Wi-Fi Guide

Cross-Domain Development Kit XDK110
Platform for Application Development

Bosch Connected Devices and Solutions

BOSCH
Invented for life

XDK110: Guide Wi-Fi

Document revision	2.0
Document release date	17.08.17
Workbench version	3.0.0
Document number	BCDS-XDK110-GUIDE-WIFI
Technical reference code(s)	
Notes	Data in this document is subject to change without notice. Product photos and pictures are for illustration purposes only and may differ from the real product's appearance. **Subject to change without notice**

XDK Wi-Fi Guide

PLATFORM FOR APPLICATION DEVELOPMENT

The Wi-Fi API provides several interfaces to manage the Wi-Fi functionality on the XDK. XDK applications can implement this API to communicate over the Simplelink stack with surrounding Wi-Fi networks. All network information such as IP-address, connection state, gateway or subnet mask are available. There are API features for scanning and initiating several connections.

Table of Contents

1. API OVERVIEW 3
1.1 XDK WI-FI API 3

2. API USAGE..5
2.1 FREERTOS - XDK BASICS 5
2.2 INITIALIZING WI-FI DRIVER AND WI-FI STACK...6
2.3 SCANNING FOR WI-FI-NETWORKS...6
2.4 IP SETTINGS 7
2.4.1 STATIC IP SETTINGS 9
2.4.2 DHCP WITHOUT CALLBACK 10
2.4.3 DHCP WITH CALLBACK 10
2.5 GET IP SETTINGS...12
2.6 CONNECTING TO OPEN ACCESS POINT 13
2.7 CONNECTING TO AN ACCESS POINT WITH WPA 14
2.8 DISCONNECTING..14

3. DOCUMENT HISTORY AND MODIFICATION..15

This guide postulates a basic understanding of the XDK and according Workspace. For new users we recommend going through the following guides at xdk.io/guides first:

- *Workbench Installation*
- *Workbench First Steps*

© Bosch Connected Devices and Solutions GmbH reserves all rights even in the event of industrial property rights. We reserve all rights of disposal such as copying and passing on to third parties. BOSCH and the symbol are registered trademarks of Robert Bosch GmbH, Germany.
Note: Specifications within this document are subject to change without notice.

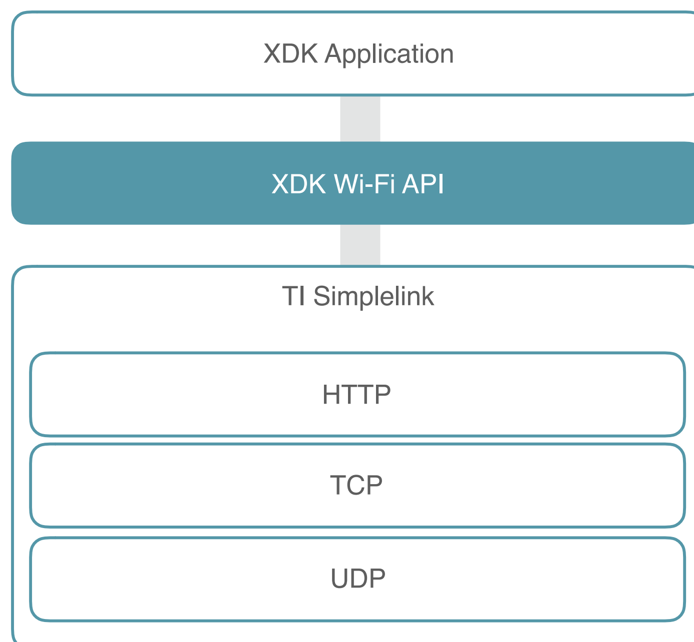
1. API Overview

1.1 XDK Wi-Fi API

In general, it is recommended to develop an application based on the highest API level the XDK framework supports. If this functionality isn't enough for the distinct purpose it is possible to access deeper API levels.

The XDK Wi-Fi API is a part of the XDK platform section and accesses the Texas Instruments Simplelink Library that is part of the library section. The Simplelink library provides a low level access to the CC3100 Wi-Fi chip of the XDK-Hardware.

Picture 1. API Hierarchy



The API is mainly built of two interfaces. To access the API, it is required to include these files at the top of the implementation file:

Code 1: Including required Wi-Fi Interfaces

```
#include <stdio.h> // If not already included in system header files

#include "BCDS_WlanConnect.h"
#include "BCDS_NetworkConfig.h"
```


© Bosch Connected Devices and Solutions GmbH reserves all rights even in the event of industrial property rights. We reserve all rights of disposal such as copying and passing on to third parties. BOSCH and the symbol are registered trademarks of Robert Bosch GmbH, Germany.
Note: Specifications within this document are subject to change without notice.

