# DFCM-NNN40-DTxR

## WiFi Software Development Kit Document

# Index:

# Revision History

| Version | Date | Reason of change | Maker |
|---|---|---|---|
| 0.1 | **2015/8/11** | Initial release | **Tsungta Wu** |
| 0.2 | **2015/10/29** | Update chapter 1, chapter 2, chapter3, and section 4.1 to sync up the information with actual development | **Tsungta Wu** |
| 0.3 | **2016/9/14** | Update chapter 1. | **Tsungta Wu** |
| | | | |
| | | | |

# WIFI mbed Software Development Kit

***A Software Development Kit (SDK) used to implement function of Wi-Fi networking function in ARM mbed for Wi-Fi/BLE Combo Module.***

## 1. Introduction

This document first describes the consistence of WIFI SDK used for Wi-Fi/BLE Combo Module in ARM mbed. Then follow by a step by step flow shows how to import this SDK in mbed. WiFi SDK APIs are summarized in chapter 4. Basic Wi-Fi functionality sample code can be found in https://developer.mbed.org/teams/Delta/code/. The very first to start with is NNN40_WiFi.

## 2. System Overview

This SDK is target to provide an interface for application to facilitate TCP/UDP communication through WIFI connectivity. This SDK leverage mbed libraries “EthernetInterface” to provide an officially supported networking stack providing the familiar Berkeley sockets programming interface. Due to the lack of an operating system on NNN40, current SDK could only support a "polling" paradigm. From the perspective of NNN40 application developer, the system overview is shown as below

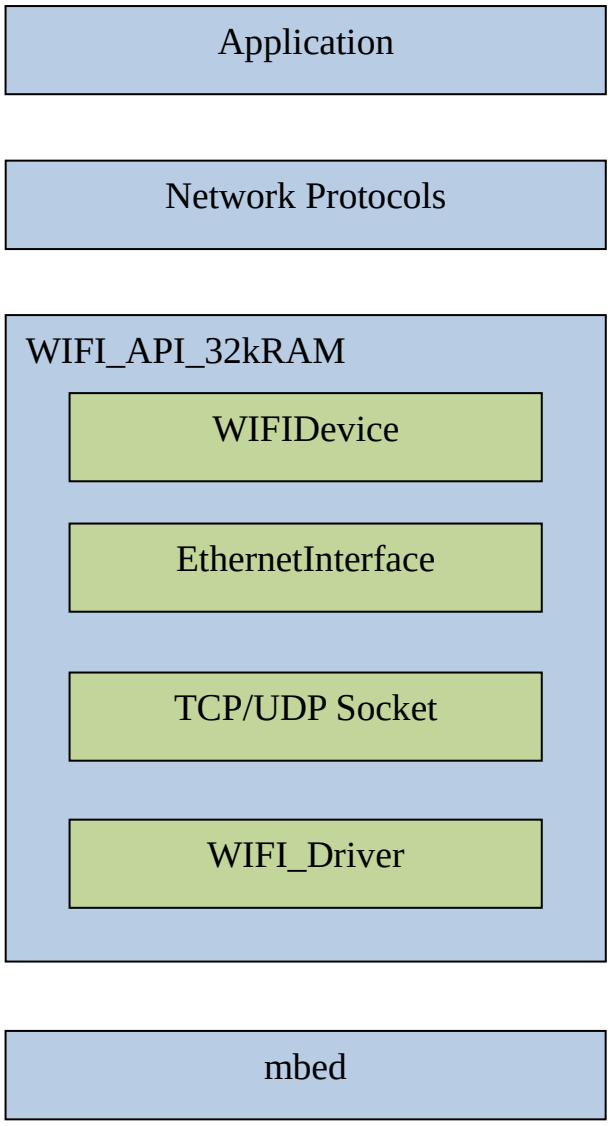


Figure 1: WIFI SDK stack architecture overview.

