SimpleLink Provisioning Mobile App Application Note

User Guide

Revision 0.2

Table of Contents

Table of Figures

Terms & Abbreviations

Abbreviation /Term	Meaning / Explanation
HTTP	Hypertext Transfer Protocol
SC	SmartConfig™
mDNS	Multicast DNS
UDP	User Datagram Protocol
Bcast	Broadcast
JSON	Javascript Object Notation

1. Introduction

1.1. Overview

This document describes the Provisioning TI's SimpleLink™ Wi-Fi® device and how to use it.
The main focus of the document is to describe the usage of the Android/iOS building blocks for UI requirements, networking and provisioning APIs required for building the mobile application.

A first step in utilizing CC3xxx in a Wi-Fi enabled application ("Wi-Fi Starter Pro") is to configure CC3xxx to a user's Wi-Fi network. This requires information on the AP, or SSID name, and the security passcode when WEP/WPA/WPA2 is enabled. Considering that embedded Wi-Fi applications will generally lack user interfaces such as keypads or touchscreens, this process can be complex without the use of advanced I/O.

SmartConfig™ leverages the standard mechanisms present in Wi-Fi to configure a CC3xxx's association information on the fly, regardless of whether user-interface is available. In this process a Wi-Fi enabled device such as a smartphone, tablet or a laptop is used to send the association information to the CC3xxx.

Additionally, SmartConfig does not depend on the host microcontroller's I/O capabilities, thereby usable by deeply embedded applications. It can be used to associate multiple devices to the same AP simultaneously. And furthermore, the device used to configure (smartphone or tablet) stays connected to the user's home network during the configuration process (as opposed to other methods that require disconnection).

Figure 1 describes this procedure in general – a device which is not connected to the network but exist on the network range, will be able to accept the information which is required for connecting to the network, using the combination of the embedded application (which is described in another document) and the mobile application which is going to be introduced in this document.

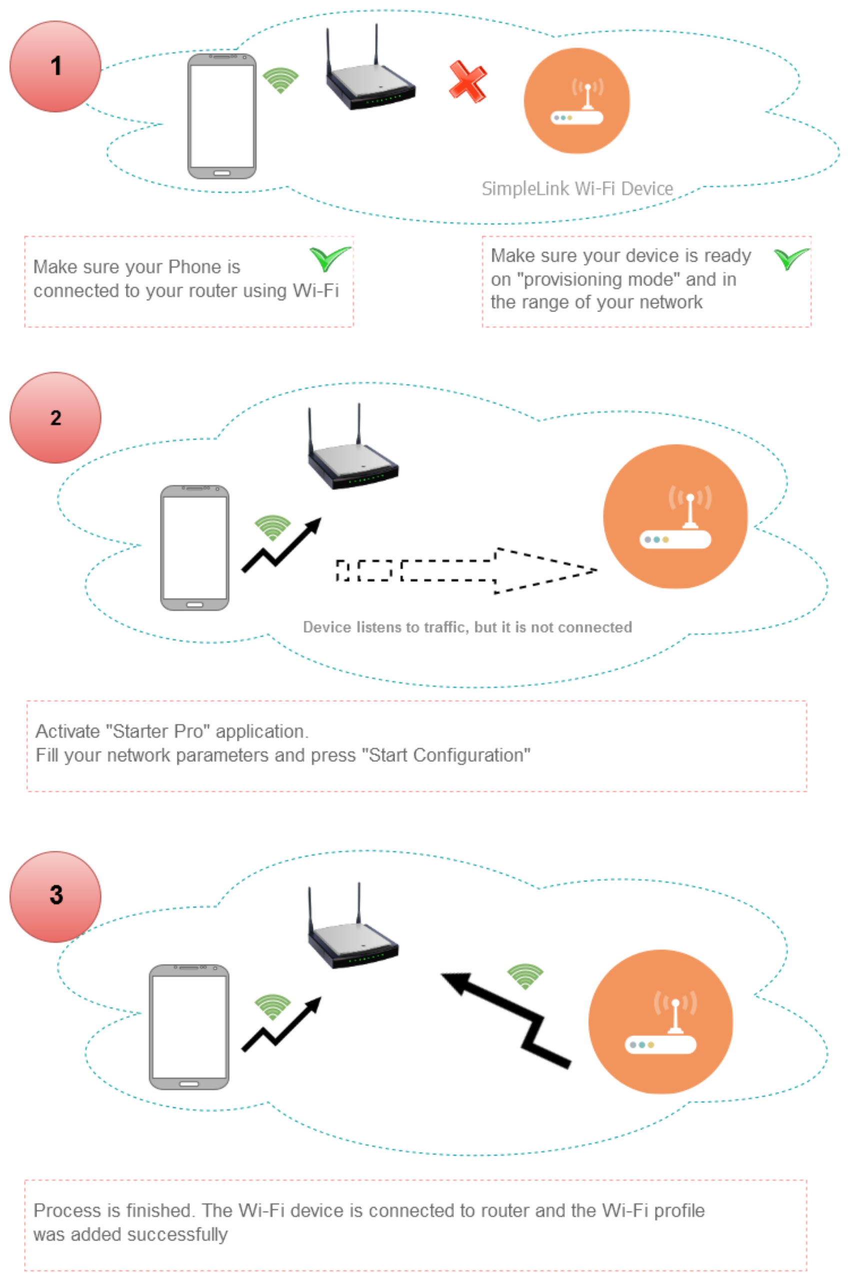
1
SimpleLink Wi-Fi Device
Make sure your Phone is connected to your router using Wi-Fi
Make sure your device is ready on "provisioning mode" and in the range of your network
2
Device listens to traffic, but it is not connected
Activate "Starter Pro" application.
Fill your network parameters and press "Start Configuration"
3
Process is finished. The Wi-Fi device is connected to router and the Wi-Fi profile was added successfully

