Developer's Guide

Ayla Device Wi-Fi for Embedded Systems

Version: 2.0

Date Released: April 29, 2017

Document Number: AY006DAW6-2

Ayla Networks

Copyright Statement

Trademarks Statement

Referenced Documents

Ayla Networks does not supply all documents that are referenced in this document with the equipment. Ayla Networks reserves the right to decide which documents are supplied with products and services.

Contact Information

Ayla Networks TECHNICAL SUPPORT and SALES

Contact Technical Support: https://support.aylanetworks.com
or via email at support@aylanetworks.com

Contact Sales: https://www.aylanetworks.com/company/contact-us

Ayla Networks REGIONAL OFFICES

Chicago
10 N. Martingale Road, Suite 400
Schaumburg, IL 601073

HEADQUARTERS
Silicon Valley
4250 Burton Drive, Suite 100
Santa Clara, CA 95054
Phone: +1 408 830 9844
Fax: +1 408 716 2621

Boston
275 Grove Street, Suite 2-400
Newton, MA 02466

Table of Contents

1 Introduction

The Ayla Device Wi-Fi (ADW) is a library package that maintains the Wi-Fi configuration and keeps the device connected. ADW supports Wi-Fi setup either with a web page or a secure Wi-Fi setup mobile application.

1.1 About this Document

This document provides information on the design and implementation of the Ayla Device Wi-Fi (ADW) software for use in embedded microprocessor systems. You may use this document as a specification for the features, algorithms, software interfaces, and integration considerations when using ADW.

NOTE This document does not describe the use of ADW on any particular software development kit (SDK). On support.aylanetworks.com, we provide an SDK-specific document for each Ayla Device Agent (ADA) package, describing the particular version of the ADA you are planning to use.

1.2 Intended Audience

This document is for software developers who wish to develop an embedded system with connectivity to Wi-Fi and the Ayla Device Service (ADS) and mobile applications.

1.3 Related Documentation

Refer to the following documents available on support.aylanetworks.com for information on Ayla products and services related to Ayla Device Wi-Fi.

- *Ayla Product Development Overview* (AY006GOV1) for an overview of the steps to bring an IoT product to market.
- *Ayla Device Agent for Embedded Agents Developer's Guide* (AY006DAR6) for information on the Ayla Embedded Agent and its use with Ayla Device Wi-Fi.
- *Ayla Command Line Interface (CLI) User Manual* (AY006UCL3)

1.4 API Conventions

For APIs described in this document, functions returning `0` on success may return `-1` or some other non-zero value on failure:

- If the type returned is `enum ada_err`, a successful return is zero and the error return is an error number less than zero.
- If the type returned is `enum wifi_error`, a successful return is zero, and the error return is a non-zero value.

1.5 Document Conventions

This document follows these Ayla documentation conventions:

- Text that you type (such as commands) and the contents of files are shown as:

```
cd wmsdk_bundle-3.1.16.1
tar xfz ada-wmsdk-src-1.0.tgz
```

- Function prototypes and other code fragments are shown in `Courier New`, a fixed-width font.
- Function names, variables, structure names and members are mentioned in-line in `Courier New`, for example: `enum ada_err`
- Ancillary information that is important to emphasize is shown as:

NOTE	The commands provided in the example assume your evaluation board is xxxx and your chip is yyyy. If otherwise, make the appropriate substitutions

- Information on system failures or hazards that could damage a product (i.e. data loss)

Make sure that the appropriate data buffering is accounted for in deployed devices, especially where the loss of data is critical to the core functionality or the services provided by the systems.

1.6 Glossary

LAN mode	This setting allows the connection and control from a mobile device that is on the same IP subnet as the device.
REST	This stands for Representative State Transfer and is a method of web-based APIs.
Wi-Fi Module	A hardware component that has an MCU containing the Ayla Embedded Agent and Wi-Fi component used to allow connectivity to the Ayla Cloud Services.

1.7 Abbreviations and Acronyms

The following acronyms are used in this document.

ABI	Application Binary Interface
ADA	Ayla Device Agent. This is a legacy term for Ayla Embedded Agent. The ADA acronym is still used in CLI commands and sources, and descriptions of either of these may refer to the Ayla Embedded Agent as the “device agent.”

