Getting started with the X-CUBE-WIFI1 Wi-Fi functions and applications software expansion for STM32Cube

Introduction

This document describes how to get started with the X-CUBE-WIFI1 expansion software for STM32Cube. X-CUBE-WIFI1 provides the complete middleware to build Wi-Fi applications using the SPWF01SA/SPWF04SA serial-to-Wi-Fi module. It is easily portable across different MCU families, thanks to STM32Cube.

This package contains software and sample applications that allow the user to:

- configure the SPWF01SA/SPWF04SA to connect to Access Points
- configure the SPWF01SA/SPWF04SA in STA, MiniAP and IBSS modes of operation
- scan available networks and choose APs to connect to
- use TCP/UDP connections to open, close and read/write to sockets/server sockets
- perform RESTful actions like HTTP-GET/POST
- perform actions on files on a webserver

The software provides implementation examples for STM32 Nucleo platforms equipped with the X-NUCLEO-IDW01M1 or X-NUCLEO-IDW04A1 expansion board, featuring the SPWF01SA/SPWF04SA Serial-to-Wi-Fi Module.

The software is based on STM32Cube technology and expands the range of STM32Cube-based packages.

1 Acronyms and abbreviations

Table 1. Acronyms and abbreviations

Acronym	Description
AP	Access Point
BSP	Base Support Package
GPIO	General Purpose Input/Output
HAL	Hardware Abstraction Layer
HTML	Hyper Text Markup Language
HTTP	Hyper Text Transfer Protocol
I²C	Inter Integrated Circuit
IBSS	Independent Basic Service Set
MCU	Microcontroller Unit
Mini-AP	Mini Access Point
RAM	Random Access Memory
REST API	Representational State Tranfer APIs
SSID	Service Set Identifier
STA	Station mode
TCP	Transmission Control Protocol
UART	Universal Asynchronous Receiver/Transmitter
UDP	User Datagram Protocol
Wi-Fi	Wireless LAN based on IEEE 802.11
WIND	Wi-Fi Indication
WLAN	Wireless Local Area Network

2 What is STM32Cube?

STM32Cube™ represents the STMicroelectronics initiative to make developers' lives easier by reducing development effort, time and cost. STM32Cube covers the STM32 portfolio.

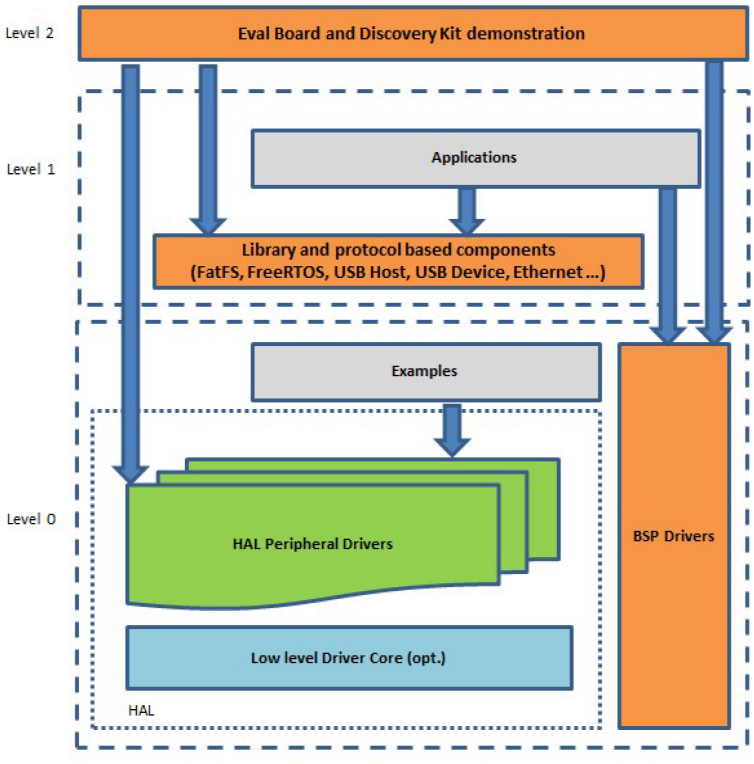
STM32Cube version 1.x includes:

- STM32CubeMX, a graphical software configuration tool that allows the generation of C initialization code using graphical wizards.
- A comprehensive embedded software platform specific to each series (such as the STM32CubeF4 for the STM32F4 series), which includes:
  - the STM32Cube HAL embedded abstraction-layer software, ensuring maximized portability across the STM32 portfolio
  - a consistent set of middleware components such as RTOS, USB, TCP/IP and graphics
  - all embedded software utilities with a full set of examples

2.1 STM32Cube architecture

The STM32Cube firmware solution is built around three independent levels that can easily interact with one another, as described in the diagram below.

Figure 1. Firmware architecture



Level 0: This level is divided into three sub-layers:

- Board Support Package (BSP): this layer offers a set of APIs relative to the hardware components in the hardware boards (Audio codec, IO expander, Touchscreen, SRAM driver, LCD drivers. etc…); it is based on modular architecture allowing it to be easily ported on any hardware by just implementing the low level routines. It is composed of two parts:

  - Component: is the driver relative to the external device on the board and not related to the STM32, the component driver provides specific APIs to the external components of the BSP driver, and can be ported on any other board.
  - BSP driver: links the component driver to a specific board and provides a set of easy to use APIs. The API naming convention is BSP_FUNCT_Action(): e.g., BSP_LED_Init(), BSP_LED_On().

- Hardware Abstraction Layer (HAL): this layer provides the low level drivers and the hardware interfacing methods to interact with the upper layers (application, libraries and stacks). It provides generic, multi-instance and function-oriented APIs to help offload user application development time by providing ready to use processes. For example, for the communication peripherals (I²C, UART, etc.) it provides APIs for peripheral initialization and configuration, data transfer management based on polling, interrupt or DMA processes, and communication error management. The HAL Drivers APIs are split in two categories: generic APIs providing common, generic functions to all the STM32 series and extension APIs which provide special, customized functions for a specific family or a specific part number.
- Basic peripheral usage examples: this layer houses the examples built around the STM32 peripherals using the HAL and BSP resources only.

Level 1: This level is divided into two sub-layers:

- Middleware components: set of libraries covering USB Host and Device Libraries, STemWin, FreeRTOS, FatFS, LwIP, and PolarSSL. Horizontal interaction among the components in this layer is performed directly by calling the feature APIs, while vertical interaction with low-level drivers is managed by specific callbacks and static macros implemented in the library system call interface. For example, FatFs implements the disk I/O driver to access a microSD drive or USB Mass Storage Class.
- Examples based on the middleware components: each middleware component comes with one or more examples (or applications) showing how to use it. Integration examples that use several middleware components are provided as well.

Level 2: This level is a single layer with a global, real-time and graphical demonstration based on the middleware service layer, the low level abstraction layer and basic peripheral usage applications for board-based functions.

3 X-CUBE-WIFI1 software expansion for STM32Cube

3.1 Overview

X-CUBE-WIFI1 is a software package that expands the functionality provided by STM32Cube.

The key features of the package are:

- complete middleware to build Wi-Fi applications using the SPWF01SA/SPWF04SA module
- abstraction APIs to configure the module in STA, MiniAP and IBSS mode
- abstraction APIs to open/close, read/write sockets/socket servers in TCP/UDP mode
- comprehensive user APIs and documentation
- easy portability across different MCU families thanks to STM32Cube
- free user-friendly license terms
- sample applications available on X-NUCLEO-IDW01M1 or X-NUCLEO-IDW04A1 board plugged onto a NUCLEO-F103RB, NUCLEO-F401RE, NUCLEO-L476RG or NUCLEO-L053R8 board

This software is used for configuring various Wi-Fi functions such as STA modes, MiniAP mode and IBSS mode on the SPWF01SA/SPWF04SA module, running with STM32. Apart from the configuration of different modes, the package also provides APIs which allow the usage of sockets, restful functions like HTTP-GET and HTTP-POST, file operations on web servers and various utility APIs as described in section Section 4.3 Hardware and software setup.

The package also includes few sample applications that the developer can use to start experimenting with the code. Please refer to section Section 4.3 Hardware and software setup for further details on the applications provided with the package.

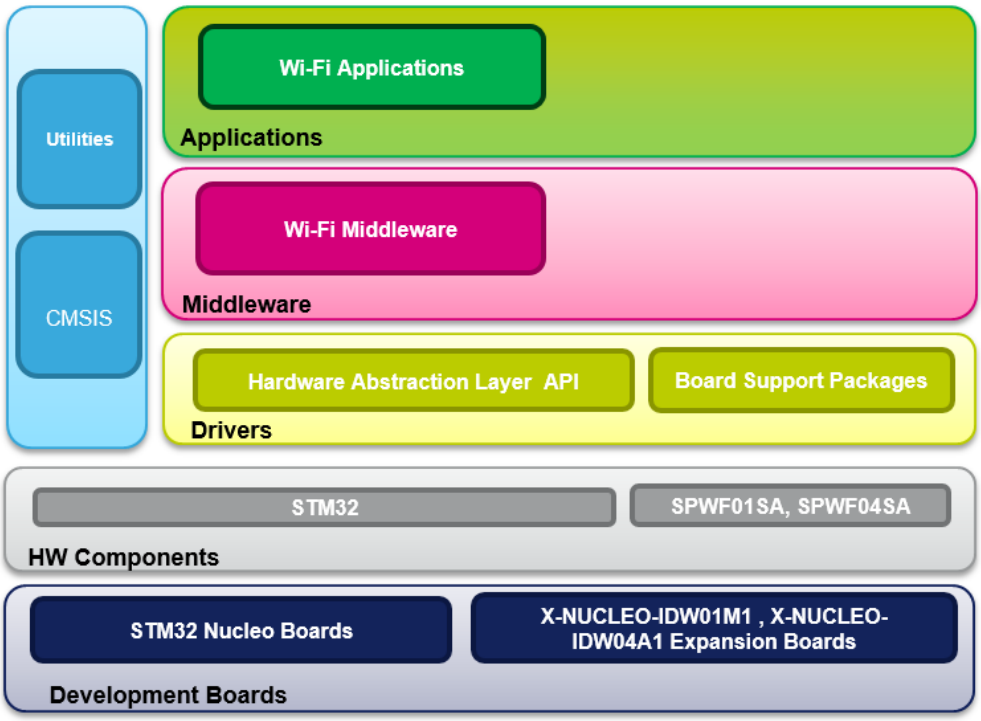
3.2 Architecture

This software is an expansion for STM32Cube; as such, it fully complies with the STM32Cube architecture and expands on it to enable development of applications involving Wi-Fi connectivity. Refer to the previous chapter for an overview of the STM32Cube architecture.

The software is based on the STM32CubeHAL hardware abstraction layer for the STM32 microcontroller. The package extends STM32Cube by providing a board support package (BSP) for the Wi-Fi expansion board and some middleware components for serial communication with a PC.