Figure 1 – Overview

1.2. Provisioning Mode

Wi-Fi Provisioning is usually done once, while connecting a new device to the network or in case of a changes on the local network which requires configurations changes. Following the completeness of the provisioning steps, CC3xxx device saves the accepted Wi-Fi information as an encrypted profile. This profile connects automatically to the network when it is available, while the device is activated as Wi-Fi as a station role.

“AP mode” provisioning method connects to the device as a Wi-Fi Station and sends the configurations. By using this mode the user should know which device to connect, by its published SSID while acting on AP role.

2. Top Level Architecture

2.1. Top Level blocks

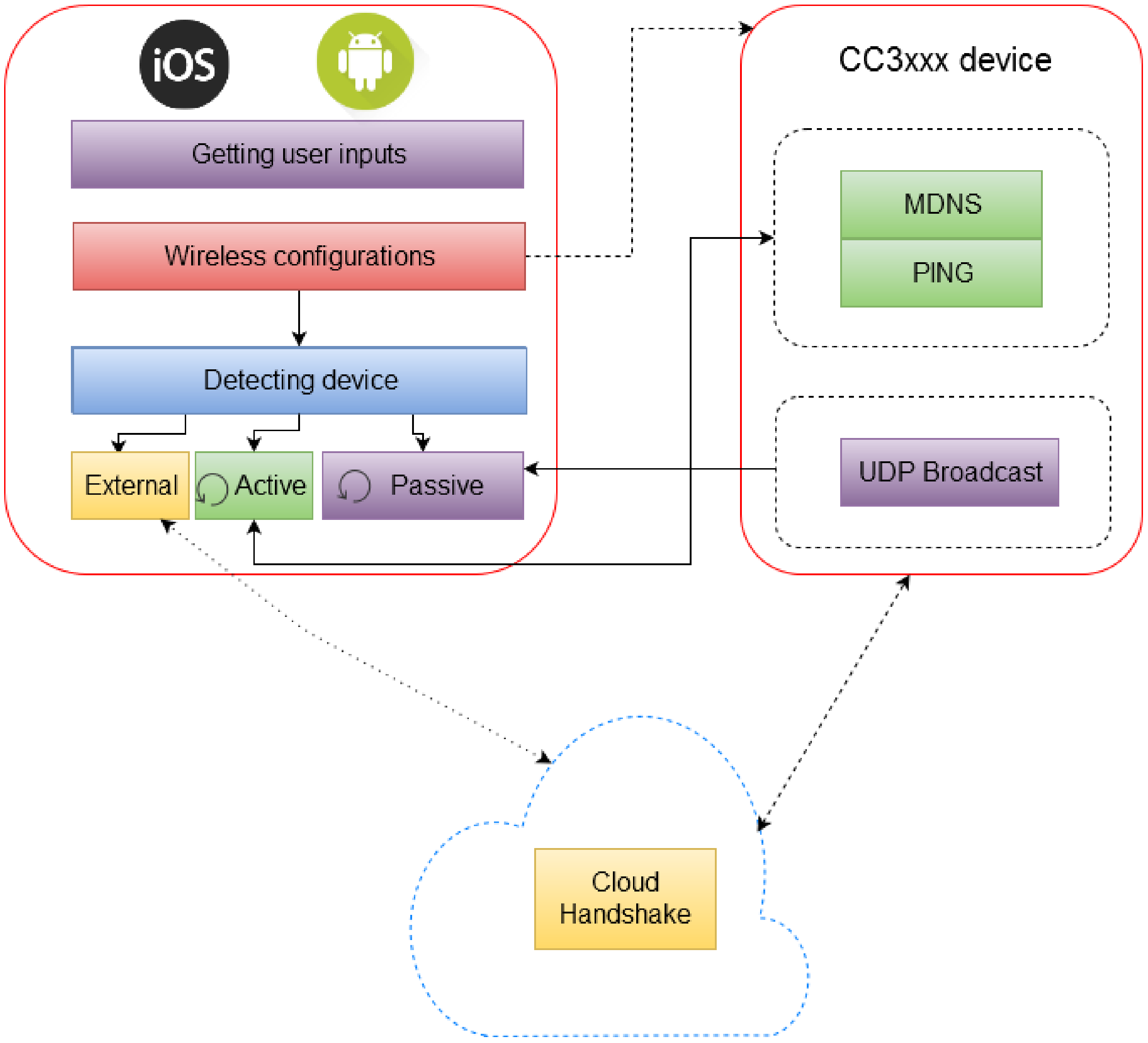

Figure 2 – Top level architecture

2.2. Provisioning steps

There are 4 steps for completing provisioning:

- **User Inputs –** Use inputs from the user about the network.
- **Sending configurations** – Mobile is sending, device is receiving.
- **Finding the device on the network** – Mobile is searching the device; device is publishing information and responses to network queries.
- **Connecting to the device and getting** feedback – Mobile connects to device, over the shared network, device is responding.

2.3. Step 1: User inputs

This is the first step of using the application. The Mobile phone should be connected to the network using Wi-Fi as a starting point, before activating provisioning.

While starting the application, the network data should be filled so that the mobile device will be able to transmit it to the network.
The mandatory fields on both options is the network SSID (which is usually the active Wi-Fi connection of the mobile phone) and network password, in case of using a secured network.

Device name, if not set, will keep the default device name as it was before.

2.4. Step 2: Sending configurations

The second phase is to send the information to the device, using the active network though the device is not a part of this network, yet.

In this case the mobile side is responsible for transmitting the information to the network and the device listens to the information transmitted (even if it is a secured network) and after verifying the correctness of the information will create and store a profile of this network.
The profile is now activated and connects to the network.

2.5. Step 3: Finding the device on the network

The third phase is to find device's IP address, after it is connected to the Wi-Fi network and acquired an IP address from the DHCP server.
The mobile is going to use 3 options for detecting the device on the network:

- Listening to UDP broadcast packets from the device specifying the name and ip address
- Listening UDP multicast packets from mDNS on the network, and filtering by services supported by the device
- Sending broadcasts ping packets and catching ping response packets from the devices on the network.

