

WiFi Software Users Guide

Contents

1	Introduction	4
1.1	Purpose	4
1.2	Scope of this Manual	4
1.3	Acronyms and Terms	4
1.4	Conventions	5
1.5	References	5
1.6	Additional Information	5
2	Product Overview	6
3	Installation Components	7
3.1	CCES Installation	7
3.1.1	Web Installation	8
3.1.2	Local Installation	9
4	Example Projects	10
4.1	Wi-Fi Examples	10
4.2	Opening Example	10
4.3	Example Layout	10
4.4	Running the example	11
4.5	Configuring a Local MQTT Broker	13
4.6	Configuring a Local MQTT Subscriber or Publisher	18
5	Wi-Fi Software	22
5.1	Overview	22
5.2	Communication Model	22
5.3	Software Organization	23
5.4	Software Use	23
5.4.1	Command and Response	24
5.4.2	Receiving Unexpected Events	25
5.4.3	Data Received in Events	26

Error: RuntimeException occurred while performing an XHTML storage transformation (null)

1 Introduction

1.1 Purpose

This document describes the Wifi Software Pack for CrossCore Embedded Studio® (CCES) .

Note that this document references example applications that use the Wifi Software Pack but are not actually located in this package . Supported Board Support Packs will contain these examples.

1.2 Scope of this Manual

This document details what is in the Wifi Software Pack as well as how to install and work with it. Additionally, it covers how to build and run the example applications that accompany this package.

This document is intended for users who want to write software using Wifi Software Pack. It assumes some familiarity with the C/C++ programming language and CCES.

1.3 Acronyms and Terms

ADI	Analog Devices, Inc.
API	Application Programming Interface
BSP	Board Support Pack
CCES	CrossCore Embedded Studio®
CMSIS	Cortex® Microcontroller Software Interface Standard
DFP	Device Family Pack
HRM	Hardware Reference Manual
NoOS	No Operation System
RTE	Run-Time Environment

1.4 Conventions

Throughout this document, we refer to important installation locations. These locations are defined here.

- `<cces_root>`
 - The default CCES installer for CCES 2.6.0 places the product at location **C:**
/Analog Devices/CrossCore Embedded Studio 2.6.0, but the install location may vary depending on user preferences.
 - The default packs are placed at location `<cces_root>/ARM/packs`
`/AnalogDevices`.

1.5 References

1. CrossCore Embedded Studio® (CCES) [<http://www.analog.com>]
2. ARM CMSIS PACK [<http://www.keil.com/cmsis/pack>]

1.6 Additional Information

For more information on the latest ADI processors, silicon errata, code examples, development tools, system services and devices drivers, technical support and any other additional information, please visit our website at www.analog.com/processors.

2 Product Overview

The Wi-Fi Software Pack (ADI-WifiSoftware) provides a software framework for the ESP8266 Wi-Fi chip. The software is divided into three layers:

- Framework Layer: Control flow for applications with no operating system.
- Companion Layer: Encoding and decoding of Wi-Fi packets.
- Transport Layer: Reading and writing of Wi-Fi packets.

The package acts as a library to be used with Analog Devices' processors. Examples for this pack can be found in the supported Board Support Packs.

3 Installation Components

Before installing the Wifi Software Pack, the following should be installed.

- CrossCore Embedded Studio ® 2.6.0
- Device Family Pack for the targeted processor
- Board Support Pack for the targeted board ([**Optional**]: only to get example code)

This software is released in the form of a CMSIS Pack file. CCES will extract the contents of the Pack file into the CCES installation directory. This allows for a clean partitioning of software delivered by ADI and software created by the user. The Wi-Fi Software Pack contents (software framework, documentation, etc.) are placed at the following location

- CrossCore Embedded Studio® : `<cces_root>/ARM/packs/AnalogDevices/ADI-WifiSoftware/x.y.z`

where **x.y.z** is the installed pack version number. Figure 1 shows the contents that will be placed at this location after the installation has completed.

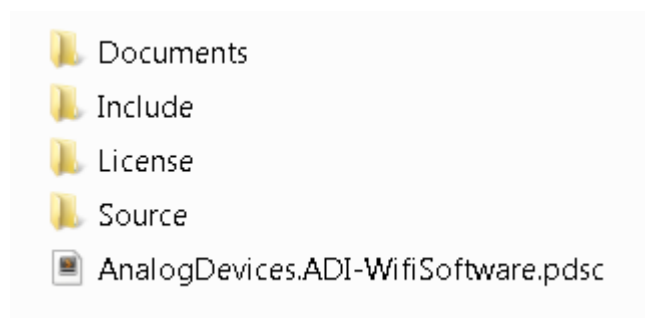


Figure 1. Installation Directory Structure

3.1 CCES Installation

To install a new Pack or update an existing Pack go to CMSIS Pack Manager perspective, shown in Figure 2. If the Pack Manager perspective was not opened previously, the CMSIS Pack Manager icon may not be present on the toolbar as shown below. In that case, the Pack Manager perspective can be opened by clicking *Window Perspective Open Perspective Other Pack Manager*. There are two methods that can be used to install the Pack described below.

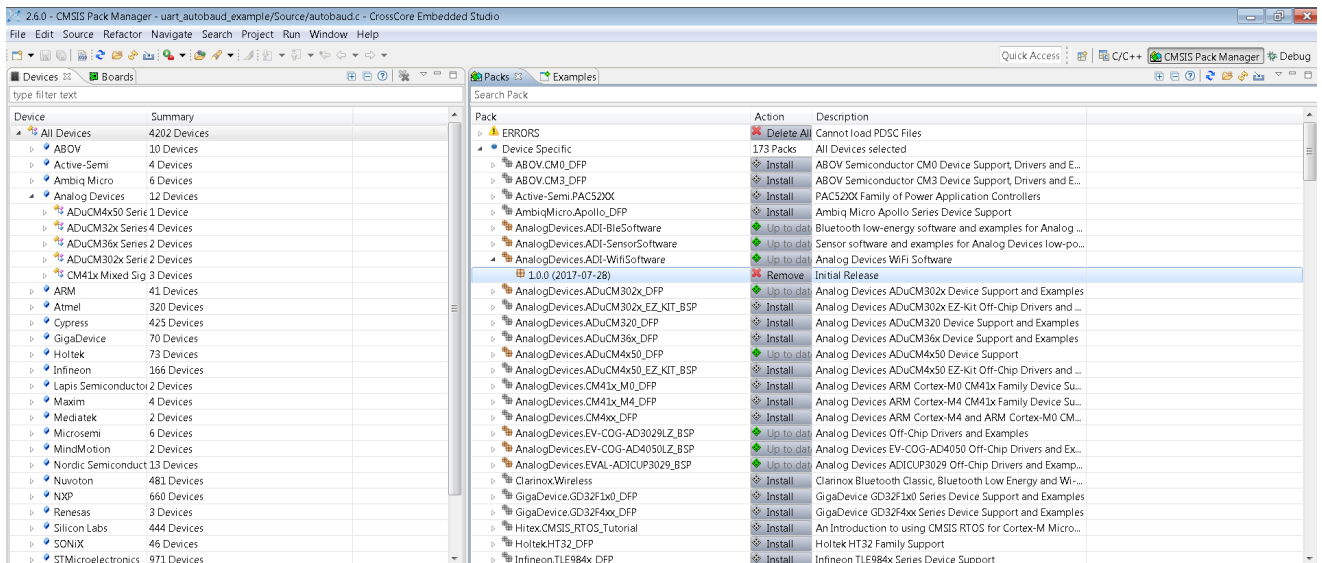


Figure 2. CMSIS Pack Manager Perspective

3.1.1 Web Installation

The Pack can be installed directly from the web using CCES, the user does not need to download the file and open it with CCES. This can be done by first refreshing the CMSIS Pack Manager (the blue arrows in the top left of the *Packtab*). This will display a list of available Pack files as shown in Figure 3. Clicking on a supported processor such as the "ADuCM302x Series" will show the Pack in the *Pack* tab as "AnalogDevices.ADI-WifiSoftware". Click "Install" and accept the license agreement in order to install the Pack .

