# MQTT Programmer's Guide

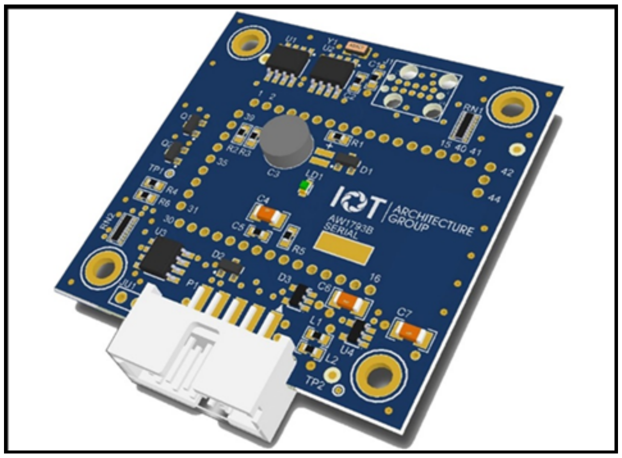
for the

# Wisplet® S2W IOT Engine

## CONTENTS

# 1. INTRODUCTION

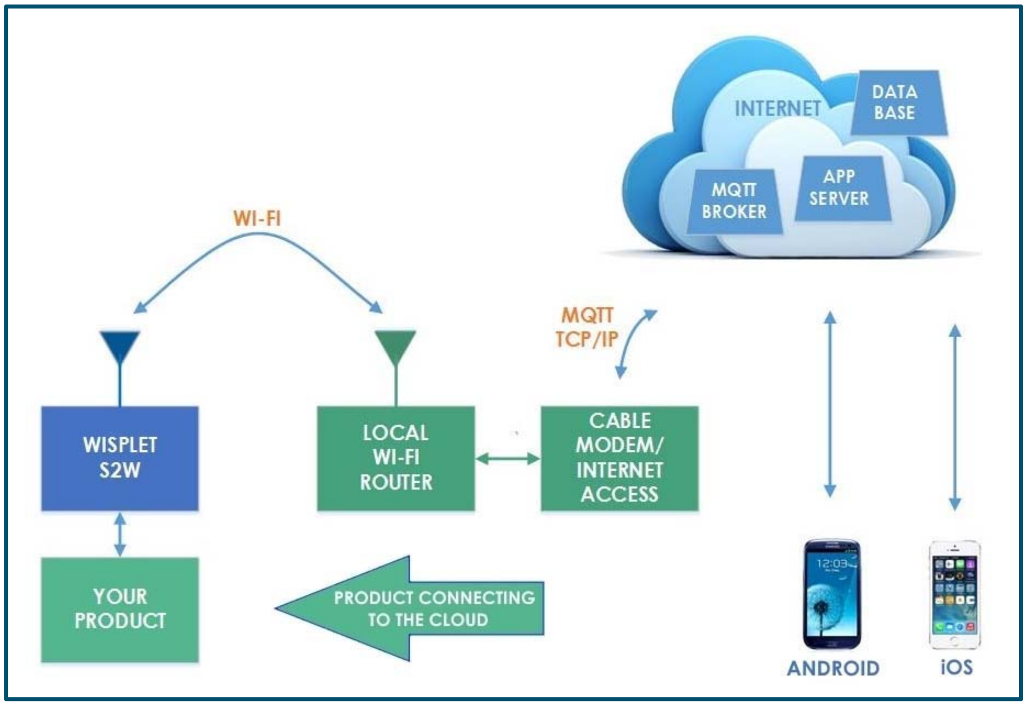
The Wisplet S2W is a small, low-cost, internet connectivity hardware board specifically designed for Internet-of-Things applications.



***WISPLET S2W HARDWARE MODULE***

*FIGURE 1.1*

The Wisplet S2W provides internet connectivity on one side, and connectivity to a hardware product on the other side, for the purpose of providing a way for that hardware product to connect to a cloud application.



***WISPLET S2W CONNECTS YOUR PRODUCT TO THE CLOUD***

*FIGURE 1.2*

In addition to including the necessary radio circuitry for providing internet connectivity, the Wisplet provides an implementation of the MQTT protocol, which is fast becoming the industry standard for Internet-of-Things (IOT) applications.

