Alexa Ready application on Talaria TWO uses AWS IoT Embedded C Device SDK available on Talaria TWO to interact with the Device Shadow Service on AWS IoT Core.

Device makers can have their own Alexa Skill implementation hosted on their cloud to bridge the Alexa Voice Service (AVS) and device maker’s AWS IoT Core Cloud Service. Post this, AWS IoT Embedded C Device SDK available on Talaria TWO can be used to make their smart home device endpoints which can be controlled using Alexa.

This Application Note describes the APIs used to achieve this on Talaria TWO and guides the reader to a demo of Alexa Ready application using InnoPhase Smart Home Demo Alexa Skill to control a device endpoint running on Talaria TWO EVB and behaving as an Alexa Smart Home Switch.

This is achieved by Alexa Voice Service (AVS) Cloud interacting with the Alexa Skill hosted at InnoPhase IoT’s AWS Cloud which eventually controls the Talaria TWO EVB represented as an IoT device endpoint in InnoPhase Inc.’s AWS IoT Core Cloud.

# AWS IoT Device Shadow Service

Device Shadow is the always-available device state in the AWS cloud. A device's shadow is a JSON document that is used to store and retrieve current state information for a device. The AWS IoT Device Shadow service adds and maintains shadows to AWS IoT Thing objects. It is also known as Thing Shadow.

Shadows can make a device’s state available to apps and other cloud services whether the device is connected to AWS IoT or not.

To get and set the state of a device from AWS IoT Core, the Device Shadow service can be used over MQTT and HTTP. Applications and web services generally use HTTP to interact with Device Shadow and Embedded devices often use MQTT to interact with Device Shadow.

Common use cases for Device Shadow include backing up device state, or sending commands to devices and as a reliable data store for devices, applications, and other cloud services to share data. While devices, applications, and other cloud services are connected to AWS IoT Core, they can access and control the current state of a device through its shadows. Device Shadow Service enables devices, applications, and other cloud services to connect and disconnect without losing a device's state.

# AWS IoT Embedded C Device SDK 3.0.1 Features

AWS IoT Embedded C Device SDK version 3.0.1 is available with Talaria TWO FreeRTOS SDK which allows embedded applications to securely connect to the AWS IoT Core platform. It simplifies access to the PUB/SUB functionality of the AWS IoT Core broker via MQTT and provides convenient APIs for handling MQTT topics reserved for Device Shadow Service to interact with Thing Shadows.

AWS IoT Embedded C Device SDK was specifically designed for resource constrained devices (running on micro-controllers and RTOS), and follows a layered architecture as shown in Figure 1.

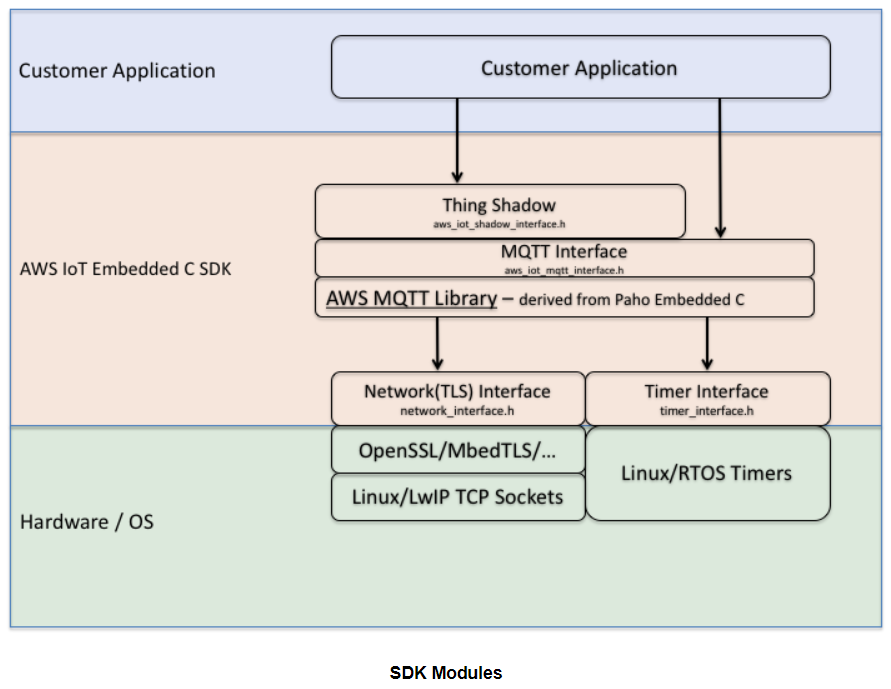


Figure : SDK architecture & modules

## MQTT Connection

AWS IoT Embedded C Device SDK provides functionality to create and maintain a mutually authenticated TLS connection over which it runs MQTT. This connection is used for any further publish operations and allow for subscribing to MQTT topics which will call a configurable callback function when these topics are received.

## Thing Shadow

AWS IoT Embedded C Device SDK implements the specific protocol for Thing Shadows to retrieve, update and delete Thing Shadows. This protocol is implemented to ensure correct versioning and support for client tokens.

# AWS IoT Core Device Shadow Service Protocol

This section describes device communications with Shadows using Shadow Service Protocol.

It internally uses MQTT messages which is the preferred method for embedded devices to communicate with the AWS IoT Device Shadow service.

It abstracts the necessary MQTT topic subscriptions by automatically subscribing to and unsubscribing from the reserved topics for Device Shadow Service as needed for each API call. Inbound state change requests are automatically signaled via a configurable call back.

Shadow communications emulate a request/response model using the publish/subscribe communication of MQTT. As MQTT is used, the shadow needs to connect and disconnect.

