Chapter 5: Connecting to Access Points (AP)

Objective

At the end of this chapter you will understand the fundamentals of operating as a Wi-Fi Station (STA) and connecting to a Wi-Fi Access Point (AP). You will have an introduction to the TCP/IP Networking stack and you will have a basic understanding of the first three layers of the Open Systems Interconnection (OSI) reference model for a network stack (i.e. physical, datalink and network layers). You will also have an understanding of the Wi-Fi datalink layer which handles connections and encryption. Finally, you will understand some of the basics of IP networking (addresses, netmasks) and the role of the WICED Device Configuration Table (DCT).

Most importantly, you will be able to use WICED to connect your IoT device to a Wi-Fi Network.

Time: 1 ¾ Hours

Fundamentals

TCP/IP Networking Stack

Almost all complicated systems manage the overall complexity by dividing the system into layers. The "Network Stack" or more accurately, the "TCP/IP Network Stack" is exactly that: a hierarchical system for reliably communicating over multiple networking mediums (Wi-Fi, Ethernet, etc.). Each layer isolates the user of that layer from the complexity of the layer below it, and simplifies the communication for the layer above it. You might hear about the OSI Network Model which is another, similar way to describe networking layers; however, it is easier to envision IP networks using the TCP/IP model.

Each layer takes the input of the layer above it and then embeds that information into one or more of the Protocol Data Units (PDUs) of that layer. A PDU is the atomic unit of data for a particular layer: e.g. the Datalink Layer takes an IP packet and divides it up into 1 or more Wi-Fi Data Link Layer Frames. The physical layer takes Datalink Layer Frames and divides them up into bits.

Layer	Protocol	Protocol Data Unit	Comment
Layer 5 Application	DNS, DHCP, MQTT, HTTP, etc.	Data	The layers below the application provide the mechanism to trade useful data. The application layer is the actual protocol to do something useful in the device e.g. HTTP (get or put data), DNS (find a IP address from a name), MQTT (publish or subscribe) etc.
Layer 4 Transport	TCP	(TCP) Segments	Reliable, ordered, error checked stream of bytes – think of it as a pipe between

	UDP	(UDP) Datagram	computers or as a phone call. An unreliable connectionless datagram flow (packets of data) – think of it like dropping an envelope in the mail to the post office, you don't know it is received until the other side confirms and delivery order is not guaranteed.
Layer 3 Network	IP	Packets	An IP network can send and receive IP packets with source and destination IP addresses to anywhere on the Internet. The IP layer deals with addressing and routing of packets.
Layer 2 Data-Link	802.11 MAC	Frame	A frame is the atomic unit of transmission in the network. Each frame is no more than one Maximum Transmission Unit (MTU) of data which is specific to each data-link layer. All of the data from the layers above are broken into frames by the data link layer. Converts bits into unencrypted frames. This layer only communicates on the Local Area Network
Layer 1 Physical	802.11(a,b,g, n,ac)	Bits	Sends and receives streams of bits over the Wi-Fi Radio; handles carrier access and arbitration for the network medium.

(Physical/Datalink) Wi-Fi Basics

There are two ends of a Wi-Fi network: the Station (i.e. the IoT device) and the Access Point (i.e. the wireless router). In order for a Station to connect to a Wi-Fi Access Point, it must have the following information: **SSID** and **Encryption Scheme**. The WICED chip will take care of selecting the proper band and channel. In addition, in order to send data, all Wi-Fi Datalink Frames are labeled with the source and destination **MAC Addresses**.

SSID (the name of the wireless network)

SSID stands for Service Set Identifier. The SSID is the network name and is composed of 0-32 bytes (a.k.a. octets - which is the same as an 8-bit byte - but for some reason which is lost in the mists of history, networking guys always call them octets). The name does not have to be human readable (e.g. ASCII) but because it is uncoded bytes, it is effectively case sensitive (be careful).

Band (either 2.4GHz or 5GHz)

Wi-Fi radios encode 1's and 0's with one of a number of different modulation schemes depending on the type of Wi-Fi network (a,b,g,n,ac,ax) and operating mode. The types of encoding are transparent to your IoT application since the chip, radio, and firmware will virtualize this for you. The data is then transmitted into the 2.4GHz or 5GHz band (which band is important). Note that 5GHz band has higher throughput and less latency but less range while the opposite is true for 2.4GHz band.

Channel number

The available channels are band (2.4GHz) and geographically (location) specific. Additionally, the FCC regulates which channels and bands may be used for different operating regions of the world. At the Wi-Fi layer, this is configured via a country-code setting which maps to a set of available channels for that region. 2.4GHz is pretty simple, there are channels 1-14 with 1-11 available all over the world. 5GHz is region specific and regulatory bodies (e.g. the FCC) will mandate which channels you may use depending on the region.