The following figure outlines the software architecture of the package:

Figure 2. **X-CUBE-WIFI1 software architecture**

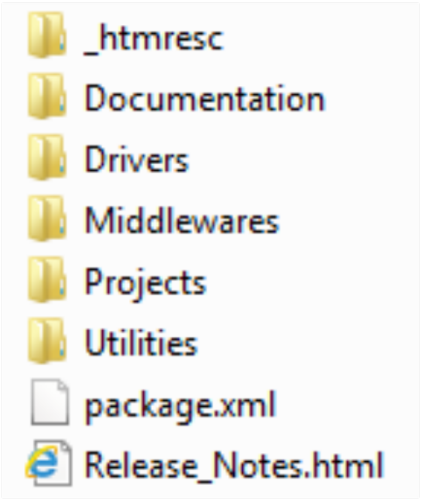


The application software accesses and controls the Wi-Fi expansion board via the HAL and BSP described below.

- **STM32Cube HAL driver layer**: this layer provides a generic multi instance simple set of APIs (application programming interfaces) to interact with the upper layers (application, libraries and stacks). It is composed of generic and extension APIs. It is directly built around a generic architecture and allows the layers that are built upon, such as the middleware layer, to implement their functionalities without dependencies on the specific hardware configuration for a given Microcontroller Unit (MCU). This structure improves the library code reusability and guarantees an easy portability on other devices.
- **Board Support Package (BSP) layer**: the software package needs to support the peripherals on the STM32 Nucleo board apart from the MCU. This software is included in the board support package (BSP). This is a limited set of APIs which provides a programming interface for certain board specific peripherals, e.g. the LED, the user button etc. This interface also helps in identifying the specific board version. In case of Wi-Fi expansion board, it provides the programming interface for SPWF01SA/SPWF04SA serial-to-Wi-Fi module. It provides support for initializing and reading SPWF01SA/SPWF04SA data.

3.3 Folder structure

Figure 3. **X-CUBE-WIFI1 package folder structure**



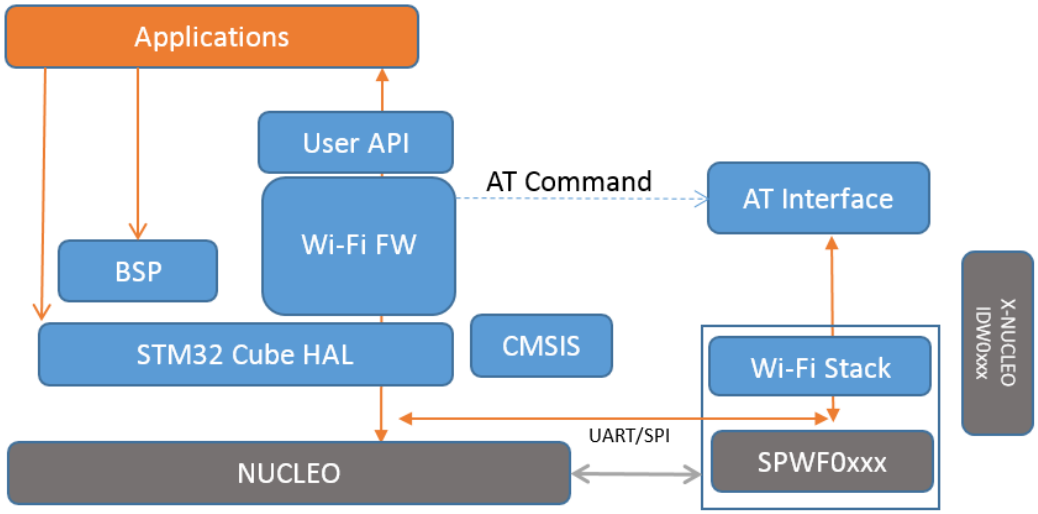
The following folders are included in the software package:

- _htmresc: contains image resources used by Release_Notes.html
- Documentation: contains a compiled HTML file generated from the source code and documentation detailing the software components and APIs.
- Drivers: contains the HAL drivers and the board specific drivers for each supported board or hardware platform, including the onboard components and the CMSIS layer - a vendor-independent hardware abstraction layer for the Cortex-M processor series.
- Middlewares: contains the Wi-Fi middleware and the core source code to manage the data sent by the SPWF01SA/SPWF04SA module and implement all the user-sde API's.
- Projects: contains a sample application to access various X-NUCLEO-IDW01M1 or X-NUCLEO-IDW04A1 board configurations for the NUCLEO-L053R8, NUCLEO-F401RE, NUCLEO-L476RG and NUCLEO-F103RB platforms.
- Utilities: contains utility programs related to server socket and socket client. These programs are readily available to test the sample applications provided in the package.

3.4 APIs

Detailed technical information decribing the user API functions and parameters can be found in the *X-CUBE-WIFI1.chm* compiled HTML file located in the **Documentation** folder.

Figure 4. Detailed software architecture



From the figure above, we can see that the AT command infrastructure is abstracted to an API that the user can easily manipulate for Wi-Fi module operation. The AT commands pass through the X-CUBE HAL layers and drivers, down to the hardware and through the UART/SPI communication interface to the SPWF01SA module. The module firmware processes the AT commands and the results are communicated back to the platform microcontroller throught the UART/SPI.

Details regarding the user APIs follow.

- `wifi_reset(void)`: active low WIFI_RST GPIO pin for 200 ms; Wi-Fi module resets and user receives indication.
- `wifi_init(wifi_config *config)`: initiates the timers, allocates memory to buffers and starts the background process to acquire Wi-Fi signals through the UART/SPI; it creates the Wi-Fi instance and configures it according to the `wifi_config` data structure

Note: This function must be called by the user before using the functional APIs. The user must provide the API with the wifi_config structure which can be used to set a number of parameters during initialization of the wi-fi module. Further details are available in Section 3.7.1 Wi-Fi module/interface selection.

- `wifi_restore(void)`: restores the Wi-Fi factory values.
- `wifi_enable(wifi_bool enable)`: enables/disables the Wi-Fi radio interface.
- `wifi_ap_start(uint8_t * ssid, uint8_t channel)`: starts the Wi-Fi module in Mini AP mode with a given SSID; the AP is located in the selected channel.
- `wifi_connect(uint8_t * ssid, uint8_t * sec_key, WiFi_Priv_Mode priv_mode)`: connects the Wi-Fi to a given AP; the user must provide the AP SSID, security key, and network privacy mode.
- `wifi_disconnect(void)` **:** disconnect the Wi-Fi from the AP.
- `wifi_adhoc_create(uint8_t * ssid, WiFi_Priv_Mode priv_mode)`**:** create an ad-hoc network (IBSS) with given SSID and privacy settings.
- `wifi_standby(uint8_t arg_standby_time)`: put the module in stand-by for a specific interval; timer expiration and subsequent module wakeup of the is signaled by a callback which needs to be implemented by the user.

Note: You can put the module in stand-by for an indefinite amount of time, in which case, a wakeup can be triggered by the specific wakeup API (see below).

- `wifi_wakeup(wifi_bool enable)`: resume module operation from either stand-by or sleep (enable=1), or return to sleep state after a temporary wakeup (enable=0).
- `wifi_network_scan(wifi_scan * scan_result, uint8_t max_scan_number)`: scan the Wi-Fi network and return the list of the available APs with relative capabilities (RSSI and SSID); the user

typically allocates an array of scan results and the API populates it with scan results according to the `wifi_scan` data structure, with the RSSI and the SSID string.

Note: The user must also provide the desired number of scan results to be read in a single API call; the maximum is 15.

- `wifi_set_socket_certificates(TLS_Certificate ca_certificate, TLS_Certificate tls_certificate, TLS_Certificate certificate_key, TLS_Certificate client_domain, uint32_t tls_epoch_time)`: this API is used to set the TLS parameters of the module. This API is updated from version 1.1.0 with the addition of epoch time parameter. All three modes of anonymous, one-way and mutual authentication are supported and can be given as the first parameter. The usage is “m” for mutual authentication and “o” for one-way authentication. For anonymous connections, this API does not need to be called. The previous API without the epoch time in X-CUBE-WIFI1 v1.1.0 is deprecated. Older applications will still compile and run with the default epoch time of Mon, 25 Jan 2016 13:14:17 GMT.
- `wifi_socket_client_ open ( uint8_t * hostname, uint32_t port_number , uint8_t * protocol, uint8_t * sock_id )`: opens a network socket client; the module supports up to 8 open TCP/UDP socket clients.
- `wifi_socket_client_ write ( uint8_t sock_id , uint16_t DataLength,char * pData )`: writes data to an open socket client; the maximum supported data length is 4096 bytes, but the SW currently supports the writing of 1024 bytes in a single write operation.
- `wifi_socket_client_ close ( uint8_t sock_close_id )`: closes an open socket client, whose id is provided as a parameter to this API.
- `wifi_list_open_client_socket(void)`: lists all opened socket clients.
- `wifi_socket_server_open(uint32_t port_number, uint8_t * protocol, uint8_t * sock_id)`: opens the socket server at a designated port number; so the module starts listening for incoming clients; data from remote clients is received through a callback function (see Section 3.6.1 Wi-Fi user event callbacks) implemented by the user. The SPWF01SA module supports only one socket server and the third parameter "sock_id" is not required. In case of the SPWF04 module this parameter returns the socket server id created by the module itself.
- `wifi_socket_server_close(uint8_t server_id, uint8_t client_id)`: closes an opened server socket. For the SPWF01 module, the argument is "void" as there is just one single server socket supported. For the SPWF04 module, you have to provide the server id and the client id of the socket to be closed.
- `wifi_socket_server_write(uint8_t server_id, uint8_t client_id, uint16_t DataLength,char * pData)`: writes data in the server socket. For the SPWF04, you have to provide the id of the client and server socket in the first 2 parameters. For the SPWF01, these two parameters don't exist. The maximum supported data length is 4096 bytes, but the software currently supports the writing of 1024 bytes in a single write operation.
- `wifi_list_bound_client_socket(void)`: lists all bound socket clients on all socket servers.
- `wifi_http_get (uint8_t * hostname, uint8_t * path, uint32_t port_number )`: sends a HTTP-GET request to a specific host; the user must provide the hostname, the path and the port number.

Note: the HTTP-GET result is returned to the user with a callback (see Section 3.6.1 Wi-Fi user event callbacks).

- `wifi_http_ post ( uint8_t * url_path )`: sends an HTTP-POST request to a specific host.
- `wifi_fw_ update ( uint8_t * hostname, uint8_t * filename_path , uint32_t port_number )`: updates firmware over-the-air (FOTA) with an “.ota” firmware file hosted by the user on the desired server; the user must provide the hostname and the complete path of the file on the server, including the filename and the port number.
- `wifi_gpio_ init ( GpioPin pin, char* dir , char irq )`: configures any of the 15 GPIO pins of the module for the direction of the pin (input or output) and the setting of the interrupt if needed (e.g., rising edge, falling edge, or both).
- `wifi_gpio_ read ( GpioPin pin, uint8_t * val , uint8_t * dir )`: reads the current configuration of the GPIO pin; the val parameter reads the current value on the pin and the direction parameter returns 1 for “in” and 0 for “out”.
- `wifi_gpio_ write ( GpioPin pin, GpioWriteValue value)`: writes the desired value of the pin in the GPIO.