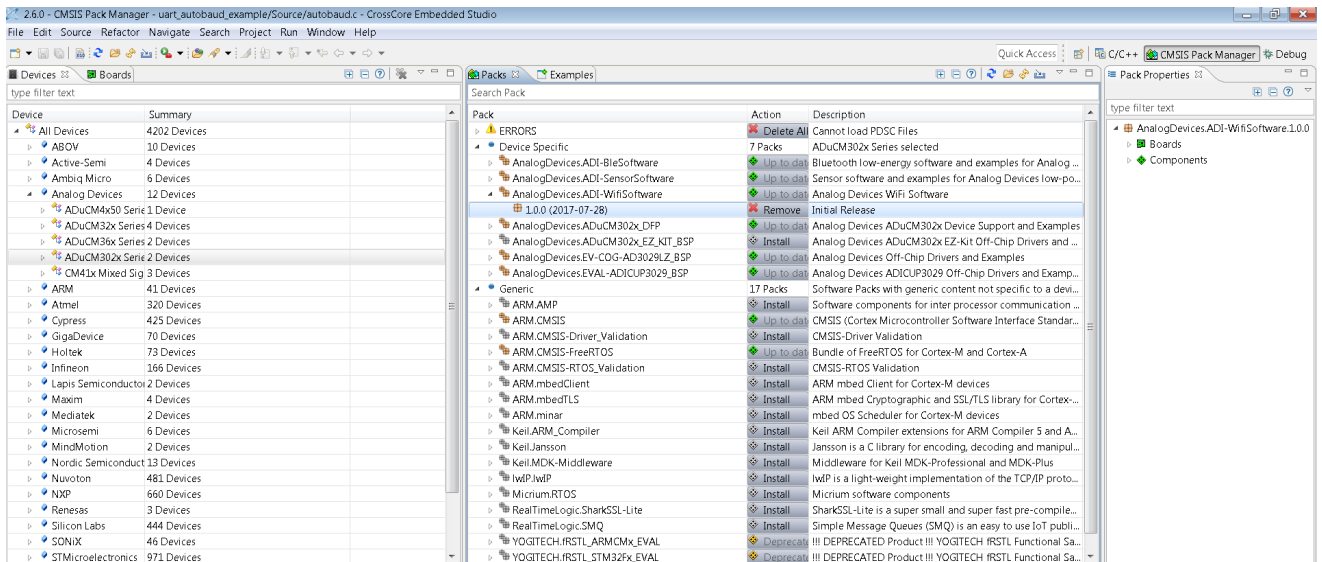


Figure 3. Available Pack Files

3.1.2 Local Installation

If the user has already obtained the Pack file, it can be installed without using the method described above. Click "Import Existing Packs" (the folder icon in the *Pack* tab) and then browse to the Pack file.

4 Example Projects

4.1 Wi-Fi Examples

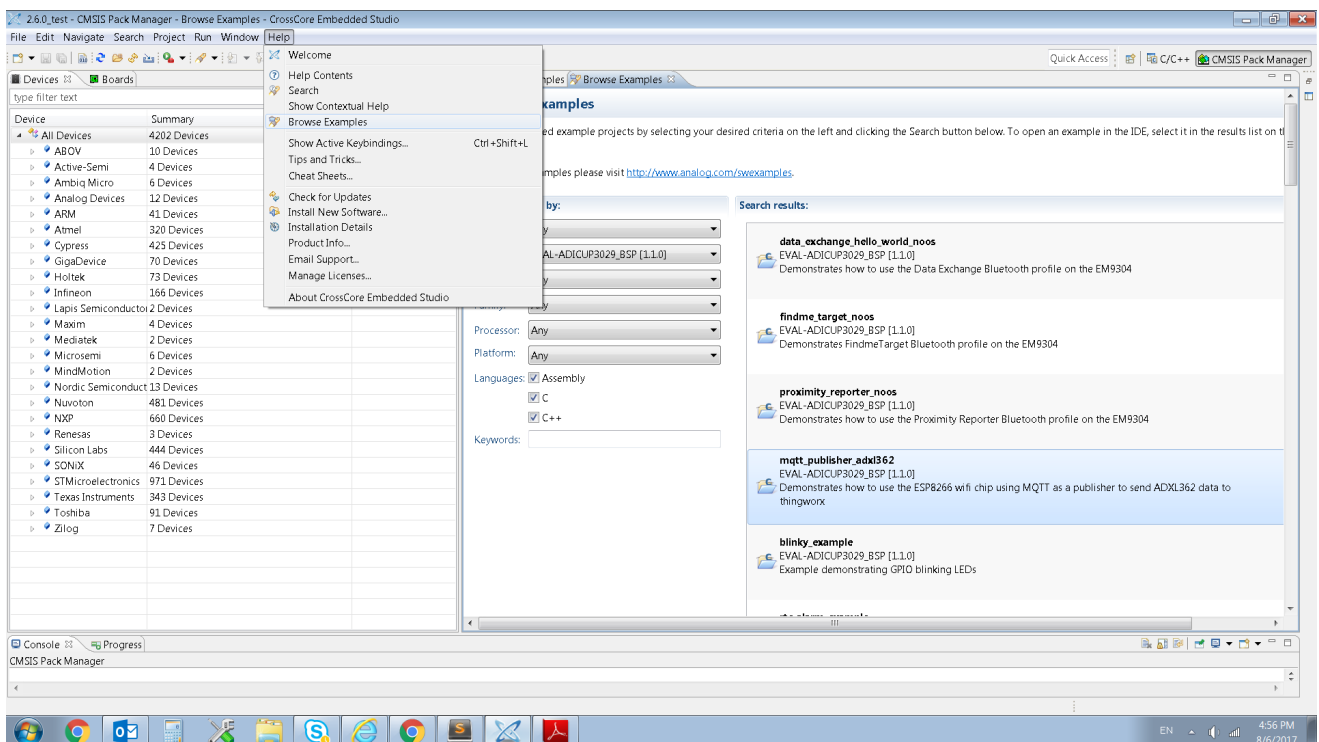
Board Support Packs that support the Wifi Software pack contain the example(s) given below.

- ESP8266 MQTT Publisher ADXL362 Accelerometer example

For a list of supported Board Support Packs, please consult the www.analog.com.