However, from the station point of view (and therefore for this class) none of this matters since when you try to join an SSID the WICED SDK will scan all the channels looking for the correct SSID.

Encryption (Open, WEP, WPA, WPA2)

In order to provide security for Wi-Fi networks it is common to use data link layer encryption. The types of network encryption are Open (i.e. no security), Wireless Equivalent Privacy (WEP) which is not completely secure (but may be OK for some type of limited legacy applications), Wi-Fi Protected Access (WPA) and WPA2 which has largely displaced WPA (you must support WPA2 to use the Wi-Fi logo on your product). There are two versions of WPA1/2: one called "Personal" or "Pre Shared Key" (PSK) and one called "Enterprise".

WEP and WPA PSK both use a password—called a key—to encrypt the data. The WEP encryption scheme is not recommended as it is very easy to compromise (e.g. using tools like Wireshark and AirSnort). The PSK key scheme of WPA is very secure as it uses AES (Advanced Encryption Standard). However, sharing keys is a painful, unsecure process because it means that everyone has the same key.

To solve the key distribution problem, most enterprise networking solutions use WPA2 Enterprise which requires use of a RADIUS server to handle authentication of each station individually.

Enterprise security is an oncoming crisis for the IoT market and is a differentiating feature of WICED – when you use WICED, this is all taken care of for you – auto-magically!

Media Access Control (MAC) Address

The Wi-Fi MAC address is a 48-bit unique number comprised of an OUI (Organizationally Unique ID) and a station ID. The first three bytes of the MAC address are the OUI field and that is assigned by IEEE to be unique per manufacturer (e.g. Cypress). In order for the datalink layer to send a frame it must address the frame with a source and destination MAC address. Other devices on the network will only pass frames into the higher levels of the stack that are addressed to them. Remember that the Datalink Layer does not know anything about the higher layers (e.g. IP). Finally, the most significant bit of the most significant byte (e.g. bit 47) specifies a multicast (Group) address and the special address of all 1's (e.g. ff:ff:ff:ff:ff:ff) is a broadcast address (send to everyone).

The datalink layer needs to be able to figure out the MAC address of a particular IP Address in order to send it out on the Wi-Fi network. In order to figure out this mapping there is a protocol called Address Resolution Protocol (ARP).

ARP

Inside of every device there is an ARP table that has a map of MAC Address to IP address. In order to discover the MAC address of an IP address, an "ARP request" is broadcast to the network. All devices attached to a network listen for ARP requests. If you hear an ARP request with your IP address in it, you respond with your MAC address. From that point forward both sides add that information to their ARP table (and in fact if you hear others ARPing you can update your table as well). The brilliant part of this scheme is that if you ARP for an IP address that is not on your local network, the router will respond with its MAC address (the subject of the next section).

IP Networking
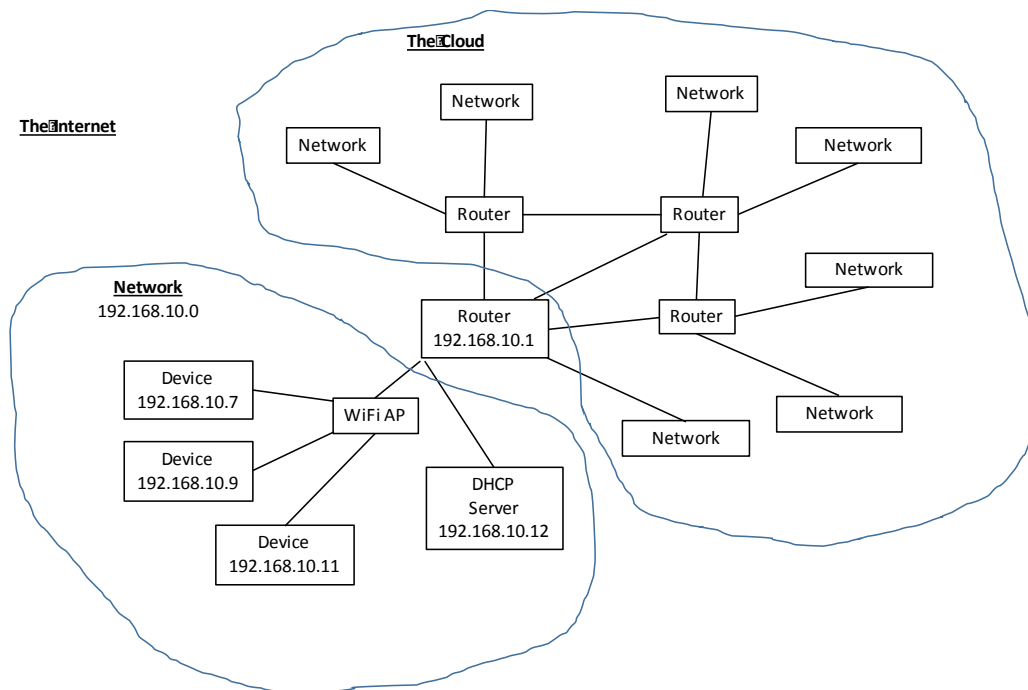
The Internet is a mesh of interconnected **IP networks**. **The Cloud** is all of the Internet that is accessible by your network, but may also mean servers that are attached to a network somewhere on the Internet.