- `wifi_file_ create ( char *pFileName,uint16_t alength , char * databuff )`: creates a new file which is stored in RAM; the maximum file size is 4096 bytes.
- `wifi_file_delete (char * pFileName )`: deletes an existing file; static files may not be deleted, only overridden.
- `wifi_file_ list ( void)`: lists each stored file stored in the file system with its name, size and the location (I=Internal FLASH Memory, D=RAM Memory, E=External FLASH Memory); the list data is sent to the user with the callback `ind_wifi_file_data_available(uint8_t * data_ptr)`. The user needs to implement this function to access the list data.
- `WiFi_Status_t wifi_file_print_content(uint8_t *FileName, int16_t offset, int16_t length)`: shows the content of a file. The file content data is sent to the user via the callback `ind_wifi_file_data_available(uint8_t * data_ptr)`. For the SPWF04, the user has to provide the offset and the length of the file content to be printed on the console. For the SPWF01, only the file name is provided and the whole file is printed on the console. The user has to implement this function to access the content.
- `wifi_ping(uint8_t * hostname)`: sends ping messages to the "hostname" which is provided as a parameter for this API.
- `wifi_file_rename(char *FileName, char *Mod_FileName)`: renames a file from FileName to Mod_FileName.
- `wifi_file_image_create(uint8_t * HostName,uint8_t * FileName, uint32_t port_number)`: update the file system with the file system image in HostName, naming it with the FileName, and the port_number to be accessed.
- `wifi_get_IP_address(uint8_t *ip_addr)`: returns the IP address assigned to the module after connecting to AP or after mini-AP has been set up.
- `wifi_get_MAC_address(uint8_t *mac_addr)`: returns the module MAC address.
- `WiFi_Status_t wifi_websocket_client_open(uint8_t * hostname, uint32_t port_number, uint8_t * protocol, uint8_t * sock_id)`: opens a websocket client and connects to a websocket server.
- `WiFi_Status_t wifi_websocket_client_write(uint8_t sock_id, uint32_t DataLength,char * pData)`: writes to a websocket client id with specified data and data length.
- `WiFi_Status_t wifi_list_open_web_client_socket(void)`: lists opened websockets. Data are received through the callback `ind_wifi_socket_list_data_available(uint8_t * data_ptr)`.
- `WiFi_Status_t wifi_websocket_client_close(uint8_t sock_close_id)`: closes a websocket id with normal closure status (option to provide a specific status code, as in the AT command is not currently provided).
- `WiFi_Status_t wifi_mqtt_connect(uint8_t * hostname, uint32_t port_num)`: opens a connection with an MQTT broker available at hostname and port_num.
- `WiFi_Status_t wifi_mqtt_publish(uint8_t *topic, uint32_t DataLength, char *pData )`: publishes a message to an MQTT broker specifying the topic, data and data length.
- `WiFi_Status_t wifi_mqtt_disconnect(void)`: disconnects the connection with an MQTT broker.
- `WiFi_Status_t wifi_mqtt_subscribe(uint8_t *topic)`: subscribes to any specified topic from a connected broker.
- `WiFi_Status_t wifi_mqtt_unsubscribe(uint8_t *topic)`: unsubscribes from a specified topic from a connected broker.
- `WiFi_Status_t wifi_tftp_get(uint8_t * hostname, uint32_t port_number, uint8_t * filename)`: uses TFTP protocol to get a file specified by *filename* from a host specified by *hostname*.
- `WiFi_Status_t wifi_send_mail(uint8_t *hostname, uint32_t port_num, uint8_t *from, uint8_t *to, uint8_t *subject, uint32_t length, char *Data)`: sends a secure e-mail via the SMTP protocol.
- `WiFi_Status_t wifi_fill_input_buffer(uint32_t DataLength, char * Data)`: used to send a message to a remote interface (SSI), as in the case of an **Input to Remote** (WIND:56) callback which has to be implemented by the callback `ind_wifi_inputssi_callback(void)`.
- `WiFi_Status_t wifi_file_erase_external_flash(void)`: erases the external httpd filesystem (available only on SPWF01SA).

- `WiFi_Status_t wifi_unmount_user_memory(int8_t volume, int8_t erase)`: unmounts or erases user memory volumes specified by *volume* and *erase* parameter (refer to AT command list for further information on the available options).

3.5 Sample applications

Application examples using the X-NUCLEO-IDW01M1 or X-NUCLEO-IDW04A1 expansion board with either a NUCLEO-F103RB, NUCLEO-F401RE, NUCLEO-L476RG or NUCLEO-L053R8 board are provided in the “Projects” directory, with ready-to-build projects available for multiple IDEs.

The X-NUCLEO-IDW01M1 and X-NUCLEO-IDW04A1 are expansion boards compatible with the STM32 Nucleo series of boards. The X-NUCLEO-IDW01M1 and X-NUCLEO-IDW04A1 integrate the SPWF01SA/SPWF04SA module and the embedded FW in this board enables the user to create a Wi-Fi network and other functions with simple AT commands. The AT command list is detailed in the SPWF01SA/SPWF04SA user manual available on www.st.com. The software described in this document provides an user friendly API which abstracts the AT command list to meaningful functional APIs.

This section describes the X-NUCLEO-IDW01M1 and X-NUCLEO-IDW04A1 expansion board application software delivered as part of the X-CUBE-WIFI1 package.

3.5.1 Client socket in STA mode

In this application, the Wi-Fi module is configured as STA. The example runs a network scan to check if the desired AP is available. Once connection with the AP is established, the module tries to open a socket connection with the remote server socket.

A remote socket is opened on the module behaving as a socket client. The server socket program is called `tcp socket server`. The `server.exe` program running on a PC connected to the same network as the module opens a socket at port 32000 and waits for a client to connect.

In the example, the host server name can be configured in the source code. Once connection with the server is established, the socket writes to the socket. In this particular socket, the server in question is an echo server (supplied in the utilities folder of this package).

The screenshots below demonstrate (with debug printouts enabled) the sequence whereby the module first tries to connect to the remote socket and then proceeds to write data to it; the server socket replies by echoing the data back to the client socket. The client socket receives the data via the `ind_wifi_socket_data_received()` callback implemented by the user. The data is returned as a pointer, from which the user can retrieve the data.

The following is the Tera Term output for this application:

Figure 5. **Scan result and connection to AP**

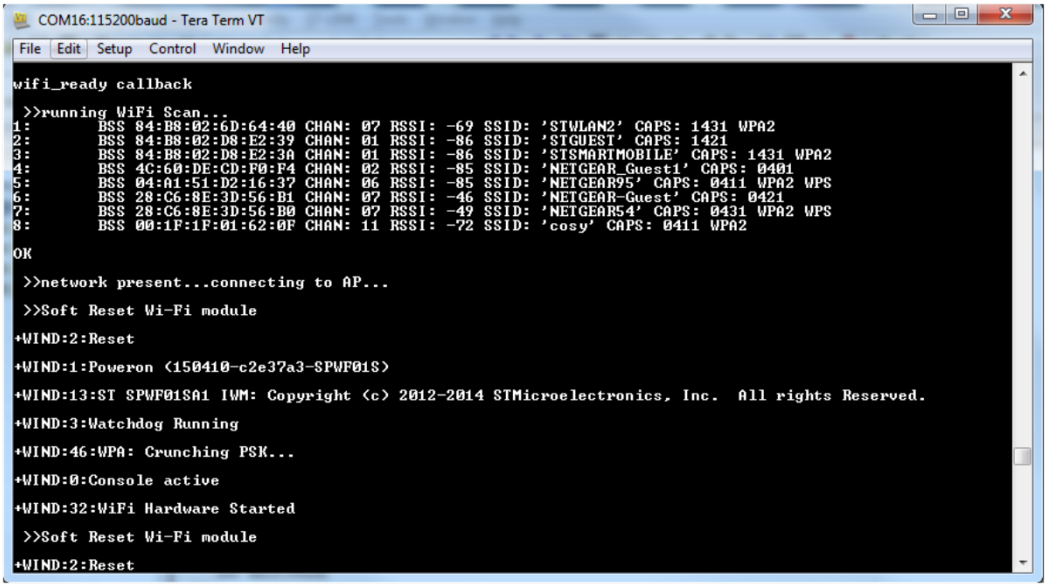
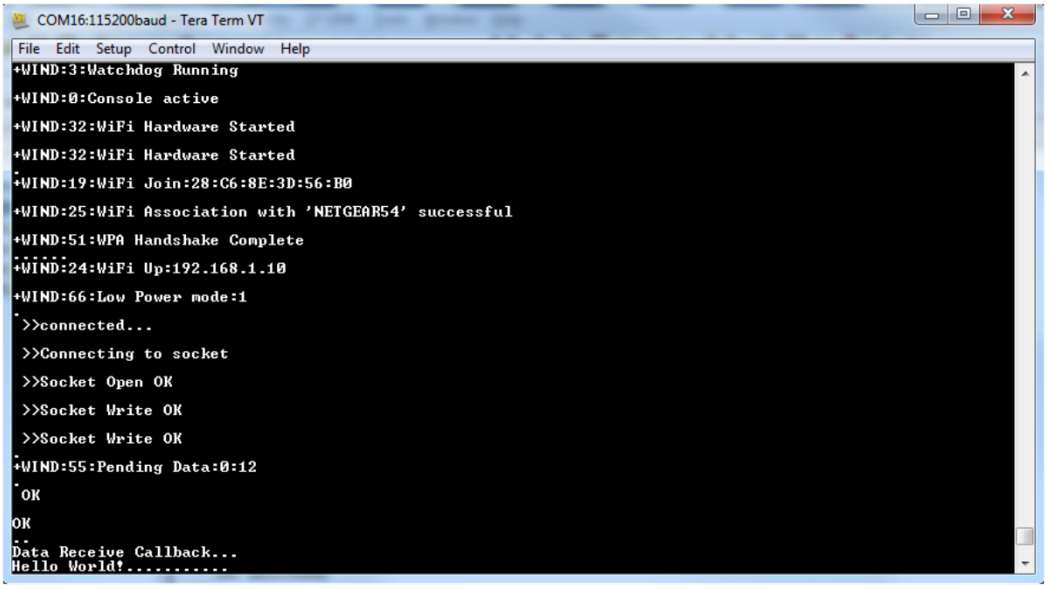
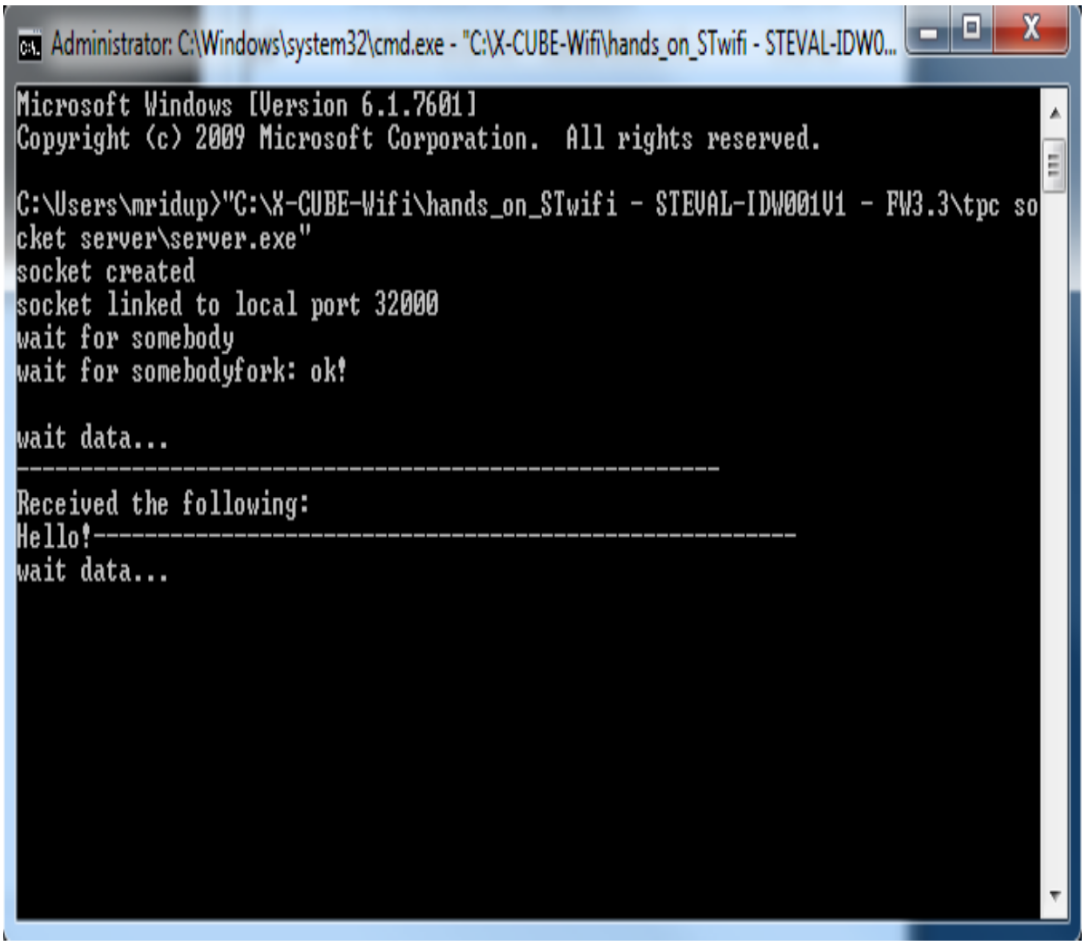


