User's Guide
SWRU472A–February 2017–Revised April 2019

SimpleLink™ Wi-Fi® CC3x20 and CC3x35 Provisioning for Mobile Applications

This user's guide describes Texas Instruments SimpleLink™ Wi-Fi® provisioning solution for mobile applications, specifically on the usage of the Android™ and iOS® building blocks for UI requirements, networking, and provisioning APIs required for building the mobile application.

Terms and Abbreviations

Abbreviation or Term	Meaning and Explanation
AP	Access Point
APIs	Application Interfaces
Bcast	Broadcast
HTTP	Hypertext Transfer Protocol
JSON	Javascript Object Notation
mDNS	Multicast DNS
SC	SmartConfig
UDP	User Datagram Protocol
UI	User Interface

Contents

Trademarks

1 Introduction

1.1 Overview

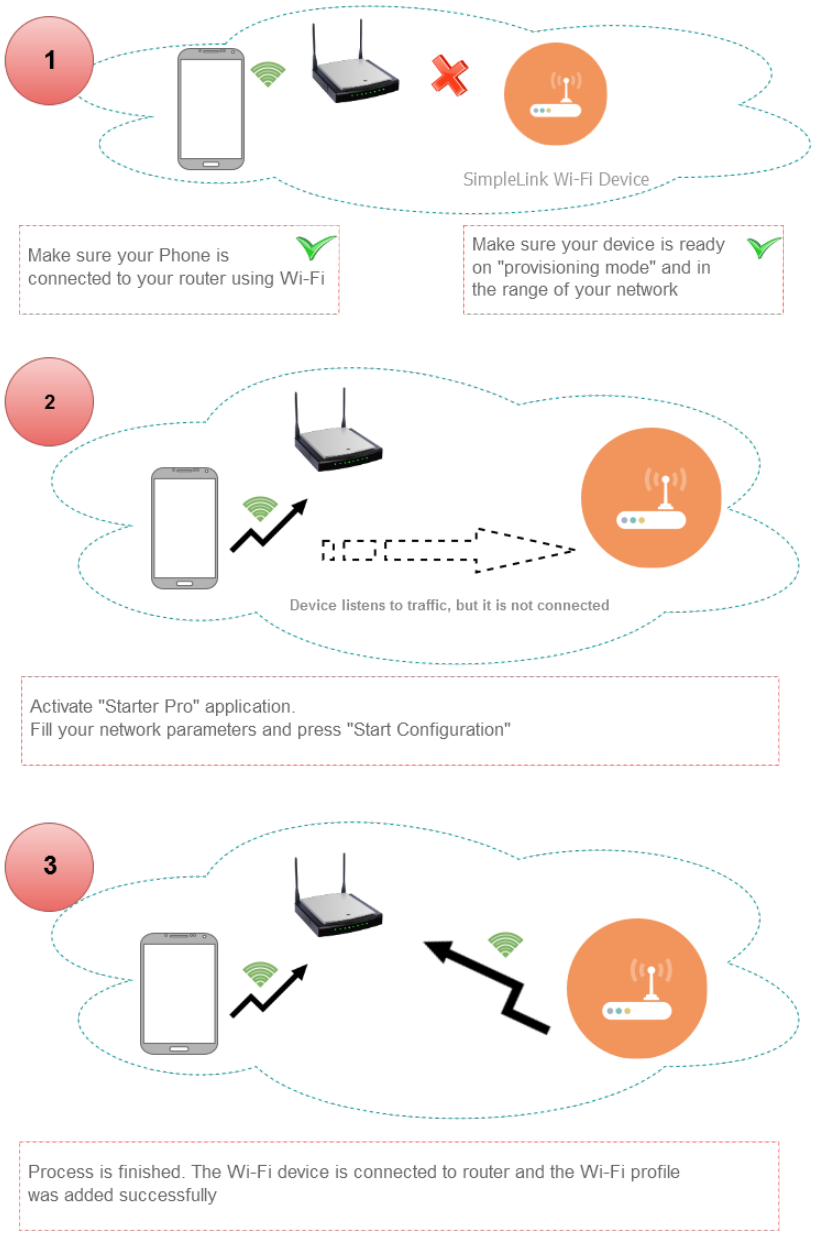
This document describes the Texas Instruments SimpleLink™ Wi-Fi® provisioning solution for mobile applications. The main focus of this document is on the usage of the Android™ and iOS® building blocks for UI requirements, networking, and provisioning APIs required for building the mobile application.

The CC31xx and CC32xx devices are part of the SimpleLink™ microcontroller (MCU) platform which consists of Wi-Fi®, *Bluetooth*® low energy, Sub-1 GHz and host MCUs, which all share a common, easy-to-use development environment with a single core software development kit (SDK) and rich tool set. A one-time integration of the SimpleLink™ platform enables you to add any combination of the portfolio's devices into your design, allowing 100 percent code reuse when your design requirements change. For more information, visit www.ti.com/simplelink.

The first step in utilizing a CC32xx/CC31xx device in a Wi-Fi®-enabled application is to connect the device to a Wi-Fi® network (access point). This process is called Wi-Fi® provisioning and it involves loading the information of the access point (SSID name and security credentials) to the device. This process can be complex considering that embedded Wi-Fi® applications generally lack user interfaces such as keypads or touch screens.

Figure 1 illustrates a general overview of this procedure: a device which is not connected to the network, but exists in the network range, can accept the information required for connecting to the network, using the combination of the embedded application (see *CC31xx, CC32xx SimpleLink™ Wi-Fi® Internet-on-a chip™ Solution Device Provisioning*) and the mobile application.