There are three actions a device can perform on the shadow - Get, Update and Delete. Every shadow action consists of a request topic, a successful response topic (accepted), and an error response topic (rejected). On performing any action, the acknowledgment will be received in either accepted topic or rejected topic.

On performing any action, the acknowledgment will be received in either accepted or rejected.

For Example: If there is a need to perform a GET on a Thing Shadow the following messages will be sent and received:

1. A MQTT Publish on the topic - $aws/things/{thingName}/shadow/get
2. Subscribe to MQTT topics - $aws/things/{thingName}/shadow/get/accepted and $aws/things/{thingName}/shadow/get/rejected.

If the request was successful, the things json document is received in the accepted topic.

Similarly, for another example, when an Update is performed to a Thing Shadow, one of the two things can happen and can be acknowledged.

The update action could be accepted by the Thing Shadow and the version of the JSON document will be updated. The update request could also be rejected.

This can be known by subscribing to the two topics: $aws/things/{thingName}/shadow/update/accepted and $aws/things/{thingName}/shadow/update/rejected.

## Initialization of the device on first connection to AWS IoT Core

After a device registers with AWS IoT, it subscribes to these MQTT messages for the shadows that it supports. The ShadowTopicPrefix can refer to either a named or an unnamed shadow, as described in Table 1.

Shadows can be named or unnamed (classic). The topics used by each differ only in the topic prefix. Table 1 shows the topic prefix used by each shadow type.

|  |  |
| --- | --- |
| **ShadowTopicPrefix value** | **Shadow type** |
| $aws/things/thingName/shadow | Unnamed (classic) shadow |
| $aws/things/thingName/shadow/name/shadowName | Named shadow |

Table : Topic prefix used by each shadow type

|  |  |  |
| --- | --- | --- |
| **Topic** | **Meaning** | **Action a device should take when this topic is received** |
| ShadowTopicPrefix/delete/accepted | The delete request was accepted and AWS IoT deleted the shadow. | The actions necessary to accommodate the deleted shadow, such as stop publishing updates. |
| ShadowTopicPrefix/delete/rejected | The delete request was rejected by AWS IoT and the shadow was not deleted. The message body contains the error information. | Respond to the error message in the message body. |
| ShadowTopicPrefix/get/accepted | The get request was accepted by AWS IoT, and the message body contains the current shadow document. | The actions necessary to process the state document in the message body. |
| ShadowTopicPrefix/get/rejected | The get request was rejected by AWS IoT, and the message body contains the error information. | Respond to the error message in the message body. |
| ShadowTopicPrefix/update/accepted | The update request was accepted by AWS IoT, and the message body contains the current shadow document. | Confirm the updated data in the message body matches the device state. |
| ShadowTopicPrefix/update/rejected | The update request was rejected by AWS IoT, and the message body contains the error information. | Respond to the error message in the message body. |
| ShadowTopicPrefix/update/delta | The shadow document was updated by a request to AWS IoT, and the message body contains the changes requested. | Update the device's state to match the desired state in the message body. |
| ShadowTopicPrefix/update/documents | An update to the shadow was recently completed, and the message body contains the current shadow document. | Confirm the updated state in the message body matches the device's state. |

Table : ShadowTopicPrefix

After subscribing to the messages in the preceding table for each shadow, the device tests to see if the shadows that it supports have already been created by publishing a /get topic to each shadow. If a /get/accepted message is received, the message body contains the shadow document, which the device uses to initialize its state. If a /get/rejected message is received, the shadow is created by publishing an /update message with the current device state.

## Processing messages while the device is connected to AWS IoT Core

There are three key value pairs of device states in shadow JSON document which a device needs to be concerned about.

1. Reported
2. Desired
3. Delta

All these keys are under the state.

If the device state is changed using a physical interaction, then Publishing an /update message with a desired message body that has the device’s physically changed state is needed. When it is done, other entities connected with Device Shadow Service get a delta callback notifying them with the change.

The device always receives a delta message if there is any difference between the desired and the reported section of the device and if the device has subscribed for the delta topic using the API [aws\_iot\_shadow\_register\_delta()](http://aws-iot-device-sdk-embedded-c-docs.s3-website-us-east-1.amazonaws.com/aws__iot__shadow__interface_8h.html" \l "a2b6000e1a7590cff3b6e66601b0c4934" \o "This function is used to listen on the delta topic of #AWS_IOT_MY_THING_NAME mentioned in the aws_iot...).

While a device is connected to AWS IoT, it can receive /update/delta messages if the desired state is changed by another party, and should keep the device state matched to the changes in its shadows by:

1. Reading all /update/delta messages received and synchronizing the device state to match.
2. Publishing an /update message with a reported message body that has the device’s current state, whenever the device's state changes.

While a device is connected, it publishes these messages when indicated.

|  |  |  |
| --- | --- | --- |
| **Indication** | **Topic** | **Payload** |
| The device's state has changed. | ShadowTopicPrefix/update | A shadow document with the reported property. |
| The device's desired state has changed (physical interaction). | ShadowTopicPrefix/update | A shadow document with the desired property. |
| The device might not be synchronized with the shadow. | ShadowTopicPrefix/get | (empty) |
| An action on the device indicates that a shadow will no longer be supported by the device, such as when the device is being remove or replaced | ShadowTopicPrefix/delete | (empty) |

Table : Indication

## Processing messages when the device is reconnected to AWS IoT Core

When a device with one or more shadows connects to AWS IoT, it should synchronize its state with that of all the shadows that it supports by:

1. Reading all /update/delta messages received and synchronizing the device state to match.
2. Publishing an /update message with a reported message body that has the device’s current state.

# AWS IoT Embedded C Device SDK - Shadow Service APIs and Structures

APIs available in Talaria TWO AWS IoT Device SDK to effectively use the AWS IoT Device Shadow Service are as follows:

## aws\_iot\_shadow\_init()

This API takes care of initializing the IoT client and the internal book-keeping data structures of Thing Shadow before use.