Figure 6. **Client socket read/write**



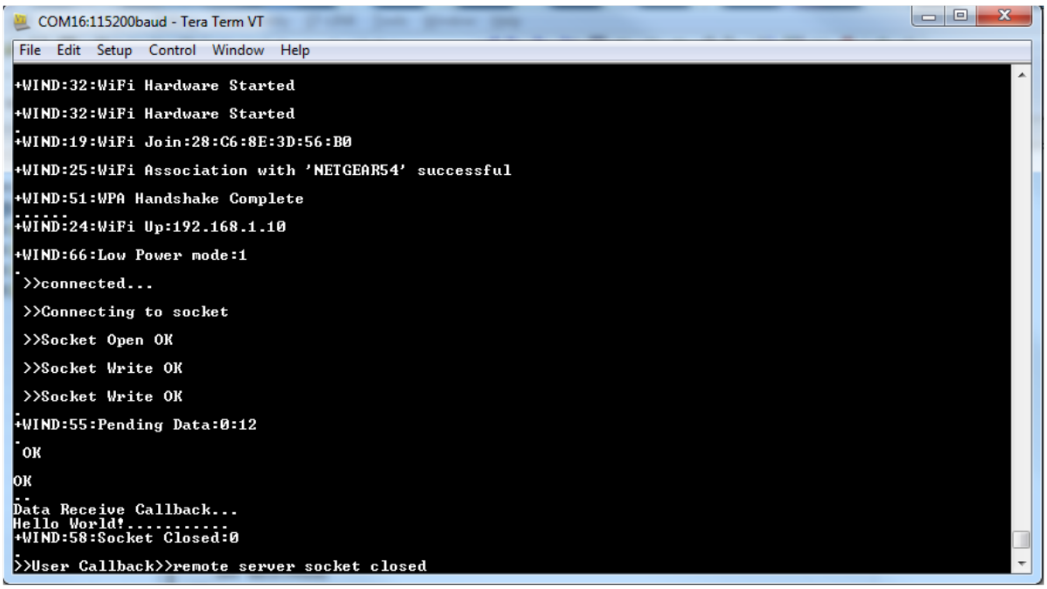
The following screenshot is of the `tcp socket server` program, where the data from the client is received.

Figure 7. **TCP socket server program**



When the server socket is closed by the user, there is another `ind_wifi_socket_client_remote_server_closed()` callback, which must also be implemented by the user; see below.

Figure 8. **Remote server socket closed callback**

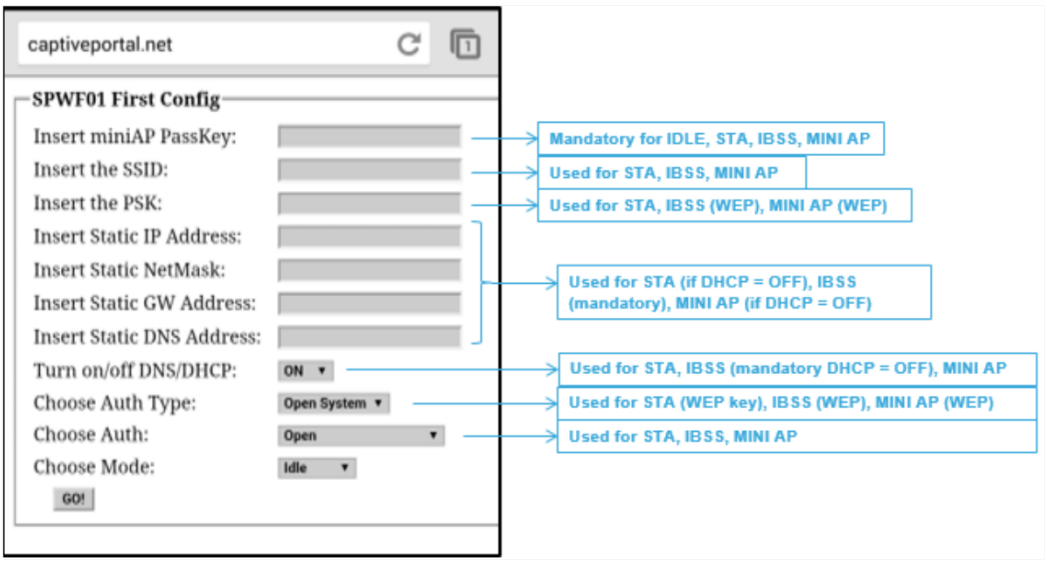


3.5.2 Socket server in MiniAP mode

In this application, the Wi-Fi module is configured as MiniAP. The miniAP network can be accessed using any Wi-Fi equipped device like a PC. In this example, the network SSID name is *SPWF01_AP*.

On connecting to the MiniAP, we can open “captiveportal.net” (the `ap_domain_name` can be changed by customizing the `wifi_configuration` structure) in our browser, which opens the page shown below.

Figure 9. **Captiveportal.net screen shot**

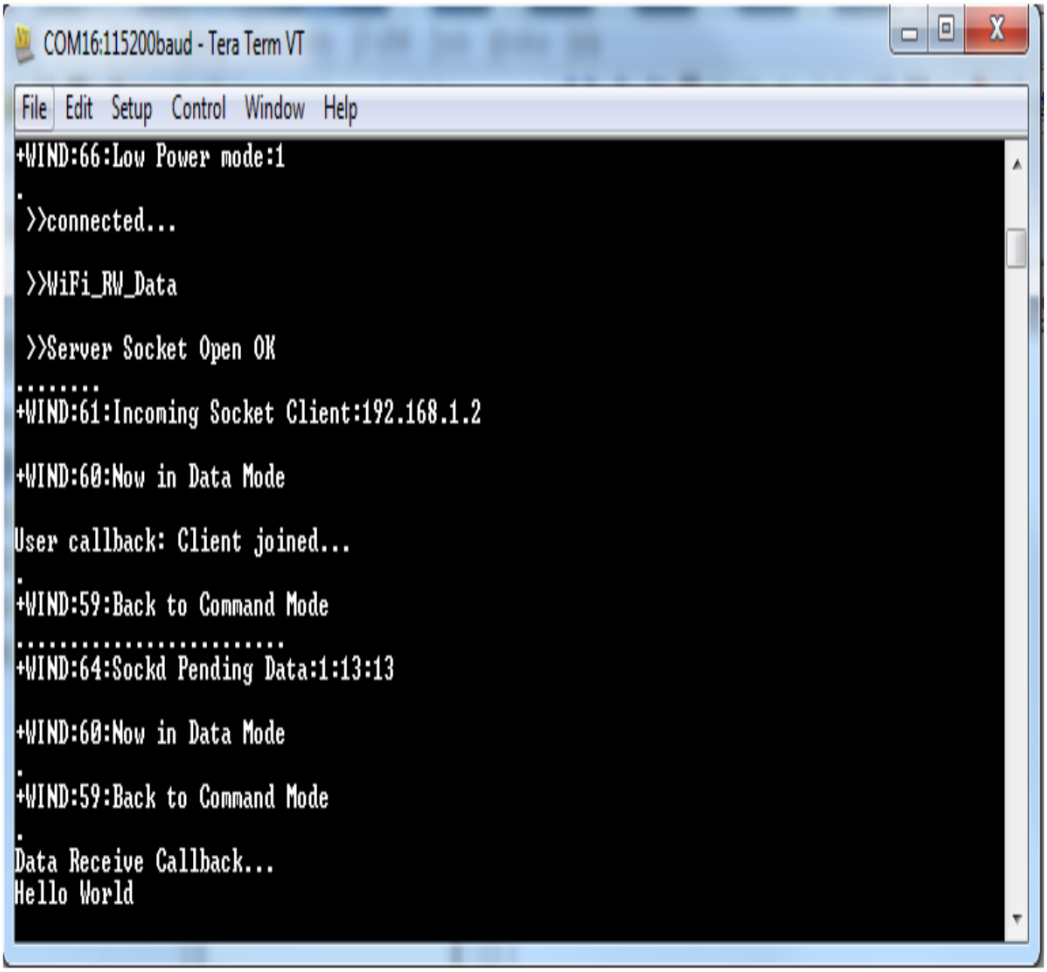


This page allows setting the module to one of IDLE, STA, IBSS or MINI AP modes. By setting the necessary parameters, we can convert the MiniAP configuration to STA mode and then have the module connect to the desired AP.