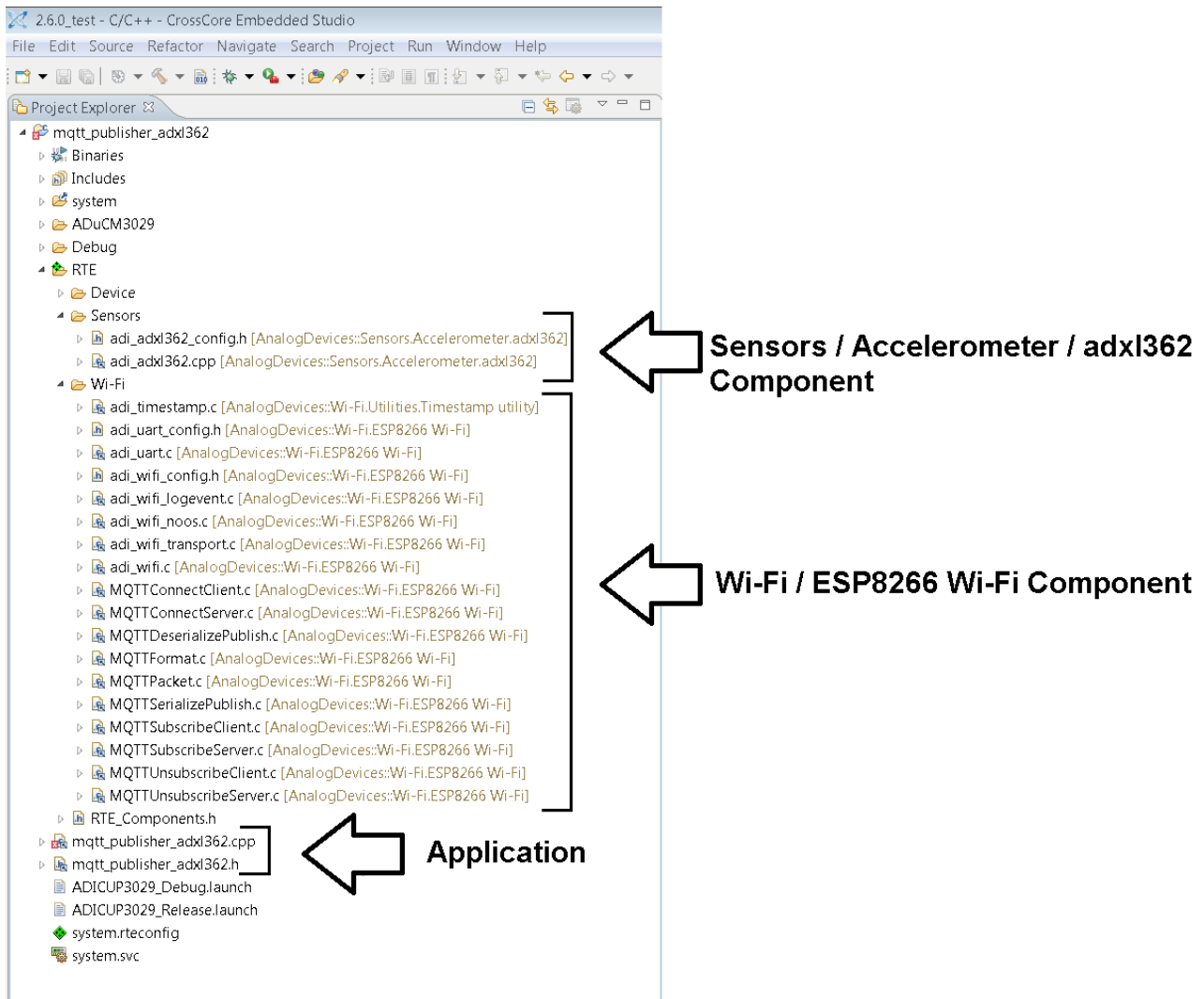
4.2 Opening Example

Examples can be opened by using CCES Browse Examples window as shown below. Double click on the example to open the project.



4.3 Example Layout

The layout of the MQTT publisher example is given below. All Wi-Fi examples follow similar layout.



4.4 Running the example

Follow below steps in order to run the project

1. Open the project and build
2. Set up hardware as described in the associated project Readme file
3. Set up the Broker as described in the section of this document [Configuring a Local MQTT Broker](#)
4. Set up the Subscriber as described in the section of this document [Configuring a Local MQTT Subscriber](#)

5. Configure the parameters pictured in the image below.
 - a. **aWifiSSID** should be the SSID of the network the ESP8266 will be connecting to. For example "IoTSSID"
 - b. **aWifiPassword** should be the password associated with **aWifiSSID**. If there is no password, leave this as an empty string. Otherwise an example of this field configure is "password1"
 - c. **aMQTTBrokerIp** is the IP address of the broker. To get this information read the section [Configuring a Local MQTT Broker](#). An example of this field is "192.10.1.1"
 - d. **aMQTTBrokerPort** is the port used to connect to the broker. If using the local Mosquitto MQTT broker this should be "1883"
6. Press Run Debug C/C++ menu option
7. Run the application

```
#include <axl/adxl362/adi_adxl362.h>
#include <framework/noos/adi_wifi_noos.h>

/* Defined in pinmux.c. */
extern "C" int32_t adi_initpinmux(void);

/***** Wi-Fi Configuration *****/
/* SSID of the access point. */
uint8_t aWifiSSID[] = "";

/* Password of the access point. */
uint8_t aWifiPassword[] = "";

/***** MQTT Configuration *****/

/* IP address of the broker to publish to. */
uint8_t aMQTTBrokerIp[] = "";

/* Port of the broker to publish to. */
uint8_t aMQTTBrokerPort[] = "";

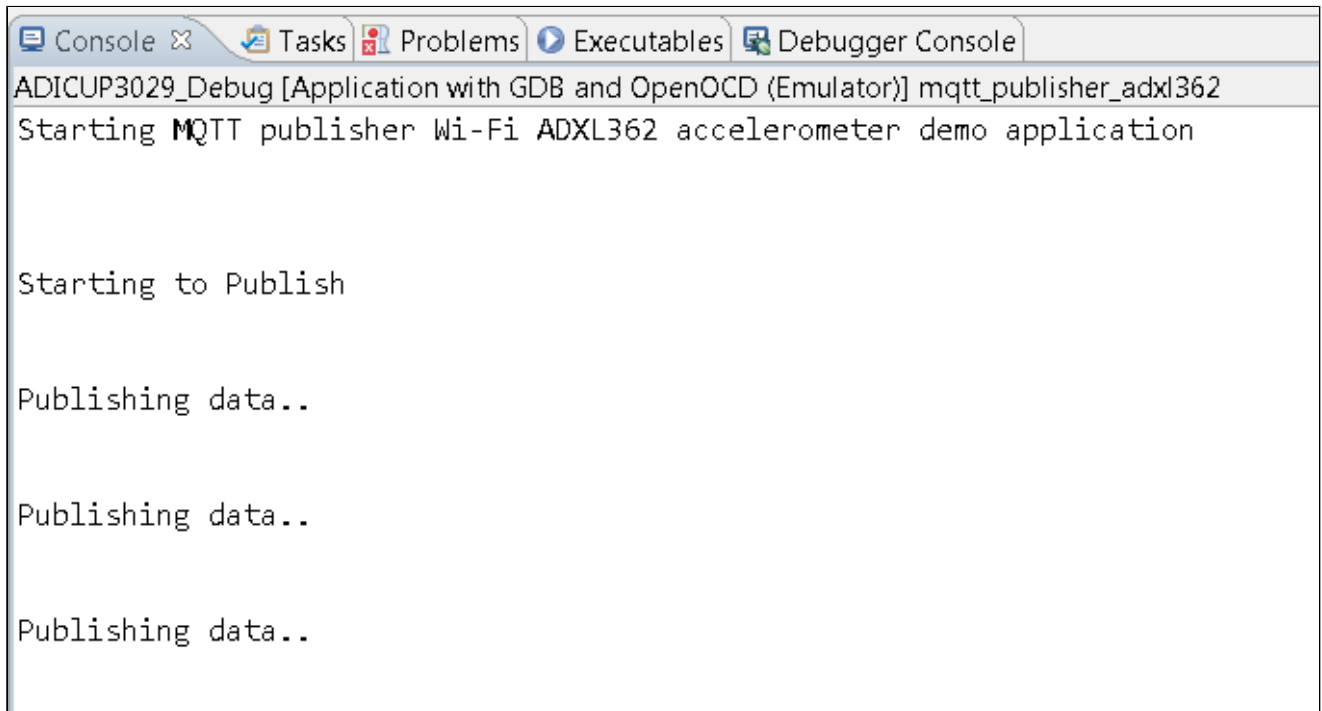
/*! MQTT publisher name. */
uint8_t aMQTTPublisherName[] = "ADI_WIFI_MQTT";

/*! MQTT Topic name. */
uint8_t aMQTTTopicName[] = "TOPIC_ADI_ADXL362";

/*! MQTT publish packet quality of service. */
#define ADI_WIFI_MQTT_PUBLISHER_QOS (0u)

/*! MQTT publisher version. */
#define ADI_WIFI_MQTT_PUBLISHER_VERSION (3u)
```

