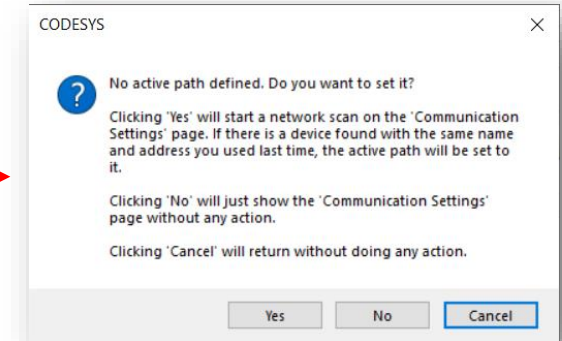
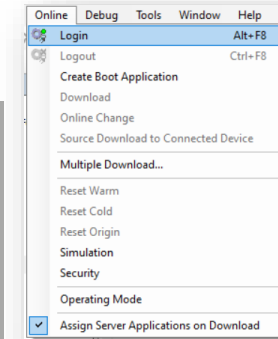
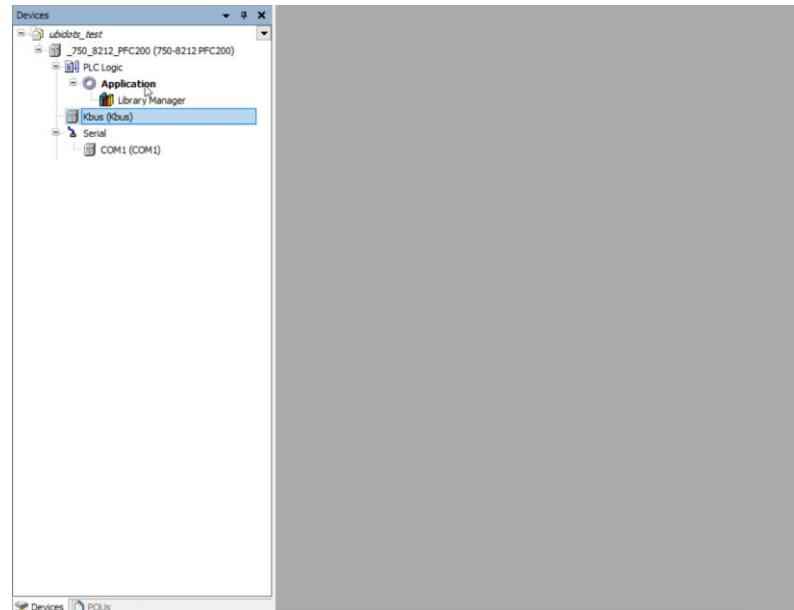
3c)

- Create a POU and Task configuration
- Setup a gateway and attempt to go online. Start by selecting 'Online' and 'Login'. Codesys will prompt you to create an active path
- Scan the network and select the PFC200. You'll be prompted for a username and password.

Default: admin/wago

- Login and now you should be online!
- Go back offline. Go to library manager, add library, and install WagoAppCloud & WagoAppJSON

### Video: POU/Task config



Public



## Connecting the Wago PFC200 to Ubidots Via Codesys

### The Code (Publish): MQTT Library

For this application, we'll be using the FbPublishMQTT\_2 function block.

**sTopic** will be a string representing the MQTT topic = 'v2.0/devices/{DEVICE\_LABEL}'

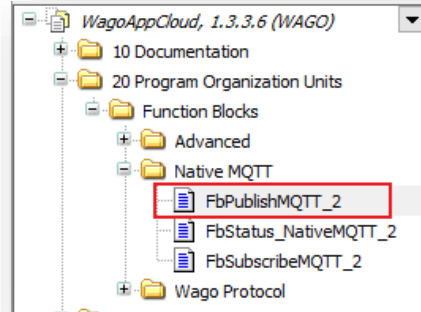
In this example the topic is as follows:  
'v2.0/devices/joes\_test'

**eQualityOfService = 1** which indicates that the MQTT publisher will send the message at least once to the MQTT Broker aka Ubidots.

**dwSize** will be a dynamic variable that determines the length of the message string that will be sent to the broker, in bytes.

**aData** is the pointer of our JSON payload data to be published to the Broker. The data must be copied over from the JSON string to this aData array.

**xTrigger** will act as the trigger to begin publishing data and the outputs **xBusy**, **xError**, **oStatus** are troubleshooting indicators.