ADS	Ayla Device Service (the cloud)
ADW	Ayla Device Wi-Fi
AP	Access Point
LAN	Local Area Network
MCU	Microcontroller unit
mDNS	Multicast Domain Name System
NTP	Network Time Protocol
REST	Representative State Transfer
RISC	Reduced Instruction Set Computing
RSSI	Received Signal Strength Indicator
SSID	Service Set Identifier
STA	Station Computer
WEP	Wired Equivalent Privacy
WMI	Wi-Fi Module Interface
WPS	Wi-Fi Protected Setup

1.8 Revision History

Table 1 provides a summary of updates to the content of this document.

Table 1: Revision History of Content Updates and Changes

Revision #	Date (yyyy-mm-dd)	Change
1.0	2015-08-31	Initial version.
2.0	2016-11-21	Updates for Ayla Embedded Agent 1.2 and ADW 1.2 Added new APIs: • `adw_wifi_is_enabled()` • `adw_wifi_force_ap_mode()` • `adw_wifi_unforce_ap_mode()`

2 Device Wi-Fi Features

Ayla Device Wi-Fi (ADW) maintains the Wi-Fi configuration and connection between the device and the cloud. The ADW software is an optional package with the Ayla Embedded Agent, but the ADW requires the Ayla Embedded Agent. The product releases covered in this document are the ADW 1.2, which is currently integrated with Ayla Embedded Agent 1.2.

This section describes limitations in the ADW and interfaces to the Ayla Embedded Agent and the host application.

2.1 Limitations in the ADW 1.2 Release

Wi-Fi Protected Setup (WPS) is not supported or planned for future releases.

In some Software Development Kits (SDKs), the AP and STA mode can be used simultaneously only when the AP joined by the STA is not in the same network used by AP mode, 192.168.0.1. If the network being joined conflicts with the AP network, the AP shuts down and the STA connection is maintained. This provides a good Wi-Fi setup, but not a good user experience. You can change the compiled-in AP mode IP address to something that is less likely to conflict. The mobile applications use the gateway address obtained using DHCP. Refer to Step 4 in Section 6.

2.2 Interfaces to the Ayla Embedded Agent

ADW relies on the following interfaces that may be provided, but are required by the Ayla Embedded Agent:

- C library functions
- Console logging
- Configuration system
- Encryption services
- HTTP client
- HTTP server
- OS functions: timers, threads, and callbacks
- Networking interfaces

NOTE We assume you are using the current release of the Ayla Embedded Agent.

2.3 Interfaces to the Host Application

The ADW relies on functionality provided by the product integrator. This primarily consists of configuration and an event handler responsible for the indications of connectivity, such as LEDs or other displays. The functionality of the product integrator may also be used to start the Ayla Embedded Agent and launch other services when connectivity is established and to stop services when connectivity is lost.

3 Major Components of Ayla Device Wi-Fi

This section provides pertinent information on the main components of Ayla Device Wi-Fi (ADW). Each subsection is an ADW component.

3.1 Wi-Fi State Machine

The ADW Wi-Fi state machine manages the Wi-Fi connection, AP mode, and scanning. The main objective of this component is to select a profile and connect accordingly.

3.2 Wi-Fi Module Interface

The Wi-Fi Module Interface (WMI) is a set of internal APIs that are a part of the platform-specific interface to the software development kit (SDK).

3.3 Wi-Fi HTTP Pages

The ADW HTTP server pages are used during the Wi-Fi setup to obtain scan results and initiate the Wi-Fi connection.

When a mobile device first connects to ADW in AP mode, the server can re-direct all requests to the ADW page for Wi-Fi setup using the browser or through a mobile app.

Wi-Fi setup using a browser is not secure because the Wi-Fi passphrase is revealed when sent. Whereas, Wi-Fi setup through the mobile app always uses a cryptographic key exchange protocol on top of HTTP.

3.4 Configuration

Configuration of the ADW relies on the features used by the Ayla Embedded Agent running on top of the SDK. The Wi-Fi module has three configuration sets in use:

- running configuration
- startup configuration
- factory configuration

The *running* configuration is stored in RAM, and if the configuration is not saved, it is erased on each reset.

The *startup* and *factory* configurations are stored in flash and the concatenation of them is used to load the running configuration during initialization. If the same name is used for both the factory and startup configurations, the last one saved takes precedence.

There is a *setup mode* that is usually enabled during initial configuration in the OEM factory settings. You may only enter certain CLI commands in setup mode. When a `save` is done in setup mode, the running configuration is written to the factory configuration. When the configuration is saved outside of setup mode, the running configuration is written to the startup configuration.