Debug console screen shot



The screenshot shows a console window from an IDE. The title bar includes tabs for 'Console', 'Tasks', 'Problems', 'Executables', and 'Debugger Console'. The console text is as follows:

```
ADICUP3029_Debug [Application with GDB and OpenOCD (Emulator)] mqtt_publisher_adxl362
Starting MQTT publisher Wi-Fi ADXL362 accelerometer demo application

Starting to Publish

Publishing data..

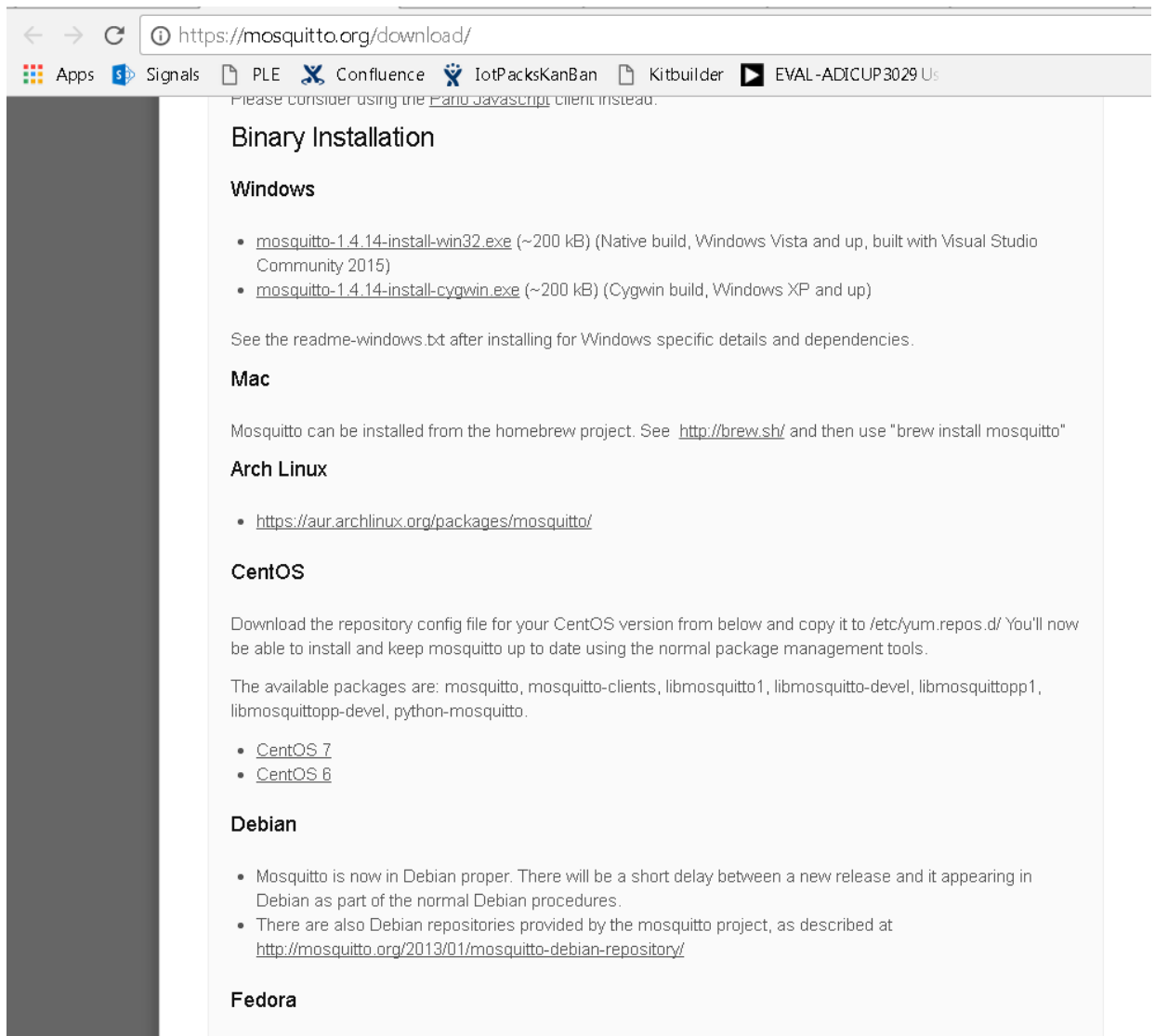
Publishing data..

Publishing data..
```

4.5 Configuring a Local MQTT Broker

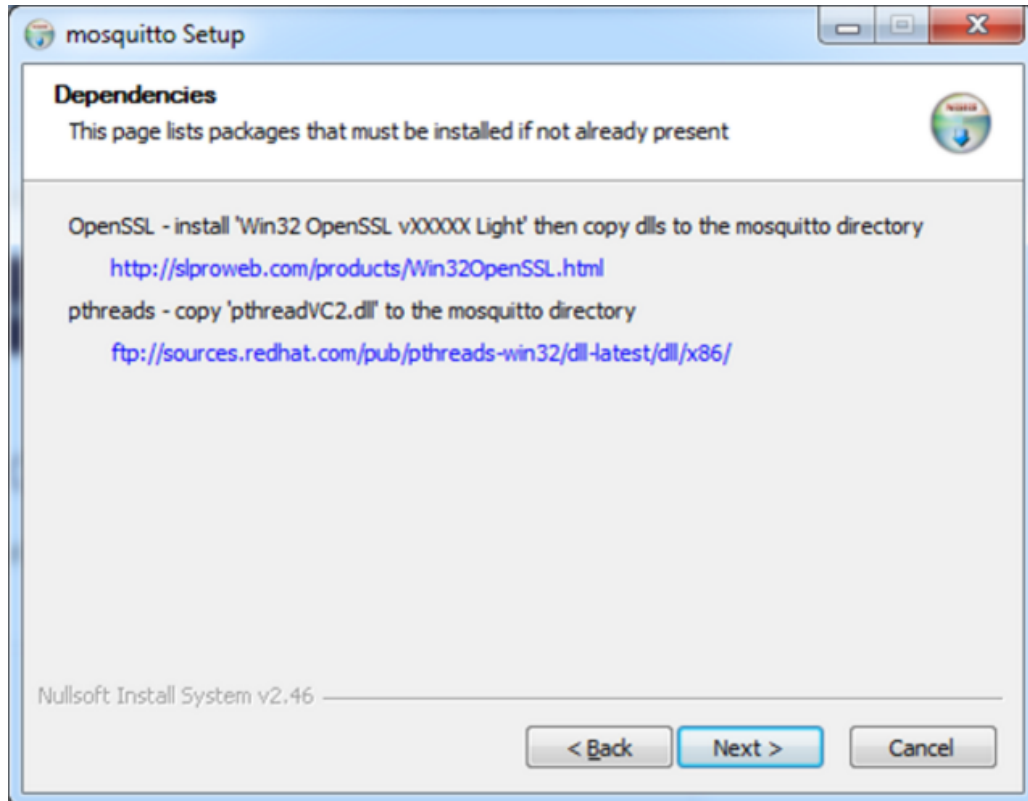
Note: Make sure the host machine that is running the local MQTT broker is connected to the same network as the board with the ESP8266.