Table 1. Wi-Fi API Interfaces

Interface	Description
`BCDS_WlanConnect.h`	Wi-Fi connection interface
`BCDS_NetworkConfig.h`	Wi-Fi configuration interface

The `BCDS_WlanConnect.h` interface provides several functions to make the Wi-Fi Hardware available and to do things like searching for networks or initiating a connection to them. In general, this interface provides the connection functions of the API.

Table 2. Wi-Fi API connection functions

Function	Description
`WlanConnect_Init()`	Initialize Wi-Fi driver and Wi-Fi stack. This function must be called before calling any other Wi-Fi API function
`WlanConnect_DeInit()`	Deinitialize the Wi-Fi driver and the Wi-Fi stack
`WlanConnect_ScanNetworks()`	Scan for surrounded Wi-Fi networks
`WlanConnect_Open()`	Connect to a network without security permissions
`WlanConnect_WPA()`	Connect to a network with a WPA passphrase
`WlanConnect_Disconnect()`	Disconnect the current network connection
`WlanConnect_CurrentNwStatus_T()`	Delivers the current status of the Wi-Fi network connection

The `BCDS_NetworkConfig.h` interface provides basic functions to make general connection settings and to get informations about the current network connection.

Table 3. Wi-Fi API configuration functions

Function	Description
`NetworkConfig_Ipv4Value()`	Convert a given IP in a hexadecimal state
`NetworkConfig_Ipv4Byte()`	Extract a byte from a acquired IP address
`NetworkConfig_SetIpStatic()`	Set the configuration to a static IP
`NetworkConfig_SetIpDhcp()`	Set the configuration to dynamic IP by using DHCP
`NetworkConfig_GetIpSettings()`	Handle Wi-Fi IP settings. Return a structure that includes the connection information

© Bosch Connected Devices and Solutions GmbH reserves all rights even in the event of industrial property rights. We reserve all rights of disposal such as copying and passing on to third parties. BOSCH and the symbol are registered trademarks of Robert Bosch GmbH, Germany.
Note: Specifications within this document are subject to change without notice.

2. API Usage

First of all there will be a small introduction to the SSID network parameter. This is necessary in order to understand how scanning for Wi-Fi networks and connections to Wi-Fi networks works.

SSID
The SSID (Service Set Identifier) describes the name of a Wi-Fi network. It is a case-sensitive string with a byte length up to 32 bytes. Devices are able to identify different access points (AP) after boot up and establish a Wi-Fi connection to them. Similar to this the SSID can be used to search for a specific network.

2.1 FreeRTOS - XDK Basics

This section describes the basic operation how to execute a part of code with the freeRTOS operating system. Therefore operating tasks are used for the execution of the wifi code examples. It is to mention that there are timer tasks too, but this section will only give a short introduction to operating tasks. The reason for this is, some operating function calls require the task to be an operating task and cannot be called in timer tasks. The following code describes how to create and start the necessary operating task.

Note: `appInitSystem()` need to be placed a the end of the implementation file.

Code 2. Create operating task

```
void appInitSystem(void * CmdProcessorHandle, uint32_t param2)
{
    if (CmdProcessorHandle == NULL) {
        printf("Command processor handle is null \n\r");
        assert(false);
    }
    BCDS_UNUSED(param2);

    int taskStackDepth = 768;
    int taskPriority = 1;
    // Create task
    if (pdPASS
        != xTaskCreate(wifiApplication,
        (const char * const) "wifiApplication",
        taskStackDepth, NULL, taskPriority,0))
    {
    /* Assertion due to: SW timer was not started, due to insufficient heap
```

The `appInitSystem()` is used as recommended function to create the task and only the body should be modified. The operating task is created and executed in the if condition. This includes a request if enough memory is reserved for the task. If this is not the case an assertion is triggered.
`taskStackDepth` and `taskPriority` are necessary parameters and have to be set manually for the initialization. `taskStackDepth` is the variable for the stack size of the task and has to be valued sufficient. `taskPriority` is necessary for the scheduler that determines the task execution order. It is recommended that only advanced XDK users change these parameters. Additionally the name of the function that should be called by the operating task has to be passed. The structure of the called function has to be the same as in code 3.