All **devices** on the Internet have a legal **IP address** and belong to an (IP) **Network** that is defined by a **Netmask**. **Routers** are devices that connect IP networks by taking IP packets from one network and forwarding them along to the correct next network. This is a complicated task and is outside of the scope of this class, but it is the reason that Cisco is worth $151B. For the purposes of this class you should just think that once you have connected to the network that your packets are magically transported to the other end.

An IP Address uniquely identifies an individual device with a 32-bit number that is generally expressed as four hex-bytes separated by periods. E.g. 192.168.15.7. IP addresses are divided into two parts: the network address (which is the first x number of bits) and the client address which are the last 32-x bits. The netmask defines the split of network/client. E.g. the netmask for 192.168.15.* is 255.255.255.0

An IP Network (sometimes called an IP Subnetwork) is the collection of devices that all share the same network address e.g. all of the devices on 192.168.15.* (netmask 255.255.255.0) are all part of the same IP Network.

Most commonly, IP addresses for IoT type devices are assigned dynamically by a Dynamic Host Control Protocol (DHCP) server. To dynamically assign a DHCP address you first send a Layer-2 broadcast datagram requesting an IP address (DHREQUEST). When a DHCP server hears the request it responds with the required information. DHCP is integrated into WICED and handles this exchange of information for you automatically when enabled.

Device Configuration Table (DCT)

The device configuration table is a section of the WICED flash with a predefined format that is used to store fundamental information about the system (i.e. client AP SSID, client AP passphrase, etc.). It can also be used to store your application information. The DCT is used by the WICED firmware to "do the right thing". For example, *wiced_network_up()* reads the network information from the DCT and connects to the specified network.

The table is built during the make process and written into the flash along with your application. The DCT can also be modified (and written) on the fly by your application.

When building a WICED App you can either use the default DCT or you can make a custom one or a custom section of one. To preconfigure the Wi-Fi section of the DCT table you need to create a .h file (generally called wifi_config_dct.h with the correct #defines. You then need to add "WIFI_CONFIG_DCT_H := wifi_config_dct.h" to the makefile so that the your custom DCT is built.

 You can get a template for the file in the directory "include/default_wifi_config_dct.h". Note that the name of the file is hard-coded in most projects to be "wifi_config_dct.h" so you must rename it.

```
 1  #pragma once
 2
 3  /* This is the soft AP used for device configuration */
 4  #define CONFIG_AP_SSID        "WICED_AWS"
 5  #define CONFIG_AP_CHANNEL     1
 6  #define CONFIG_AP_SECURITY    WICED_SECURITY_WPA2_AES_PSK
 7  #define CONFIG_AP_PASSPHRASE "12345678"
 8
 9  /* This is the soft AP available for normal operation (if used)*/
10  #define SOFT_AP_SSID          "WICED Device"
11  #define SOFT_AP_CHANNEL       7
12  #define SOFT_AP_SECURITY      WICED_SECURITY_WPA2_AES_PSK
13  #define SOFT_AP_PASSPHRASE    "WICED_PASSPHRASE"
14
15  /* This is the default AP the device will connect to (as a client)*/
16
17  #define CLIENT_AP_SSID        "Guest"
18  #define CLIENT_AP_PASSPHRASE ""
19
20  #define CLIENT_AP_BSS_TYPE    WICED_BSS_TYPE_INFRASTRUCTURE
21  #define CLIENT_AP_SECURITY    WICED_SECURITY_OPEN
22  #define CLIENT_AP_CHANNEL     6
23  #define CLIENT_AP_BAND        WICED_802_11_BAND_2_4GHZ
24
25  /* This is the network interface the device will work with */
26  #define WICED_NETWORK_INTERFACE    WICED_STA_INTERFACE
```

The device can operate in three modes as seen in the table above: Configuration AP (lines 4-7), Soft AP (10-13), and Client Mode (lines 17-23). It is also possible to have multiple network interfaces as well as support Wi-Fi and Ethernet (line 26).   The configuration AP is used for devices that want to allow other devices to connect to them to perform configuration of the WICED system over Wi-Fi. The soft AP is used for devices that will act as a Wi-Fi access point during normal operation. The client is used for devices that will connect to an existing Wi-Fi network as a station. For the purposes of this chapter we will only be a CLIENT so you will only need to touch 17-23.