When a factory reset is performed, either by the `reset factory` CLI command or through an API, the startup configuration is erased, causing the device to revert to the factory configuration that existed when setup mode was disabled.

The OEM should disable setup mode before performing the Wi-Fi setup and connecting to any networks. Otherwise, the new Wi-Fi profile is saved to the factory configuration and is not erased by a factory reset.

3.5 Logging

ADW log messages use the module name `wifi` under the ADA logging subsystem.

NOTE Refer to the *Ayla Device Agent for Embedded Agents Developer's Guide* for information on the ADA logging subsystem.

3.6 CLI Interfaces

The Ayla Embedded Agent provides a small number of CLIs that may be used by the ADW. These interfaces allow some debugging and manufacturing setup of the device. See also Section 7.2.4.

NOTE Refer to the *Ayla Command Line Interface (CLI) User Manual* for more information.

4 Multithreading Considerations

Ayla Device Wi-Fi (ADW) code runs in the Ayla Embedded Agent network thread and is asynchronous for the most part. ADW may also launch other threads to perform synchronous operations when necessary. Events and calls from other threads are safe and either performed to completion or queued.

5 Memory Requirements

Memory requirements for the Ayla Device Wi-Fi (ADW) depend on the features and the chosen Software Development Kit (SDK). The estimates in Table 2 below are based on the memory requirements of a typical small RISC architecture.

Table 2 provides estimated memory requirements for the ADW code and data.

Table 2: Revision History of Content Updates and Changes

Subsystem	Flash Size (bytes)	RAM Size (bytes)
ADW Code	16660	16660
ADW initialized data	408	408
ADW BSS data	0	5160
ADW read-only	7600	7600
ADW web pages	6808	6808
TOTAL	31476	36636

NOTE You may not need to include code, read-only data, and web pages in both RAM and Flash on some platforms.

6 Integrating Ayla Device Wi-Fi

This section provides the steps required to integrate and use Ayla Device Wi-Fi (ADW) with the Ayla Embedded Agent. As stated earlier, these are delivered together in the same package.

NOTE You may use the code in `demo/ayla_demo/demo_wifi.c` as an example for handling initialization and events.

1. In `demo_wifi.c`, call `demo_wifi_init()`before calling `ada_init()`.
2. Modify `OEM_AP_SSID_PREFIX` in `conf.h` as follows to use as your desired prefix for the AP-Mode SSID.

   The format of the AP-Mode SSID is `Ayla-<mac-address>`, for example:

   `Ayla-123456789012`

   Change the prefix from “Ayla” to something more appropriate for your product.

   If you want to change the pattern to something else, change the code in `demo_wifi_init()`.
3. Consider turning on `conditional AP-mode` by doing `adw_wifi_ap_conditional_set(1)`. This disables the AP-Mode when there is an enabled profile, which is a good security measure, especially if the device loses connectivity and is using AP-Mode registration.
4. Change the IP parameters if your application prefers another set. On some SDKs, you may be advised to use a network that does not conflict with the station-mode network. The IP parameters for AP-Mode are provided in the header file in `libadw/include/adw/wifi_conf.h`. The definitions are as follows:
   - `ADW_WIFI_AP_IP` - defaults to 192.168.0.1
   - `ADW_WIFI_AP_NETMASK` - defaults to 255.255.255.0
   - `ADW_WIFI_AP_DHCP_START` defaults to 192.168.0.21
   - `ADW_WIFI_AP_DHCP_END` - defaults to 192.168.0.49
   - `ADW_WIFI_AP_DHCP_LEASE` - Lease time, defaults to 300 seconds.
5. During manufacturing, enter the CLI command, `wifi enable` to turn on the Wi-Fi. You may prefer to use the equivalent C APIs instead, which is `adw_wifi_enable()`.
6. When all of the desired factory settings have been made, disable the setup mode and save the settings to the factory configuration using either the CLI `setup_mode disable`, or the C API, `ada_conf_setup_mode(0)`.

NOTE This is an important step so that a factory reset can be performed later by the device or from the cloud.

7 Application Programming Interfaces (APIs)

This section describes the APIs that the Ayla Device Wi-Fi (ADW) supplies to the Ayla Embedded Agent and the host application.