In this application, the module opens the socket server and waits for a client to connect to it. The snapshot below shows how the module waits after opening the socket and then client connects to the server socket. The debug

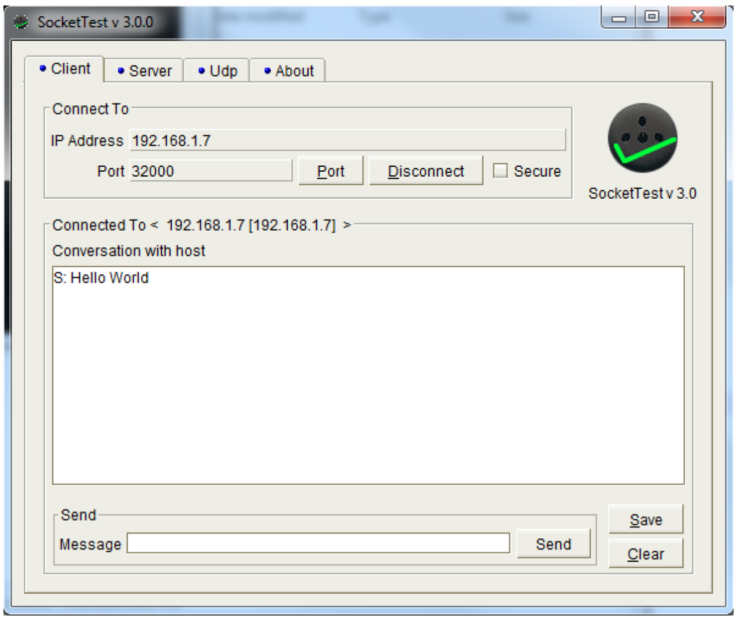
printouts are the Wi-Fi asynchronous indications received by the module. While receiving data from the client, the module switches from command mode to data mode and then back to command mode once all the data has been received by the module. All this is handled by the firmware.

Figure 10. Server socket Tera Term screen shot



The SocketTest3 client program used for this application is in the **Utilities** folder.

Figure 11. **SocketTest3 program**

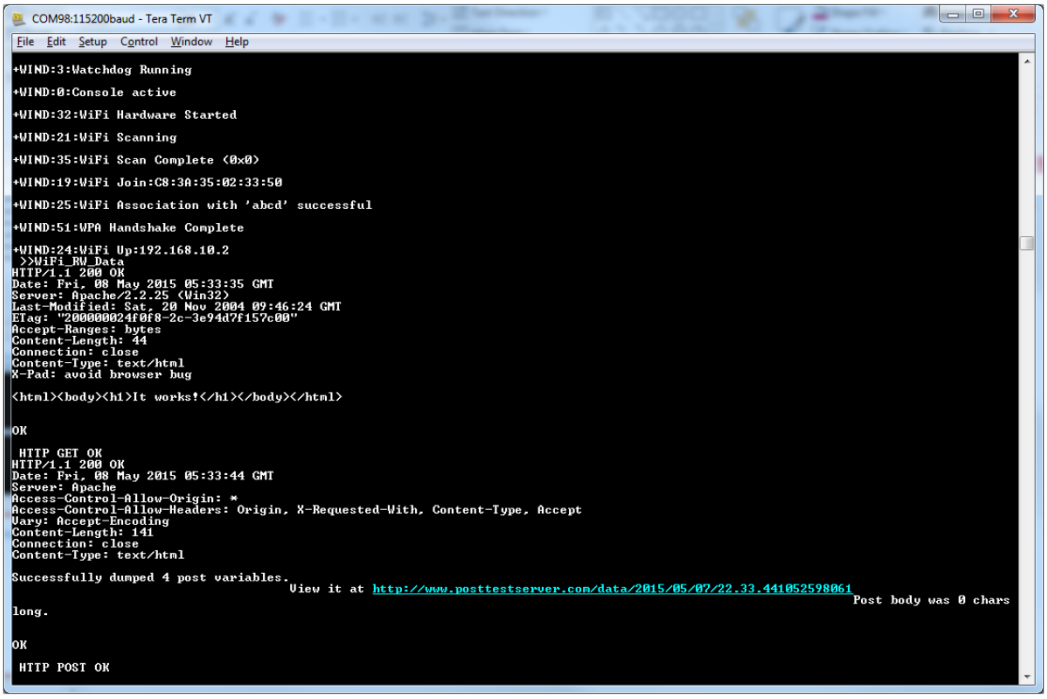


This program connects to the server socket created by the module by specifying the IP address and the open port number. After connecting to the socket, a test write can be performed by entering the text into the **Message** field.

3.5.3 HTTP-Request

This program uses the RESTful APIs of the Wi-Fi module and executes the HTTP-GET and POST operations. First, the program executes the HTTP-GET operation from the httpbin.org Web Server and returns GET data. Next, it attempts an HTTP-POST operation on the remote posttestserver.com server with 4 variables, with the desired results shown in the screenshot below.

Figure 12. **HTTP GET/POST operation**



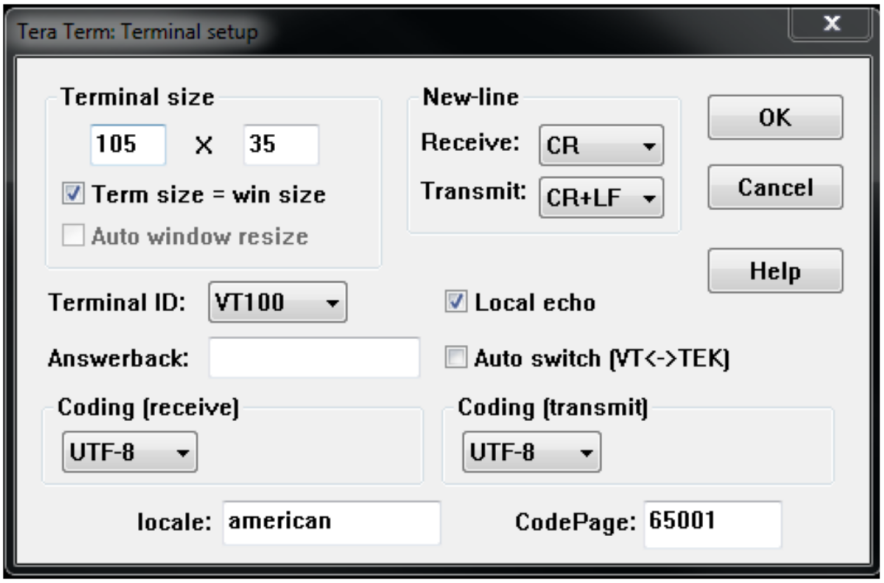
3.5.4 Wi-Fi Virtual Com application

An application is provided along with the package to simulate a virtual communication port over USB. In this application, a serial terminal like Tera Term can be opened to the STM32 Nucleo board and used to directly send AT commands to the module and the results returned directly to the terminal. This application is for users who want to understand how the AT commands behave and experiment with the AT commands directly on the module.

To run this application, FLASH the application on the STM32 Nucleo and start using the terminal to send AT commands.

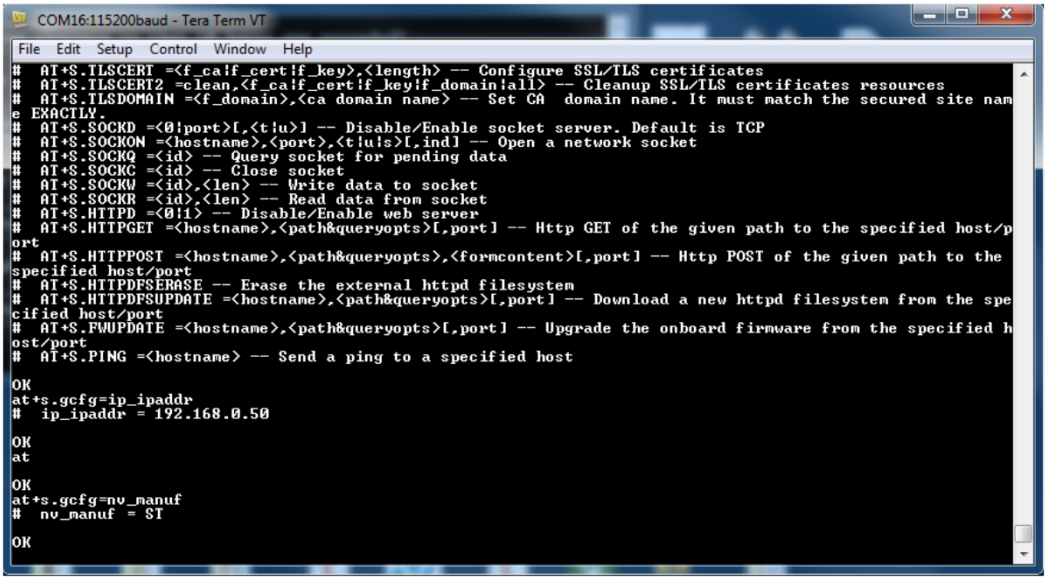
In the Terminal Setup option in Setup, select CR+LF in the Transmit listbox and check the local echo checkbox.

Figure 13. **Tera Term setup for VCOM**



The AT command to be sent to the module is then simply typed directly in the Tera Term window, as shown below.

Figure 14. **VCOM application screen shot**



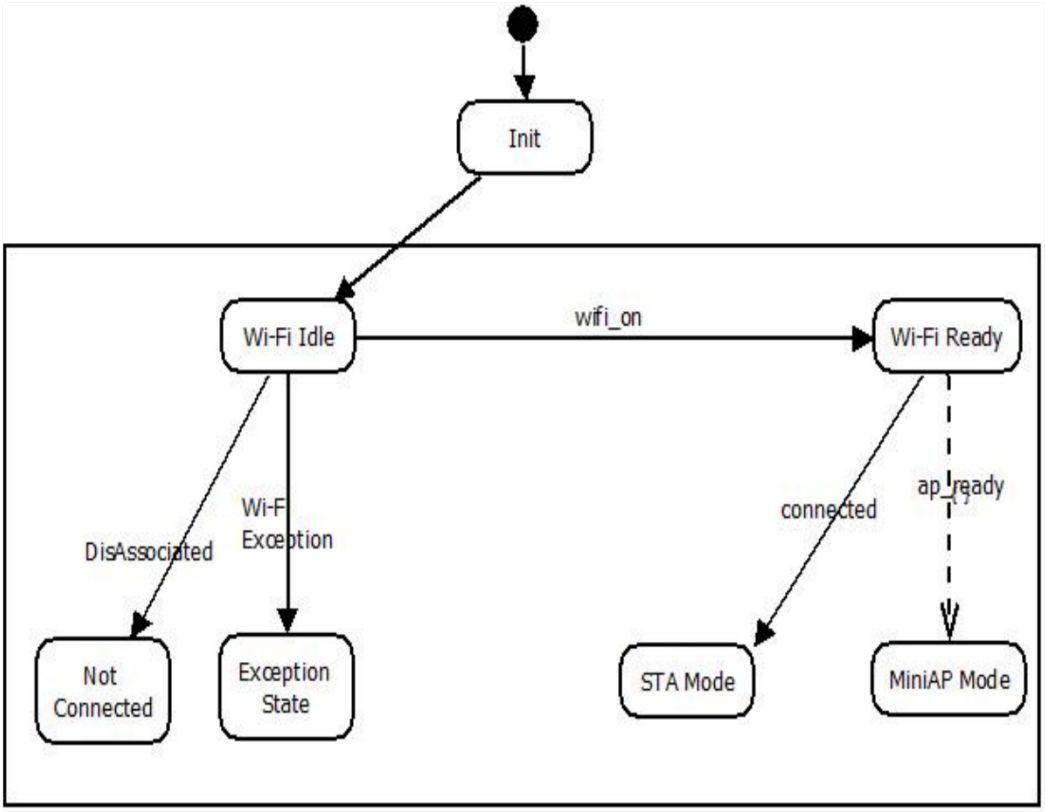
3.6 Sample application user states and state machine

In most of the sample applications in the package, the SW module is set up to run as a Wi-Fi station mode and the board boots and connects to the desired AP chosen by the user.