#### Function Description

This function block transmit data to the cloud in the own data structure. The PLC application engineer can choose the topic and his own data structure. The Quality of Service 0, 1 and 2 are supported. To write the data in a JSON format the library WagoAppJSON could be used.

#### Note

If using the Native MQTT Protocol with the function blocks FbPublishMQTT\_2 and/or FbSubscribeMQTT\_2 it is necessary to configure the Data Protocol Native MQTT in the web based management.

#### Note

For secure data traffic it is recommended to send maximum 65kByte in one task interval.

#### Note

This function block must be executed in the background task (set the task priority to 15).

### FbPublishMQTT\_2 (FB)

#### Interface variables

Scope	Name	Type	Comment
Input	sTopic	STRING(255)	MQTT Topics are structured in a hierarchy similar to folders and files in a file system using the forward slash (/) as a delimiter. Using this system you can create a user friendly and self descriptive naming structures of you own choosing. Topic names are: Case sensitive; use UTF-8 strings.
	eQualityOfService	eQualityOfService	Quality of Service: 0,1,2
	xRetain	BOOL	Set to true to make the message retained
	dwSize	DWORD	DataCount to be transmitted
	aData	POINTER TO BYTE	Array of data which should be transmitted
Inout	xTrigger	BOOL	Trigger the transmission of data
Output	xBusy	BOOL	Transmission in progress
	xError	BOOL	Indicates that an Error has occurred.
	oStatus	FbResult	Status object with detailed information about a happend error. (Listed in eStatus) The content of the error object could be displayed via the FbShowResult from the WagoSysErrorBase library.

#### Function

Generate a MQTT message to publish to the cloud.



## Connecting the Wago PFC200 to Ubidots Via Codesys

```
PROGRAM Main
VAR
  oFbPublishMQTT_2 : WagoAppCloud.FbPublishMQTT_2(eConnection := eConnectionId.Connection1);
  aBuffer          : ARRAY[0..1999] OF BYTE;
  dwBytesCount     : DWORD;
  sPayload         : STRING(1024);
  xTrigger         : BOOL;
  TimerOn: TON;
  dwBusyCounter: DWORD;
  dwErrorCounter: DWORD;
END_VAR
```

### Example one: Simple Hard Code

Here we focus on initializing the MQTT publish function block as well as all of the variables the FB requires.

It's important to initialize the MQTT function block by specifying the specific cloud connection. Here we are using cloud connection 1.

Create a timer to act as the publishing interval, which in this case is every 1 second.

Create the JSON message payload. It's important to note that the type is a string with the format of quotes around the variable name and nothing around the value.

Example of one variable: '{"var\_name": var\_value}'

Example of two variables: '{"var\_name\_1": var\_1\_val, "var\_name\_2": var\_2\_val}'

Once the timer has completed the elapsed time and the previous publishing function is complete, store length of the payload and copy the payload value to the publishing buffer array.

Actuate the publish function block with the data and length of data you'd like to transmit