7.1 Interfaces Supplied to the Ayla Embedded Agent

The Ayla Embedded Agent requires ADW or any alternative Wi-Fi subsystem to supply the interfaces described in this section. The descriptions in Table 3 focus on the behavior of the interface under ADW.

NOTE Refer to *Ayla Device Agent for Embedded Agents Developer's Guide* for the interface requirements of the Ayla Embedded Agent.

Table 3: Interfaces Supplied to the Ayla Embedded Agent

Function	Description
`adap_wifi_features_get()`	`enum ada_wifi_features adap_wifi_features_get(void)` • Returns a mask of features provided by this version of ADW. • Can depend on compile options and the version of ADW. • Returns `AWF_SIMUL_AP_STA`, indicating that simultaneous AP and Station mode can be used. • Does not set the WPS-related feature bits.
`adap_wifi_in_ap_mode()`	`int adap_wifi_in_ap_mode()` Returns a non-zero value if the AP interface is active, and zero if the interface is not active.
`adap_wifi_get_ssid()`	`int adap_wifi_get_ssid(void *buf, size_t len)` • Adds the SSID of the connected network to the supplied buffer. The `len` argument gives the total length of the buffer available, which must be at least 33 bytes long. • Returns a value that is the length of the SSID in bytes. The value is either `0` or a negative number if the station interface is not connected to a network.
`adap_wifi_stayup()`	`void adap_wifi_stayup(void)` Instructs the ADW that the interface is in use and connectivity should be maintained. The Ayla Embedded Agent uses this to inform the ADW that the internal web server is in use so that AP-mode remains active for some unspecified time longer.

Table 3: Interfaces Supplied to the Ayla Embedded Agent (Continued)

Function	Description
`adap_wifi_show_hist()`	`void adap_wifi_show_hist(int to_log)` Causes the last several Wi-Fi connection attempts to be logged. If the `to_log` argument is non-zero, the history entries are logged to the console as Wi-Fi information messages, and if the argument is `0`, the entries are sent to the console as CLI output using `printcli()`.

7.2 Interfaces Used by the Host Application

The interfaces in this section may also be used by the host application. The descriptions focus on the behavior of the interface under ADW.

7.2.1 Configuration APIs

Table 4 provides the interfaces that set the initial configuration options for ADW.

Table 4: Interfaces that Set the Initial Configuration Options for ADW

Function	Description
`adw_wifi_ap_conditional_set()`	`void adw_wifi_ap_conditional_set(int conditional)` Sets or clears the conditional flag for AP-mode. If conditional is non-zero, the AP mode is not started if there is an enabled network profile. This prevents the device from being accessible in AP-mode when the configured network is down. The default is `0`, meaning to start AP-mode whenever no profiles can be joined.
`adw_wifi_ap_ssid_set()`	`void adw_wifi_ap_ssid_set(const char *ssid)` • Sets the SSID for AP-mode • Sets the AP-mode security to none (open) • Enables the AP profile
`adw_wifi_ios_setup_app_set()`	`void adw_wifi_ios_setup_app_set(const char *name)` Provides the custom URL to which iPhones and iPads are redirected when joining the device in AP-mode. This should be the URL associated with the setup application.

Table 4: Interfaces that Set the Initial Configuration Options for ADW (Continued)

Function	Description
`adw_wifi_save_policy_set()`	`void adw_wifi_save_policy_set(int ap_connect, int serv_connect)` **NOTE:** Call this function before loading the configuration. During an attempt to connect, these settings determine when, if ever, a new Wi-Fi profile is persisted in the configuration in flash. During Wi-Fi setup for a new network, a profile is created and a connection is attempted, but the profile is not enabled and persisted until some success condition is met. When `ap_connect` is non-zero, the profile is enabled and persisted if the AP is connected and a DHCP address is received. When `serv_connnect` is non-zero, the profile is enabled and persisted when the cloud service is successfully contacted and the device is authenticated. This function is typically only necessary during testing. If this function is never called, `serv_connect` defaults to `1`, and `ap_connect` defaults to `0`.

7.2.2 Control APIs

Table 5 provides the interfaces that initialize ADW or allow control of its functions after initialization.