1. Download and install Mosquitto. Pick the binary installation that matches your machine . (<https://mosquitto.org/download/>)



- a. Double click the downloaded .exe file, and install the Mosquitto program

- b. During the install, Mosquitto will ask you to make sure you have 2 other programs installed.



c. Click the links within the Windows install wizard **BEFORE** completing the Mosquitto install.

- i. **OpenSSL** (<http://slproweb.com/products/Win32OpenSSL.html>). Download the Win32 OpenSSL v1.0.2c Light setup file and install it.

Win32 OpenSSL Screenshot



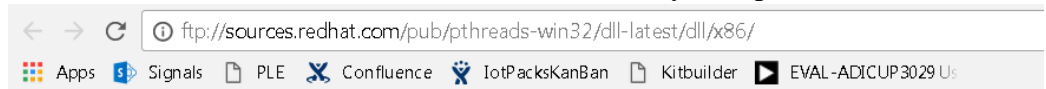
Screenshot of the Win32 OpenSSL Installer.

Download Win32 OpenSSL









Download Win32 OpenSSL today using the links below!

File	Type	Description
Win32 OpenSSL v1.0.2c Light	3MB Installer	Installs the most commonly used essentials of Win32 OpenSSL v1.0.2c (Recommended for users by the creators of OpenSSL). Note that this is a default build of OpenSSL and is subject to local and state laws. More information can be found in the legal agreement of the installation.

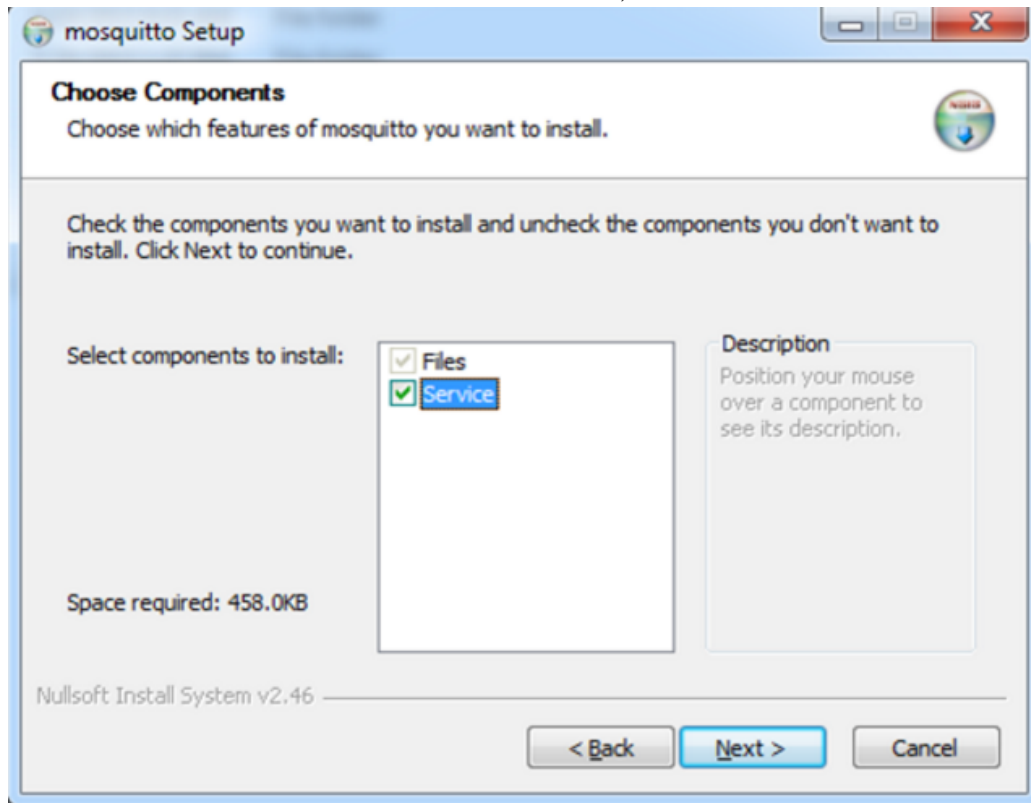
- ii. **pthread** (<ftp://sources.redhat.com/pub/pthreads-win32/dll-latest/dll/x86/>). Right click Pthreadvc2.dll and save it somewhere on the system path.



Index of /pub/pthreads-win32/dll-latest/dll/x86/

Name	Size	Date Modified
 [parent directory]		
 md5.sum	293 B	2/4/15, 7:00:00 PM
 pthreadGC2.dll	117 kB	5/26/12, 8:00:00 PM
 pthreadGCE2.dll	119 kB	5/26/12, 8:00:00 PM
 pthreadVC2.dll	54.5 kB	5/26/12, 8:00:00 PM
 pthreadVCE2.dll	60.5 kB	5/26/12, 8:00:00 PM
 pthreadVSE2.dll	56.0 kB	5/26/12, 8:00:00 PM
 sha512.sum	866 B	2/4/15, 7:00:00 PM

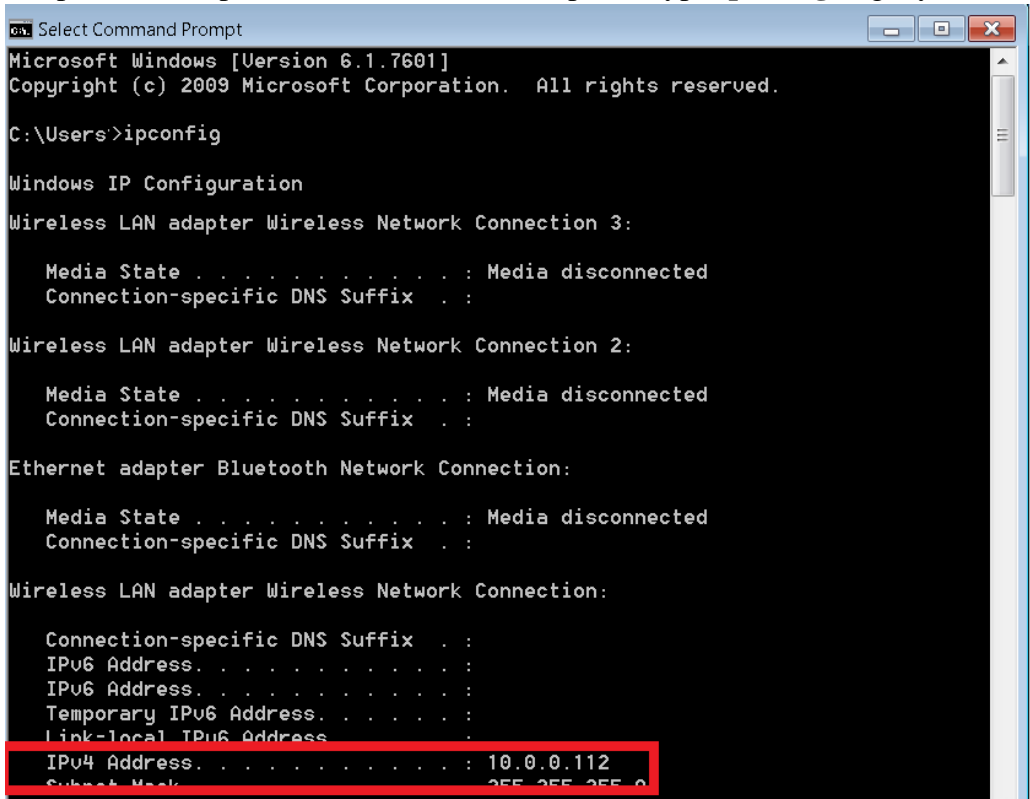
- d. Click next and finish the installation. The installer may ask if you want to install Services as well. Make sure that box is checked, as shown below.