Figure 1. General Overview



1.2 Provisioning Methods

Wi-Fi® provisioning is usually done once, either while connecting a new device to the network or after an update on the local network which requires configuration changes. After the provisioning, the CC32xx device saves the accepted Wi-Fi® information as an encrypted profile. Using this profile, the CC3xxx/CC31xx device is able to automatically connect to the network when it is available as a Wi-Fi® station.

There are two main provisioning methods to connect a CC3xxx device to a desired access point using TI's mobile application for Wi-Fi® provisioning, Wi-Fi® Starter Pro:

- SmartConfig™ – Using the SmartConfig algorithm eliminates the need to know the CC32xx/CC31xx device identity ahead of time. The process configures any listening device. Using this method, the user does not need to know the device name, or any other device identity.
- AP mode – The mobile application connects to the device as a Wi-Fi® station, and sends the network information for the desired network connection. By using this mode, the user should know which device to connect to according to its published SSID, while functioning as an AP.

The user can select which mode to use using the Wi-Fi® Starter Pro app: SmartConfig™ or AP mode. Wi-Fi® Starter Pro contains a Settings tab for handling configurations; switching between SmartConfig™ and AP mode is one of the options on this tab.

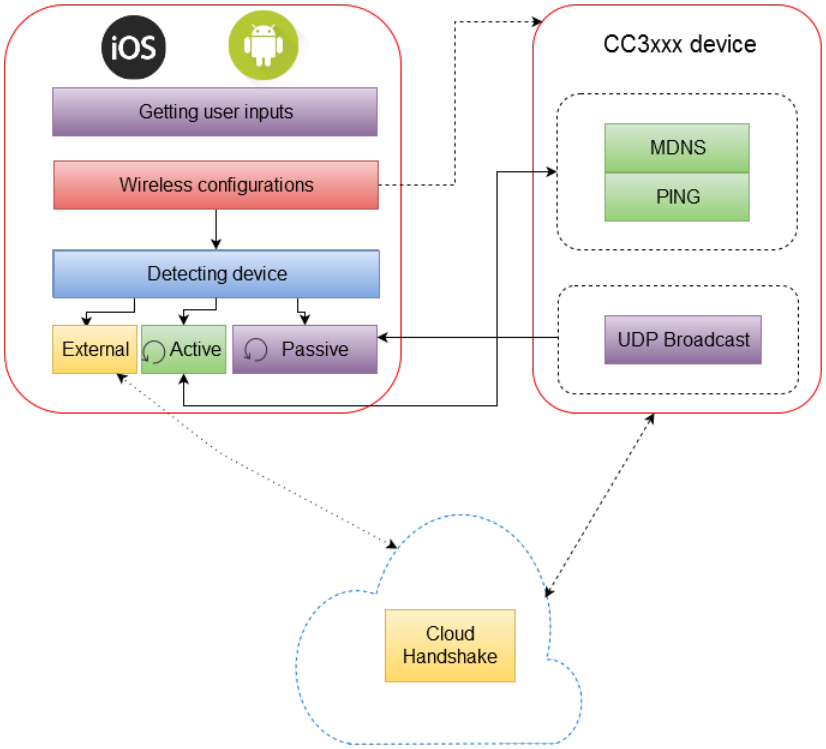
The preferred mode for the CC3200/CC3100 devices is AP mode. The preferred mode for the CC3220/CC3235/CC3120/CC3135 devices is SmartConfig™.

2 Top-Level Architecture

2.1 Top-Level Blocks

Figure 2 illustrates the top-level architecture.

Figure 2. Top-Level Architecture



2.2 Basic Provisioning Steps

There are four steps for completing provisioning:

1. User Inputs – The user provides information about the identity of the network.
2. Sending configurations – The mobile app sends network credentials to the device.
3. Finding the device on the network – The mobile app searches for the device. The device publishes information such as services and device name, and responds to network queries.
4. Connecting to the device and getting feedback – The mobile app connects to the device over the shared network, and confirms that the device is responding.

2.2.1 Step 1: User Inputs

This is the first step of the application. The mobile phone should be connected to the Wi-Fi® network before the provisioning is activated.

If using a secured network, the mandatory fields for both options (SmartConfig™ or AP mode) are the network SSID (usually the active Wi-Fi® connection of the mobile phone) and password.

If the device name is not set to a new value, then the default device name is used.

2.2.2 Step 2: Sending Configurations

The second step is to send the information to the non-configured device, using the active network. The active network is the link between the Mobile phone and the Router.

In this case, the mobile app side is responsible for transmitting the information to the network while the device listens to the information transmitted (even if it is a secured network). After verifying the validity of the information, the device creates and stores a profile of this network that is kept on the device's storage.

The saved profile is now activated and connected to the network.

2.2.3 Step 3: Finding the Device on the Network

The third step is to find the IP address of the device after it is connected to the Wi-Fi® network, and acquire an IP address from the DHCP server.

The mobile app uses three options for detecting the device on the network:

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

If the network is not totally isolated, the device will be detected and verified by the mobile application in one or more of the three options described.

2.2.4 Step 4: Connecting to the Device and Getting Feedback

The final step is to check if the provisioning completed successfully. This is done by sending a query from the mobile app to the device, asking for the provisioning results. This step is performed assuming the device IP address is already known from previous step, and that the device supports HTTP requests.

After the query is sent to the device, there are several possible responses that can be received. A successful response means that the provisioning completed. In case of a timeout or a failure, AP fallback mode is automatically suggested to the user by the mobile application.

2.2.5 Fallback Step: Confirmation Failed

If the mobile app failed to connect to the device and receive feedback, the mobile app should connect to the device directly. The app can act as an AP in case of a connection failure, and the device switches to the configurations stage as normal. In this case, the device is configured but the confirmation feedback is transferred using the direct connection.