Table 5: Interfaces that Initialize ADW or Allow Control After Initialization

Function	Description
`adw_wifi_init()`	`void adw_wifi_init(void)` • Initializes the ADW and the Wi-Fi subsystem. • Must be called before `ada_init()`.
`adw_wifi_page_init()`	`void adw_wifi_page_init(int redirect)` **NOTE:** Call after `adw_wifi_init()` and before `ada_init()`. Initializes the web pages used by the Wi-Fi subsystem. If non-zero, the `redirect` parameter indicates that requests from mobile devices should be redirected to the Wi-Fi setup page. This parameter should normally be `1`.
`adw_wifi_enable()`	`void adw_wifi_enable(void)` Enables and starts the Wi-Fi manager.
`adw_wifi_disable()`	`void adw_wifi_disable(void)` Disables the Wi-Fi manager. If either the station or AP interfaces are active, they are disconnected.

Table 5: Interfaces that Initialize ADW or Allow Control After Initialization (Continued)

Function	Description
`adw_wifi_is_enabled()`	`int adw_wifi_is_enabled()` Returns a non-zero value if the Wi-Fi manager is enabled.
`adw_wifi_profile_erase()`	`int adw_wifi_profile_erase(unsigned int index` Erases the profile selected by the argument index. Valid indices are from `0` to `ADW_WIFI_PROF_AP - 1` for station profiles, and `ADW_WIFI_PROF_AP` for the AP profile. The return value is `0` on success and `-1` if an invalid profile index is given. If the station profile currently in-use is erased, the module disconnects from the network. If the AP profile is erased and the module is currently in AP mode, this function does not have an immediate effect. The module remains in AP mode until the module times out or Wi-Fi is disabled.
`adw_wifi_profile_sta_erase()`	`void adw_wifi_profile_sta_erase(void)` Erases all station profiles. If currently connected, the module disconnects from the network.
`adw_wifi_profile_ap_erase()`	`void adw_wifi_profile_ap_erase(void)` Erases the AP profile. If the module is currently in AP mode, this call does not stop this mode. The module remains in AP mode until the module times out or Wi-Fi is disabled.
`adw_wifi_force_ap_mode()`	`void adw_wifi_force_ap_mode(void)` Makes the AP interface active and the Station interface inactive, allowing Wi-Fi setup to complete using the AP interface. This remains in effect until reversed by: • `adw_wifi_unforce_ap_mode()` • a successful connection attempt • reboot of the device This state is not persisted in the configuration.
`adw_wifi_unforce_ap_mode()`	`void adw_wifi_unforce_ap_mode(void)` Cancels the behavior set up by `adw_wifi_force_ap_mode()`, allowing an enabled profile (that was previously configured) to join.

7.2.3 Status APIs

The host application may use the Status APIs described below to determine the status of the Wi-Fi connection.

- `adw_wifi_configured()`

  ```
  int adw_wifi_configured(void)
  ```

  Returns a non-zero if any network profiles are enabled.

- `adw_wifi_event_register()`

  ```
  void adw_wifi_event_register(void (*handler)(enum
  adw_wifi_event_id, void *),
                               void *arg)
  ```

  Registers a callback to the application function `handler()`. The callback is done when certain events occur, such as joining or leaving a network or starting or stopping AP mode. This allows the application to indicate the status using the LEDs or to start or stop network services. The event ID is any of those listed in Table 6.

NOTE You can only register a handler in this manner one time.

Table 6: The Event ID Used by the Handler

Event Enum Symbol	Description
`ADW_EVID_ENABLE`	Subsystem enabled.
`ADW_EVID_DISABLE`	Subsystem disabled.
`ADW_EVID_ASSOCIATING`	Starting to join a network.
`ADW_EVID_SETUP_START`	Starting to use WPS (not used yet) or similar setup method.
`ADW_EVID_SETUP_STOP`	Stopping WPS or similar setup activity.
`ADW_EVID_STA_DOWN`	Station interface is down.
`ADW_EVID_STA_UP`	Station interface is up.
`ADW_EVID_STA_DHCP_UP`	Station DHCP address has been acquired.
`ADW_EVID_AP_START`	AP interface started.
`ADW_EVID_AP_DOWN`	AP interface stopped.
`ADW_EVID_AP_UP`	AP interface is up.
`ADW_EVID_RESTART_FAILED`	Restarting for error recovery has failed. Rebooting is advised.