- e. Make sure the following DLLs are in the same directory as the Mosquitto.exe. If not, please copy them there.
- libeay32.dll, ssleay32.dll (Look for these files in the OpenSSL-Win32 or OpenSSL-Win32\Bin folder)
 - pthreadVC2.dll
- f. Re-Run the Mosquitto.exe file, in order to complete the installation process.
2. Open a Command Prompt and navigate to the folder where mosquitto is installed. The default location is C:\Program Files (x86)\mosquitto.
3. Type mosquitto.exe -v to start the broker in verbose mode:

```
C:\Program Files (x86)\mosquitto>mosquitto.exe -v
1496151655: mosquitto version 1.4.11 (build date 20/02/2017 23:24:29.40) starting
1496151655: Using default config.
1496151655: Opening ipv6 listen socket on port 1883.
```

4. The mosquitto broker is now running locally and has the same IP as your machine and by default runs on port 1883. Open a new Command Prompt and type **ipconfig** to get your local



```
Select Command Prompt
Microsoft Windows [Version 6.1.7601]
Copyright (c) 2009 Microsoft Corporation. All rights reserved.

C:\Users>ipconfig

Windows IP Configuration

Wireless LAN adapter Wireless Network Connection 3:

    Media State . . . . . : Media disconnected
    Connection-specific DNS Suffix  . :

Wireless LAN adapter Wireless Network Connection 2:

    Media State . . . . . : Media disconnected
    Connection-specific DNS Suffix  . :

Ethernet adapter Bluetooth Network Connection:

    Media State . . . . . : Media disconnected
    Connection-specific DNS Suffix  . :

Wireless LAN adapter Wireless Network Connection:

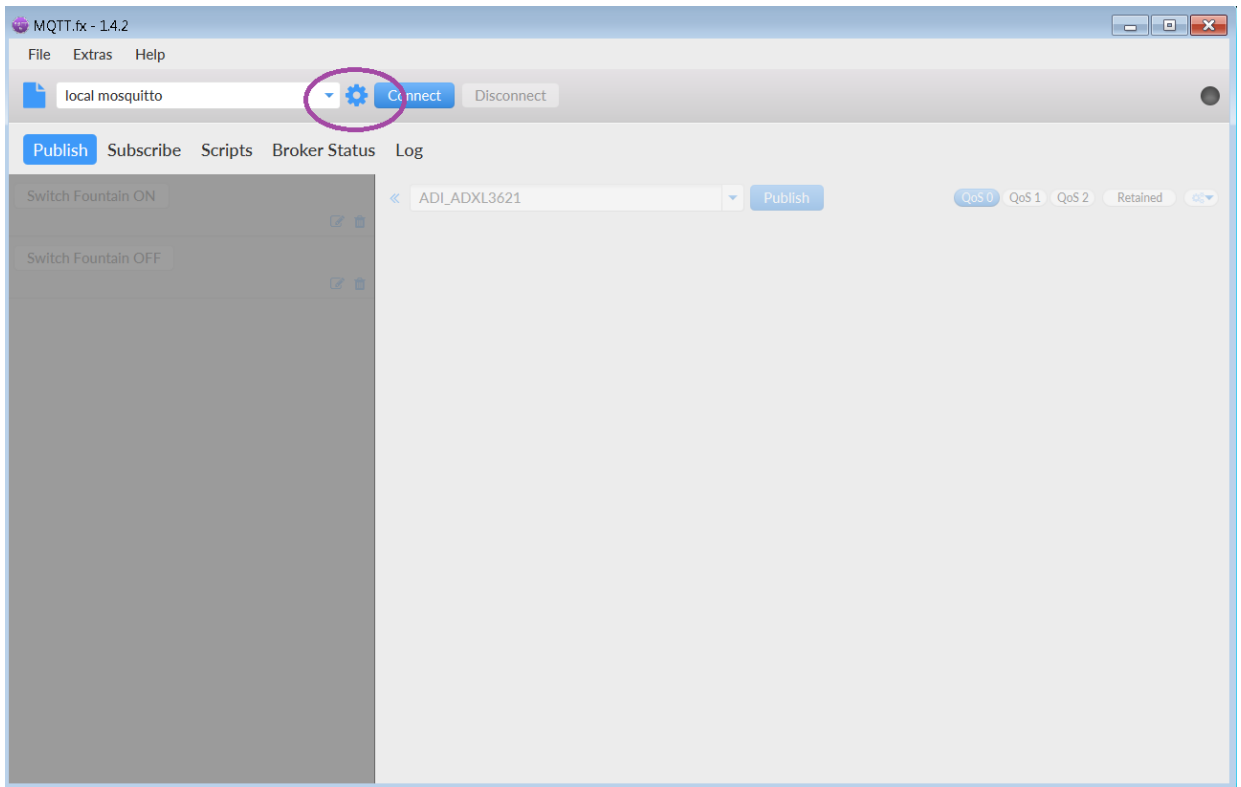
    Connection-specific DNS Suffix  . :
    IPv6 Address. . . . . :
    IPv6 Address. . . . . :
    Temporary IPv6 Address. . . . . :
    Link-Local IPv6 Address. . . . . :
    IPv4 Address. . . . . : 10.0.0.112
    Subnet Mask . . . . . : 255.255.255.0
```

IP address.

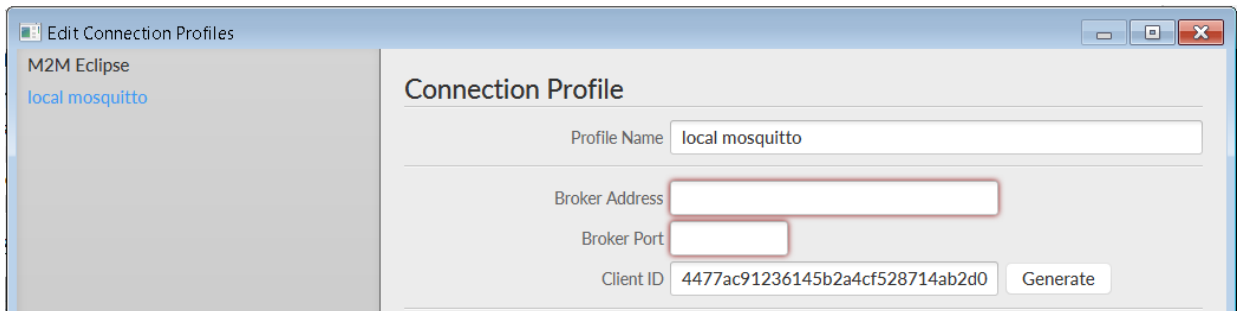
4.6 Configuring a Local MQTT Subscriber or Publisher