To find the definition (or possible definitions) of the #defines you can highlight, right click, and select "Open declaration". For example, if you open the declaration of "WICED_SECURITY_OPEN", it will take you to:

```
144 typedef enum
145 {
146     WICED_SECURITY_OPEN          = 0,
147     WICED_SECURITY_WEP_PSK       = WEP_ENABLED,
148     WICED_SECURITY_WEP_SHARED    = ( WEP_ENABLED   | SHARED_ENABLED ),
149     WICED_SECURITY_WPA_TKIP_PSK  = ( WPA_SECURITY  | TKIP_ENABLED ),
150     WICED_SECURITY_WPA_AES_PSK   = ( WPA_SECURITY  | AES_ENABLED ),
151     WICED_SECURITY_WPA_MIXED_PSK = ( WPA_SECURITY  | AES_ENABLED | TKIP_ENABLED ),
152     WICED_SECURITY_WPA2_AES_PSK  = ( WPA2_SECURITY | AES_ENABLED ),
153     WICED_SECURITY_WPA2_TKIP_PSK = ( WPA2_SECURITY | TKIP_ENABLED ),
154     WICED_SECURITY_WPA2_MIXED_PSK = ( WPA2_SECURITY | AES_ENABLED | TKIP_ENABLED ),
155
156     WICED_SECURITY_WPA_TKIP_ENT  = ( ENTERPRISE_ENABLED | WPA_SECURITY  | TKIP_ENABLED ),
157     WICED_SECURITY_WPA_AES_ENT   = ( ENTERPRISE_ENABLED | WPA_SECURITY  | AES_ENABLED ),
158     WICED_SECURITY_WPA_MIXED_ENT = ( ENTERPRISE_ENABLED | WPA_SECURITY  | AES_ENABLED | TKIP_ENABLED ),
159     WICED_SECURITY_WPA2_TKIP_ENT = ( ENTERPRISE_ENABLED | WPA2_SECURITY | TKIP_ENABLED ),
160     WICED_SECURITY_WPA2_AES_ENT  = ( ENTERPRISE_ENABLED | WPA2_SECURITY | AES_ENABLED ),
161     WICED_SECURITY_WPA2_MIXED_ENT = ( ENTERPRISE_ENABLED | WPA2_SECURITY | AES_ENABLED | TKIP_ENABLED ),
162
163     WICED_SECURITY_IBSS_OPEN     = ( IBSS_ENABLED ),
164     WICED_SECURITY_WPS_OPEN      = ( WPS_ENABLED ),
165     WICED_SECURITY_WPS_SECURE    = ( WPS_ENABLED | AES_ENABLED),
166
167     WICED_SECURITY_UNKNOWN       = -1,
168
169     WICED_SECURITY_FORCE_32_BIT  = 0x7fffffff
170 } wiced_security_t;
```

You can see from the figure above that WICED supports just about any type of Wi-Fi security you can think of.

All of the DCT information is mapped into flash by the WICED SDK. Typically you won't need to know about it since you just choose your settings in the wiced_config_dsct.h file, but if you want to read/modify some of the DCT settings from the firmware you will need to understand how the values are stored in flash.

The WICED SDK provides a predefined structure for the DCT mapping in the file platform_dct.h which can be found in the WICED/platform/include folder).

```
740 typedef struct
741 {
742     platform_dct_header_t            dct_header;
743     platform_dct_mfg_info_t          mfg_info;
744     platform_dct_security_t          security_credentials;
745     platform_dct_wifi_config_t       wifi_config;
746     platform_dct_ethernet_config_t   ethernet_config;
747     platform_dct_network_config_t    network_config;
748     platform_dct_bt_config_t         bt_config;
749     platform_dct_p2p_config_t        p2p_config;
750     platform_dct_ota2_config_t       ota2_config;
751     platform_dct_version_t           dct_version;
752 } platform_dct_data_t;
753
```

As you can see from the table above, the DCT is divided into sections. As an example, wifi_config is a structure of type platform_dct_wifi_config_t that contains information about the Wi-Fi configuration including the known access points. If you right click and do Open Declaration on platform_dct_wifi_config_t you will see:

```
611 typedef struct
612 {
613     wiced_bool_t                device_configured;
614     wiced_config_ap_entry_t     stored_ap_list[CONFIG_AP_LIST_SIZE];
615     wiced_config_soft_ap_t      soft_ap_settings;
616     wiced_config_soft_ap_t      config_ap_settings;
617     wiced_country_code_t        country_code;
618     wiced_aggregate_code_t      aggregate_code;
619     wiced_mac_t                 mac_address;
620     uint8_t                     padding[2];  /* ensure 32bit aligned size */
621 } platform_dct_wifi_config_t;
622
```