3 Provisioning - SmartConfig™

3.1 Overview

TI's SmartConfig™ algorithm sends connection information to the device without a real connection. SmartConfig™ is responsible for sending the data, and the networking blocks (ping, UDP listener, mDNS) are responsible for getting the new IP of the connected device after the connection.

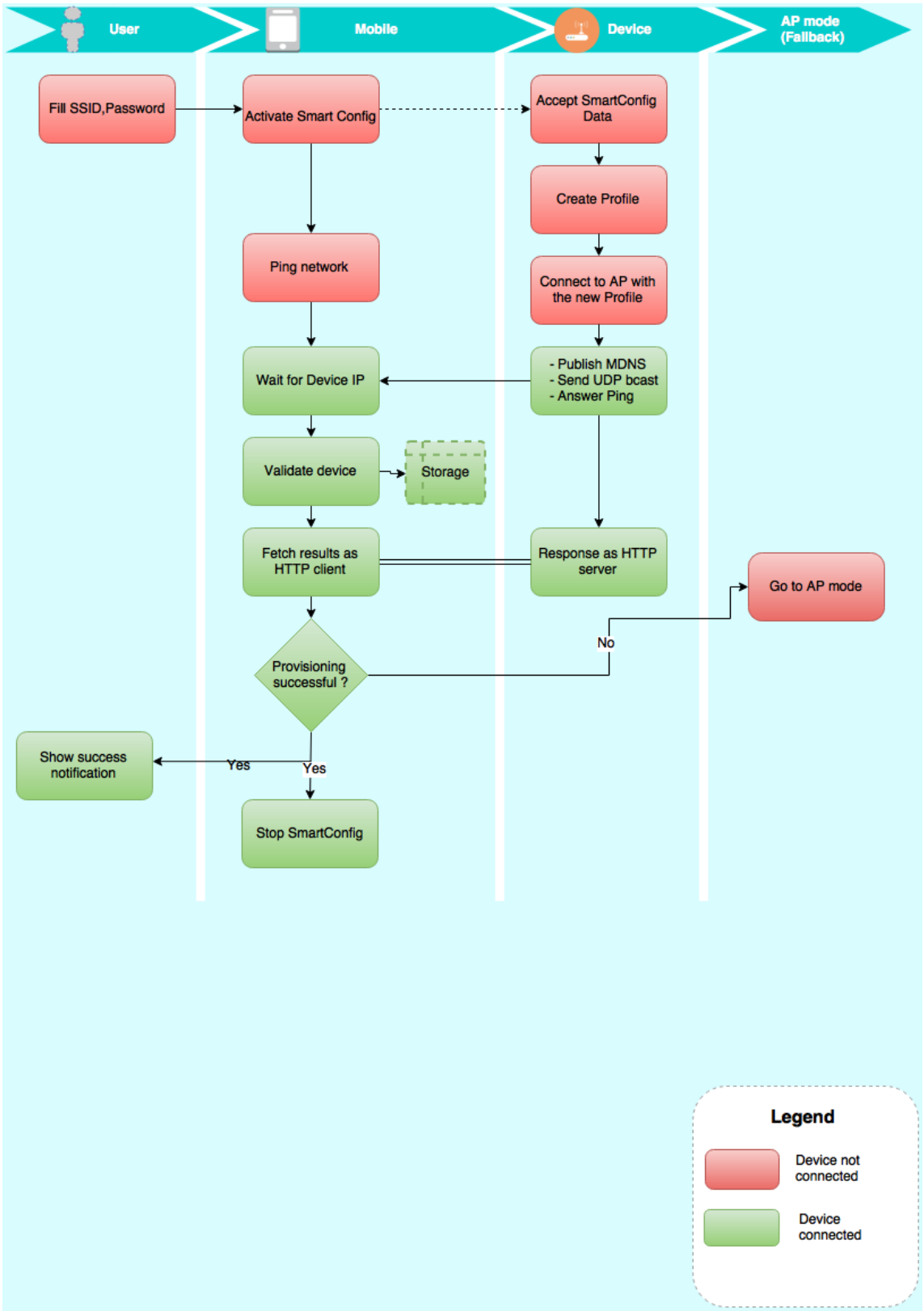
SmartConfig™ leverages the standard mechanisms present in Wi-Fi® to configure a CC32xx/CC31xx device's association information on the fly, regardless of whether a user interface is available. In this process, a Wi-Fi®-enabled device such as a smartphone, tablet, or a laptop sends the association information to the CC32xx//CC31xx device.

Additionally, SmartConfig™ does not depend on the I/O capabilities of the host microcontroller, and thus can be used by embedded applications. SmartConfig™ can be used to associate multiple devices to the same AP simultaneously. Additionally, the configured device (such as a smartphone or tablet) stays connected to the user's home network during the configuration process (as opposed to other methods that require disconnection).

3.2 SmartConfig™ Flow Chart

Figure 3 illustrates the SmartConfig™ flow chart.