After the board is started and reset, the Wi-Fi SSID of the AP which the module will connect to, the password, the priv mode and the security key are set.

By default, the state machine of the Wi-Fi module is always set to the “Receive Indication” mode in which the SW running on the STM32 Nucleo is set up to receive WIND notifications from the Wi-Fi module.

Figure 15. **Application state machine**



On initialization, the Wi-Fi module is initialized with the buffer memories, the timers are started and the Wi-Fi module is reset. Following initialization, the state machine of the Wi-Fi application is set to `wifi_reset` or `wifi_idle` to wait for the following state of the Wi-Fi module. In the background, the Wi-Fi module is continuously processing WIND messages and changing the state machine of the Wi-Fi module firmware. The user can monitor state changes in the function callbacks called by the module when a state change is detected for the following initial events:

- Hardware started
- Wi-Fi connected to AP
- Wi-Fi miniAP started

On hardware startup, the function callback `ind_wifi_ready()` is called and the user can change this state from `wifi_reset` / `wifi_idle` to `wifi_ready`. When the `wifi_ready` state is achieved, the user can choose whether to configure the Wi-Fi module as a station mode device or as a miniAP mode device.

Depending on user requirements, the user calls the necessary configuration function, e.g., `wifi_connect()` to configure the device as a station device. The Wi-Fi module is configured with the required parameters setup by the user (SSID, Password, etc.) and the configuration is saved in the module.

After calling this function, when the device is connected to the access point specified by the user, the callback function `ind_wifi_connected()` is called by the module and the user can choose to change the user state machine to `wifi_connected`.

Connecting to an AP or forming a miniAP is only a small part of what a user can do with the Wi-Fi module. Other functions include opening a socket, writing to a socket, and performing restful operations like httpget or httppost functions.

3.6.1 Wi-Fi user event callbacks

The User can register callbacks based on WIND notifications from the Wi-Fi module.

WIND notifications are asynchoronous indications or messages from the SPWF01SA/SPWF04SA module, which can also arrive while an AT command is in progress.

List of call back APIs:

- `void ind_wifi_warning(WiFi_Status_t warning_code)`: this callback provides a warning signal user with a code that can be compared in the callback context with the warning codes listed in the `WiFi_Status_t` definition to take suitable action.
- `void ind_wifi_error (WiFi_Status_t error_code)`: this callback provides an error signal user with a code that can be compared in the callback context with the error codes listed in the `WiFi_Status_t` definition to take suitable action.
- `void ind_wifi_connection_error(WiFi_Status_t status_code)`: connection errors are also described in the `WiFi_Status_t` definition. Connection errors are usually errors followng connection with an AP and include disassociation (removal from the network) and deauthentication events. The user must therefore implement this function to monitor the connection with the AP. Other errors like Scan failure and Join failure are also reported through this callback.
- `void ind_wifi_connected(void)`: when the Wi-Fi module is connected to the desired AP, this callback is generated. This callback is specifically generated when the module obtains the IP address and the connection with the AP is running. After this callback, the user state machine should be changed in order to commence the following set of application activities.
- `void ind_wifi_ap_ready(void)`: this callback is meant to be used when configuring the module as a mini access point (miniAP). Whenever the miniAP is up and running, this callback is sent to the user.
- `void ind_wifi_ap_client_joined(uint8_t * client_mac_address)`: in miniAP mode, whenever a client joins the miniAP this function is called with the MAC address of the client.
- `void ind_wifi_ap_client_left(uint8_t * client_mac_address)`: in miniAP mode, whenever a client leaves the miniAP, this function is called with the MAC address of the client.
- `void ind_wifi_wifi_ready(void)`: this callback can be used to determine when the Wi-Fi module is ready and initialized to be used for further configuration. The user must wait for this callback before proceeding with further configuration of the module in station or miniAP mode. Monitoring this callback is therefore of utmost importance.
- `void ind_wifi_packet_lost(WiFi_Status_t status_code`: this signals the user when data packets are lost in transmission.
- `void ind_wifi_gpio_changed(void)`: this callback can be used to monitor whether a GPIO line has been changed.
- `ind_wifi_socket_data_received(int8_t server_id, int8_t socket_id, uint8_t * data_ptr, uint32_t message_size, uint32_t chunk_size, WiFi_Socket_t socket_type)`: this callback is used by the user to check for data reception both for server and client sockets. The data is not explicitly read by the user, but uses this callback to capture any incoming data from server/client sockets. The data is provided as a pointer to a memory area and the user is advised to immediately copy the data to the user memory which should ideally be declared in the user's own program. The memory area pointed to by the data pointer is volatile is flushed immediately following callback termination, so there is no guarantee that the data will remain. In `socket_id`, the callback also provides the information about the socket ID number on which this data is available.

  In `server_id`, the callback provides the information about the server id number on which data is available, if this is a server socket, otherwise this value is -1.

  Apart from the data pointer, the callback also passes chunk size and whole data packet size information. Hence, for each callback, the user can calculate how many remaining callbacks there are and the exact quantity of data bytes that are yet to arrive. For example, the total data the user might receive could be 1100 bytes; in this case, the user receives three consecutive callbacks like:

  – First callback: chunk_size: 512B, message_size: 1100B
  – Second callback: chunk_size: 512B, message_size: 1100B
  – Third callback: chunk_size: 76B, message_size: 1100B

  The `socket_type` parameter indicates the type of socket (web or network(TCP/UDP)).
- `void ind_wifi_socket_client_remote_server_closed(uint8_t * socketID, WiFi_Socket_t socket_type)`: this callback is used by the module firmware to inform the user when a remote server

socket which the client was connected to is closed. The user can then proceed to reopen the socket or ensure that this socket is no longer used.

- `void ind_wifi_socket_server_data_lost(void)`: any socket server data lost in transmission is signaled to the user through this callback.
- `void ind_wifi_resuming(void)`: whenever the Wi-Fi module enters sleep/standby mode and subsequently resumes or wakes up, this callback informs the user that the module has resumed normal operation.
- `void ind_wifi_socket_server_client_joined(void)`: whenever a new client connects to a server socket on the module, the user receives this callback to indicate that a new client has joined. This is helpful in keeping track of how many clients are active.
- `void ind_wifi_socket_server_client_left(void)`: this callback signals the user when a client leaves or disconnects from a server socket.
- `void ind_wifi_ http_data_available(uint8_t * data_ptr, ,uint32_t message_size)`: this callback indicates when data from a http-get is available to the user. The user needs to copy the data to a local buffer before exiting this callback. There should not be any intensive buffer operations in these kinds of callback.
- `void ind_wifi_file_data_available(uint8_t * data_ptr)`: this callback signals whenever data from a file operation (`wifi_file_list()`/`wifi_file_print_content()`) is available to the user. The user needs to copy the data to his own local buffer before exiting this callback.
- `void ind_wifi_mqtt_data_received(uint8_t client_id, uint8_t *topic,uint32_t chunk_size,uint32_t message_size, uint32_t total_message_size, uint8_t *data_ptr)`: receives data from MQTT broker.
- `void ind_wifi_mqtt_closed(uint8_t client_id)`: to be implemented to catch an MQTT close action by MQTT broker.
- `void ind_wifi_inputssi_callback(void)`: raised in case of WIND:56 (**Input to Remote**). This function should prompt the user to set the input SSI text field by calling the function `wifi_fill_input_buffer(uint32_t DataLength, char * Data)`.
- `void ind_wifi_output_from_remote_callback(uint8_t *data_ptr)`: raised when data are sent from a remote SSI interface (particularly, when WIND:57 arrives).
- `void ind_wifi_socket_list_data_available(uint8_t * data_ptr)`: returns the socket list for any calls to list open, bound or web sockets.

While users can manipulate the callbacks to execute meaningful steps in their applications, you should not perform intensive operations which hold the callback function for too long. Any necessary processing should be kept brief to allow exiting the callbacks quickly.

3.6.2 Wi-Fi User State

The sample user applications include the following user states:

- `wifi_state_reset`
- `wifi_state_ready`
- `wifi_state_idle`
- `wifi_state_connected`
- `wifi_state_connecting`
- `wifi_state_disconnected`
- `wifi_state_activity`
- `wifi_state_error`
- `wifi_undefine_state`

These are only indicative and the user is free to write other states as needed, even if most of the foreseeable states would be similar to those listed above.

3.7 Wi-Fi application implementation

In the following sections, certain aspects of writing Wi-Fi applications using the X-CUBE-WIFI1 firmware is detailed, like how to initialize the Wi-Fi module and what the status codes to expect from the APIs are.

3.7.1 Wi-Fi module/interface selection

The user has to select, during compiling time, the Wi-Fi module which interface is intended to be used.

Select the appropriate configuration from the project settings.

Note: All project IDEs (IAR, Keil and SW4STM32) have different methods for configuration management (refer to Figure 16. STM32L4xx_Nucleo_SPWF01 IAR workspace for an IAR example).

The supported configurations are:

- Nucleoxx_SPWF01
- Nucleoxx_SPWF04_UART
- Nucleoxx_SPWF04_SPI

The configurations use different macros in the project settings.

Figure 16. STM32L4xx_Nucleo_SPWF01 IAR workspace

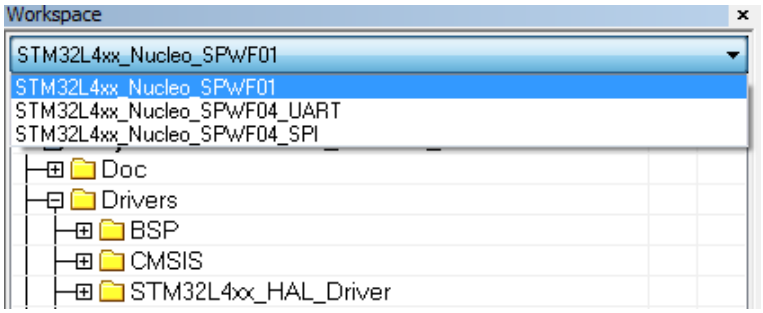


Table 2. Macros for supported configurations

Macro name	Description
Nucleoxx_SPWF01:	Uses the UART interface for SPWF01 (the default and only interface).
Nucleoxx_SPWF04_UART :	SPWF04SA module is used in the expansion board (X-NUCLEO-IDW01M1) with UART interface.
Nucleoxx_SPWF04_SPI:	SPWF04SA module is used in the expansion board (X-NUCLEO-IDW04A1) with SPI interface.

These macros are defined/non-defined for each platform supported (e.g. for L0 and F1 platforms the SPWF04SA module is not defined as the X-NUCLEO-IDW04A1 is not supported for these two platforms).

3.7.2 Debug option

The source code debug prints on the serial console can be enabled or disabled using the DEBUG_PRINT macro which is defined in the file wifi_conf.h in the Inc folder of each application.

3.7.3 Wi-Fi Status Codes

The Wi-Fi function return status signals are mostly handled using the `Wifi_Status_t` structure. This is the standard return type for most of the user API functions and the user should be aware that both errors and warnings status types are returned as `Wifi_Status_t` type.