The second entry "stored_ap_list" is an array of type "wiced_config_ap_entry_t". The first element (i.e. index 0) of this array contains information for the access point that the STA connects to as a client. If you right click on wiced_config_ap_entry_t and do Open Declaration, you will see:

```
571 typedef struct
572 {
573     wiced_ap_info_t details;
574     uint8_t         security_key_length;
575     char            security_key[ SECURITY_KEY_SIZE ];
576 } wiced_config_ap_entry_t;
577
```

The first entry in this structure (details, which is a structure of type wiced_ap_info_t) contains details of the access point that the client will connect to. If you rght click on wiced_ap_info_t and do Open Declaration, you will see:

```
162 typedef struct wiced_ap_info
163 {
164     wiced_ssid_t         SSID;            /**< Service Set Identification (i.e. Name of Access Point)
165     wiced_mac_t          BSSID;           /**< Basic Service Set Identification (i.e. MAC address of A
166     int16_t              signal_strength; /**< Receive Signal Strength Indication in dBm. <-90=Very poo
167     uint32_t             max_data_rate;   /**< Maximum data rate in kilobits/s
168     wiced_bss_type_t     bss_type;        /**< Network type
169     wiced_security_t     security;        /**< Security type
170     uint8_t              channel;         /**< Radio channel that the AP beacon was received on
171     wiced_802_11_band_t  band;            /**< Radio band
172     struct wiced_ap_info* next;           /**< Pointer to the next scan result
173 } wiced_ap_info_t;
```

Many of the entries in this structure are also structures. You can explore each of the individual structures to see what values they contain.

The DCT may exist as a series of flash rows inside of the application processor (i.e. if it has internal flash), or it may exist in a serial flash attached to the Wi-Fi chip. In order to read from the DCT you need to call the function *wiced_dct_read_lock()* which will read the DCT into a RAM buffer which you can then modify and then write back to the flash with the function *wiced_dct_write()*.

You provide the *wiced_dct_read_lock()* call with a pointer to a pointer to an empty structure which will be filled with the DCT Wi-Fi data. The type of structure depends on which section of the DCT that you want to read (the section is a parameter to the *wiced_dct_read_lock()* function). For example, if you want to read the DCT_WIFI_CONFIG_SECTION, then the pointer type would be platform_dct_wifi_config_t.

You can find the list of section names in the wiced_dct_common.h file which is located in WICED/platform/MCU. Here are the sections available:

```
74  /* DCT section */
75  typedef enum
76  {
77      DCT_APP_SECTION,
78      DCT_SECURITY_SECTION,
79      DCT_MFG_INFO_SECTION,
80      DCT_WIFI_CONFIG_SECTION,
81      DCT_INTERNAL_SECTION, /* Do not use in apps */
82      DCT_ETHERNET_CONFIG_SECTION,
83      DCT_NETWORK_CONFIG_SECTION,
84      DCT_BT_CONFIG_SECTION,
85      DCT_P2P_CONFIG_SECTION,
86      DCT_OTA2_CONFIG_SECTION,
87      DCT_VERSION_SECTION, /* Do not use in apps */
88  } dct_section_t;
```

When you are done with the RAM copy of the DCT you need to free it by calling the function *wiced_dct_read_unlock()*.

If the flash is "internal" and directly accessible by the processor you can call *wiced_dct_read_lock()* with the writable parameter set to false, in which case the *wiced_dct_read_lock()* will give you a pointer to the flash instead of making a copy in RAM. You can only do this if you are only going to read the DCT. That is, if you want to be able to write to the DCT, the writable parameter must be set to true.

The DCT functions are documented under Components→WICED Application Framework→DCT

The WICED Wi-Fi SDK

In order to attach to a Wi-Fi network you must call the *wiced_network_up()* function. That API call has three parameters: the networking interface to use; which method to use to get your IP address etc.; which static IP parameters to use (or NULL). Here is the API from the WICED documentation:

```
wiced_result_t wiced_network_up ( wiced_interface_t          interface,
                                  wiced_network_config_t    config,
                                  const wiced_ip_setting_t * ip_settings
                                )
```