1. Download and install MQTTfx (<http://www.mqttfx.org/>)

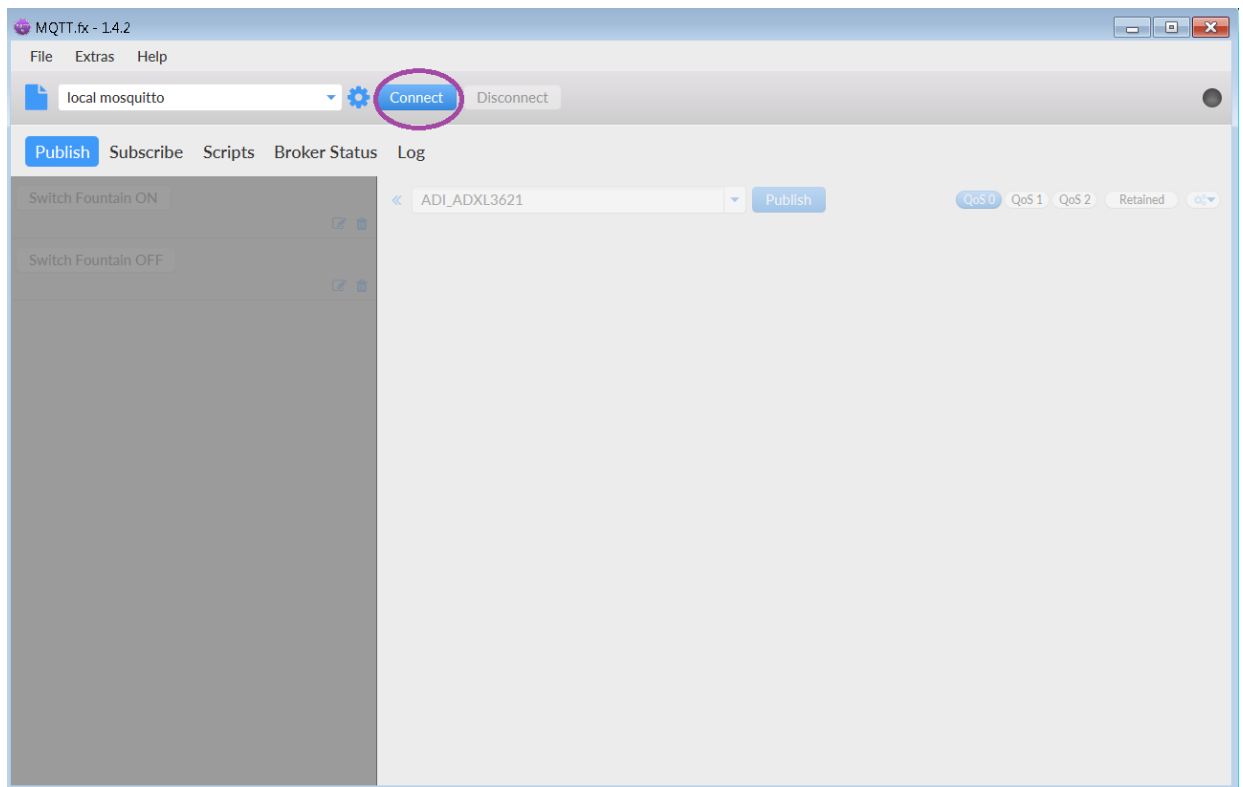
2. Click on the gear to configure the broker settings.



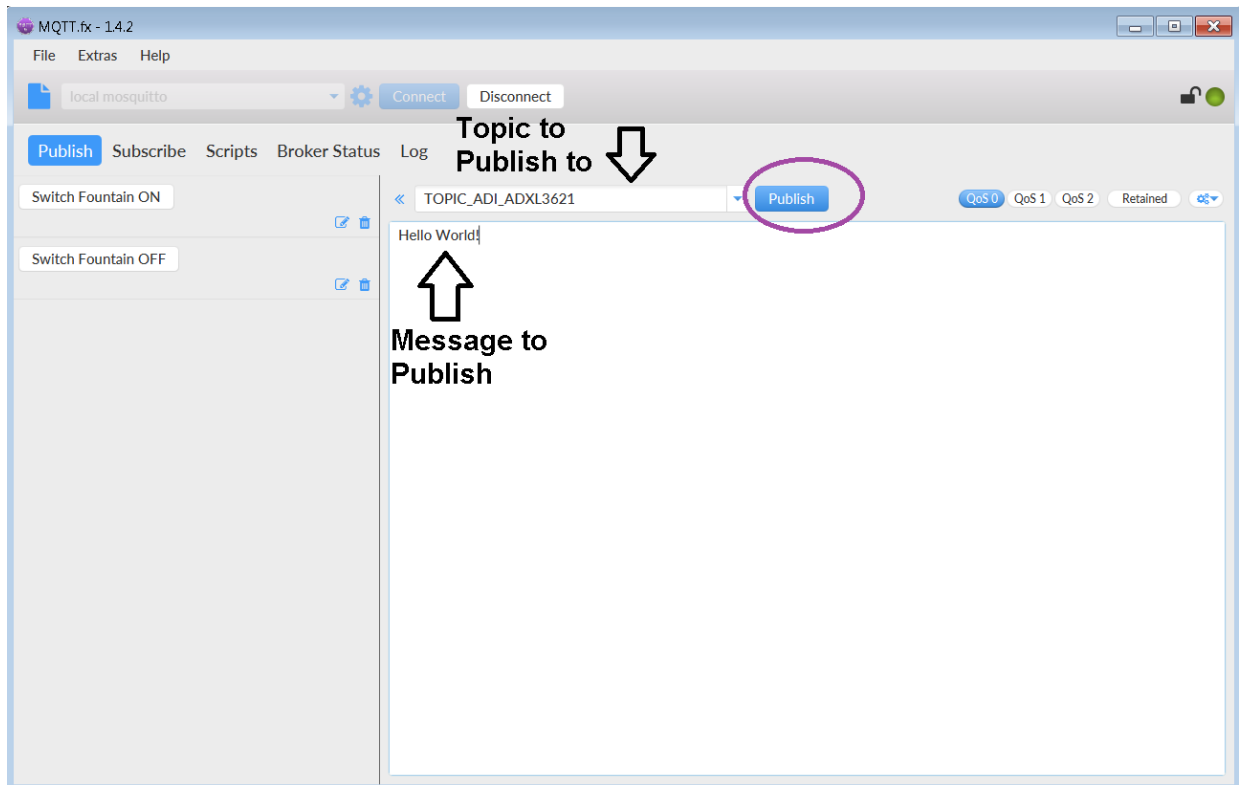
3. Supply the Port and IP address of the broker you wish to connect to. To get this information please read the section [Configuring a Local MQTT Broker](#).



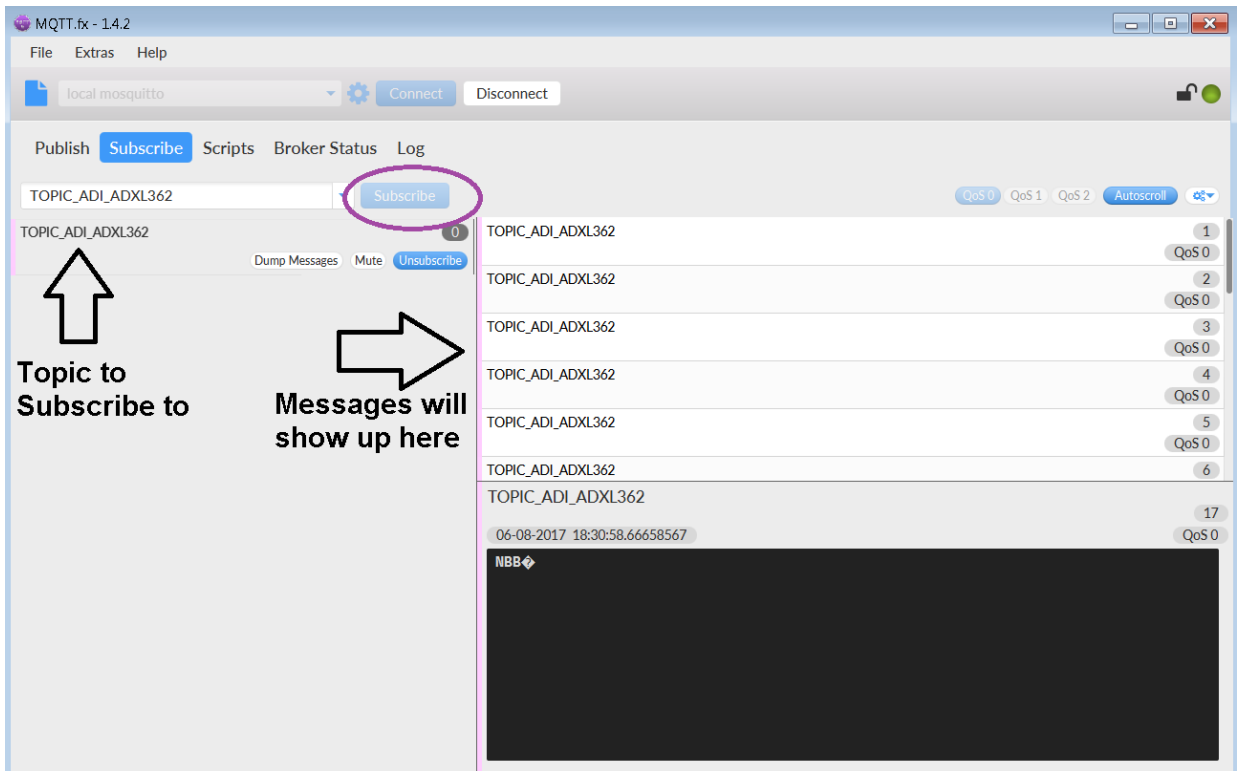
4. Now hit the "Connect" button to connect to the broker.