From the bottom, mbed is the official SDK provide high –level Peripheral APIs to microcontroller coding, refer to this link for details http://developer.mbed.org/handbook/mbed-SDK. WIFI_Driver is the low level driver for embedded Wi-Fi chipset controlling. On the top of

WIFI_Driver, there are high level UDP/TCP Socket API and the EthernetInterface APIs for a specific configuration for the Ethernet transport. WIFIDevice is the high level APIs used to configure Wi-Fi AP link in Station (STA) infrastructure mode AccessPoint (aka. SoftAP) infrastructure mode for Ethernet connection

Networking applications will very likely rely on a specific protocol (HTTP, NTP, FTP, etc). User can refer to a list of network protocol developed by the mbed community from this link http://developer.mbed.org/handbook/TCP-IP-protocols-and-APIs. Note that due to memory constrain of low power Cortex-M0 core which have limited 256 KB Flash and 32 KB RAM, there are about 87 KB Flash and 17.5 KB RAM left for the application. The detail memory usage is summarized in the following table

| Memory usage | Flash (KB) | RAM (KB) |
|---|---|---|
| Total available on Cortex-M0 | 256 | 32 |
| WIFI SDK | 57 | 4.5 |
| BLE SDK* | 112 | 10 |
| WIFI Application available (maximum) | 87 | 17.5 |

* BLE SDK is inherited in mbed NNN40 platform

# 3. Wi-Fi SDK Functionality List

## 3.1 Wi-Fi SDK Available Functionality List

This section lists the available application-level functionalities

- IEEE 802.11 b/g/n connectivity
- Station (STA) infrastructure mode
- AccessPoint (SoftAP) infrastructure mode
- Fast AP connection, rapidly reconnect to the last AP
- Support WEP/WPA/WPA2 security
- DHCP Client and static IPv4 addressing
- DNS client
- TCP Server/Client

- UDP Server/Client
- Support BLE and WIFI coexistence (refer to WIFIDevice.h)
- Support SoftAP mode (refer to sample code NNN40_WiFi_SoftAP)
- Support Websocket library (refer to sample code NNN40_Websocket)
- Support XmppClient library (refer to sample code NNN40XmppClient)
- Support http server (refer to sample code NNN40_APmodeToSTAmodeByHTTPServer)

## 3.2Function Limitations

This section lists the function limitations due to the computation power and memory constrain of low power core MCU.

- Maximum socket connection: 2 TCP and 2 UDP simultaneously
- Maximum TCP/UDP packet payload: 1400 bytes for TCP and 1400 bytes for UDP
- Maximum application layer TCP throughput: 7.5KB
- Maximum application layer UDP throughput: 20KB
- The fastest receive internal for UDP packet: 100ms

# 4. Wi-Fi SDK APIs Summary Table

## 4.1WIFIDevice APIs Summary Table

| Command Syntax | Command Parameters | Description | Remark |
|---|---|---|---|
| int sleep(void) | No input parameter | Disable WIFI and set into sleep mode to conserve energy (no WIFI function is available at this point).<br>\return 0 on success, a negative number on failure | |
| int enableCoexistance() | No input parameter | Enable the setting of two coexistence control pins (COEX_B and COEX_W) are short to each other.<br>\return 0 on success, a negative number on failure | |
| int apScan(void (*eventCallback)(scanApInfo result)) | \result callback function | Scan for available access point on all channels<br>\return number of scanned WIFI access point | |