The parameter wiced_interface_t specifies which network interface to use. The WICED-SDK supports the ability to use multiple networks at the same time e.g. Wi-Fi & Ethernet. To find the definition, I went to the definition of wiced_interface_t (by highlighting, right clicking and selecting Open Declaration) in the SDK. For the purposes of this class we will always use the WICED_STA_INTERFACE, meaning we are always going to be a station (i.e. client), never an access point.

```
typedef enum
{
    WICED_STA_INTERFACE      = WWD_STA_INTERFACE,       /**< STA or Client Interface  */
    WICED_AP_INTERFACE       = WWD_AP_INTERFACE,        /**< softAP Interface         */
    WICED_P2P_INTERFACE      = WWD_P2P_INTERFACE,       /**< P2P Interface            */
    WICED_ETHERNET_INTERFACE = WWD_ETHERNET_INTERFACE,  /**< Ethernet Interface       */

    WICED_INTERFACE_MAX,        /** DO NOT USE - MUST BE AFTER ALL NORMAL INTERFACES - used for counting interfaces */
    WICED_CONFIG_INTERFACE   = WICED_AP_INTERFACE | (1 << 7), /**< config softAP Interface  */
} wiced_interface_t;
```

The next parameter in the *wiced_network_up()* call is how to configure the network, meaning how you specify the IP address, Netmask, Router etc. You can either set it statically or you can use DHCP. The WICED SDK can turn on a DHCP server inside of your device to serve DHCP requests from all over the network. This would be useful if you were acting as the access point. However, for the purposes of the class we will use "WICED_USE_EXTERNAL_DHCP_SERVER" so that you get your IP information from the DHCP running in the class's router. Here is a screen shot of the options:

```
typedef enum
{
    WICED_USE_EXTERNAL_DHCP_SERVER, /**< Client interface: use an external DHCP server */
    WICED_USE_STATIC_IP,            /**< Client interface: use a fixed IP address */
    WICED_USE_INTERNAL_DHCP_SERVER  /**< softAP interface: use the internal DHCP server */
} wiced_network_config_t;
```

If you are using an external DHCP server, then you don't need to specify the ip_settings. In that case, just use NULL for the third parameter.

If you are not using an external DHCP server, you need to statically specify the IP networking parameters by passing a structure called wiced_ip_setting_t. That structure has three elements as can be seen here:

```
/** IP address settings */
typedef struct
{
    wiced_ip_address_t ip_address;   /**< IP address       */
    wiced_ip_address_t gateway;      /**< Gateway address */
    wiced_ip_address_t netmask;      /**< Netmask          */
} wiced_ip_setting_t;
```

WICED_RESULT_T

Throughout the WICED SDK, a value from many of the functions is returned telling you what happened. The return value is of the type "wiced_result_t" which is a giant enumeration. Some values that we return include WICED_SUCCESS, WICED_PENDING and WICED_ERROR. If you look at the wiced_result_t you will not see those values because the enumeration is built up hierarchically to make it easier to maintain. Here it the top level of the hierarchy:

```
typedef enum
{
    WICED_RESULT_LIST       ( WICED_            )  /*      0 -    999 */
    WWD_RESULT_LIST         ( WICED_WWD_        )  /*   1000 -   1999 */
    WLAN_RESULT_LIST        ( WICED_WLAN_       )  /*   2000 -   2999 */
    WPS_BESL_RESULT_LIST    ( WICED_BESL_       )  /*   3000 -   3999 */
    RESOURCE_RESULT_LIST    ( WICED_RESOURCE_   )  /*   4000 -   4999 */
    TLS_RESULT_LIST         ( WICED_TLS_        )  /*   5000 -   5999 */
    PLATFORM_RESULT_LIST    ( WICED_PLATFORM_   )  /*   6000 -   6999 */
    TCPIP_RESULT_LIST       ( WICED_TCPIP_      )  /*   7000 -   7999 */
    BT_RESULT_LIST          ( WICED_BT_         )  /*   8000 -   8999 */
    P2P_RESULT_LIST         ( WICED_P2P_        )  /*   9000 -   9999 */
    FILESYSTEM_RESULT_LIST( WICED_FILESYSTEM_ )  /* 10000 - 10999 */
} wiced_result_t;
```