Assuming the code has run with no issues, you should be able to reload the Ubidots page and find the new values auto populated in the device tab.

```
TimerOn(IN := TRUE, PT := T#1S);
sPayload := '{"var1": 23, "var2": 46, "var3": 1}';

IF TimerOn.Q THEN
  xTrigger := TRUE;
  TimerOn(IN := FALSE);

  IF NOT oFbPublishMQTT_2.xBusy THEN

    // Copy payload string
    dwBytesCount := Length(sPayload);
    MemCopy(pDest := ADR(aBuffer), pSource := ADR(sPayload), udiSize := dwBytesCount);

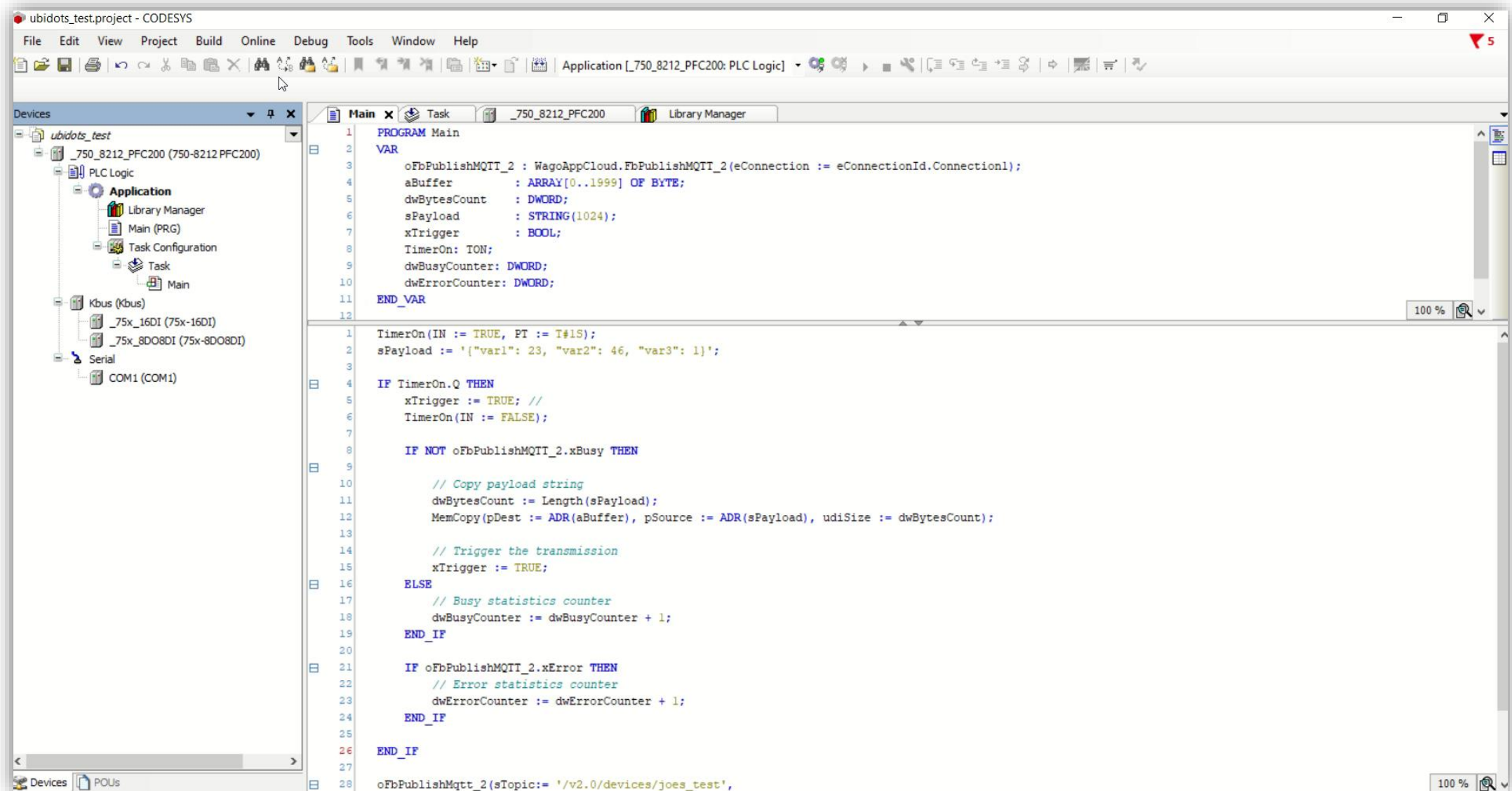
    // Trigger the transmission
    xTrigger := TRUE;
  ELSE
    // Busy statistics counter
    dwBusyCounter := dwBusyCounter + 1;
  END_IF

  IF oFbPublishMQTT_2.xError THEN
    // Error statistics counter
    dwErrorCounter := dwErrorCounter + 1;
  END_IF

END_IF

oFbPublishMqtt_2(sTopic:= '/v2.0/devices/joes_test',
  eQualityOfService:= 1,
  dwSize := dwBytesCount,
  aData := aBuffer,
  xTrigger := xTrigger);
```

## Connecting the Wago PFC200 to Ubidots Via Codesys





## Connecting the Wago PFC200 to Ubidots Via Codesys

### The Code (Publish): JSON Library

For this application, we'll be utilizing the Fb\_JSON\_Writer\_02 function block in the WagoAppJSON library.