In case the network is not totally isolated, the device is going to be detected and verified by the mobile application in one or more of the 3 options described.

2.6. Step 4: Connecting to the device and getting feedback

The final step is to check if the device completed provisioning successfully. This is done by sending a query from the Mobile to the device, asking for the provisioning results. This step is performed assuming we already know the device IP address from previous step, and the device is supporting HTTP requests.
There are several response options. Upon successful response the provisioning is completed. In case there is a timeout or a failure, AP fallback mode is suggested to the user by the Mobile application.

2.7. Fallback step: Confirmation failed

In case the mobile side failed to connect to the device and get a feedback, the mobile should connect to the device directly and the device will switch to the configurations stage as usual. In this case, the device was configured but the confirmation feedback will be transferred using the direct connection.

3. Provisioning – AP Mode

3.1. Overview

AP mode is the process of using device AP role for configurations. After selecting the desired AP for configurations, all settings (ssid,password,device name,uuid) are sent by HTTP protocol. When the configurations are ready, the device will switch to STA role and will use the profile (ssid + password) to connect to the local network. At this stage, the mobile applications will connect to the same AP as well and will search for the SimpleLink, using one or more of the 3 methods: Ping, mDNS and Udp brodcasts.

In case there is a problem finding SimpleLink device, the Device is going to change back his role to AP. The Mobile side will connect to the SimpleLink device, again, but this time to close the loop and to fetch the provisioning results.