© Bosch Connected Devices and Solutions GmbH reserves all rights even in the event of industrial property rights. We reserve all rights of disposal such as copying and passing on to third parties. BOSCH and the symbol are registered trademarks of Robert Bosch GmbH, Germany.
Note: Specifications within this document are subject to change without notice.

Code 3. Operating Task Wi-Fi function implementation

```
static void wifiApplication(void* handle)
{
    (void) handle;
    // Wi-Fi implementation here
    vTaskDelete( NULL );
}
```

Note: The operating task will execute the code inside the wifiApplication function once. If it is necessary to execute a function more than once with an operating task a infinite loop has to be programmed manually.
For more information, please check the freeRTOS guide about operating and timer tasks.

2.2 Initializing Wi-Fi Driver and Wi-Fi Stack

Before scanning for or connection to Wi-Fi networks can be done, it is required to make an initialization of the Wi-Fi driver and the Wi-Fi stack.
The code block 4 below shows the call of the according Wi-Fi initialization function `WlanConnect_Init()`. It is required to call this function before any other function of the Wi-Fi API can be called.

Code 4. Initialization Wi-Fi driver and Wi-Fi stack

```
WlanConnect_Init();
```

2.3 Scanning for Wi-Fi-networks

If the Wi-Fi is initialized, it is possible to scan for surrounding Wi-Fi networks before connecting to them.
The following code block 5 describes how to implement the function body to scan for surrounding Wi-Fi networks. An explanation of the required variables is shown in table 4.

Table 4. WlanConnect scanning types

Variable	Description
`Retcode_T`	Return state for programable request logics
`WlanConnect_ScanInterval_T`	Defined time period in which intervals the scan occurs
`WlanConnect_ScanList_T`	A struct which contains the scanned network informations

© Bosch Connected Devices and Solutions GmbH reserves all rights even in the event of industrial property rights. We reserve all rights of disposal such as copying and passing on to third parties. BOSCH and the symbol are registered trademarks of Robert Bosch GmbH, Germany.
Note: Specifications within this document are subject to change without notice.

Code 5. Scan for surrounding Wi-Fi networks

```
Retcode_T retScanStatus;
WlanConnect_ScanInterval_T scanInterval = 5;
WlanConnect_ScanList_T scanList;
portTickType delay500msec = 500;

retScanStatus = WlanConnect_ScanNetworks(scanInterval, &scanList);

if (retScanStatus == RETCODE_OK) {
    // Print SSID networks
    for (int i = 0; i < scanList.NumOfScanEntries; i++) {
        if (0 != scanList.ScanData[i].SsidLength) {
            printf("Found SSID number %d is : %s\n\r", i,
                    scanList.ScanData[i].Ssid);

            vTaskDelay( delay500msec / portTICK_RATE_MS);
        }
    }
}
else if (retScanStatus == RETCODE_NO_NW_AVAILABLE) { /* No networks found */ }
else {
    // Scan failed
}
```

The `WlanConnect_ScanNetworks()` function scans for surrounding networks in the committed time intervals. Results of the scan are saved in a scanList variable. The return state retScanStatus provides the result of the scan request. There are three listed request states used for the Wi-Fi network scan: `RETCODE_OK`, `RETCODE_NO_NW_AVAILABLE` and an else case if the scan failed.

2.4 IP Settings

As before a small introduction to the SSID, a short overview of the basic IP setting parameters follow. These are necessary for understanding a manual IP setup. If the IP is set to static, it is important to adapt the parameters to the according access point configuration.

DNS
Stands for "Domain Name System." Domain names serve as memorizable names for websites and other services on the Internet. However, computers access Internet devices by their IP addresses. DNS translates domain names into IP addresses, allowing you to access an Internet location by its domain name.

Gateway
A gateway is a hardware device that acts as a "gate" between two networks. It may be a router, firewall, server, or other device that enables traffic to flow in and out of the network.
While a gateway protects the nodes within network, it also a node itself. The gateway node is considered to be on the "edge" of the network as all data must flow through it before coming in or going out of the network. It may also translate data received from outside networks into a format or protocol recognized by devices within the internal network.
A router is a common type of gateway used in home networks.

© Bosch Connected Devices and Solutions GmbH reserves all rights even in the event of industrial property rights. We reserve all rights of disposal such as copying and passing on to third parties. BOSCH and the symbol are registered trademarks of Robert Bosch GmbH, Germany.
Note: Specifications within this document are subject to change without notice.