Parameter pClient is a new MQTT Client to be used as the protocol layer. Will be initialized with pParams.

Returns an IoT Error Type defining successful/failed Initialization.

|  |
| --- |
| IoT\_Error\_t aws\_iot\_shadow\_init(AWS\_IoT\_Client \*pClient, ShadowInitParameters\_t \*pParams); |

## aws\_iot\_shadow\_connect()

This API does the TLSv1.2 handshake and establishes the MQTT connection to connect to the AWS IoT Thing Shadow service over MQTT.

Parameter pClient is MQTT Client used as the protocol layer, pParams holds Shadow Connection parameters.

Returns an IoT Error Type defining successful/failed Connection.

|  |
| --- |
| IoT\_Error\_t aws\_iot\_shadow\_connect(AWS\_IoT\_Client \*pClient, ShadowConnectParameters\_t \*pParams); |

## aws\_iot\_shadow\_yield()

This API is called to yield the current thread to the underlying MQTT client and Shadow. It ensures the expired requests of Shadow actions are cleared and Timeout callback is executed.

It also ensures that the MQTT client gets the time to manage PING requests to monitor the health of the TCP connection as well as periodically check the socket receive buffer for subscribe messages.

This function could be used in a separate thread waiting for the incoming messages, ensuring the connection is kept alive with the AWS Service.

All callbacks used in the SDK will be executed in the context of this function.

Parameter pClient is MQTT Client used as the protocol layer, timeout is the maximum time in milliseconds the yield function will wait for a message and/or read the messages from the TLS buffer.

Returns an IoT Error Type defining successful/failed Yield.

|  |
| --- |
| IoT\_Error\_t aws\_iot\_shadow\_yield(AWS\_IoT\_Client \*pClient, uint32\_t timeout); |

## (\*fpActionCallback\_t)()

This is a Function Pointer typedef used as the callback for the actions Update, Get and Delete.

This function will be called from the context of thread which called aws\_iot\_shadow\_yield().

Parameter pThingName is Thing Name of the response received, action tells that the response is of which action (Update, Get or Delete), status informs if the action was Accepted/Rejected or Timed out, pReceivedJsonDocument is received JSON document when Accepted, pContextData is the void\* data passed in during the action call (Update, Get or Delete).

|  |
| --- |
| typedef void (\*fpActionCallback\_t)(const char \*pThingName, ShadowActions\_t action, Shadow\_Ack\_Status\_t status, const char \*pReceivedJsonDocument, void \*pContextData); |

|  |
| --- |
| /\*\*  \* @brief Thing Shadow Acknowledgment enum  \*/  typedef enum {  SHADOW\_ACK\_TIMEOUT, SHADOW\_ACK\_REJECTED, SHADOW\_ACK\_ACCEPTED  } Shadow\_Ack\_Status\_t;  /\*\*  \* @brief Thing Shadow Action type enum  \*/  typedef enum {  SHADOW\_GET, SHADOW\_UPDATE, SHADOW\_DELETE  } ShadowActions\_t; |

## aws\_iot\_shadow\_update()

This API is the used to perform an Update action to a Thing Name's Shadow.

Update is one of the most frequently used functionalities by a device. In most cases the device may be just reporting few params to update the thing shadow in the cloud.

If no callback or if the JSON document does not have a client token, then the Update Action will just publish the update and not track it.

The following steps are performed on using this function:

1. Subscribe to Shadow topics - $aws/things/{thingName}/shadow/update/accepted and $aws/things/{thingName}/shadow/update/rejected
2. Wait for two seconds for the subscription to take effect
3. Publish on the update topic - $aws/things/{thingName}/shadow/update
4. The response will be handled in the aws\_iot\_shadow\_yield() function. In case of timeout or if no response is received, the subscription to shadow response topics is un-subscribed.

On the contrary, if the persistent subscription is set to TRUE then the un-subscribe will not be done. The topics will always be listened to.

Parameter pClient is MQTT Client used as the protocol layer, pThingName is the Thing Name of the shadow that needs to be Updated, pJsonString contains a JSON document which update action expects. The JSON String should be a null terminated string. This JSON document should adhere to the AWS IoT Thing Shadow specification. To help in the process of creating this JSON document- SDK provides JSON handling APIs explained later in this app note.

Parameter callback is the callback that will be used to inform the caller of the response from the AWS IoT Shadow service. Callback could be set to NULL if response is not important.

Parameter pContextData is an extra parameter that could be passed along with the callback. Should be set to NULL if not used.

Parameter timeout\_seconds is the time the SDK will wait for the response on either accepted or rejected topic before declaring timeout on the action.

Parameter isPersistentSubscribe should be set to true to avoid repeated subscription and un-subscription if every time a device updates the same shadow, as mentioned above. If the update action on the Thing Name is a one-off update, then isPersistentSubscribe should be set to false.

Returns an IoT Error Type defining successful/failed Update action.

|  |
| --- |
| IoT\_Error\_t aws\_iot\_shadow\_update(AWS\_IoT\_Client \*pClient, const char \*pThingName, char \*pJsonString, fpActionCallback\_t callback, void \*pContextData, uint8\_t timeout\_seconds, bool isPersistentSubscribe); |

## aws\_iot\_shadow\_get()

This API is the used to perform a Get action to a Thing Name's Shadow.

This is usually used to get the config of a device at boot up. It is like the Update API internally, except it does not take a JSON document as the input. In case of success, JSON document is received over the accepted topic.

All the other parameters are same as explained in API aws\_iot\_shadow\_update().