3.2. AP mode Flow Chart

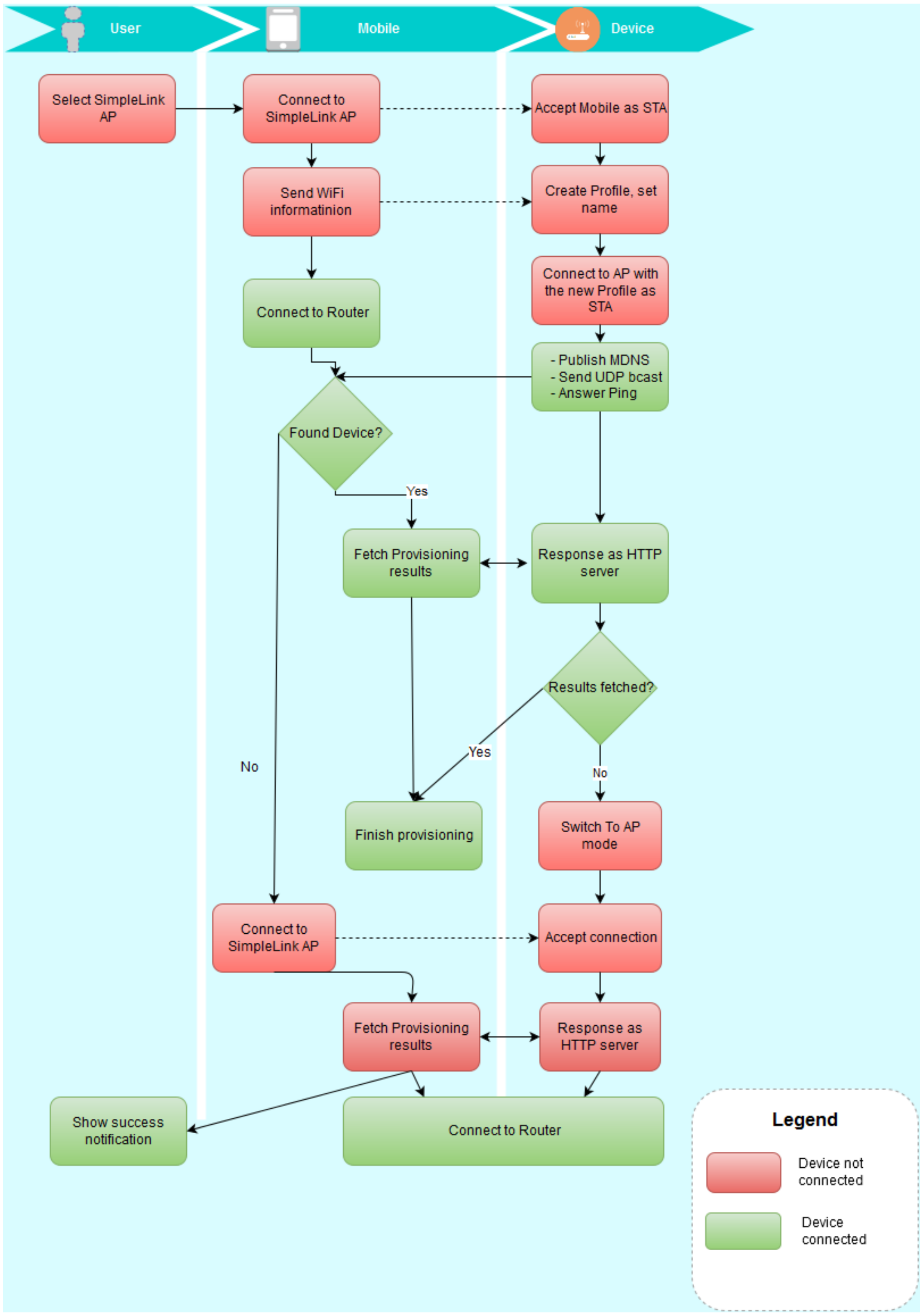

Figure 3 – Flow chart

3.3. Block Diagram

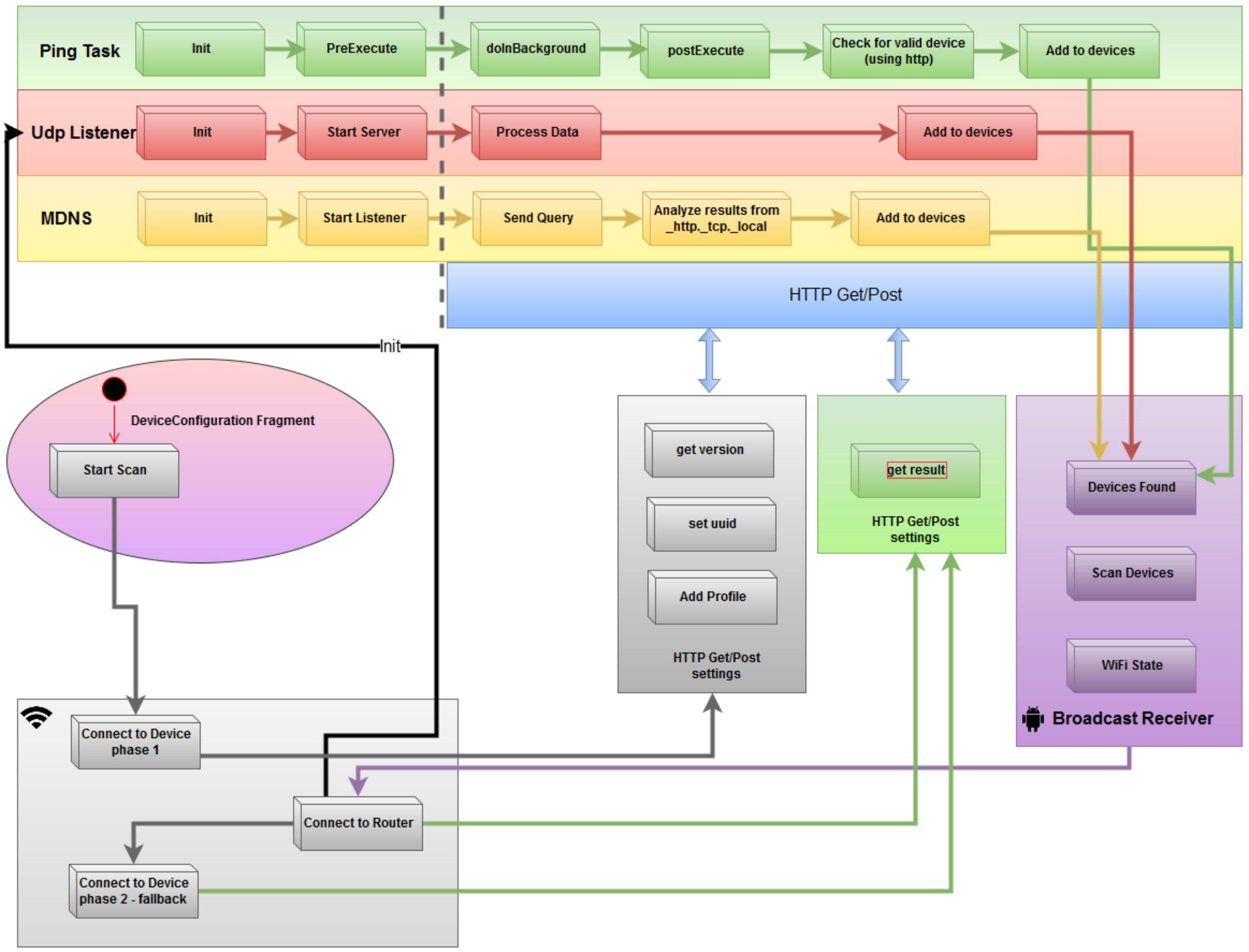

Figure 4 – Android block diagram

3.4. UDP listener

UDP listener is a UDP Server, running on a specific thread (Android UI thread or iOS thread).
It should be activated as a background task so that it could find UDP broadcast from the device, upon acquiring IP address from the network.

The Device should send a few messages, with delays between them to publish its ip address and name.
The data is published using port number “1501” and is contains device name and device ip address, comma separated on a textual format.

Upon successful receiving and parsing the data, UDP listener callback is activated and the device is added to the device containers, after it is converted to JSON format.
There is no use for UDP listener while the device is used as “AP” mode.

For example – starting UDP server on Android:

```
udpBcastServer = new UdpBcastServer(mCallback);
```

3.5. mDNS listener

mDNS listener assumes the device, using its default configurations, support mDNS & HTTP server.