Subnet mask
A subnet mask is a number that defines a range of IP addresses that can be used in a network. (It is not something you wear on your face to keep subnets out.) Subnet masks are used to designate subnetworks, or subnets, which are typically local networks LANs that are connected to the Internet. Systems within the same subnet can communicate directly with each other, while systems on different subnets must communicate through a router. Therefore, subnetworks can be used to partition multiple networks and limit the traffic between them.

DHCP
The Dynamic Host Configuration Protocol (DHCP) is a standardized network protocol for dynamically distribution network configuration parameters like IP addresses. With the XDK it is possible to implement a connection based on DHCP too. There are two different DHCP modes, with and without a callback.
It is possible to turn DHCP off and set the network parameters manually.

The table below shows the possible IP modes.

Table 5. IP Setting modes

IP modes
static IP application
DHCP IP application with callback
DHCP IP application without callback

Every IP mode will be explained to give the user all the information to setup a mode for his own purpose.

© Bosch Connected Devices and Solutions GmbH reserves all rights even in the event of industrial property rights. We reserve all rights of disposal such as copying and passing on to third parties. BOSCH and the symbol are registered trademarks of Robert Bosch GmbH, Germany.
Note: Specifications within this document are subject to change without notice.

2.4.1 Static IP settings

The manual implementation of the IP settings is the configuration below with static network parameters. This configuration allows the setup of the network parameters like DNS, Gateway or Subnet Mask. Therefore it is necessary to disable the introduced DHCP and set the network parameters manually in the code. The following table describes which network parameters are needed to set the static IP.

Table 6. NetworkConfig_IpSettings_T IPv4 parameters

Variable	Description
`NetworkConfig_IpSettings_T`	A struct that contains the parameters for a static configuration
Struct Variable	**Description**
`ipV4`	32 Bit Internet Protocol Version 4 Address
`ipV4DnsServer`	32 Bit Address of the local DNS Server
`ipV4Gateway`	32 Bit Address of the local Router/Gateway
`ipV4Mask`	32 Bit Subnetmask of the local Network

The following code block 6 shows an example how the network parameters can be set for a static IP configuration. The `NetworkConfig_Ipv4Value()` function converts the committed decimal IP parameters in hexadecimal IP parameters.

Code 6. Set IPV4 parameters

```
NetworkConfig_IpSettings_T myIpSet;

myIpSet.isDHCP = (uint8_t) NETWORKCONFIG_DHCP_DISABLED;
myIpSet.ipV4 = NetworkConfig_Ipv4Value(192, 168, 0, 2);
myIpSet.ipV4DnsServer = NetworkConfig_Ipv4Value(192, 168, 0, 1);
myIpSet.ipV4Gateway = NetworkConfig_Ipv4Value(192, 168, 0, 1);
myIpSet.ipV4Mask = NetworkConfig_Ipv4Value(255, 255, 255, 0);
```

The code block 7 shows the implementation of the static IP setup. A request state `retStatusSetIp` is returned and it can be determined if the IP setup was configured successful or not. The `NetworkConfig_SetIpStatic()` function set the committed IP parameters in the interface.

© Bosch Connected Devices and Solutions GmbH reserves all rights even in the event of industrial property rights. We reserve all rights of disposal such as copying and passing on to third parties. BOSCH and the symbol are registered trademarks of Robert Bosch GmbH, Germany.
Note: Specifications within this document are subject to change without notice.

Code 7. Set up static IP

```
Retcode_T retStatusSetIp;
retStatusSetIp = NetworkConfig_SetIpStatic(myIpSet);

if (retStatusSetIp == RETCODE_OK) {
    // Set IP succeed
}
else {
    // Set IP static failed
}
```

2.4.2 DHCP without callback

A dynamic implementation of the IP setup can be done with a IP setup by using DHCP without a callback. In opposition to the static IP setup the DHCP setup only requires a function call to acquire the IP parameters.
The code below shows the DHCP implementation and uses similar to the static IP setup a request state `retStatusSetIp` if the IP setup was successfully configured. Similar to the static IP configuration the `NetworkConfig_SetIpDhcp()` function provides the dynamic setting of the IP parameters.