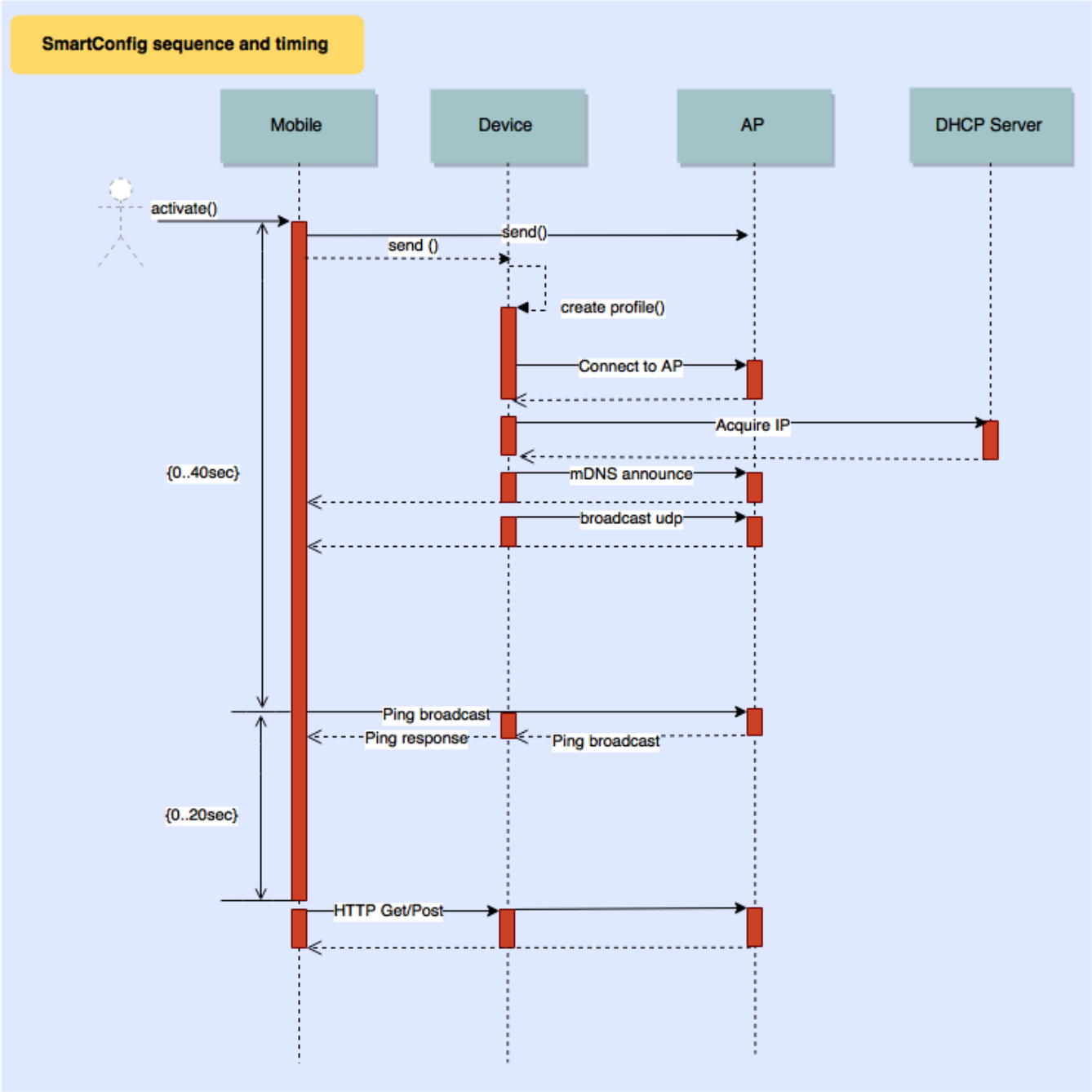
Figure 3. SmartConfig™ Flow Chart



3.3 SmartConfig™ Sequence

Figure 4 illustrates the SmartConfig™ sequence.