An example of how the events should be handled by the host application is in the demo program for the platform, which is provided with ADW. For example, for the `ADW_EVID_AP_UP` event, the application should start a web server and DNS server for handling Wi-Fi setup. When the `ADW_EVID_AP_DOWN` event occurs, the DNS server may be stopped. On some platforms, these operations may be performed internally by ADW. On other platforms, these operation need to be done by the host application. Consult the demo code for the requirements on your platform.

When the `ADW_STA_DHCP_UP` event is called, the ADA client should start, along with the web server, mDNS server, and possibly the NTP client, depending on the platform.

NOTE Not all events need to be handled; some may be ignored. The handler must ignore any events that are not recognized.

- `int adw_wifi_in_ap_mode()`

  ```
  int adw_wifi_in_ap_mode(void)
  ```

  Returns a non-zero if the AP interface is currently active.

7.2.4 Command Line Interface

This subsection provides the APIs for various command line interfaces (CLI). The host application may use these APIs to include in the host application as main commands or subcommands. The command name used for each one may be changed by the application to something different than specified, but the usage messages indicate the name commonly used, like `compiled in`.

NOTE Generally, these commands are only used during setup or development. After a product ships, it is undesirable to have an exposed CLI.

- `CLI function arguments`

  Most of these functions (like those described below) have prototypes similar to:

  ```
  void adw_wifi_cli(int argc, char **argv)
  ```

  An exception is the `adw_wifi_show()` function, which takes no argument. A wrapper for a `show wifi` command generally calls this API.

  Each command is invoked on the CLI as `wifi arg1 arg2 arg3`, with the arguments separated by spaces or some other application-specified separator. The command parser may be provided by the SDK or the host application. The host application may add ADW commands to its own command set.

NOTE Though not required, it is useful for the parser to accept quoted-string tokens to pass arguments containing spaces to the commands. Without that ability, the CLI is not able to specify an SSID that contains a space.

The first parameter, `argc`, provides a count of the number of arguments, including the command itself. `argc` should always be at least 1. The second parameter, `argv`, is a pointer to an array of `argc` string pointers, followed by a `NULL` pointer.

For example, the CLI for "`wifi profile 0 profile erase`" can be called as:

```
char *argv[6] = {"wifi", "profile", "0", "profile", "erase", NULL};
adw_wifi_cli(5, &argv);
```

- `adw_wifi_show()`

  `void adw_wifi_show(void)`

  Prints the Wi-Fi status on the serial debug console. Following is an example of the typical output.

```
Wi-Fi associated with SSID QA-ASUS
RSSI -51 antenna 0

STA MAC address 00:50:43:21:ee:57
IP       192.168.1.249
netmask  255.255.255.0
DNS      192.168.1.1

AP MAC address 00:50:43:21:ee:57
Net Down, Link Up

profiles:
Index              Network  Security
    0              QA-ASUS    WPA2_Personal
   AP    Ayla-00504321ee57             none

scan results:
            Network     Type Chan Signal             Security
          QA-LINKSYS       AP    1    -32  WPA2 Personal Mixed
            QA-DDWRT       AP    6    -33  WPA2 Personal Mixed
          QA-NETGEAR       AP    2    -34        WPA2 Personal
              QA-WEP       AP    6    -35                 WEP
          QA-WPA-AES       AP    6    -35                 WPA
         QA-WPA-TKIP       AP    6    -36                 WPA
AirPort Time Capsule       AP   11    -40  WPA2 Personal Mixed
           QA-TPLINK       AP   10    -43  WPA2 Personal Mixed
        QA-WPA2-TKIP       AP   11    -44        WPA2 Personal
             QA-NONE       AP   11    -44                None
         QA-WPA2-AES       AP   11    -44        WPA2 Personal
             QA-ASUS       AP    9    -45        WPA2 Personal
          QA-DDWRT02       AP   11    -46  WPA2 Personal Mixed
              AY-QA2       AP    6    -46  WPA2 Personal Mixed
   Ayla-cc52afc6d701       AP    3    -48                None
   Ayla-cc52afc6d809       AP    3    -48                None

connection history:
    2016-11-21T22:09:15Z anon ssid Q*S bssid d410 status (final) 0 none
      IP 192.168.1.249 mask 255.255.255.0 default route 192.168.1.1
      DNS 192.168.1.1, 0.0.0.0
```

Notice in the example above that the output starts with the mode and signal and then provides the MAC and IP parameters for the Station and AP interfaces.

NOTE The amount of information provided in the output may change in future versions or depending on the current settings and conditions.