The error codes returned as `Wifi_Status_t` type are:

- WiFi_TIME_OUT_ERROR
- WiFi_NOT_SUPPORTED
- WiFi_NOT_READY
- WiFi_SSID_ERROR
- WiFi_AT_CMD_BUSY
- WiFi_CONFIG_ERROR
- WiFi_STA_MODE_ERROR
- WiFi_AP_MODE_ERROR
- WiFi_AT_CMD_RESP_ERROR
- WiFi_AT_FILE_LENGTH_ERROR
- WiFi_HAL_UART_ERROR
- WiFi_IN_LOW_POWER_ERROR
- WiFi_HW_FAILURE_ERROR
- WiFi_STACK_OVERFLOW_ERROR
- WiFi_HARD_FAULT_ERROR
- WiFi_MALLOC_FAILED_ERROR
- WiFi_INIT_ERROR
- WiFi_START_FAILED_ERROR
- WiFi_EXCEPTION_ERROR
- WiFi_UNHANDLED_IND_ERROR

The warning codes returned as `Wifi_Status_t` type are:

- WiFi_SIGNAL_LOW_WARNING
- WiFi_HEAP_TOO_SMALL_WARNING
- WiFi_POWER_SAVE_WARNING

The connection error codes returned as `Wifi_Status_t` type are:

- WiFi_DE_AUTH
- WiFi_DISASSOCIATION
- WiFi_JOIN_FAILED
- WiFi_SCAN_BLEWUP
- WiFi_SCAN_FAILED

The packet lost error codes returned as `Wifi_Status_t` type are:

- WiFi_UNHANDLED_IND_ERROR
- WiFi_RX_MGMT
- WiFi_RX_DATA
- WiFi_RX_UNK

4 System setup guide

4.1 Hardware description

This section describes the hardware components required for developing an application using the Wi-Fi expansion board for STM32 Nucleo.

4.1.1 STM32 Nucleo platform

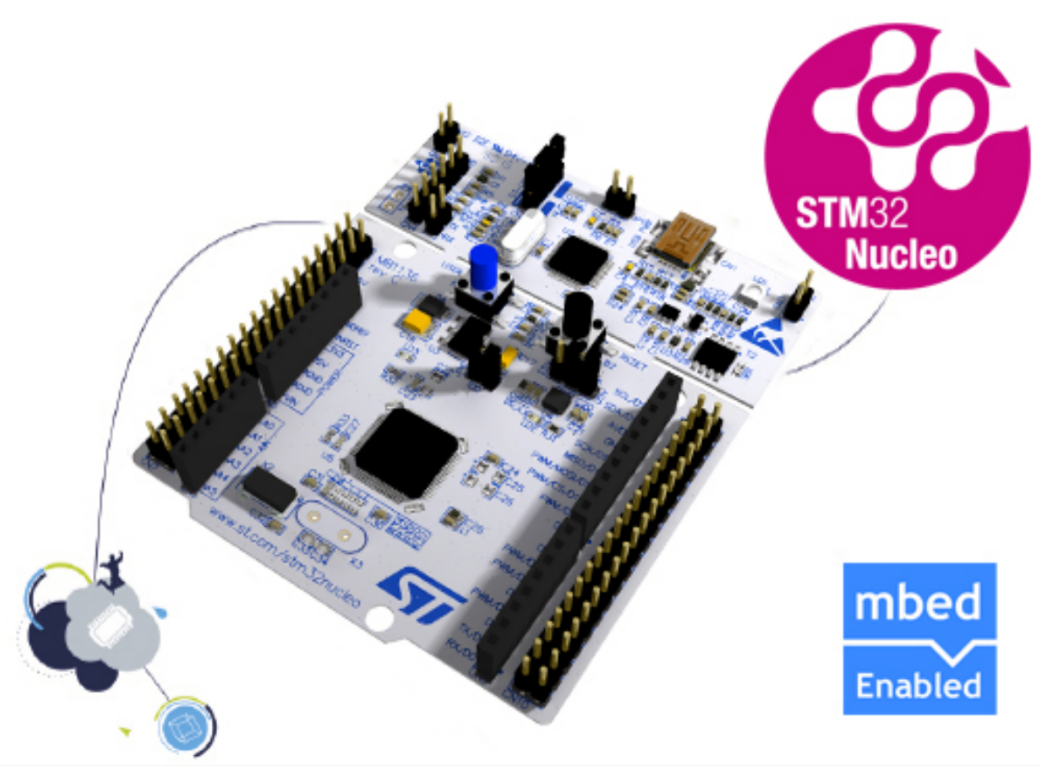
STM32 Nucleo development boards provide an affordable and flexible way for users to test solutions and build prototypes with any STM32 microcontroller line.

The Arduino™ connectivity support and ST morpho connectors make it easy to expand the functionality of the STM32 Nucleo open development platform with a wide range of specialized expansion boards to choose from.

The STM32 Nucleo board does not require separate probes as it integrates the ST-LINK/V2-1 debugger/ programmer.

The STM32 Nucleo board comes with the comprehensive STM32 software HAL library together with various packaged software examples.

Figure 17. STM32 Nucleo board



Information regarding the STM32 Nucleo board is available at www.st.com/stm32nucleo

4.1.2 X-NUCLEO-IDW01M1 expansion board

The X-NUCLEO-IDW01M1 is a Wi-Fi evaluation board based on the SPWF01SA module, which expands the STM32 Nucleo boards.

The CE, IC and FCC certified SPWF01SA module has an embedded STM32 MCU, a low-power Wi-Fi b/g/n SoC with integrated power amplifier and power management and an SMD antenna.

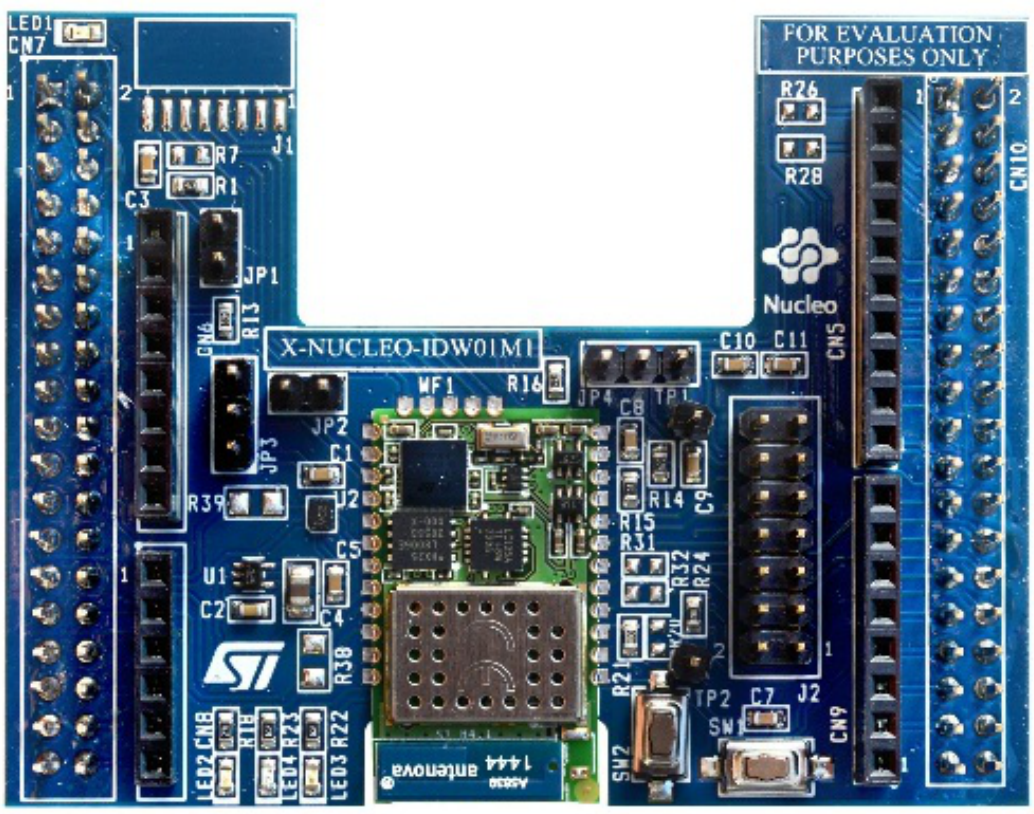
The SPWF01SA module is also equipped with 1 MByte of external FLASH for firmware update over-the-air (FOTA).

The firmware features a complete software IP stack to open up to 8 TCP/UDP sockets, as well as dynamic web pages with SSI to interact with the module and a REST API (get & post) for conveniently transferring files to/from servers in the cloud. The module can simultaneously behave as a socket server and socket client.

The firmware supports secure sockets with TLS/SSL encryption, ensuring secure end-to-end communications with the cloud, with or without authentication. The module operates as a client STA, IBSS, or miniAP (with up to 5 client STAs).

The X-NUCLEO-IDW01M1 interfaces with the MCU on the STM32 Nucleo board via the UART serial port; the user can easily access the stack functions using the AT command. X-NUCLEO-IDW01M1 is compatible with both the ST morpho and Arduino UNO R3 connector layout.

Figure 18. X-NUCLEO-IDW01M1 Wi-Fi expansion board



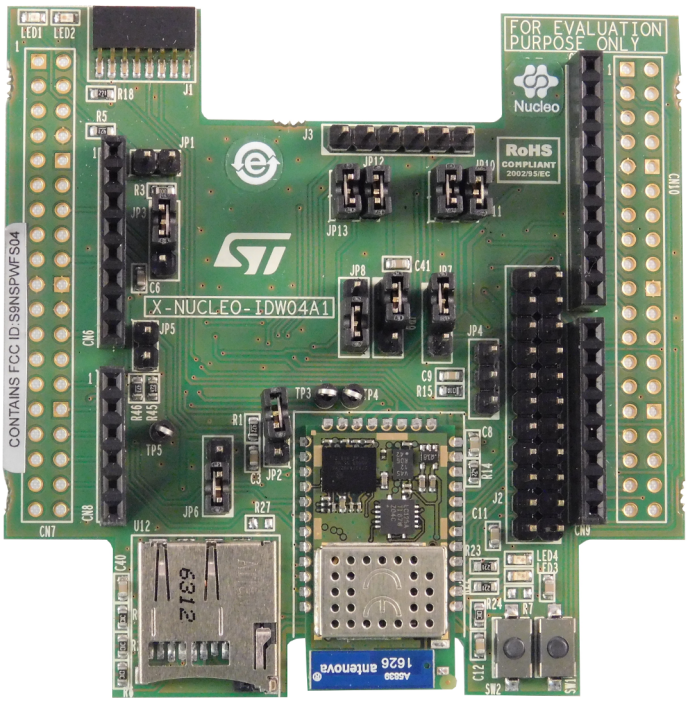
Information regarding the expansion board is available on www.st.com at http://www.st.com/x-nucleo.

4.1.3 X-NUCLEO-IDW04A1 expansion board

The X-NUCLEO-IDW04A1 is a Wi-Fi evaluation board based on the SPWF04SA module, which expands the STM32 Nucleo boards.

The SPWF04SA module has an embedded STM32 MCU, a low-power Wi-Fi b/g/n SoC with integrated power amplifier and power management and an SMD antenna.

Figure 19. **X-NUCLEO-IDW04A1 expansion board**



Information regarding the expansion board is available on www.st.com at http://www.st.com/x-nucleo.