Figure 4. SmartConfig™ Sequence

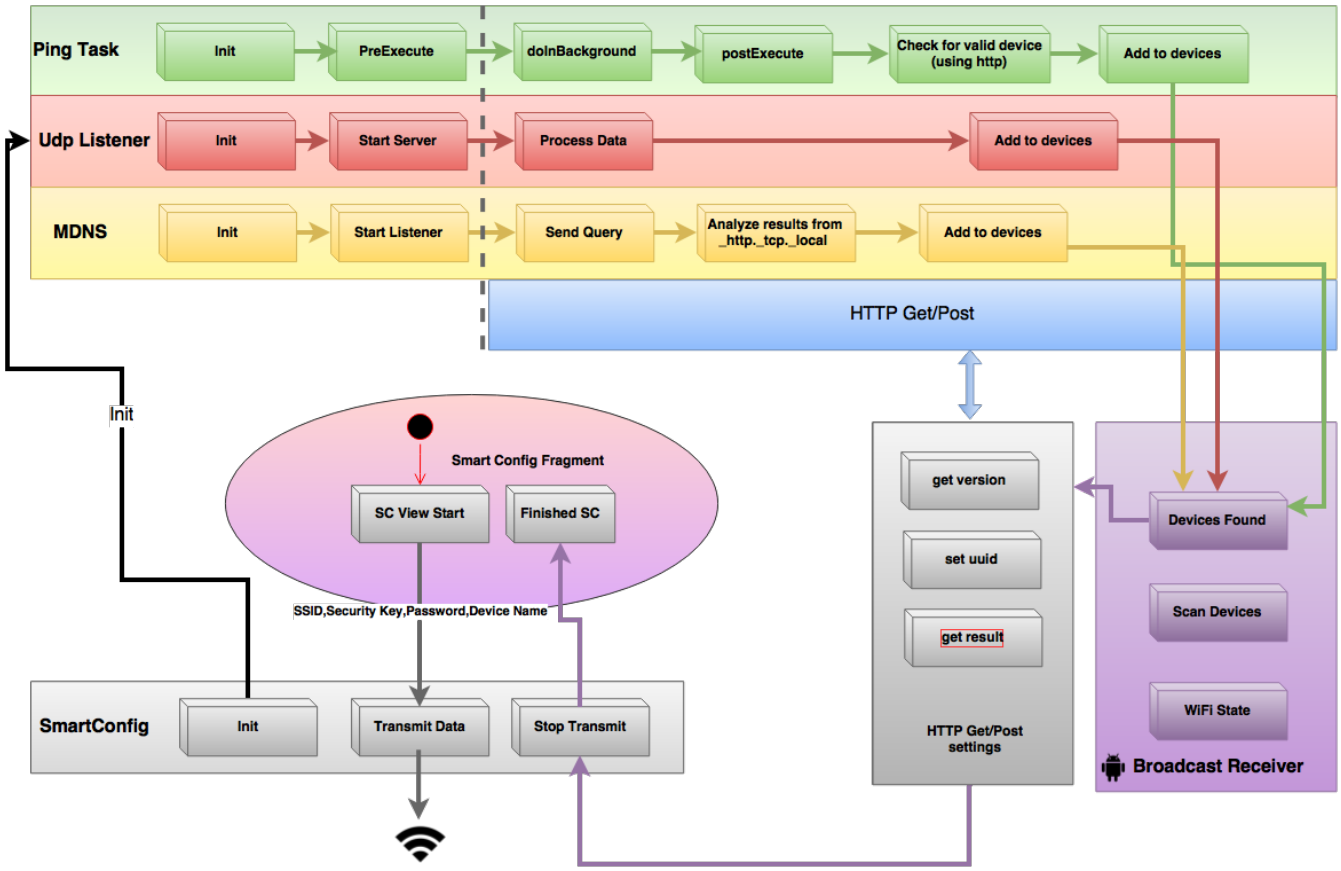


4 Android™ Block Diagram

4.1 Overview

Figure 5 illustrates the Android™ block diagram.

Figure 5. Android™ Block Diagram



4.2 SmartConfig™ Initialization

The SmartConfig™ library contains a set of APIs to activate SmartConfig™. These APIs cover the initialization of SmartConfig™ objects, and set configurations such as SSID, password, and start/stop transmission.

The transmission works repeatedly until it is stopped. The repeated data transmission improves the probability for success on a limited Wi-Fi® coverage network or on a congested environment.

Activate SmartConfig™ when the phone, the AP, and the target device are all on the same local network, with no hidden devices. For example, if the phone is not within the range of the device, but both the device and the phone are in the range of the AP. The device and the phone are hidden.

The SmartConfig™ command is activated by the following class:

Android™:

class SmartConfig

Table 1. Android™ Parameters

String	Value
Listener	A listener object, whose class derives from FirstTimeConfigListener base class; can be null.
FreeData	Null in case of no device name. If the device name exists, the first byte should be value 0x3, second byte specifies device string length, and the rest are device name characters.
Key	The Wi-Fi key to configure
EncryptionKey	A key used by the protocol to encrypt the Wi-Fi key (must be shared with the target device); can be null
String IP	Destination IP address of packets. This should be the IP address of the Wi-Fi access point or router.
String Ssid	The Wi-Fi network name to configure
Group	Group – should be 0
Token	Token should be null

For example:

```
try {

   smartConfig = new SmartConfig(smartConfigListener, freeData, passwordKey, paddedEncryptionKey,
gateway, SSID, (byte) 0, "");
} catch (SocketException e) {
   Log.e(TAG, "Failed to create instance of smart config");
   return;
}
```

iOS®:

class FirstTimeConfig::initWithData

Table 2. iOS® Parameters

String	Value
String IP	Destination IP address of packets. This should be the IP address of the Wi-Fi access point or router.
String Ssid	The Wi-Fi network name to configure
FreeData	Null in case of no device name. If the device name exists, the first byte should be value 0x3, second byte specifies device string length, and the rest are device name characters.
Key	The Wi-Fi key to configure
EncryptionKey	A key used by the protocol to encrypt the Wi-Fi key (must be shared with the target device); can be null.
numberOfSetups	4
numberOfSyncs	10
syncLength1	3
syncLength2	23
delayInMicroSeconds	10000

4.3 SmartConfig™ Transmission

Starts SmartConfig™ packets transmission, based on the init values.

transmitSettings()

For example:

```
smartConfig.transmitSettings();
```

4.4 Stopping SmartConfig™ Transmissions

SmartConfig™ continues to broadcast network credentials until stopped by the user. SmartConfig™ is unaware of the success or failure of the transmission. Therefore, it should be stopped by controlling layers after one of the following scenarios:

- Activated for 40 sec + 20 sec for getting network indications , but the device was not detected. It is assumed that after this period of time, that the device is not connected to the network, the configuration is wrong, or the device is not on the correct mode for provisioning.
- The device was detected successfully, and there is no need to keep sending the data again until time-out expires.

stopTransmitting()

For example:

```
smartConfig.stopTransmitting();
```

4.5 UDP Listener

UDP listener is a UDP server running on a specific thread (Android™ UI thread or iOS® thread). UDP listener should be activated as a background task, so that it can find UDP broadcasts from the device, upon acquiring an 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 contains both the device name and the device IP address, with a comma separation in a textual format.

Upon successfully receiving and parsing the data, the UDP listener callback is activated, and the device is added to the device containers after it is converted to JSON format.

For example – starting UDP server on Android™:

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

4.6 mDNS Listener

mDNS listener assumes the device, using its default configurations, supports both mDNS and HTTP server. mDNS listener will be active as longn as the provisioning process is ngoing.

In this case, upon acquiring an IP address, the device announces its services by using multicast messages. HTTP is one of the services to be published on the network.

Because the HTTP service contains some known information, this information is used to add the device containing this specific information to the device list.

The txt fields on the checked mDNS packet should contain the string "srcvers=1D90645" to confirm the correctness of this device. If a valid CC32xx/CC31xx device detected using mDNS, construct a JSON message with the device name and its IP address, and add it to the device container. The device information and container are only examples of using the information and sending it across mobile application layers. This implementation can be modified by the mobile programmer.

On Android™ – mDNSHelper handles mDNS initialization and callback functions for accepting and parsing incoming mDNS information from the local network.