| | | | |
|---|---|---|---|
| void setNetwork(uint8_t* SSID, uint8_t* PW, uint8_t priority) | \SSID name of access point to connect | Set SSID, password and priority to connect. | |
| | \PW password of the given SSID | | |
| | \priority range from 0 to 2, set 0 for the highest priority | | |
| void WIFIDevice::setAccessPoint(char* SSID, char* PW, security_t security=SECURITY_WPA2_AES_PSK, uint8_t channel=1) | \param SSID name of access point in AP mode | Set SSID, password, security type and channel in AP mode | |
| | \param PW password of the given SSID | | |
| | \param security type of security in AP mode | | |
| | \param channel range from 1 to 14 | | |
| wifiSpInfo read_WIFI_SP_version() | No input parameter | Read WIFI Service Pack version | |
| | | \return wifiSpInfo | |
| int storage_erase4KB(uint32_t address) | \param address range from 0x00 to 0x3F000 (must be a multiple of 0x1000) | Flash memory will be erased in groups of 4KB sector | |
| | | \return 0 on success, a negative number on failure | |
| int storage_write(uint32_t address, uint8_t *data, uint16_t len) | \param address flash memory address to be written, range from 0x00 to 0x3FFFF | Write one (multiple) byte(s) of data into flash, must perform erase before the flash memory area can be overwrite. | |
| | \param data pointer to the buffer containing data to be written | | |
| | \param len length of the data to be written to flash | \return 0 on success, a negative number on failure | |
| int storage_read(uint32_t address, uint8_t *data, uint16_t len) | \param address flash memory address to be read, range from 0x00 to 0x3FFFF | Read one (multiple) byte(s) of data from flash. | |
| | \param data pointer to the buffer containing data to be read | | |
| | \param len length of the data to be read from flash | \return 0 on success, a negative number on failure | |

## 4.2 EthernetInterface APIs Summary Table

On the top of WIFIDevice APIs, there are EthernetInterface and Socket APIs that follow mbed standard APIs to facilitate TCP/UDP communication. Details description and examples can be found from the following links.

EthernetInterface link: http://developer.mbed.org/handbook/Ethernet-Interface

Socket link: http://developer.mbed.org/handbook/Socket

The following tables provide a list of supported APIs that compatible with ARM official APIs

| Command Syntax | Command Parameters | Description | Remark |
|---|---|---|---|
| int init() | No input parameter | Initialize the interface and configure it to use DHCP (no connection at this point).<br>\return 0 on success, a negative number on failure | |
| int init(const char* ip, const char* mask, const char* gateway) | \ip the IP address to use<br>\mask the IP address mask<br>\gateway the gateway to use | Initialize the interface and configure it with the following static configuration (no connection at this point).<br>\return 0 on success, a negative number on failure | The setting of mask and gateway are not supported |
| int connect(unsigned int timeout_ms=35000) | \timeout_ms timeout in ms (default: (35)s). | Bring the WI-FI connection up, start DHCP if needed.<br>\return 0 on success, a negative number on failure | |
| int disconnect() | No input parameter | Bring the interface down<br>\return 0 on success, a negative number on failure | |
| char* getMACAddress() | No input parameter | Get the MAC address of your Ethernet interface<br>\return a pointer to a string containing the MAC address | |
| char* getIPAddress() | No input parameter | Get the IP address of your Ethernet interface<br>\return a pointer to a string containing the IP address | |

| | | | |
|---|---|---|---|
| char* getGateway() | No input parameter | Get the Gateway address of your Ethernet interface<br>\return a pointer to a string containing the Gateway address | |
| char* getNetworkMask() | No input parameter | Get the Network mask of your Ethernet interface<br>\return a pointer to a string containing the Network mask | |

## 4.3Socket APIs Summary Table

| Command Syntax | Command Parameters | Description | Remark |
|---|---|---|---|
| TCPSocketServer | Instantiate a TCP Server | | |
| int bind(int port) | \port The port to listen for incoming connections on. | Bind a socket to a specific port.<br>\return 0 on success, -1 on failure. | |
| int listen(int backlog = 1) | \backlog number of pending connections that can be queued up at any one time [Fixed backlog = 1]. | Start listening for incoming connections.<br>\return 0 on success, -1 on failure. | |
| int accept ( TCPSocketConnection &connection) | \connection A TCPSocketConnection instance that will handle the incoming connection. | Accept a new connection.<br>\return 0 on success, -1 on failure. | |
| void set_blocking(bool blocking, unsigned int timeout=1500) | \blocking true for blocking mode, false for non-blocking mode.<br>\timeout timeout in ms [Default: (1500)ms]. | Set blocking or non-blocking mode of the socket and a timeout on blocking socket operations. | |
| int close(bool shutdown=true) | \shutdown free the left-over data in message queues | Close the socket.<br>\return 0 on success, -1 on failure. | |
| TCPSocketConnection | TCP socket connection. | | |
| int connect (const char *host, const int port) | \host The host to connect to. It can either be an IP Address or | Connects this TCP socket to the server. | |