Returns an IoT Error Type defining successful/failed Get action.

|  |
| --- |
| IoT\_Error\_t aws\_iot\_shadow\_get(AWS\_IoT\_Client \*pClient, const char \*pThingName, fpActionCallback\_t callback, void \*pContextData, uint8\_t timeout\_seconds, bool isPersistentSubscribe); |

## aws\_iot\_shadow\_delete()

This API is the used to perform a Delete action to a Thing Name's Shadow.

It is generally the responsibility of the accompanying web service / app to do the delete so it is not a very common use case for the device.

It is like the Update function internally, except it does not take a JSON document as the input. The Thing Shadow referred by the Thing Name will be deleted.

All the other parameters are same as explained in API aws\_iot\_shadow\_update().

Returns an IoT Error Type defining successful/failed Delete action.

|  |
| --- |
| IoT\_Error\_t aws\_iot\_shadow\_delete(AWS\_IoT\_Client \*pClient, const char \*pThingName, fpActionCallback\_t callback, void \*pContextData, uint8\_t timeout\_seconds, bool isPersistentSubscriptions); |

## aws\_iot\_shadow\_register\_delta()

This API is the used to listen on the delta topic of #AWS\_IOT\_MY\_THING\_NAME mentioned in the aws\_iot\_config.h file. Any time a delta is published, the JSON document will be delivered to the pStruct->cb. If the parsing done by the SDK is not needed, then use the jsonStruct\_t key set to "state".

Parameter pClient is MQTT Client used as the protocol layer and pStruct is the struct used to parse JSON value. To help in the process of parsing this JSON document- SDK provides JSON handling APIs explained later in this app note.

Returns an IoT Error Type defining successful/failed delta registration.

|  |
| --- |
| IoT\_Error\_t aws\_iot\_shadow\_register\_delta(AWS\_IoT\_Client \*pClient, jsonStruct\_t \*pStruct); |

## aws\_iot\_shadow\_reset\_last\_received\_version()

This API is the used Reset the last received version number to zero and is useful when the Thing Shadow is deleted and the local version needs to be rest.

|  |
| --- |
| void aws\_iot\_shadow\_reset\_last\_received\_version(void); |

## aws\_iot\_shadow\_get\_last\_received\_version()

This API is the used get the last received version number for a JSON document.

Version of a document is received with every accepted/rejected and the SDK keeps track of the last received version of the JSON document of #AWS\_IOT\_MY\_THING\_NAME shadow.

One exception to this version tracking is that the SDK ignores the version from update/accepted topic. Rest of the responses will be scanned to update the version number.

Reason behind this is, accepting version change for update/accepted may cause version conflicts for delta message if the update message is received before the delta.

Returns version number of the last received response.

|  |
| --- |
| uint32\_t aws\_iot\_shadow\_get\_last\_received\_version(void); |

## aws\_iot\_shadow\_enable\_discard\_old\_delta\_msgs()

This API enables the ignoring of delta messages with old version number.

As the MQTT is as protocol layer, there could be more than 1 of the same messages if we use QoS 0. To avoid getting called for the same message, this functionality should be enabled. If enabled, all the old messages will be ignored.

|  |
| --- |
| void aws\_iot\_shadow\_enable\_discard\_old\_delta\_msgs(void); |

## aws\_iot\_shadow\_disable\_discard\_old\_delta\_msgs()

This API disables the ignoring of delta messages with old version number.

|  |
| --- |
| void aws\_iot\_shadow\_disable\_discard\_old\_delta\_msgs(void); |

## aws\_iot\_shadow\_set\_autoreconnect\_status()

This API is the used to enable or disable autoreconnect feature. Any time a disconnect happens the underlying MQTT client attempts to reconnect if this is set to true.

Parameter pClient is MQTT Client used as the protocol layer, newStatus holds the value to set the autoreconnect option to.

Returns an IoT Error Type defining successful/failed operation.

|  |
| --- |
| IoT\_Error\_t aws\_iot\_shadow\_set\_autoreconnect\_status(AWS\_IoT\_Client \*pClient, bool newStatus); |

## aws\_iot\_shadow\_disconnect()

This API is used to disconnect from the AWS IoT Thing Shadow service over MQTT. This closes the underlying TCP connection.

Parameter pClient is MQTT Client used as the protocol layer.

Returns an IoT Error Type defining successful/failed disconnect status.

|  |
| --- |
| IoT\_Error\_t aws\_iot\_shadow\_disconnect(AWS\_IoT\_Client \*pClient); |

## aws\_iot\_shadow\_free()

This API is used to clean shadow client and free up memory that was dynamically allocated for the client.

Parameter pClient is MQTT Client that was previously created by calling aws\_iot\_shadow\_init().

Returns an IoT Error Type defining successful/failed freeing.

|  |
| --- |
| IoT\_Error\_t aws\_iot\_shadow\_disconnect(AWS\_IoT\_Client \*pClient); |

## aws\_iot\_shadow\_init\_json\_document()

This API initializes the JSON document with Shadow expected name/value and fills the JSON Buffer with a null terminated string. This function should always be used First, before using iot\_shadow\_add\_reported() and/or iot\_shadow\_add\_desired(), and finally iot\_finalize\_json\_document() is called.

The caller of the API needs to ensure the size of the buffer is enough to hold the entire JSON document.

Parameter pJsonDocument is the JSON document filled in this char buffer, maxSizeOfJsonDocument is maximum size of the pJsonDocument that can be used to fill the JSON document.

Returns an IoT Error Type defining if the buffer was null or the entire string was not filled up.