Code 8. Set up dynamic IP

```
// Set the DHCP without callback
Retcode_T retStatusSetIp;
retStatusSetIp = NetworkConfig_SetIpDhcp(0);

if (retStatusSetIp == RETCODE_OK) {
    // Waiting for IP
}
else {
   // Setting Ip over DHCP failed
}
```

2.4.3 DHCP with callback

Sometimes it is useful to execute a piece of code directly after setting the IP parameters. Therefore this implementation provides a dynamic IP setup by using DHCP with callback. This implementation provides a callback if the dynamic IP setup was configured or not. An if condition is for this case not necessary. To implement an IP setup by using DHCP with callback a callback function is required. A callback is simply a reference of a function which is given to an interface and will be called later. The code block 9 shows an implementation of a callback function. A simple request including the `NETWORKCONFIG_IPV4_ACQUIRED` enumeration defines if the IP was acquired successfully.

© Bosch Connected Devices and Solutions GmbH reserves all rights even in the event of industrial property rights. We reserve all rights of disposal such as copying and passing on to third parties. BOSCH and the symbol are registered trademarks of Robert Bosch GmbH, Germany.
Note: Specifications within this document are subject to change without notice.

Code 9. Set up the callback function

```
void myDhcpIpCallbackFunc(NetworkConfig_IpStatus_T returnStatus) {

    if ( returnStatus == NETWORKCONFIG_IPV4_ACQUIRED) {
        printf("Callback Function : IP was acquired using DHCP\n\r");
    }
    else {
        // DHCP Request failed
    }
}
```

The following code block 10 shows the implementation of the DHCP with callback setup. For this example the variable in the table 7 is needed.

Table 7. Dynamic DHCP callback types

Variable	Description
`NetworkConfig_IpCallback_T`	A variable that is used for the IP callback

The dynamic IP setup below needs to be configured as before with the call of the `NetworkConfig_SetIpDhcp()` function. Similar to the other examples there is a request state `retStatusSetIP` if the setup execution was successful. Note that this additional request isn't necessary. The request status can be checked into the callback.

Code 10. Set up dynamic IP with callback

```
Retcode_T retStatusSetIp;
NetworkConfig_IpCallback_T myIpCallback;
// Set the IP callback
myIpCallback = myDhcpIpCallbackFunc;

// Set the DHCP with callback
retStatusSetIp = NetworkConfig_SetIpDhcp(myIpCallback);

if (retStatusSetIp == RETCODE_OK) {
    // Waiting for IP
}
else {
    // Set DHCP with Callback failed
}
```

Note: If no IP settings are made, the Wi-Fi API automatically uses the dynamic IP setup using DHCP without callback.

© Bosch Connected Devices and Solutions GmbH reserves all rights even in the event of industrial property rights. We reserve all rights of disposal such as copying and passing on to third parties. BOSCH and the symbol are registered trademarks of Robert Bosch GmbH, Germany.
Note: Specifications within this document are subject to change without notice.

2.5 Get IP Settings

The previous section described how to setup the network IP parameters. This section describes how to access the acquired IP. To do so the following code block 11 implements the access of the acquired IP and prints it to the console.

Code 11. Get ip settings

```
NetworkConfig_IpSettings_T myIpGet;
Retcode_T retStatusGetIp;

retStatusGetIp = NetworkConfig_GetIpSettings(&myIpGet);

if (retStatusGetIp == RETCODE_OK) {
    printf("The static IP was retrieved : %u.%u.%u.%u \n\r",
             (unsigned int) (NetworkConfig_Ipv4Byte(myIpGet.ipV4, 3)),
             (unsigned int) (NetworkConfig_Ipv4Byte(myIpGet.ipV4, 2)),
             (unsigned int) (NetworkConfig_Ipv4Byte(myIpGet.ipV4, 1)),
             (unsigned int) (NetworkConfig_Ipv4Byte(myIpGet.ipV4, 0)));
}
else {
    // Get IP settings failed
}
```

Therefore the `NetworkConfig_IpSettings_T` struct is needed to save the acquired IP. Next, the `NetworkConfig_GetIpSettings()` function accesses the IP settings and saves the acquired IP in the `myIPGet` variable. Same as before the `retStatusGetIp` return state delivers a result if the access of the acquired IP was successful. The `NetworkConfig_Ipv4Byte()` function extracts a byte from the acquired IP address in the `myIPGet` variable. This function call can be used for example to print the acquired IP to the console.

© Bosch Connected Devices and Solutions GmbH reserves all rights even in the event of industrial property rights. We reserve all rights of disposal such as copying and passing on to third parties. BOSCH and the symbol are registered trademarks of Robert Bosch GmbH, Germany.
Note: Specifications within this document are subject to change without notice.