In this case, upon acquiring an ip address, the device is going to announce it services by using multicast messages. The http is one of the services going to be published to the network.

Since http service contains some known information, we rely on this information for adding the device containing this specific information to our device list.

The txt fields on the mDNS packet we check should contains the string "srcvers=1D90645" for confirming the correctness of this device.
In case a valid SimpleLink device is detected using mDNS, we construct a JSON message with the device name and its ip address and add it to our device container.

On Android – mDNSHelper is handling mDNS initialization and callback function for accepting and parsing incoming mDNS information from the local network.
There is no use for mDNS listener while the device is used as “AP” mode.

3.6. Ping task

Ping task is a background task, which is activated after SmartConfig is done, to find devices on the local network and it is based on sending a broadcast ICMP ECHO request (ping request) from the device and waiting for ECHO reply (ping reply) from devices on the local network.

SimpleLink device is designed to reply to broadcast ping requests by default. Upon accepting response from the network devices, the mobile application will filter only SimpleLink devices by querying specific http request – getting the device version. In case we got a valid response, it will indicate this is a simple link device and we can add it to our devices container in a JSON format.

Ping class is handling ping generation and task activity of accepting ping response and creating an http request for validation the device name.

There is no use for mDNS listener while the device is used as “AP” mode.

3.7. HTTP requests for AP mode

Since HTTP queries are usually long and may take time to complete (depends on Host speed and interface, link quality) all HTTP actions are wrapper on an async thread and should not stall any UI activity.

3.7.1. Get Device Version

The HTTP call is based on “http://[ip address]/param_product_version.txt”
This specific API is good for both “R1” and for “R2” since it should return the current version.