4.7 Ping Task

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

The SimpleLink™ device is designed to reply to broadcast ping requests by default. Upon accepting a response from the network devices, the mobile application filters only SimpleLink™ devices by querying specific http request and getting the device version. In case of a valid response, it indicates that this is a SimpleLink™ device, and it can be added to the devices container in a JSON format.

Ping class handles both ping generation, accepting ping responses, and creating an HTTP request for validating the device name.

4.8 HTTP Requests for SmartConfig™

HTTP requests send or request information from the mobile app side (HTTP client) to the device side (HTTP server). There are some services used by SmartConfig™, and other services used by AP mode.

Each service usually has two options: R1, which is the first revision of the SimpleLink™ device, and supports different APIs than R2. Most HTTP APIs require an input parameter to indicates whether the API should act as R1 or as R2.

Because HTTP queries are usually long and may take time to complete (depending on the host speed, interface, or link quality), all HTTP actions are wrapped on an async thread, and should not stall any UI activity.

4.8.1 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, and it returns the current version.

After getting version information, this information should be used on the next HTTP API as one of the input parameters.

Device method:

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

4.8.2 Get Configurations Results

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

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

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

4.8.3 Get Response Number Code

The string fetched from the previous API, getCGFResultFromDevice, should be converted to a number by using the string in Table 3.

Table 3. String Values

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)
```

4.8.4 Get Error Message From Number

This API converts the number into a readable string. This string is shown to the user as a result of the provisioning actions upon completion, as shown in Table 4.

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

Table 4. Error String

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);
```

4.8.5 Get Device Name

This API reads the device name from the device.

The device name is presented on the UI, if the user decides to get the default device name and not to set a device name.

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

Texas Instruments™ recommends setting a device name, because if other devices respond to mDNS, the mobile device cannot decide which one belongs to the active one, as no name was set.