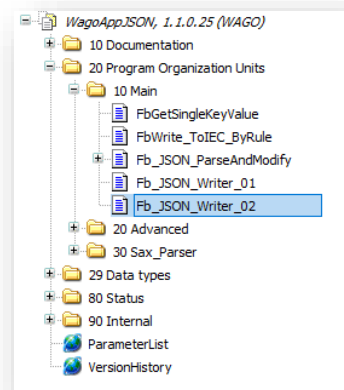
The functionality of this library is broken up into three parts: Creating the template string, the value string, and the execution of the function block

**sJSON\_BaseFrame** will house the template of your JSON string. Any dynamic variables will have the placeholder '#Parameter' within the template string.

**aParameterValues** is an array of all variables that are represented in the template string as #Parameter. Since this array is a string, you'll need to convert your values into strings before putting them into the array.

**xTrigger** will cause the JSON function block to trigger its execution and create a JSON string represented by the string template and value array.

**sOutput** will be the final JSON string result.



### Fb\_JSON\_Writer\_02 (FB)

#### Interface variables

Scope	Name	Type	Comment
Input	sJSON_BaseFrame	STRING(JSON_MAX_STRING)	String containing the general JSON data
	xDeleteFormattingCharacters	BOOL	delete Tab and CR LF
	oStatus	WagoSysErrorBase.FbResult	Status details
Output	xError	BOOL	Error occurred
	xDone	BOOL	JSON data generated without error
	aParameterValues	POINTER TO STRING(JSON_MAX_PARAMETER_STRING)	An array containing the variable JSON values, start index must be 0 e.g. [0..9]
Input	xTrigger	BOOL	Activate the build process
Inout	sOutput	STRING(JSON_MAX_STRING)	Result string

#### Function

Generate a JSON string from a base template and an array with appropriate JSON values Other than Fb\_JSON\_Writer\_01 this block allows some more flexibility by skipping key value pairs from the basic template string.

```
//Define a template string, which contains ``#Parameter`` wherever a variable value should be inserted.
VAR
  MyTemplateString:String(JSON_MAX_STRING):='
    {"Menu": {"id": "#Parameter", "value": "#Parameter", "popup": {"menuitem": [{"value": "#Parameter", "onclick": "#Parameter"}, {"value": "#Parameter", "onclick": "#Parameter"}]}}}
  ';
// Define a second variable from type array which contains the variable values already converted to a string.
  MyValueArray:ARRAY[0..7] OF STRING:=['CountryCode','DE','4.1','down1','4.2','down2','4.3','down3'];
END_VAR

Assuming you just want to generate from time to time a different string like:

{"Menu":{"id":"CountryCode","value": "DE","popup": {"menuitem": [{"value": "4.1","onclick": "down1"}]}}}

than the parameter array should look like:

MyValueArray:ARRAY[0..7] OF STRING:=['CountryCode','DE','4.1','down1','##','##','##','##'];
```

## Connecting the Wago PFC200 to Ubidots Via Codesys

### The Code (Publish): Example 2 Parameterized JSON

Initializing the Template as  
MyTemplateString

Creating the value array of size  
three because we are passing three  
variables

Initializing the three integer values  
and creating a new xJSON trigger  
variable

### Variable Declaration

```
PROGRAM Main
VAR

//Variables for MQTT publish
oFbPublishMQTT_2 : WagoAppCloud.FbPublishMQTT_2(eConnection := eConnectionId.Connection1);
aBuffer          : ARRAY[0..1999] OF BYTE;
dwBytesCount     : DWORD;

dwBusyCounter: DWORD;
dwErrorCounter: DWORD;
xTrigger       : BOOL;

//Variables for JSON_Writer
oFbJSON : WagoappJSON.Fb_JSON_Writer_02;
MyTemplateString : STRING(JSON_MAX_STRING) := '{"val1": #Parameter, "val2" : #Parameter, "val3": #Parameter}'; //String template
MyValueArray : ARRAY[0..2] OF STRING; //Array of values, must start at index 0
xJSONTrigger  : BOOL;

value_var1 : INT; //Raw values to be transmitted. Must be converted to string first.
value_var2 : INT;
value_var3 : INT;

TimerOn: TON;
sPayload : STRING(2000);

END_VAR
```



## Connecting the Wago PFC200 to Ubidots Via Codesys

### The Code (Publish): Example 2 Parameterized JSON

Individually setting values to the variables

We are then setting each value array element to the string form of the variable via Int\_to\_string conversion function

Add the JSON trigger to the timing if statement and then run the function block with sPayload as the output parameter.