The Wisplet also performs rules processing, so that it can be configured to understand the various sensors built into the hardware product to which it is attached. The Wisplet can be configured to know which sensor value to report to the cloud at a defined interval in status reports, and can also know which ranges of values for each sensor that should be considered to be a condition that should generate an immediate alert to the cloud.

In addition to being able to report data from the hardware product to the cloud, the Wisplet can also deliver control messages from the cloud to the hardware product. This could be something as simple as telling the hardware product to display a particular value on an LED display on the product, or something more complicated like telling the hardware product to unlock or open a door.

# 2. MQTT INTERFACE

There are three interfaces to the Wisplet S2W:

- its UART which is how it talks to your product,
- its configuration web pages, used to select your Wi-Fi access point, and
- via MQTT commands.

This document is directed towards the MQTT interface. There are 2 components to a MQTT message:

- the topic string (aka the address to send the payload), and
- the actual message data (payload).

## 2.1 Topic String Definition

Topic strings are case sensitive, hierarchal in their definition and different levels are separated via a fore slash ( **/** ) character. The convention adopted by the Wisplet is to use all lowercase characters in the topic string to avoid any issues of mixed case.

The convention used by the Wisplet requires that the first level of the topic string is always the client ID, the unique identifier that is used by the MQTT client when connecting to the MQTT broker, who is publishing the message. For the Wisplet, this is its MAC address, expressed via 12 hexadecimal characters (0-9 a-f) all lower case. For the cloud or other non-Wisplet device this could be anything. At the time this document was written, this portion of the topic string is ignored by the Wisplet for any messages it receives. For future compatibility, it may be useful to carefully define a unique name to be used here in case the Wisplet subscriptions are tightened up to only permit messages from a specific source. Whatever is used, be careful to keep that field 12 characters or less, no spaces or other special characters, recommend using just A-Z a-z 0-9 – and _ for now.

The second level of the topic depends on the direction of the message.

- For messages being sent by a Wisplet to the supervisor of the system, aka the north side, it is simply the string ***msg*** to keep it short and meaningful.
- For those messages being sent to a Wisplet, it is the MAC address of that Wisplet device that you want to receive that message. This address is in the same format as the first level that of the topic string.

There is a special variation of this second level MAC address that uses wildcards to permit targeting of multiple Wisplets via a single message. The wildcard character is defined as a lower case letter **x**, and it can be used to substitute for a specific hexadecimal character in the second level address. Wildcards are only valid when used in a specific order, left to right, starting with the most significant character of the MAC address.

For example, let's assume that we want to send a message to a Wisplet with MAC address abcdef123456 and our MAC address is 1a2b3c4d5e6f, then the topic string will begin like: 1a2b3c4d5e6f/abcdef123456 with the remainder of the topic string to be defined later. Sending a message to a topic string with this definition will only reach one and only one Wisplet, the Wisplet with MAC address abcdef123456. With one wildcard, xbcdef123456, we can make the same message affect up to 16 Wisplets, since the first hexadecimal character can be anything 0-9 a-f. If we use two wildcards, the address becomes xxcdef123456 and that address can affect up to 256 Wisplets. There are 13 possible addresses possible using this wild card scheme, ranging from a specific address with no wildcards at all, up to 12 wildcards which means all Wisplets. Using our above example we can create the following addresses:

`abcdef123456` – up to $16^0$ = 1 device

`xbcdef123456` – up to $16^1$ = 16 devices

`xxcdef123456` – up to $16^2$ = 256 devices

`xxxdef123456` – up to $16^3$ = 4,096 devices

`xxxxef123456` – up to $16^4$ = 65,536 devices

`xxxxxf123456` – up to $16^5$ = 1,048,576 devices

`xxxxxx123456` – up to $16^6$ = 16,777,216 devices

`xxxxxxx23456` – up to $16^7$ = 268,435,456 devices

`xxxxxxxx3456` – up to $16^8$ = 4,294,967,296 devices

`xxxxxxxxx456` – up to $16^9$ = 68,719,476,736 devices

`xxxxxxxxxx56` – up to $16^{10}$ = 1,099,511,627,776 devices

`xxxxxxxxxxx6` – up to $16^{11}$ = 17,592,186,044,416 devices

`xxxxxxxxxxxx` – all devices, up $16^{12}$ = 281,474,976,710,656 devices

Care must be taken when using this type of addressing because you could flood the network with a lot of traffic from Wisplets replying back. But this addressing scheme can be very useful for such tasks as firmware updates where you could stage the update process. For example by using 11 wildcards, that leaves the last character as defined, and with 16 possible choices for that character (0-9 a-f) you could address the entire device pool of Wisplets with 16 separate topic string definitions, giving you a way to stage your firmware updates and lessen the burden on the server providing the update. If you choose 2 wildcards, then you stage over 256 topic strings and messages. The less wildcards used, the more stages you can define and roll out a firmware update or query a group of Wisplets.

To take advantage of this wild card scheme, the Wisplet will subscribe to 13 topic strings, with each topic being a variation of its MAC address as documented above. In this manner, the Wisplet does not need to process every message received to see if it is for it, something which could be time consuming in large eco system. Instead, the Wisplet will only get a message if that message was addressed to it via an exact address or some version of the wild cards.

## 2.2 Message Payload Definition

The message payload varies with each message/reply. See the following message definitions for more details.

**Legend**

<from> = MACaddressOfSendingWisplet OR the device sending to a Wisplet

<to> = targetWispletMACaddress or wildcard address for that Wisplet

Wisplet firmware version: A.B.C.D where

A = UART interface to customer protocol rev

B = MQTT client version

C = OS Version

D = build number

While A,B and C may jump to reflect changes in the supporting libraries, you should use D (build number) when checking for new features, etc.

### 2.2.1 Status Report

- Requires Wisplet firmware version = any
- sent via rules timed event, on power up, or as a reply upon request
- Direction = from Wisplet
- topic string format = <from>/msg/**status**
- payload format
  - <PID name as defined in rules package> : <PID value> (The number of PIDs included depends on what ones were included in the status report in the rules package and what ones have values read from the customer's device)
  - rssi : <the RSSI (signal strength) value for the Wi-Fi connection to the access point>
  - uptime : <32-bit msec counter for system uptime, will wrap around after 49.7 days>

```
Example: { "p0": "0.3", "p1": "0.0", "p2": 0, "p3": 0, "p4": 0, "p5": 0,
"p6": 1, "p8": 0, "p9": 0, "p11": 0, "p12": 0, "p13": 0, "p14": 0, "p15": 2,
"p17": 3, "rssi": -62, "uptime": 105619 }
```

### 2.2.2 Alert Message

- Requires Wisplet firmware version = any
- sent as a result of a rules alert rule (PID value triggered an alert)
- Direction = from Wisplet
- topic string format = <from>/msg/**alert**
- payload format
    - PID : <pid number from rules package>
    - PIDname : <name of PID from rules package>
    - PIDvalue : <value when alert got triggered>
    - T1 : <threshold 1 value for the rule that triggered the alert>
    - T2 : <threshold 2 value for the rule that triggered the alert, may not be relevant>
    - Rule : <name of rule that got triggered, for ex. If Less Than T1>
    - RuleNumber : <# of rule that was triggered>

```
Example: { "PID": 4, "PIDName": "Pot", "PIDValue": "0.00", "T1": "1.00",
"T2": "0.00", "Rule": "If Less Than T1", "RuleNumber": 2 }
```

### 2.2.3 Bad Host Alert Message

- Requires Wisplet firmware version = any
- sent after no comms with host after a period of time has been triggered an alarm (just in case host has gone down, currently set to 5 minutes)
- Direction = from Wisplet
- topic string format = <from>/msg/**badhost**
- payload format will be one of the reasons listed below, indicating what code was running on the customer's (host) side when the lack of communications error was triggered
    - Reason : "customer board bootloader "
    - Reason : "customer board application"
    - Reason : "unknown"

```
Example: { "Reason": "customer board application" }
```

### 2.2.4 Get Firmware Versions

- Requires Wisplet firmware version = 2.144.352.16 or later
- Request from Cloud to return firmware versions of Wisplet and Customer boards