2.6 Connecting to open Access Point

An open Access Point is a Wi-Fi network without any security permissions and therefore no authentication is necessary. If the Wi-Fi stack is initizialized (code block 4) the IP settings can be done and a connection can be established. The previous codes showed how to configure the necessary networks IP settings and access the acquired IP. The following code shows how to connect to an open Wi-Fi network. Therefore the variables of table 9 below are needed.

Table 9. Wi-Fi network parameters

Variable	Description
`WlanConnect_SSID_T`	Contains the SSID of the Wi-Fi network
`WlanConnect_PassPhrase_T`	Contains the password of the Wi-Fi network

The code block 12 shows how to connect to a Wi-Fi network using the `WlanConnect_Open()` function. The committed parameters are the network SSID and a connection callback, if needed. In this particular the example the function was called without the connection callback. If a callback is needed it can be implemented like the DCHP with callback code block 8 and 9. Similar to the other codes a request state `retStatusConnect` is implemented if the connection was successfully established.

Code 12. Connect to an open Wi-Fi network

```
Retcode_T retStatusConnect;
WlanConnect_SSID_T connectSSID = "networkSSID";

retStatusConnect = (Retcode_T) WlanConnect_Open(connectSSID, 0);
if (retStatusConnect == RETCODE_OK) {
    printf("Connected successfully.\n\r");
}
else {
    // Connection failed
}
```


© Bosch Connected Devices and Solutions GmbH reserves all rights even in the event of industrial property rights. We reserve all rights of disposal such as copying and passing on to third parties. BOSCH and the symbol are registered trademarks of Robert Bosch GmbH, Germany.
Note: Specifications within this document are subject to change without notice.

2.7 Connecting to an Access Point with WPA

WPA stands for Wi-Fi Protected Access. WPA is a security protocol designed to create secure wireless (Wi-Fi) networks. It is similar to the WEP protocol, but offers improvements in the way it handles security keys and the way users are authorized.
Similar to the implementation with an open Access Point the network IP settings have to be configured to establish a connection. A password is also needed to connect to this network. This means both variables from table 9 are required. The following code block 13 shows the implementation how to connect to a WPA Wi-Fi network. The difference to the open Access Point configuration is the password which is committed to the `WlanConnect_WPA()` function. Similar to the implementation of an open Access Point the function also contains the option to use a callback if the connection is established. The example similarly includes a request state if the connection was successful.

Code 13. Connect to a Wi-Fi WPA network

```
WlanConnect_SSID_T connectSSID = "networkSSID";
WlanConnect_PassPhrase_T connectPassPhrase = "networkPassword";
Retcode_T retStatusConnect;

retStatusConnect = (Retcode_T) WlanConnect_WPA(connectSSID,
             connectPassPhrase,0);
if (retStatusConnect == RETCODE_OK) {
    printf("Connected successfully.\n\r");
}
else {
    // Connection failed
}
```

2.8 Disconnecting

Finally if a connection should be closed, code 14 shows an implementation how to disconnect from a connected Wi-Fi network. Similar to the codes before a request state `retStatusDisconnect` is used if the connection was successfully terminated.
The `WlanConnect_Disconnect()` function call provides with a committed zero statement the disconnection from a connected Wi-Fi network.

Code 14. Disconnect from a Wi-Fi network

```
Retcode_T retStatusDisconnect;

retStatusDisconnect = (Retcode_T) WlanConnect_Disconnect(0);

if (retStatusDisconnect == RETCODE_OK) {
    printf("Disconnection successfully.\n\r");
}
else {
    // Disconnection failed
}
```


© Bosch Connected Devices and Solutions GmbH reserves all rights even in the event of industrial property rights. We reserve all rights of disposal such as copying and passing on to third parties. BOSCH and the symbol are registered trademarks of Robert Bosch GmbH, Germany.
Note: Specifications within this document are subject to change without notice.

3. Document History and Modification

Rev. No.	Chapter	Description of modification/changes	Editor	Date
2.0		Version 2.0 initial release	AFS	2017-08-17

© Bosch Connected Devices and Solutions GmbH reserves all rights even in the event of industrial property rights. We reserve all rights of disposal such as copying and passing on to third parties. BOSCH and the symbol are registered trademarks of Robert Bosch GmbH, Germany.
Note: Specifications within this document are subject to change without notice.