5. To Publish a message hit the "Publish" button.



6. To subscribe to a topic, go to the subscribe tab and press the button "Subscribe". If messages are being published on this topic, they will show up on this page.



5 Wi-Fi Software

5.1 Overview

The Wifi Software Pack contains software that interfaces with the ESP8266, and provides the user with a simple, high-level interface for performing Wi-Fi and MQTT operations. The Wi-Fi software can be incorporated into an application by adding the "Wi-Fi ESP8266" component in the RTE configuration (described in the previous section). This will add the Wi-Fi source files, a Wi-Fi static configuration, and the necessary include paths to the project. This section describes the organization of the Wi-Fi software and how to use it.

Note that this section will cover the software a high-level, please see the Doxygen documentation located in *Documents/WiFi_Software_Doxygen_index.html* for specific details about each API, data structure, enumeration, and macro.

5.2 Communication Model

At a high level, a processor and ESP8266 communicate using the AT command set over a UART or SPI interface. After a command is sent from the processor, the ESP8266 will reply with a response. The ESP8266 can also issue events to the processor. Unlike responses, events are asynchronous. Figure 9 illustrates this communication model.

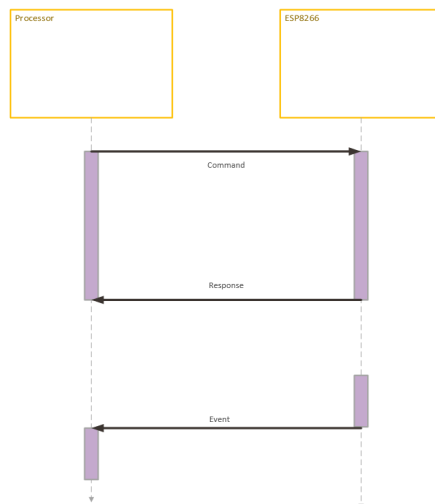


Figure 9. Basic Communication Model

5.3 Software Organization

The Wi-Fi software is partitioned into three layers, built on each other and providing different levels of abstraction from the communication model.

Layer	Include	Description
NoOS Framework Layer	<code><framework/noos/adi_wifi_noos.h></code>	<i>Control flow</i> This layer is used to manage control flow by blocking until specific responses or events are received. The layer will also handle unexpected events by passing them to the application via a callback function.
Companion (Radio) Layer	<code><radio/adi_wifi.h></code>	<i>Encoding and decoding of packets</i> This layer deals with the encoding and decoding of packets sent and received from the ESP8266. These packets use the AT command set.
Transport Layer	<code><transport/adi_wifi_transport.h></code>	<i>Writing and reading packets</i> This layer implements the interface used to communicate with the ESP8266. The user has the option to set this to SPI or UART.

5.4 Software Use

Applications interact with the companion and framework layer APIs when performing Wi-Fi operations. Depending on the mode of operation, the call sequence will vary. This section will explain how to use these APIs for the various modes of operation. Also, the process of receiving data from the ESP8266 will vary based on these modes.

There are two modes of operation for the Wi-Fi software:

- Command and response
- Receiving unexpected events

5.4.1 Command and Response

The simplest form of communication with the ESP8266 is by issuing commands and receiving a response directly after. Applications perform this action by using the APIs located in the Companion Layer. The command-response sequence is completed by only a single function call from the application as shown below. This is a blocking function call. The ESP8266 operates serially, handing a single AT command at a time.

```
#include <radio/adi_wifi.h>

...

ADI_WIFI_RESULT eResult;

eResult = adi_wifi_radio_XXX(...);
```

Example 1. Command-Response API Call

The function arguments will vary depending on the API, and are described in the Doxygen documentation. If the response packet will contain data the application needs, the API will have a pointer where this data will be stored. The application must allocate the memory, pass the pointer, and wait for the function call to complete successfully before using the data. An example is shown below.

```
#include <radio/adi_wifi.h>

...

ADI_WIFI_RESULT eResult;
uint8_t aVersion[256u];

// The response will contain the version info, which will be stored in aVersion
eResult = adi_wifi_radio_GetVersionInfo(aVersion, 256u);

if (eResult == ADI_WIFI_SUCCESS)
{
    // aVersion has been filled with the correct data
}
```



```
else
{
// aVersion has not been filled
}
```

Example 2. Data Received in Response

5.4.2 Receiving Unexpected Events

The ESP8266 can issue events to the processor even if the application did not explicitly trigger it with a command. An example of this would be an application running as a MQTT subscriber. Any published data will be received unexpectedly and asynchronously. In a single-threaded model, the application must periodically pass control to the Framework Layer to handle any unexpected events. An API has been provided to perform this action. If an event occurs during the period in which the application has passed control to the Framework Layer, control will be given back to the application via a callback function. The callback function event contains the data received in the event. The Doxygen documentation contains details on which events pass data through this callback and how it should be interpreted. An example of this flow is shown below.

```
#include <radio/adi_wifi.h>
#include <framework/noos/adi_wifi_noos.h>

...

static void ApplicationCallback(void * pCBParam, uint32_t Event, void * pArg)
{
    switch (Event)
    {
        case CMD_NONE:
            adi_wifi_ParseSubscriberData();

            break;

        ...

    }
}

int main(void)
{
```

```

ADI_WIFI_RESULT eResult;

/* Initialize the ESP8266 Wi-Fi module */
eResult = adi_wifi_radio_Init(adi_wifi_ApplicationCallback)

...

while(1u)
{
    // This function call will block for a given period of time and dispatch e

    eResult = adi_wifi_DispatchEvents(...);

    // Application specific code

    ...

}

return 1;
}

```

Example 4. Dispatching Events to the Callback

Note that even though the application callback is executed at the thread level, it should be minimal to avoid blocking other events from being received. Commands shall not be issued from the application level. This concludes the discussion on the two modes of operation for the Wi-Fi software. Using the software described here, applications can easily interface with the ESP8266 and perform specific actions on Wi-Fi or MQTT events.

5.4.3 Data Received in Events

The interface for receiving data from response packets was simple and straightforward - when the command-response API completes, the memory allocated by the application is filled. For events, it is slightly more complicated due to their asynchronous nature. To simplify the application software, the Companion Layer will cache the data received from an event in local memory. The application then calls the getter API to extract the cached data. The getter API is located in the Companion Layer and is called `adi_wifi_GetData()`.

The cached data is only valid until the next event function call, since the companion layer reuses the response buffer as command buffer in order to save memory. More explicitly, for an unexpected event, the application should to call the getter function from the application callback.