4.8.6 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)
```

5 Provisioning – AP Mode

5.1 Overview

AP mode is the process of using the AP role of the device for configurations. After selecting the desired AP for configurations, all settings (ssid, password, device name, and uuid) are sent by HTTP protocol. When the configurations are ready, the device switches to station (STA) role, and uses the profile (ssid + password) to connect to the local network. At this stage, the mobile applications also connect to the same AP and will search for the SimpleLink™ device, using one or more of the three methods: ping, mDNS, and UDP broadcasts.

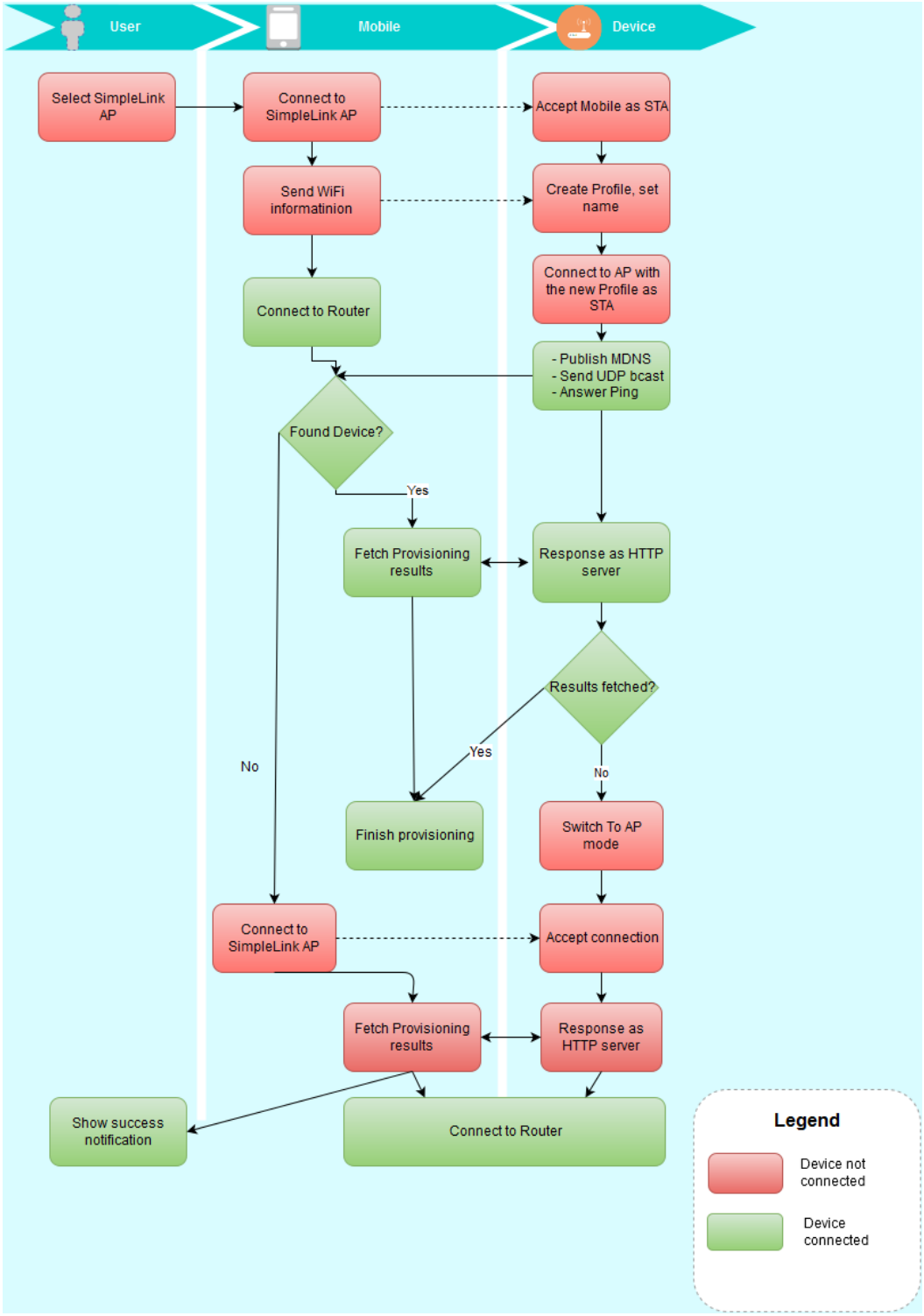
If there is a problem finding the SimpleLink™ device, the device changes the role back to AP. The mobile side connects to the SimpleLink™ device again, but this time to close the loop and fetch the provisioning results.

On TI's mobile application ("Wi-Fi® Starter Pro") the transition between AP mode and SmartConfig™ is set by a button on the "Settings" screen.

5.2 AP Mode Flow Chart

Figure 6 illustrates the AP mode flow chart.

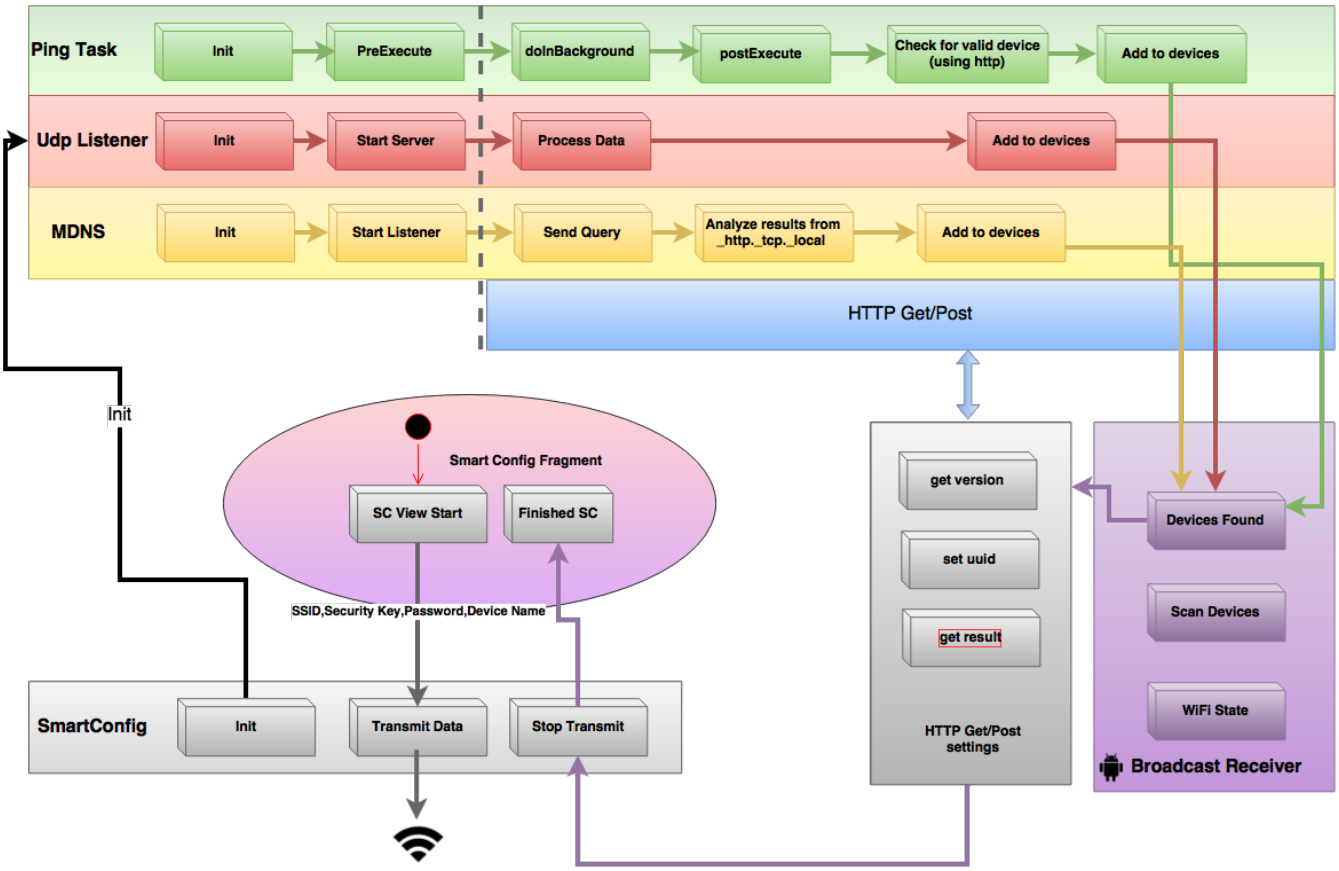
Figure 6. AP Mode Flow Chart



5.3 Block Diagram

Figure 7 illustrates the Android™ block diagram.

Figure 7. Android™ Block Diagram



5.4 UDP Listener

See Section 4.5.

UDP listener should be activated when the mobile application connects to the local network as a station. There is no use for UDP listener while the device is in AP mode.

5.5 mDNS Listener

See Section 4.6.

mDNS listener should be activated when the mobile application connects to the local network. There is no use for mDNS listener while the device is in AP mode.

5.6 Ping Task

See Section 4.7.

Ping task should be activated when the mobile application connects to the local network. There is no use for mDNS listener while the device is in AP mode.

5.7 HTTP Requests for AP Mode

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

5.7.1 Get SSID List From Device

This API requests the APs list from the device. The API accepts the list of scanned APs. The query does not initiate a scan, but fetches the latest APs from the device list. It is activated while pressing Wi-Fi® network in AP mode.

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

5.7.2 Rescan Networks on Device

API for setting scan interval settings.

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

5.7.3 Add Profile

This API is a direct profile activation API for storing a new profile. It accepts connection details such as 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)
```

5.7.4 Notify Device Profile is Ready

Upon setting a new profile, the mobile app side schedules an activation of this profile by using this API. The API notifies the device that the mobile app is ready for the next step, and the device should restart in Wi-Fi® station mode to start using the profile set previously.