- Direction = to Wisplet (command)
- Topic string format = <from>/<to>/**version**
- Payload format = <ignored, can be anything or null>

- Direction = from Wisplet (reply)
- Topic String format = <from>/msg/**version**
- Payload format
  - Wisplet : <string with Wisplet firmware version info>
  - If customer board is in its bootloader and the Wisplet reads its bootloader version
    - CustomerBL : <BL version>
  - If customer board is in its application and the Wisplet reads its application version
    - CustomerApp : <if customer board is in app and we get its app version>
- The customer versions are returned if they have been already read before this command is received.
- The Wisplet version reply will be sent without cloud request upon power up and connection to the cloud broker
- The Wisplet version reply will also be sent after a Wisplet bootloading operation.

```
Example: { "Wisplet": "2.144.352.22 2016 Apr 7", "CustomerApp": 19 }
```

### 2.2.5 Get Wi-Fi Signal Strength (RSSI)

- Requires Wisplet firmware version = 2.144.352.21 or later
- Request from Cloud to return Wi-Fi signal strength for connection to access point

- Direction = to Wisplet (command)
- Topic string format = <from>/<to>/**rssi**
- Payload format = <ignored, can be anything or null>

- Direction = from Wisplet (reply)
- Topic String format = <from>/msg/**rssi**
- Payload format
  - o rssi : <RSSI value >

```
Example: { "rssi": -34 }
```

### 2.2.6 Update PIDs Command

- Requires Wisplet firmware version = any
- sent to Wisplet to change PID value(s)

- Direction = to Wisplet (command)
- Topic string format = <from>/<to>/**updates/pids**
- Payload format is PID number and its new value
  - o PID# : <new value>
- Can update multiple PIDs in same message

```
Example:  { "5": 5 }
```

- Direction = from Wisplet (reply)
- Topic String format = <from>/msg/**updates/ackpids**
- Payload format for each updated PID is its number and the result code for that update
  - o PID# : { "success" : "true" } OR
  - o PID# : { "success" : "false", "info" : "<string with error information>" }

```
Example: { "5": { "success": "true" } }
```

### 2.2.7 Update Keep Alive Setting Command

- Requires Wisplet firmware version = 2.144.352.20 or later
- sent to Wisplet to change its Keep Alive Setting

- Direction = to Wisplet (command)
- Topic string format = <from>/<to>/**updates/newkas**
- Payload format is PID number and its new value
  - interval : <new value in minutes, 1-20>

```
Example: { "interval":  5 }
```

- Direction = from Wisplet (reply)
- Topic String format = <from>/msg/**updates/ackkas**
- Payload format for updated keep alive setting
  - success : "true"   OR
  - success : "false", "info" : "<string with error information or error number>"
- If successful, the Wisplet will go offline to make the change take effect.  It should come back online very quickly.

```
Example: { "success": "true" }
```

### 2.2.8 Status Message Now Command

- Requires Wisplet firmware version = 2.144.352.20 or later
- sent to Wisplet to get its status message now, instead of waiting for the rules based interval to elapse before the message is sent.

- Direction = to Wisplet (command)
- Topic string format = <from>/<to>/**updates/newstatusnow**
- Payload format is:
  - allpids : “false” - which will send whatever pid info it has now OR
  - allpids : “true” – which will ensure that all pids have been read before sending
- if allpids true/false is not specified, it will assume allpids = false
- does not reset timer for status messages

```
Example: { "allpids":  "false" }
```

- Direction = from Wisplet (reply)
- Topic String format = <from>/msg/**updates/ackstatusnow**
- Payload format for reply
  - success : “true” OR
  - success : “false”, “info” : “<string with error information or error number>”
    - error codes:
      - busy = cannot comply right now, busy with something else
      - no rules = no rules loaded for a status message reply
      - no comms = currently not talking to host
      - unknown OR value = if none of the above apply

```
Example: { "success": "true" }
```

### 2.2.9 Status Message Interval Override Command

- Requires Wisplet firmware version = 2.144.352.20 or later
- sent to Wisplet to change its status message reporting interval from its rules setting