**Note**: The JSON library used for this SDK is JSMN which does not use any dynamic memory allocation.

|  |
| --- |
| IoT\_Error\_t aws\_iot\_shadow\_init\_json\_document(char \*pJsonDocument, size\_t maxSizeOfJsonDocument); |

## structure jsonStruct\_t

After the initialization of the JSON document, APIs iot\_shadow\_add\_reported() and/or iot\_shadow\_add\_desired(), are used to fill the JSON document’s reported or desired section with the values we want to report/desire. The relevant structure is as follows:

|  |
| --- |
| /\*\*  \* @brief This is the struct form of a JSON Key value pair  \*/  struct jsonStruct {  const char \*pKey; ///< JSON key  void \*pData; ///< pointer to the data (JSON value)  size\_t dataLength; ///< Length (in bytes) of pData  JsonPrimitiveType type; ///< type of JSON  jsonStructCallback\_t cb; ///< callback to be executed on receiving the Key value pair  };  /\*\*  \* @brief All the JSON object types enum  \*  \* JSON number types need to be split into proper integer / floating point data types and sizes on embedded platforms.  \*/  typedef enum {  SHADOW\_JSON\_INT32,  SHADOW\_JSON\_INT16,  SHADOW\_JSON\_INT8,  SHADOW\_JSON\_UINT32,  SHADOW\_JSON\_UINT16,  SHADOW\_JSON\_UINT8,  SHADOW\_JSON\_FLOAT,  SHADOW\_JSON\_DOUBLE,  SHADOW\_JSON\_BOOL,  SHADOW\_JSON\_STRING,  SHADOW\_JSON\_OBJECT  } JsonPrimitiveType; |

## aws\_iot\_shadow\_add\_reported()

This API is used to Add the reported section of the JSON document of jsonStruct\_t.

It API takes variable number of arguments; count is the number of jsonStruct\_t types that you would like to add in the reported section.

It adds "reported":{<all the values that needs to be added>} to the JSON document.

The caller of the API needs to ensure the size of the buffer is enough to hold the reported section + the init section. The JSON document buffer needs to be initialized using iot\_shadow\_init\_json\_document() before calling this API.

Parameter pJsonDocument is the JSON document filled in this char buffer, maxSizeOfJsonDocument is maximum size of the pJsonDocument that can be used to fill the JSON document.

Parameter count is total number of arguments (jsonStruct\_t object) passed in the arguments.

Returns an IoT Error Type defining if the buffer was null or the entire string was not filled up.

|  |
| --- |
| IoT\_Error\_t aws\_iot\_shadow\_add\_reported(char \*pJsonDocument, size\_t maxSizeOfJsonDocument, uint8\_t count, ...); |

## aws\_iot\_shadow\_add\_desired()

This API is used to Add the desired section of the JSON document of jsonStruct\_t.

It API takes variable number of arguments; count is the number of jsonStruct\_t types that you would like to add in the desired section.

It adds " desired “: {<all the values that needs to be added>} to the JSON document.

The caller of the API needs to ensure the size of the buffer is enough to hold the desired section + the init section. The JSON document buffer needs to be initialized using iot\_shadow\_init\_json\_document() before calling this API.

Parameter pJsonDocument is the JSON document filled in this char buffer, maxSizeOfJsonDocument is maximum size of the pJsonDocument that can be used to fill the JSON document.

Parameter count is total number of arguments (jsonStruct\_t object) passed in the arguments.

Returns an IoT Error Type defining if the buffer was null or the entire string was not filled up.

**Note**: Both ‘desired’ and ‘reported’ section are not mandatory. Most devices might just use the reported section.

|  |
| --- |
| IoT\_Error\_t aws\_iot\_shadow\_add\_desired(char \*pJsonDocument, size\_t maxSizeOfJsonDocument, uint8\_t count, ...); |

## aws\_iot\_finalize\_json\_document()

This API is used to finalize the JSON document with Shadow expected client Token and increments the client token every time this API is called.

The caller of the API needs to ensure the size of the buffer is enough to hold the entire JSON document.

This API is to be called after using iot\_shadow\_add\_reported() and/or iot\_shadow\_add\_desired(), otherwise the JSON document after ADD operation will not be valid.

Parameter pJsonDocument is the JSON document filled in this char buffer, maxSizeOfJsonDocument is maximum size of the pJsonDocument that can be used to fill the JSON document.

Returns an IoT Error Type defining if the buffer was null or the entire string was not filled up.

|  |
| --- |
| IoT\_Error\_t aws\_iot\_finalize\_json\_document(char \*pJsonDocument, size\_t maxSizeOfJsonDocument); |

When action Update is called after finalize, there could a situation of multiple other services trying to update the same shadow. To differentiate the services / device in such situation, a client token string is included to the request.

AWS\_IOT\_MQTT\_CLIENT\_ID with a sequence number to differentiate between our own previous update requests is used as a client token. It is of the form: "clientToken": "UniqueClientID+Seq". This is also taken care by [aws\_iot\_finalize\_json\_document()](http://aws-iot-device-sdk-embedded-c-docs.s3-website-us-east-1.amazonaws.com/aws__iot__shadow__json__data_8h.html#acc1c9fe2e514d9b46d659ad4161d0a54) APIs.

## aws\_iot\_fill\_with\_client\_token()

This API fills the given buffer with client token for tracking the Response.

It adds the AWS\_IOT\_MQTT\_CLIENT\_ID with a sequence number. Every time this function is used the sequence number gets incremented.

Parameter pBufferToBeUpdatedWithClientToken is the buffer to be updated with the client token string, maxSizeOfJsonDocument is maximum size of the pJsonDocument that can be used to fill the JSON document.

Returns an IoT Error Type defining if the buffer was null or the entire string was not filled up.

|  |
| --- |
| IoT\_Error\_t aws\_iot\_fill\_with\_client\_token(char \*pBufferToBeUpdatedWithClientToken, size\_t maxSizeOfJsonDocument); |

# Auto Reconnect Feature

If Auto Reconnect feature is enabled using API: aws\_iot\_shadow\_set\_autoreconnect\_status(), an attempt to reconnect is made as part of the next yield call at any time, a disconnect happens. On reconnecting the MQTT connection, all the topics will be re-subscribed.