After getting version information, it should be used on the next HTTP APIs as one of the input parameters

Device method:

```
public static DeviceVersion getSLVersion(String baseUrl)
```

3.7.2. Get Configurations Results

This API is called after the device is already connected to the local network and the reason it is activates are:

- Check the error code stored on the device for notifying the user about the success or failure of the provisioning.
- In case the device waits for 30 seconds and this request is not activated by the mobile side, it assumes the provisioning is uncompleted and it switches to AP role as fallback.

```
public static String getCGFResultFromDevice(String baseUrl, DeviceVersion version)
```

3.7.3. Get Response number code

The string fetched from previous API “getCGFResultFromDevice” should be converted to a number by using the string by the following conversion table:

String	Value
“5” or “4”	Success

"Unknown Token"	Unknown_token
“Timeout”	Timeout
“0”	Not_Started
“1”	Ap_not_found
“2”	Wrong_Password
“3”	Ip_add_failed
Any other string	Failure

For example:

```
CFG_Result_Enum result_Enum = NetworkUtil.cfgEnumForResponse(resultString)
```

3.7.4. Get Error Message from Number

This API converts from the number to a human readable string. This string is shown to the user as a result of the provisioning actions, upon finish.

```
public static String getErrorMsgForCFGResult(CFG_Result_Enum result)
```

Enum	**String**
Success	"Provisioning Successful"
Unknown_token	"CFG_Result_Enum: Unknown_Token";
Timeout	"CFG_Result_Enum: Time_Out"
Not_Started	"The provisioning sequence has not started yet. Device is waiting for configuration to be sent"
Ap_not_found	"Could not find the selected WiFi network; it is either turned off or out of range. When the WiFi network is available please restart the device in order to connect."
Wrong_Password	"Connection to selected AP has failed. Please try one of the following: Check your password entered correctly and try again Check your AP is working\nRestart your AP"
Ip_add_failed	"Failed to acquire IP address from the selected AP. Please try one of the following: Try connecting a new device to the WiFi AP to see if it is OK Restart the WiFi AP"

Failure	"Please try to restart the device and the configuration application and try again"

For example:

```
result = NetworkUtil.getErrorMsgForCFGResult(result_Enum);
```

3.7.5. Set IoT uuid

Send iotUUID string, up to 64 characters to the device. In the case device can handle uuid, it will send it to the iot server, if configured.
This is an optional setting and by default the UI is not going to show it.

```
public static Boolean setIotUuid(String newName, String baseUrl)
```

3.7.6. Get Device Name

This API reads the device name from the device.
Device name is presented on the UI, in case the user decides to get the default device name and not to set a device name.

```
public static String getDeviceName(String baseUrl, DeviceVersion version)
```

*It is recommended to set a device name since if some devices will response to mDNS, the mobile device cannot decide which one belongs to the active one, since no name was set

3.7.7. Set Device Name

Set the device URN name, in case the “Device Name” field is being used by the user

```
public static Boolean setNewDeviceName(String newName, String baseUrl,
DeviceVersion version)
```

AP mode requires additional HTTP APIs since the profile for the device is being transferred directly from the mobile side to the device using an existing Wi-Fi connection.

3.7.8. Get SSID List from Device

This API request AP's list from the device. The API will accept the list of scanned APs.
The query will not initiate a scan. It will fetch the latest APs from the device's list.
It will be activated while pressing "WiFi network" on AP mode.

```
public static ArrayList<String> getSSIDListFromDevice(String baseUrl, DeviceVersion
version
```

3.7.9. Rescan Networks on Device

API for setting scan interval settings.

```
public static Boolean rescanNetworksOnDevice(String url, DeviceVersion version)
```

3.7.10. Add Profile

This API is a direct profile activation API for storing an new profile. It accepts connection details like SSID, password and security type and sends the data to the device, using HTTP protocol.

```
public static Boolean addProfile(String baseUrl, SecurityType securityType, String ssid,
String password, String priorityString, DeviceVersion version)
```

3.7.11. Notify Device Profile is ready

Upon setting a new profile, the mobile side schedules an activation of this profile by using this API. It notifies the device that the mobile is ready for the next step, and the device should restart as Wi-Fi station mode for starting to use the profile that was set previously.

```
public static Boolean moveStateMachineAfterProfileAddition(String baseUrl, String ssid,
DeviceVersion version)
```