- Direction = to Wisplet (command)
- Topic string format = <from>/<to>/**updates/newstatusint**
- Payload format is
    - enabled : <0 or 1, 0=no 1=yes>
    - interval : <new msec interval, u32 – only used if enabled=1>
    - duration : <# msec for this override to be allowed, non-zero value required if enabled = 1>
- Override is not stored in EE and it is cleared with a reboot or when the duration timer expires. (Effective 2.144.352.22 2016 Apr 8 firmware)
- Once disabled, it will revert back to rules interval setting

```
Example:  { "enabled": 1, "interval": 10000, "duration": 300000 }
```

- Direction = from Wisplet (reply)
- Topic String format = <from>/msg/**updates/ackstatusint**
- Payload format
    - success : "true"   OR
    - success : "false", "info" : "<string with error information or error number>"
        - error codes:
            - busy = cannot comply right now, busy with something else
            - no rules = no rules loaded for a status message reply
            - no comms = currently not talking to host
            - unknown OR value = if none of the above apply

```
Example: { "success": "true" }
```

### 2.2.10 Exit Silent Mode

- Requires Wisplet firmware version = 2.144.352.20 or later
- Sent to Wisplet to force it to exit silent mode. Silent mode is entered into by the Wisplet when customer board does its bootloading, so the constant reading of PIDs does not interfere with that bootloading operation.
- Silent mode is stored in EE and persists across power cycles of Wisplet.
- Silent mode is exited normally upon finish of bootloading, or it can time out if something goes wrong.

- Direction = to Wisplet (command)
- Topic string format = <from>/<to>/**exitsilentmode**
- Payload format is <ignored, can be anything or null>

- Direction = from Wisplet (reply) = none, no ack

### 2.2.11 Config

- Requires Wisplet firmware version = 2.144.352.22 or later
- sent to Wisplet to get its configuration and key data

- Direction = to Wisplet (command)
- Topic string format = <from>/<to>/**config**
- Payload format is <ignored, can be anything or null>

- Direction = from Wisplet (reply)
- Topic String format = <from>/msg/**config**
- Payload format
  - Wisplet : <string with Wisplet firmware version info>
  - If customer board is in its bootloader and the Wisplet reads its bootloader version
    - CustomerBL : <BL version>
  - Else if customer board is in its application and the Wisplet reads its application version
  - CustomerApp : <if customer board is in app and we get its app version>
  - RulesNumPIDs : <# of PIDs defined in the rules>
  - RulesStatusMsgInterval : <msec interval for status message in rules>
  - statusMsgOverrideEnabled : "true" or "false"
  - If override enabled, then
    - statusMsgOverrideInterval : <msec override interval>
    - statusMsgOverrideDuration : <msec override duration in cmd>
  - broker : <name of MQTT broker being used>
  - keepAliveSeconds : <# seconds for keep alive setting>

Examples:

```
{ "Wisplet": "2.144.352.22 2016 Apr 8", "CustomerApp": 3, "RulesNumPIDs":
18, "RulesStatusMsgInterval": 900000, "statusMsgOverrideEnabled": "false",
"broker": "AWG29O1L5K0Q3.iot.us-east-1.amazonaws.com", "keepAliveSeconds":
900 }
```

```
{ "Wisplet": "2.144.352.22 2016 Apr 8", "CustomerApp": 3, "RulesNumPIDs":
18, "RulesStatusMsgInterval": 900000, "statusMsgOverrideEnabled": "true",
"statusMsgOverrideInterval": 10000, "statusMsgOverrideDuration": 300000,
"broker": "AWG29O1L5K0Q3.iot.us-east-1.amazonaws.com", "keepAliveSeconds":
900 }
```

# 3. New Rules Package

TBD Later

# 4. Customer Board Firmware Update

TBD Later

# 5. Wisplet Firmware Update

TBD Later

# 6. REVISION HISTORY

## 6.1 Rev. A

First release – 2016 Apr 7 – dmr SE

## 6.2 Rev. B

Second release – 2016 Apr 7 dmr SE

- fixed topic string for update pid reply

## 6.3 Rev. C

Third release – 2016 Apr 8 dmr SE

- changed status message interval override not to persist across reboots, no longer saved in EE (per Schumacher request), has max duration field to limit how long

- added examples of messages

- added config command