| | | | |
|---|---|---|---|
| | a hostname that will be resolved with DNS. | | |
| | \port The host's port to connect to. | 0 on success, -1 on failure. | |
| bool is_connected (void) | No input parameter | Check if the socket is connected. | |
| | | \return true if connected, false otherwise. | |
| int send (char *data, int length) | \data The buffer to send to the host. | Send data to the remote host. | |
| | \length The length of the buffer to send. | \return the number of written bytes on success (>=0) or -1 on failure | |
| int send_all(char*data, int length) | \data The buffer to send to the host. | Send all the data to the remote host. | |
| | \length The length of the buffer to send. | \return the number of written bytes on success (>=0) or -1 on failure | |
| int receive(char *data, int length) | \data The buffer in which to store the data received from the host. | Receive data from the remote host. | |
| | \length The maximum length of the buffer. | \return the number of received bytes on success (>=0) or -1 on failure | |
| int receive_all (char *data, int length) | \data The buffer in which to store the data received from the host. | Receive all the data from the remote host. | |
| | \length The maximum length of the buffer. | \return the number of received bytes on success (>=0) or -1 on failure | |
| void set_blocking (bool blocking, unsigned int timeout=1500) | \blocking true for blocking mode, false for non-blocking mode. | Set blocking or non-blocking mode of the socket and a timeout on blocking socket operations. | |
| | \timeout timeout in ms [Default: (1500)ms]. | | |
| int close(bool shutdown=true) | \shutdown free the left-over data in message queues | Close the socket. | |

| | | | |
|---|---|---|---|
| | | \return 0 on success, -1 on failure | |
| UDPSocket | Instantiate an UDP Socket | | |
| int init(void) | No input parameter | Init the UDP Client Socket without binding it to any specific port. | |
| | | \return 0 on success, -1 on failure | |
| int bind(int port) | \port The port to listen for incoming connections on | Bind a UDP Server Socket to a specific port. | |
| | | \return 0 on success, -1 on failure. | |
| int set_broadcasting(bool broadcast=true) | \bool broadcast=true | Set the socket in broadcasting mode. | |
| | | \return 0 on success, -1 on failure. | |
| int sendTo( Endpoint &remote, char *packet, int length) | \remote The remote endpoint | Send a packet to a remote endpoint. | |
| | \packet The packet to be sent | \return the number of written bytes on success (>=0) or -1 on failure | |
| | \length The length of the packet to be sent | | |
| int receiveFrom (Endpoint &remote, char *buffer, int length) | \remote The remote endpoint | Receive a packet from a remote endpoint. | |
| | \buffer The buffer for storing the incoming packet data. If a packet is too long to fit in the supplied buffer, excess bytes are discarded | \return the number of received bytes on success (>=0) or -1 on failure | |
| | \length The length of the buffer | | |
| void set_blocking(bool blocking, unsigned int timeout=1500) | \blocking true for blocking mode, false for non-blocking mode. | Set blocking or non-blocking mode of the socket and a timeout on blocking socket operations. | |
| | \timeout timeout in ms [Default: (1500)ms]. | | |

| | | | |
|---|---|---|---|
| int close(bool shutdown=true) | \shutdown free the left-over data in message queues | Close the socket.<br>\return 0 on success, -1 on failure | |
| Endpoint | IP Endpoint(address, port) | | |
| void reset_address(void) | No input parameter | Reset the address of this endpoint. | |
| int set_address(const char *host, const int port) | \host The endpoint address (it can either be an IP Address or a hostname that will be resolved with DNS).<br>\port The endpoint port | Set the address of this endpoint.<br>\return 0 on success, -1 on failure (when a hostname cannot be resolved by DNS). | |
| char* get_address(void) | No input parameter | Get the IP address of this endpoint.<br>\return The IP address of this endpoint. | |
| int get_port(void) | No input parameter | Get the port of this endpoint.<br>\return The port of this endpoint | |