4. iOS vs Android development guidelines

iOS is different on the way it is activated and used in several points:

- iOS prevents Wi-Fi connection/disconnection from the application side. While using provisioning using AP mode requires connecting to Device or Mobile from the “Settings” menu and cannot be handled by the application code.
  In such cases, the application will notify the user what to do, and the user should perform the action, manually from the “Phone Settings” application.
- iOS prevents APIs for scanning Wi-Fi networks. In such cases, the application will notify the user to open “Phone Settings” and
- iOS has no API that may expose the security type of the connected device. In case of selecting an AP for provisioning (from the Device list) the user should specify if and what is the password for this AP

5. Porting Instructions

5.1.1. Generate UI fields required for your network

- SSID name
- Password
- Device name

5.1.2. Android Ap provisioning mode

1. Find your device by name and connected to it by Wi-Fi Manager Android API.
2. Check that the device can track your AP by activating device scanning from the device using “getSSIDListFromDevice“ API
3. Activate “addProfile” API with the required parameters from UI
4. Optionally, set device name
5. Activate “moveStateMachineAfterProfileAddition” to indicate the device is ready to restart after all settings are added
6. The device should connect to the required network by adding a profile and restarting at STA role.
7. Mobile application should connect to the same AP network, by Wi-Fi manager
8. Mobile should activate all the services find the new device: mDNS,Ping (broadcast) and UDP server
9. After finding the new device’s IP address, “getCGFResultFromDevice” API should be activated to fetch the result and to indicate that provisioning is done
10. In case the device is not detected or no response from it, the Mobile application will try to connect to it (assuming it moved to AP role), for getting the result and completing the provisioning
11. After getting the results, provisioning is finished in case of success.
12. In case of failure, the error should be inspected and fixed (wrong password, range issues etc.)

5.1.3. iOS Ap provisioning mode

1. Find your device manually (iOS has no API fore scanning or Wi-Fi connect/disconnect)
2. Optionally, activate get version using “getProductVersionFromUrl” API to verify you use a simplelink device
3. Optionally set device name using “setDeviceNameFromUrl”
4. Activate add profile API by “startAddingProfileProcedureFromURL” which is wrapper by “addProfile”. This API is used for adding a profile and it is followed by sending a reset request from the device by calling “moveStateMachineAfterProfileAddition” API

5. Upon adding profile, activate “mDdnsDiscoveryStart”,ping activity timer and UDP listener (“startStopUdp”)
6. After finding the new device’s IP address, “getCGFResultwithUrl” |API should be activated to fetch the result and to indicate that provisioning is done
7. In case the device is not detected or not responsding, Mobile application will try to connect to it (assuming it moved to AP role), for getting provisioning’s results
8. After getting the results, provisioning is finished in case of success.
9. In case of failure, the error should be inspected and fixed (wrong password, range issues etc.)

6. Settings

Settings tab is used for application configurations. it is a persistent storage on the application's local storage.
Using Android it is saved as a "SharedPreferences" object for persistency.
Using iOS the data is stored using "NSUserDefaults" class object.

- **Auto Device Selection** – Using AP mode – connects automatically to a SimpleLink device, if only one device is detected
- **Enable SmartConfig** – Unsupported mode. This setting should be set to False.
- **Enable QR reader** – Adds a QR code option
- **Show Device Name**- Display device name optional field
- **Show Security Key** – Show SmartConfig security key for encryption
- **Open in Devices Screen** – Changes default init screen
- **Show iotLink UUID** – Optional field for iotLink services

7. Logger and Email

Both iOS and Android application stores data logs for sending the information to TI in case any problem should be reported (passwords are not stored as a part of this log file).

The log file exist on the application's local storage and in some cases it is not exposed from the phone's UI.

The applications contains an email send button on the "Settings" screen which allows the user to fetch and send the latest log file sends it to ecs-bugreport@list.ti.com as an attachment.