You can look at the sub list by right clicking on it. That is, if you click on the WICED_RESULT_LIST you will see all of the enumerations of the form "WICED_". So, for a successful command, you will see "WICED_SUCCESS".

```
#define WICED_RESULT_LIST( prefix ) \
    RESULT_ENUM( prefix, SUCCESS,                        0 ),   /**< Success */                               \
    RESULT_ENUM( prefix, PENDING,                        1 ),   /**< Pending */                               \
    RESULT_ENUM( prefix, TIMEOUT,                        2 ),   /**< Timeout */                               \
    RESULT_ENUM( prefix, PARTIAL_RESULTS,                3 ),   /**< Partial results */                       \
    RESULT_ENUM( prefix, ERROR,                          4 ),   /**< Error */                                 \
    RESULT_ENUM( prefix, BADARG,                         5 ),   /**< Bad Arguments */                         \
    RESULT_ENUM( prefix, BADOPTION,                      6 ),   /**< Mode not supported */                    \
    RESULT_ENUM( prefix, UNSUPPORTED,                    7 ),   /**< Unsupported function */                  \
    RESULT_ENUM( prefix, OUT_OF_HEAP_SPACE,              8 ),   /**< Dynamic memory space exhausted */ \
    RESULT_ENUM( prefix, NOTUP,                          9 ),   /**< Interface is not currently Up */  \
    RESULT_ENUM( prefix, UNFINISHED,                    10 ),   /**< Operation not finished yet */            \
    RESULT_ENUM( prefix, CONNECTION_LOST,               11 ),   /**< Connection to server lost */             \
    RESULT_ENUM( prefix, NOT_FOUND,                     12 ),   /**< Item not found */                        \
    RESULT_ENUM( prefix, PACKET_BUFFER_CORRUPT,         13 ),   /**< Packet buffer corrupted */               \
    RESULT_ENUM( prefix, ROUTING_ERROR,                 14 ),   /**< Routing error */                         \
    RESULT_ENUM( prefix, BADVALUE,                      15 ),   /**< Bad value */                             \
    RESULT_ENUM( prefix, WOULD_BLOCK,                   16 ),   /**< Function would block */                  \
    RESULT_ENUM( prefix, ABORTED,                       17 ),   /**< Operation aborted */                     \
    RESULT_ENUM( prefix, CONNECTION_RESET,              18 ),   /**< Connection has been reset */             \
    RESULT_ENUM( prefix, CONNECTION_CLOSED,             19 ),   /**< Connection is closed */                  \
    RESULT_ENUM( prefix, NOT_CONNECTED,                 20 ),   /**< Connection is not connected */     \
    RESULT_ENUM( prefix, ADDRESS_IN_USE,                21 ),   /**< Address is in use */                     \
    RESULT_ENUM( prefix, NETWORK_INTERFACE_ERROR,       22 ),   /**< Network interface error */               \
    RESULT_ENUM( prefix, ALREADY_CONNECTED,             23 ),   /**< Socket is already connected */      \
    RESULT_ENUM( prefix, INVALID_INTERFACE,             24 ),   /**< Interface specified in invalid */ \
    RESULT_ENUM( prefix, SOCKET_CREATE_FAIL,            25 ),   /**< Socket creation failed */                \
    RESULT_ENUM( prefix, INVALID_SOCKET,                26 ),   /**< Socket is invalid */                     \
    RESULT_ENUM( prefix, CORRUPT_PACKET_BUFFER,         27 ),   /**< Packet buffer is corrupted */            \
    RESULT_ENUM( prefix, UNKNOWN_NETWORK_STACK_ERROR,   28 ),   /**< Unknown network stack error */      \
    RESULT_ENUM( prefix, NO_STORED_AP_IN_DCT,           29 ),   /**< DCT contains no AP credentials */ \
    RESULT_ENUM( prefix, STA_JOIN_FAILED,               30 ),   /**< Join failed */                           \
    RESULT_ENUM( prefix, PACKET_BUFFER_OVERFLOW,        31 ),   /**< Packet buffer overflow */                \
    RESULT_ENUM( prefix, ALREADY_INITIALIZED,           32 ),   /**< Module has already been inited */
```

Documentation

The relevant documentation for the networking management functions are in the WICED SDK documentation under Components→Management→Network Management.

Functions that allows you to interface with the Raw IP networking are available in the documentation under Components→IP Communication→Raw IP.

In addition, there is a document called WICED-DCT.pdf in the doc directory that includes a discussion of the DCT.

Introducers

An introducer is a method used to get IoT devices connected to the network. That is, they need to know which Wi-Fi SSID to connect to, what password to use, what encryption keys to use, etc. There are a number of possible strategies for solving this problem including:

- Connecting to the IoT device using Bluetooth and then using a phone based App to configure the device.
- Connecting the IoT device to a computer using a USB or Serial connection and then configuring the device with a computer based application.
- Starting a Wi-Fi Access Point with a web server on the IoT device, then connecting to the IoT device with a computer or a cellphone. The device configuration section of the DCT is used for this purpose.
- Preprogramming the device with the required information.

WICED supports all of these methods. In this class we will mainly use the pre-programmed method in the interest of simplicity and time. Some examples in later chapters use a Wi-Fi Access Point with a web server on the IoT device. Each of the other methods are demonstrated in the sample applications that come with the SDK.