This API ensures the profile adding step is completed and the device can restart a connection, using the new profile. The mobile should start scanning for the new device after sending this notification to the device.

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

6 iOS® versus Android™ Development Guidelines

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

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

7 Porting Instructions

7.1 Generate UI Fields Required for the Network

- SSID name
- Password
- Device name

7.2 Android™ AP Provisioning Mode

1. Find the device by name, and connect to it by using the Wi-Fi Manager Android API.
2. Check that the device can track your AP by activating device scanning from the device, using the getSSIDListFromDevice API.
3. Activate the addProfile API with the required parameters from UI.
4. Optionally, set the 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. The mobile application should connect to the same AP network, by using the Wi-Fi® manager.
8. The mobile app should activate all the services to find the new device: mDNS, ping (broadcast), and UDP server.
9. After finding the IP address of the new device, the getCGFResultFromDevice API should be activated to fetch the result and indicate that provisioning is done.
10. If the device is not detected or there is no response from it, the mobile application will try to connect to it (assuming it moved to AP role), to get the result and complete the provisioning.
11. After getting the results, provisioning is finished if successful.
12. In case of failure, the error (wrong password, range issues, and so forth) should be inspected and fixed.

7.3 iOS® AP Provisioning Mode

1. Find the device manually (iOS® has no API for scanning or Wi-Fi® connect and disconnect).
2. Optionally, activate the get version using the getProductVersionFromUrl API to verify you're using a SimpleLink™ device.
3. Optionally, set the device name using setDeviceNameFromUrl.
4. Activate the add profile API by startAddingProfileProcedureFromURL, which is wrapped by the addProfile API. This API is used for adding a profile, and it is followed by sending a reset request from the device by calling the moveStateMachineAfterProfileAddition API.
5. Upon adding a profile, activate mDdnsDiscoveryStart, and ping the activity timer and UDP listener (startStopUdp).
6. After finding the IP address of the new device, activate the getCGFResultwithUrl API to fetch the result and indicate that provisioning is complete.
7. If the device is not detected or not responding, the mobile application tries to connect to it (assuming it moved to AP role), to get provisioning results.
8. After getting the results, provisioning is finished if successful.
9. In case of failure, the error (wrong password, range issues, and so forth) should be inspected and fixed.

8 Settings

The Settings tab is used for application configurations, and is a persistent storage on the local storage of the application.

With Android™, the data is saved as a SharedPreferences object for persistency. With iOS®, the data is stored using the NSUserDefaults class object.

- Auto device selection – Using AP mode, it connects automatically to a SimpleLink™ device, if only one device is detected.
- Enable SmartConfig™ – If it is set to False (default), it runs on AP mode. If True, it runs on SmartConfig™ mode.
- Enable QR reader – Adds a QR code option.
- Show Security Key – Show SmartConfig™ security key for encryption.
- Open in Devices Screen – Changes default init screen.

9 Logger and Email

Both iOS® and Android™ applications store data logs for sending the information to Texas Instruments™, in case a problem should be reported (passwords are not stored as a part of this log file).

The log file exists on the local storage of the application, and in some cases it is not exposed from the phone UI.

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

Revision History

NOTE: Page numbers for previous revisions may differ from page numbers in the current version.

Changes from Original (February 2017) to A Revision	Page
• Updated title	1
• Changed Section 1	2
• Changed Section 1.2	4
• Changed Section 2.2.1	5
• Changed Section 2.2.2	5
• Changed Section 2.2.4	5
• Added paragraphs to Section 3.1	6
• Changed Section 4.4	11
• Changed Section 4.6	11
• Deleted Set IoT UUID subsection. Subsequent subsections renumbered	13
• Added paragraph to Section 5.1	14
• Added paragraph to Section 5.7.4	17
• Deleted last bulleted item in Section 8	19

IMPORTANT NOTICE AND DISCLAIMER

TI PROVIDES TECHNICAL AND RELIABILITY DATA (INCLUDING DATASHEETS), DESIGN RESOURCES (INCLUDING REFERENCE DESIGNS), APPLICATION OR OTHER DESIGN ADVICE, WEB TOOLS, SAFETY INFORMATION, AND OTHER RESOURCES “AS IS” AND WITH ALL FAULTS, AND DISCLAIMS ALL WARRANTIES, EXPRESS AND IMPLIED, INCLUDING WITHOUT LIMITATION ANY IMPLIED WARRANTIES OF MERCHANTABILITY, FITNESS FOR A PARTICULAR PURPOSE OR NON-INFRINGEMENT OF THIRD PARTY INTELLECTUAL PROPERTY RIGHTS.

These resources are intended for skilled developers designing with TI products. You are solely responsible for (1) selecting the appropriate TI products for your application, (2) designing, validating and testing your application, and (3) ensuring your application meets applicable standards, and any other safety, security, or other requirements. These resources are subject to change without notice. TI grants you permission to use these resources only for development of an application that uses the TI products described in the resource. Other reproduction and display of these resources is prohibited. No license is granted to any other TI intellectual property right or to any third party intellectual property right. TI disclaims responsibility for, and you will fully indemnify TI and its representatives against, any claims, damages, costs, losses, and liabilities arising out of your use of these resources.

TI’s products are provided subject to TI’s Terms of Sale (www.ti.com/legal/termsofsale.html) or other applicable terms available either on ti.com or provided in conjunction with such TI products. TI’s provision of these resources does not expand or otherwise alter TI’s applicable warranties or warranty disclaimers for TI products.

Mailing Address: Texas Instruments, Post Office Box 655303, Dallas, Texas 75265