The auto-reconnect feature could be enabled at any point of time after the aws\_iot\_shadow\_connect() is a success. It should not be enabled before aws\_iot\_shadow\_connect(). To verify if this feature is enabled or disabled, an underlying MQTT API aws\_iot\_is\_autoreconnect\_enabled() is used.

Exponential back-off is used to decide the time between two reconnect attempts.

There are two configuration parameters associated with exponential back-off:

1. AWS\_IOT\_MQTT\_MIN\_RECONNECT\_WAIT\_INTERVAL
2. AWS\_IOT\_MQTT\_MAX\_RECONNECT\_WAIT\_INTERVAL

Interval before every next try is multiplied by 2, starting with AWS\_IOT\_MQTT\_MIN\_RECONNECT\_WAIT\_INTERVAL.

After all the reconnect attempts fail based on the maximum back-off time, an attempt will be made every AWS\_IOT\_MQTT\_MAX\_RECONNECT\_WAIT\_INTERVAL.

In the following cases a network disconnect is detected:

1. As part of MQTT Keepalive functionality, if the Ping Response is not received back, then a disconnect is initiated and iot\_disconnect\_handler() is called.
2. If we are unable to send the Ping in the first place, then it is flagged as a disconnect.

Any time a disconnect is detected because of the keep alive logic then this disconnect handler is invoked. The iot\_disconnect\_handler() is invoked even if the auto-reconnect feature is enabled. It is invoked only once before the beginning of the reconnection attempt.

When auto-reconnect is attempted, API iot\_tls\_is\_connected() is called to check if the Physical Network is up and whether the TLS layer is connected or not. Every time before performing a TLS handshake, the return value of this function will be checked.

Yield return values could be one of these while using the reconnect feature:

1. NETWORK\_RECONNECTED
2. NETWORK\_ATTEMPTING\_RECONNECT
3. NETWORK\_RECONNECT\_TIMED\_OUT
4. NETWORK\_DISCONNECTED

**Note**: If the AWS IoT Embedded C Device SDK library is built with configuration network reconnect timeout enabled (#define AWS\_IOT\_MQTT\_DISABLE\_NETWORK\_RECONNECT\_TIME\_OUT 0), then auto reconnect behavior changes as detailed below.

After all reconnect attempt failure based on the maximum back-off time, NETWORK\_RECONNECT\_TIMED\_OUT is returned by aws\_iot\_mqtt\_yield(). There will be no longer reconnect attempts. If a reconnect is needed after this based on some external conditions then use aws\_iot\_mqtt\_attempt\_reconnect() API to reconnect and re-subscribe. This API could be manually used without turning on the auto-reconnect feature. It will attempt to reconnect only once. aws\_iot\_mqtt\_attempt\_reconnect() is a blocking call.

# About Alexa Smart Home Skill

‘Alexa Skills Kit’ can be used by the Customer to develop various Alexa controlled devices with different use cases.

In this application note demo, Alexa Smart Home Skill is used where voice interaction model is prebuilt and Smart Home Devices can be discovered and controlled using Amazon Alexa Smartphone App user interface.

<https://developer.amazon.com/en-US/docs/alexa/smarthome/understand-the-smart-home-skill-api.html>

Smart Home Skill also provide various APIs with predefined protocols as interface for different types of Smart Home Devices, For example: thermostat, temperature sensor, color controller and so on.

Similarly, there is Smart Home Security Skills with predefined voice interaction models and predefined protocols covering doorbell and lock control etc., use cases.

<https://developer.amazon.com/en-US/docs/alexa/device-apis/overview-smart-home-security.html>

There are around 25 predefined interfaces in Smart Home category, for which this sample code can be easily extended.

This application note demo skill uses PowerController interface and PowerState in the device shadow is updated based on Alexa commands from Alexa Voice Service.

<https://developer.amazon.com/en-US/docs/alexa/device-apis/alexa-powercontroller.html>

This is achieved by Alexa Smart Home Skill bridging between Alexa Voice Service (AVS) and AWS IoT Core service where the device endpoint resides. This shadow is replicated by the Talaria TWO running the example code with this application note.

To achieve this, customers will need to develop and deploy their own Alexa Skill.

A demo of this can be seen in action using ‘InnoPhase Smart Home Demo’ Alexa Skill and a device endpoint ‘InnoSwitch’ residing at InnoPhase AWS Endpoint Cloud. Talaria TWO EVB is used to connect to this device endpoint ‘InnoSwitch’ and be controlled by Alexa commands or Amazon Alexa Smartphone App. Next section describes how to setup this demo.

Custom Alexa Skills with customer’s own voice interaction models can also be built with your own protocol over AWS IoT Core Device Shadow Service.

# Setting up a Talaria TWO InnoSwitch Demo

The following section describes the steps needed to successfully setup the Alexa Ready Talaria TWO App with InnoPhase Smart Home Demo Alexa Skill.

Following steps are a one-time process for the individual using the skill to link the account and receive unique device name, device cert and device key or Talaria TWO ELF binary.

## Prerequisite

The User should already have an active Amazon account and an Alexa Application in User’s Smart Phone. Optionally, an Alexa enabled speaker like Echo can be used for voice interaction. Talaria TWO EVB and Talaria TWO Download Tool will be needed to program the board.

## Enable InnoPhase Smart Home Demo Alexa Skill

LWA (Login with Amazon) service from Amazon is used for securely linking the Alexa account to the Skill. There are two methods to enable Skill for your amazon account which are as follows:

### Method 1 - Enabling Via Browser

Login to amazon.com with the amazon account you want to link with the skill, and search for innophase alexa skill as shown in Figure 2.