Exercise(s)

01 Create an App that attaches to a WPA2 AES PSK network, have LED1 turn on for success and blink on failure

1. Make a new folder called 05 and a sub-folder called 01_attach_wpa2 (or copy a previous project).
2. Copy the template default_wifi_config_dct.h into your application folder (from step 1) and name it wifi_config_dct.h.
   a. Hint: Remember it is in the WICED include directory.
3. Modify wifi_config_dct.h.
   a. Hint: The network name and password are on the back cover of the manual.
4. Create and edit the makefile (don't forget to add the line for WIFI_CONFIG_DCT_H).
5. Create and edit 01_attach_open.c (use the function *wiced_network_up()* to read the DCT and start the network).
6. Check the return code and either turn on or blink the LED as appropriate.
   a. Hint: Use a serial terminal emulator to look at messages from the device as it boots and connects.

02 Attach to an open network

1. How would you modify the previous exercise to attach to an open network called WW101OPEN?
   a. Hint: There are three changes required.

03 Print Network Information

1. Copy exercise (01) and add functions to print out networking information:
- Your IP address (wiced_ip_get_ipv4_address)
- Netmask (wiced_ip_get_netmask)
- Router Gateway (wiced_ip_get_gateway_address)
- The IP address of www.cypress.com (wiced_hostname_lookup)
- MAC Address of your device (wwd_wifi_get_mac_address)

   a. Hint: look at the API guide sections *Components > IP Communication > Raw IP* and *Components > IP Communication > DNS lookup*.
   b. Hint: The addresses (IP address, Netmask, Gateway, and Cypress.com) are returned as a structure of type wiced_ip_address_t. One element in the structure (called ip.v4) is a uint32_t which contains the IPV4 address as 4 hex bytes. You can mask off each of these bytes individually and print them as decimal values separated by periods to get the format that is typically seen. For example, the netmask of 255.255.255.0 will be returned as 0xFFFFFF00.
   c. Hint: The MAC address is returned as a structure of type wiced_mac_t. This structure contains an element called octet which is an array of 6 octets (bytes). You can print each of these bytes individually separated by ":" to see the MAC address in the typical format.

04 (Advanced) Create an application that can switch between two different SSIDs

1. Hint: A second network called WW101WPA_SWITCH with password "cypresswicedwifi101s" is available for this exercise.
2. Copy the project from exercise (03).
3. Create a function that can print the SSID/Passphrase and Security that currently exists in the DCT.
4. Create a function that takes input as (char* ssid, char* passphrase, wiced_security_t security) and then writes that information into the DCT by performing the following steps:
   a. Take the network down (*wiced_network_down()*).
   b. Write the DCT with the other network's information:
      i. Use *wiced_dct_read_lock()* to get current structure.
         1. Hint: To write values you must use WICED_TRUE for the ptr_is_writable parameter.
      ii. Update the required information.
         1. Hint: For the values that are strings (i.e. ssid and passphrase):
            a. Use *strcpy()* to copy the values into the RAM buffer.
            b. Make sure you update the string length in the structure (you can use *strlen()* to find the length of the string).
      iii. Use *wiced_dct_write()* to update the DCT in flash.
      iv. Use *wiced_dct_read_unlock()* to free up the memory.
         1. Hint: the ptr_is_writable parameter must match the corresponding *wiced_dct_read_lock()* function call.
      v. Hint: See the example project *snip/dct_read_write*.
   c. Restart the network (*wiced_network_up()*).
5. Use the console as input. When the user presses '0' or '1' switch between network 0/1 - e.g. 0=WA101WPA and 1=WA101WPA_SWITCH. If the user presses 'p' call the print function that you wrote in step (1).
   a. Hint: Review the UART receive exercise from chapter 2.
6. Change the selected network and then power cycle or reset the board and notice that it will start with the new SSID since the new setting is saved in the DCT.

Recommended Reading

[1] TCP/IP Illustrated – Volume 1: The Protocols, W.R. Stevens, ISBN 0201633469 – "aka" the Networking Bible, if there is one book to get on TCP/IP networking, this is it!

[2] UNIX Network Programming – W.R. Stevens, ISBN 01394 – if you want to learn BSD Socket programming, there is no other reference – best book and the foundation of all networking software today.

[3] RFC 1122 – "Requirements for Internet Hosts – Communications Layers" ; Internet Engineering Task Force (IETF) - https://tools.ietf.org/html/rfc1122

[4] RFC 826 – "An Ethernet Address Resolution Protocol" ; Internet Engineering Task Force (IETF) - https://tools.ietf.org/html/rfc826

[5] RFC 153 – "Dynamic Host Configuration Protocol"; Internet Engineering Task Force (IETF) - https://tools.ietf.org/html/rfc1531