### Program Sequence

```

1  TimerOn(IN := TRUE, PT := T#1S);
2
3  value_var1 := 40;
4  value_var2 := 50;
5  value_var3 := 2;
6
7  MyValueArray[0] := INT_TO_STRING(value_var1);
8  MyValueArray[1] := INT_TO_STRING(value_var2);
9  MyValueArray[2] := INT_TO_STRING(value_var3);
10
11
12 IF TimerOn.Q THEN
13   xTrigger := TRUE;
14   xJSONTrigger := TRUE;
15   TimerOn(IN := FALSE);
16
17   IF NOT oFbPublishMQTT_2.xBusy THEN
18
19     // Copy payload string
20     dwBytesCount := Length(sPayload);
21     MemCopy(pDest := ADR(aBuffer), pSource := ADR(sPayload), udiSize := dwBytesCount);
22
23     // Trigger the transmission
24     xTrigger := TRUE;
25
26   ELSE
27     // Busy statistics counter
28     dwBusyCounter := dwBusyCounter + 1;
29   END_IF
30
31 IF oFbPublishMQTT_2.xError THEN
32   // Error statistics counter
33   dwErrorCounter := dwErrorCounter + 1;
34 END_IF
35
36 END_IF
37
38 oFbJson(sJSON_BaseFrame := MyTemplateString,
39   aParameterValues := MyValueArray,
40   xTrigger := xJSONTrigger,
41   sOutput := sPayload);
42
43 oFbPublishMqtt_2(sTopic:= '/v2.0/devices/joes_test',
44   eQualityOfService:= 1,
45   dwSize := dwBytesCount,
46   aData := aBuffer,
47   xTrigger := xTrigger);

```



## Connecting the Wago PFC200 to Ubidots Via Codesys

ubidots\_test.project\* - CODESYS

File Edit View Project Build Online Debug Tools Window Help

Application [ \_750\_8212\_PFC200: PLC Logic ]

Devices

- ubidots\_test
  - \_750\_8212\_PFC200 (750-8212 PFC200)
    - PLC Logic
      - Application
        - Library Manager
        - Main (PRG)
        - Task Configuration
          - Task
            - Main
    - Kbus (Kbus)
      - \_75x\_16DI (75x-16DI)
      - \_75x\_8DO8DI (75x-8DO8DI)
    - Serial
      - COM1 (COM1)

Main x Library Manager

```
13 //Variables for JSON_Writer
14 oFbJSON : WagoappJSON.Fb_JSON_Writer_02;
15 MyTemplateString : STRING(JSON_MAX_STRING) := '["val1": #Parameter, "val2": #Parameter, "val3": #Parameter]'; //String template
16 MyValueArray : ARRAY[0..2] OF STRING; //Array of values, must start at index 0
17 xJSONTrigger : BOOL;
18
19 value_var1 : INT; //Raw values to be transmitted. Must be converted to string first.
20 value_var2 : INT;
21 value_var3 : INT;
22
23 TimerOn: TON;
24 sPayload : STRING(2000);
25
26 END_VAR
27
```

```
1 TimerOn(IN := TRUE, PT := T#1S);
2
3 value_var1 := 40;
4 value_var2 := 50;
5 value_var3 := 2;
6
7 MyValueArray[0] := INT_TO_STRING(value_var1);
8 MyValueArray[1] := INT_TO_STRING(value_var2);
9 MyValueArray[2] := INT_TO_STRING(value_var3);
10
11
12 IF TimerOn.Q THEN
13   xTrigger := TRUE;
14   xJSONTrigger := TRUE;
15   TimerOn(IN := FALSE);
16
17   IF NOT oFbPublishMQTT_2.xBusy THEN
18
19     // Copy payload string
20     dwBytesCount := Length(sPayload);
21     MemCopy(pDest := ADR(aBuffer), pSource := ADR(sPayload), udiSize := dwBytesCount);
22
23     // Trigger the transmission
24     xTrigger := TRUE;
25
26   ELSE
27     // Busy statistics counter
28     dwBusyCounter := dwBusyCounter + 1;
29   END_IF
30
31 IF oFbPublishMQTT_2.xError THEN
32   // Error statistics counter
33
```

Messages - Total 1 error(s), 0 warning(s), 49 message(s)

Build 0 error(s) 0 warning(s) 49 message(s)

Description	Project	Object	Position
Last build: 0 0 0 Precompile			