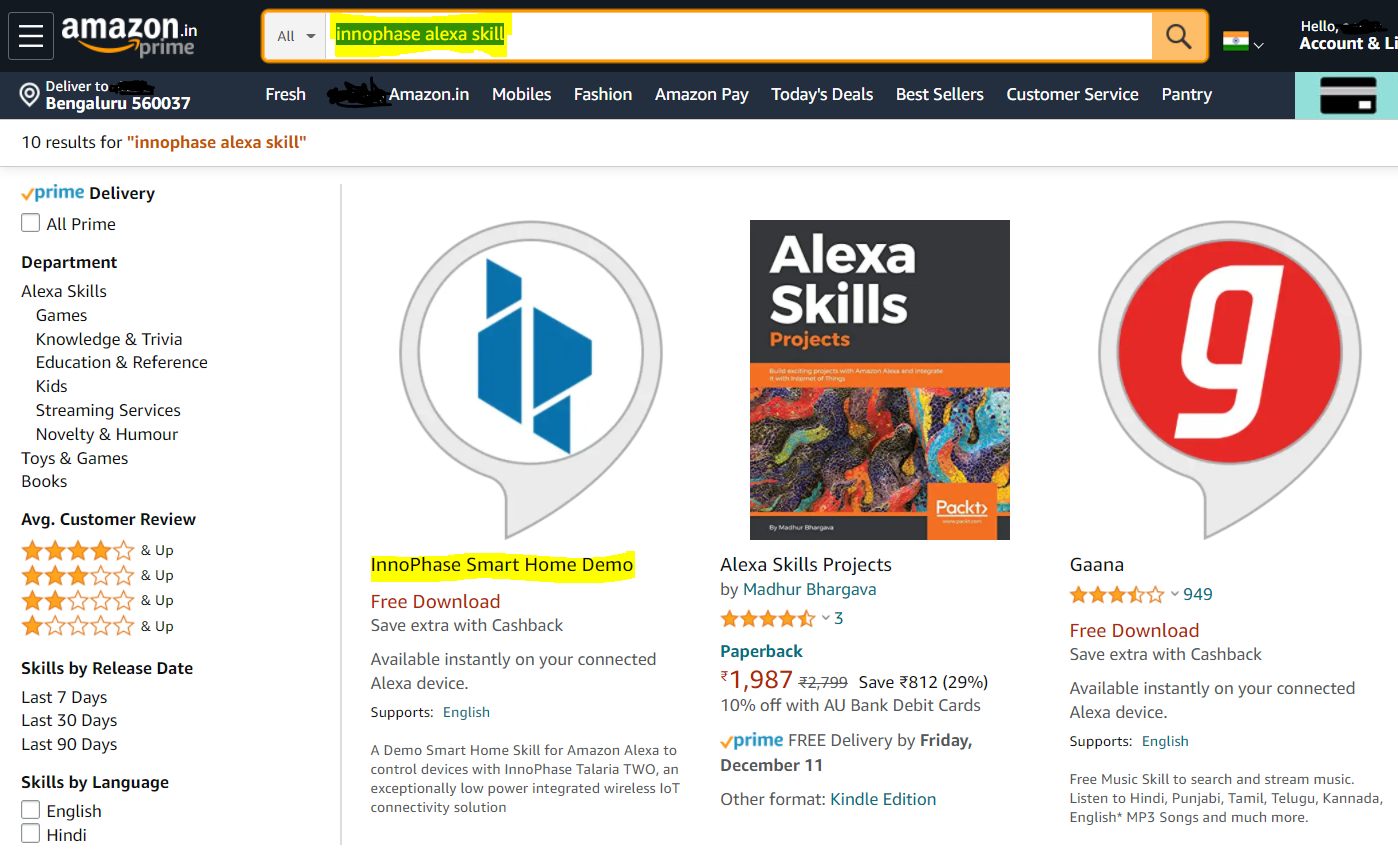


Figure : Locating and enabling skill via amazon.com in browser -- 1

Click the link of the skill and look for Enable. Once enabled from here, this skill will show up in Alexa App logged in to same account as well.