4.2 Software description

The following software components are required in order to setup a suitable development environment for creating applications for the STM32 Nucleo equipped with the X-NUCLEO-IDW01M1 or X-NUCLEO-IDW04A1 expansion board:

- X-CUBE-WIFI1 expansion for STM32Cube dedicated to X-NUCLEO-IDW01M1 or X-NUCLEO-IDW04A1 application development. The X-CUBE-WIFI1 firmware and related documentation is available on www.st.com.
- Development tool-chain and Compiler expansion for STM32Cube supporting the three following environments:
  - IAR Embedded Workbench for ARM® (EWARM) toolchain + ST-LINK
  - RealView Microcontroller Development Kit (MDK-ARM) toolchain + ST-LINK
  - System Workbench for STM32 (SW4STM32) toolchain +ST-LINK

4.3 Hardware and software setup

This section describes the hardware and software setup procedures. It also describes the system setup needed for the above.

4.3.1 Hardware setup

The following hardware components are needed:

- One STM32 Nucleo development platform (order code: NUCLEO-F103RB, NUCLEO-F401RE, NUCLEO-L476RG or NUCLEO-L053R8)

- One Wi-Fi expansion board (order code: X-NUCLEO-IDW01M1 or X-NUCLEO-IDW04A1)
- One USB type A to Mini-B USB cable to connect the STM32 Nucleo to the PC
- Wi-Fi Router or an access point (AP) which is connected to the Internet

4.3.2 Software setup

This section lists the minimum requirements for the developer to setup the SDK, run the sample application(s) on the STM32 Nucleo and do some testing.

4.3.2.1 *Development Tool-chains and Compilers*

Please select one of the Integrated Development Environments supported by the STM32Cube expansion software.

Please read the system requirements and setup information provided by the selected IDE provider.

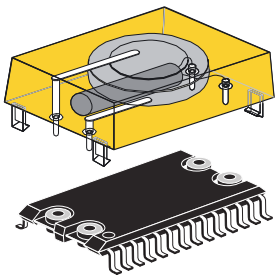
4.3.2.2 *Tera Term*

Tera Term or some other serial communication terminal emulator program is needed on the PC side for visualizing Wi-Fi notifications and messages transmitted by the sample application running on NUCLEO-F103RB / NUCLEO-F401RE / NUCLEO-L476RG / NUCLEO-L053R8 plus X-NUCLEO-IDW01M1 or X-NUCLEO-IDW04A1 expansion boards.

4.3.2.3 *Underwriters Laboratories documents for ST real-time clocks*

The Underwriters Laboratories certificates for ST's SNAPHAT SOIC and all other RTC packages are available at www.st.com.

Figure 20. SNAPHAT SOIC package



4.4 System setup instructions

This section describes how to set up the different components before writing and executing applications on the STM32 Nucleo board with the Wi-Fi expansion board.

4.4.1 STM32 Nucleo and Wi-Fi expansion boards setup

The STM32 Nucleo board integrates the ST-LINK/V2-1 debugger/programmer. The developer can download the relevant version of the ST-LINK/V2-1 USB driver at STSW-LINK008 or STSW-LINK009 (according to the Microsoft Windows OS).

The Wi-Fi expansion board X-NUCLEO-IDW01M1 or X-NUCLEO-IDW04A1 can be easily connected to the STM32 Nucleo board through the Arduino UNO R3 extension connector and the ST morpho connectors.

Note: Check to see whether resistor R21 is present on the X-NUCLEO-IDW01M1 or X-NUCLEO-IDW04A1 board. If it is, please remove it because it may interfere with JTAG debug.

4.4.2 Tera Term setup

Tera Term terminal emulation program can be used to view the streaming serial data transmitted by the sample application program running on the STM32 Nucleo board equipped with the X-NUCLEO-IDW01M1 or X-NUCLEO-IDW04A1 expansion board.

The screenshots below show the Tera Term serial port and terminal setup configurations for viewing the serial UART data transmitted by the sample application.

Figure 21. Tera Term serial port setup configuration

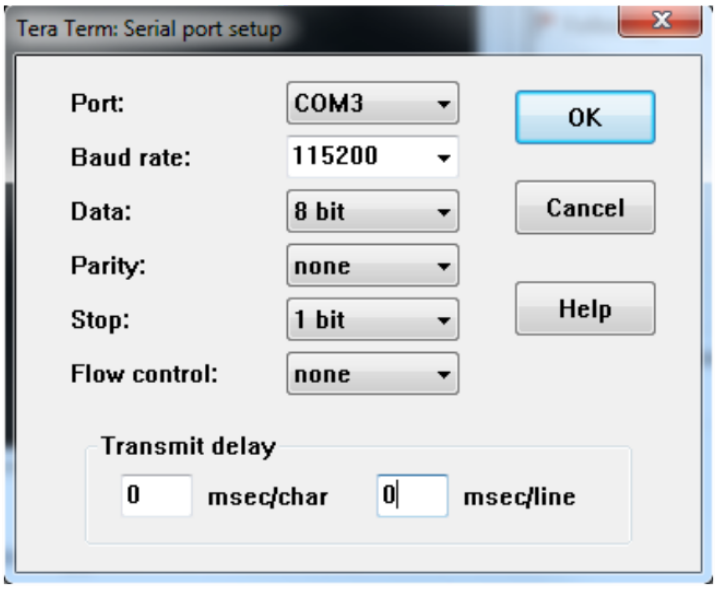
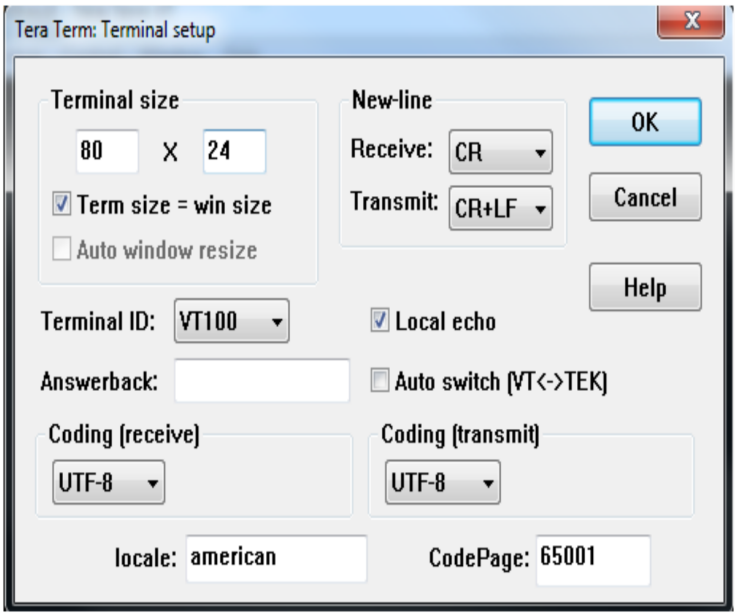


Figure 22. Tera Term terminal setup configuration



5 References

1. X-NUCLEO-IDW01M1 and X-NUCLEO-IDW04A1 documents available on www.st.com
2. UM1695: Command set reference guide for "AT full stack" for SPWF01Sx series of Wi-Fi modules, available at www.st.com
3. Wi-Fi Training- Hands On

Revision history

Table 3. **Document revision history**

Date	Version	Changes
06-Nov-2015	1	Initial release.
24-Feb-2016	2	Throughout document: - added NUCLEO-L476RG compatibility information - text and formatting changes Updated Section 3.4: "APIs" Updated Section 3.6.1: "Wi-Fi user event callbacks"
10-Mar-2017	3	Updated: Figure 1. X-CUBE-WIFI1 software architecture, Section 2.4 APIs, Figure 3. Detailed software architecture, Section 2.5.3 HTTP-Request, Section 2.6.1 Wi-Fi user event callbacks and Section 3.3.1 Hardware setup Added: Section 2.7.1 Wi-Fi module/interface selection, Section 2.7.2 Debug option and Section 3.1.3 X-NUCLEO-IDW04A1 expansion board Throughout document: added X-NUCLEO-IDW04A1 expansion board and SPWF04SA module compatibility information.
31-Jan-2018	4	Updated Section 3.4 APIs, Section 3.6.1 Wi-Fi user event callbacks, Section 3.7.1 Wi-Fi module/interface selection and Section 5 References.

Contents

1 **Acronyms and abbreviations** 2
2 **What is STM32Cube?** 3
2.1 STM32Cube architecture 3
3 **X-CUBE-WIFI1 software expansion for STM32Cube** 5
3.1 Overview 5
3.2 Architecture 5
3.3 Folder structure 6
3.4 APIs 7
3.5 Sample applications 11
3.5.1 Client socket in STA mode 11
3.5.2 Socket server in MiniAP mode 14
3.5.3 HTTP-Request 16
3.5.4 Wi-Fi Virtual Com application 17
3.6 Sample application user states and state machine 18
3.6.1 Wi-Fi user event callbacks 19
3.6.2 Wi-Fi User State 21
3.7 Wi-Fi application implementation 21
3.7.1 Wi-Fi module/interface selection 22
3.7.2 Debug option 22
3.7.3 Wi-Fi Status Codes 22
4 **System setup guide** 24
4.1 Hardware description 24
4.1.1 STM32 Nucleo platform 24
4.1.2 X-NUCLEO-IDW01M1 expansion board 24
4.1.3 X-NUCLEO-IDW04A1 expansion board 25
4.2 Software description 26
4.3 Hardware and software setup 26
4.3.1 Hardware setup 26
4.3.2 Software setup 27
4.4 System setup instructions 27

4.4.1 STM32 Nucleo and Wi-Fi expansion boards setup 27
4.4.2 Tera Term setup 27
5 References 29
Revision history 30

List of tables

Table 1. Acronyms and abbreviations . . . 2
Table 2. Macros for supported configurations . . . 22
Table 3. Document revision history . . . 30

List of figures

Figure 1. Firmware architecture . . . 3
Figure 2. X-CUBE-WIFI1 software architecture . . . 6
Figure 3. X-CUBE-WIFI1 package folder structure . . . 7
Figure 4. Detailed software architecture . . . 8
Figure 5. Scan result and connection to AP . . . 12
Figure 6. Client socket read/write . . . 12
Figure 7. TCP socket server program . . . 13
Figure 8. Remote server socket closed callback . . . 14
Figure 9. Captiveportal.net screen shot . . . 14
Figure 10. Server socket Tera Term screen shot . . . 15
Figure 11. SocketTest3 program . . . 16
Figure 12. HTTP GET/POST operation . . . 17
Figure 13. Tera Term setup for VCOM . . . 18
Figure 14. VCOM application screen shot . . . 18
Figure 15. Application state machine . . . 19
Figure 16. STM32L4xx_Nucleo_SPWF01 IAR workspace . . . 22
Figure 17. STM32 Nucleo board . . . 24
Figure 18. X-NUCLEO-IDW01M1 Wi-Fi expansion board . . . 25
Figure 19. X-NUCLEO-IDW04A1 expansion board . . . 26
Figure 20. SNAPHAT SOIC package . . . 27
Figure 21. Tera Term serial port setup configuration . . . 28
Figure 22. Tera Term terminal setup configuration . . . 28