Following is a description of the remaining output shown in the above example:

Profiles lists each stored profile, its SSID, security, and any special flags, such as `disabled` or `hidden`.

Scan results shows the results from the most recent scan, which includes the SSID, Type, channel, signal strength, and security. The signal strength is that provided by the SDK scaled to be a negative number comparable to the signal strength in dBm over the noise floor. These numbers can be compared to each other as relative strength, but are not absolute signal strength values.

Connection history shows the history of the last 3 to 5 connection attempts. Notice that only the first and last letters of the SSID and the last 2 bytes of the BSSID are shown.

- `adw_wifi_show_rssi()`

  ```
  void adw_wifi_show_rssi(int argc, char **argv)
  ```

  Shows the RSSI (signal strength). The output displays on the console after sampling a few times. The prompt may come back before the RSSI is logged.

- `adw_wifi_cli()`

  ```
  void adw_wifi_cli(int argc, char **argv)
  ```

  This is the main configuration interface for Wi-Fi.

NOTE Several of the Wi-Fi commands may not be used until a `wifi commit` command is issued. This command allows a `save` to be done before the Wi-Fi changes and changes to several of the parameters in the profile take effect.

Several parameters may be combined on the same line, but their usage can be confusing. The parameters are interpreted sequentially as if they were entered on separate wifi commands. Refer to the following example.

```
wifi profile 0 ssid mynetwork key mykey1234 security WPA2
wifi profile enable join
wifi profile 9 profile erase
```

The usage in this example can be confusing because `wifi profile 0 enable` is equivalent to:

```
wifi profile 0
wifi enable
```

However, `wifi profile 0 enable` is *not* equivalent to:

```
wifi profile 0
wifi profile enable
```

There may be changes to clear confusion like this in the future since it seems logical to infer that all the arguments apply to the profile if the profile was specified earlier in the command. Scripts using these commands must not rely on the current behavior of using multiple arguments.

The `wifi` command supports the sub-commands in Table 7.

Table 7: Subcommands Supported by the `wifi` Command

Subcommand	Description
`wifi commit`	Causes the Wi-Fi manager to act on the changes made by other Wi-Fi CLI commands. For example, if `wifi enable 1` is used, the actual join or AP mode change does not occur until `wifi commit` is used.
`wifi disable`	Immediately disables Wi-Fi.
`wifi enable [1|0]`	Enables or disables Wi-Fi. If the `0` or `1` is omitted, this acts immediately.
`wifi join`	Starts an attempt to join the selected profile even if it is disabled. This is the same as when a mobile app attempts to join a network. The profile is enabled and saved according to the `save_on_server_connect` or `save_on_ap_connect` parameters.
`wifi key <passphrase>`	Sets the Wi-Fi security key for the current profile. For WEP, the first hex key of 10 or 26 hex characters should be used.
`wifi security [none|WEP|WPA|WPA2_Personal]`	Sets the Wi-Fi security for the current profile to either `WPA2_Personal`, `WPA`, `WEP`, or `none`. **NOTE:** These entries are case-sensitive.
`wifi ssid <name>`	Sets the Wi-Fi SSID for the current profile.
`wifi profile <0-9 | ap | enable | disable>`	With values `0` to `9` or `ap`, the Wi-Fi profile is set to be acted on by the other Wi-Fi commands. The `ap` profile sets the parameters to be used when the module is in AP mode. The default profile is `0` on boot-up.
`wifi save_on_ap_connect <0|1>`	Clears or sets the behavior of the Wi-Fi manager to save new profiles chosen by the internal web server or API when the profile has successfully associated with an access point and the DHCP address is acquired. This defaults to `0`.

Subcommand	Description
`wifi save_on_server_connect <0|1>`	Clears or sets the behavior of the Wi-Fi manager to save new profiles chosen by the internal web server or API when the profile has successfully connected to the configured device service. This defaults to `1`.

With regard to the last two subcommands in Table 7, if `save_on_server_connect` is set to `1` and the agent (client) is disabled, the behavior is as if `save_on_ap_connect` is set to `1`. Refer to the following example:

```
wifi profile 0
wifi ssid mynetwork
wifi key mykey1234
wifi security WPA2_Personal
wifi profile enable
wifi join

wifi profile 9
wifi profile erase
```

4250 Burton Drive, Santa Clara, CA 95054
Phone: +1 408 830 9844
Fax: +1 408 716 2621