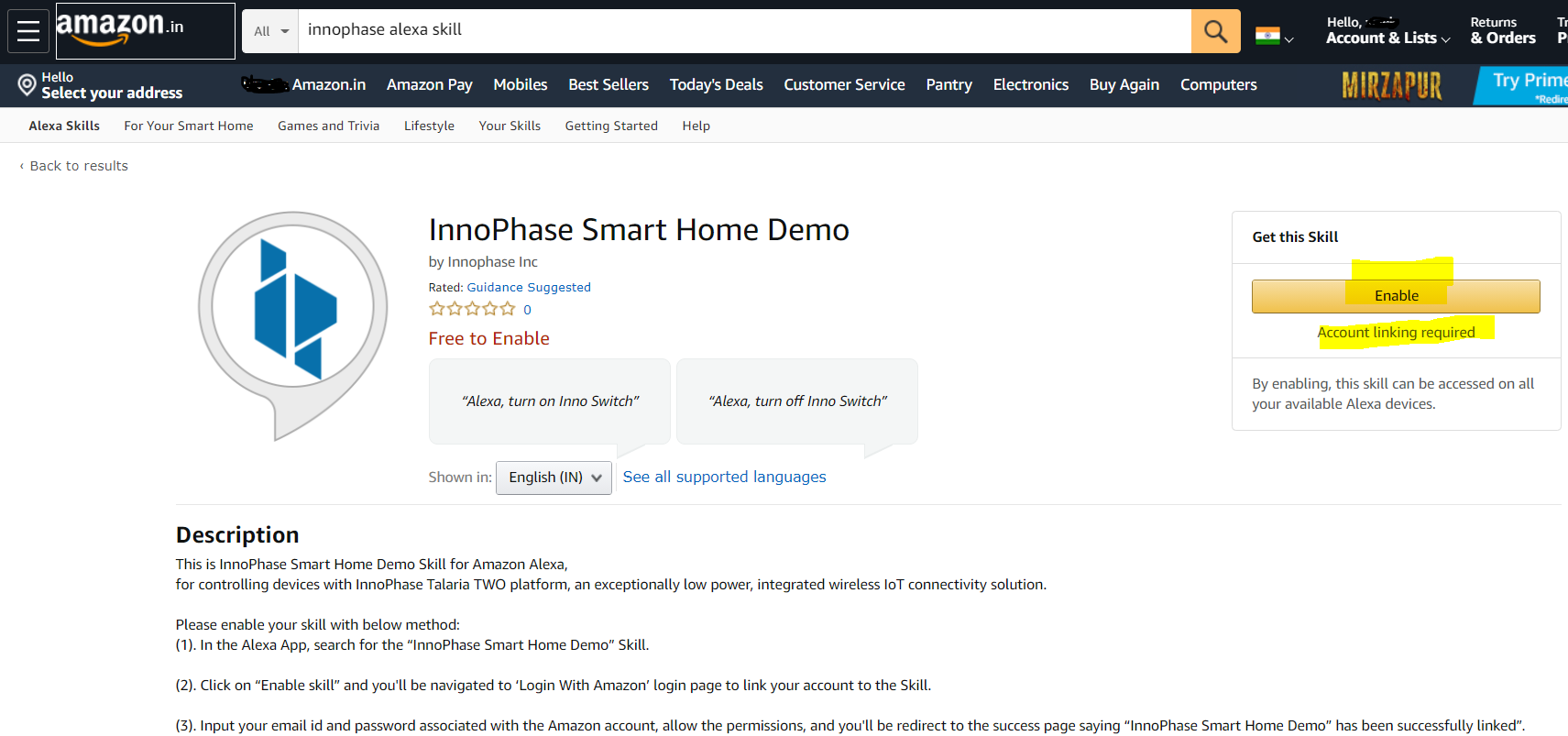


Figure : Locating and enabling skill via amazon.com in browser -- 2

### Method 2 - Enabling Via Amazon Alexa App

Search for InnoPhase Smart Home Demo from Alexa App Skill Section -> Browse Skills -> Search, locate the skill and tap Enable To Use, as shown in Figure 4.

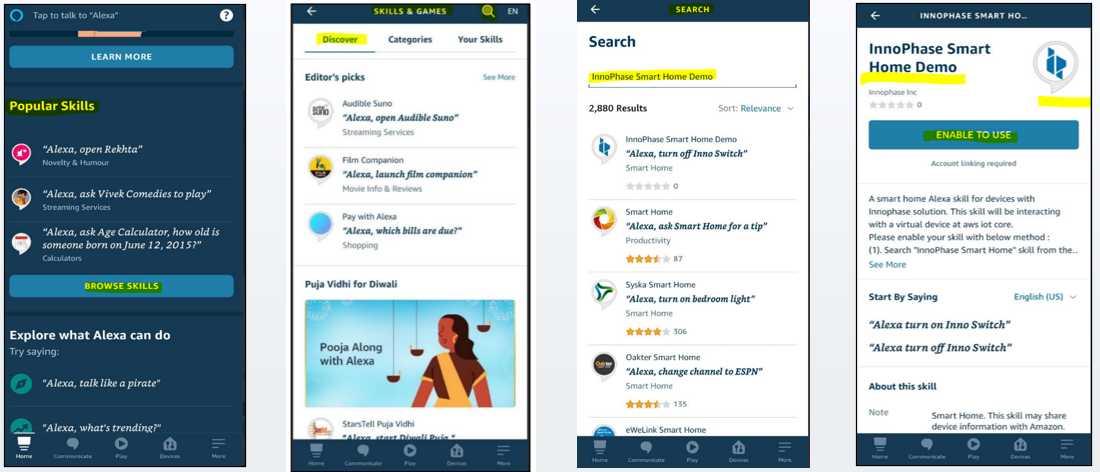


Figure : Locating and enabling skill via Alexa phone app

## Linking the Alexa account to the Skill

Follow the login instructions that appears on subsequent screens and allow the permissions as required as described below.

Clicking enable in previous steps will open a new page (or redirect to next screen) where it asks for an Amazon account, as shown in Figure 5.

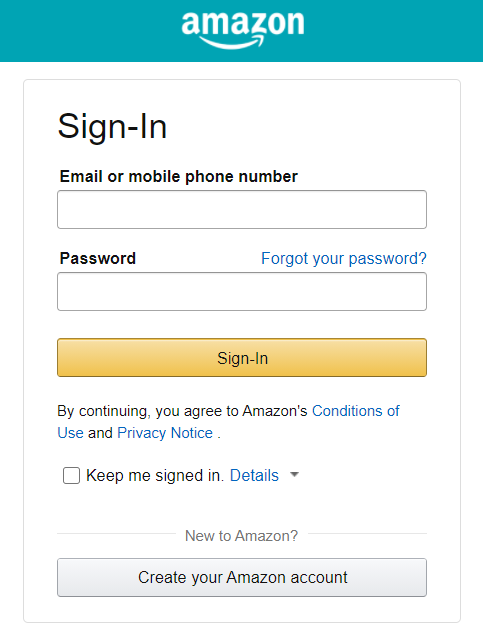


Figure : Amazon account – credentials asked for account linking

Ensure you use the same account you have used for Alexa app to login.

**Note**: If the User has already logged in to the Amazon account in the same browser but in a different tab, then the step in Figure 5 might be skipped and you will directly see a prompt as shown in Figure 6.

Provide permission for accessing email address and name by the Skill.

Graphical user interface

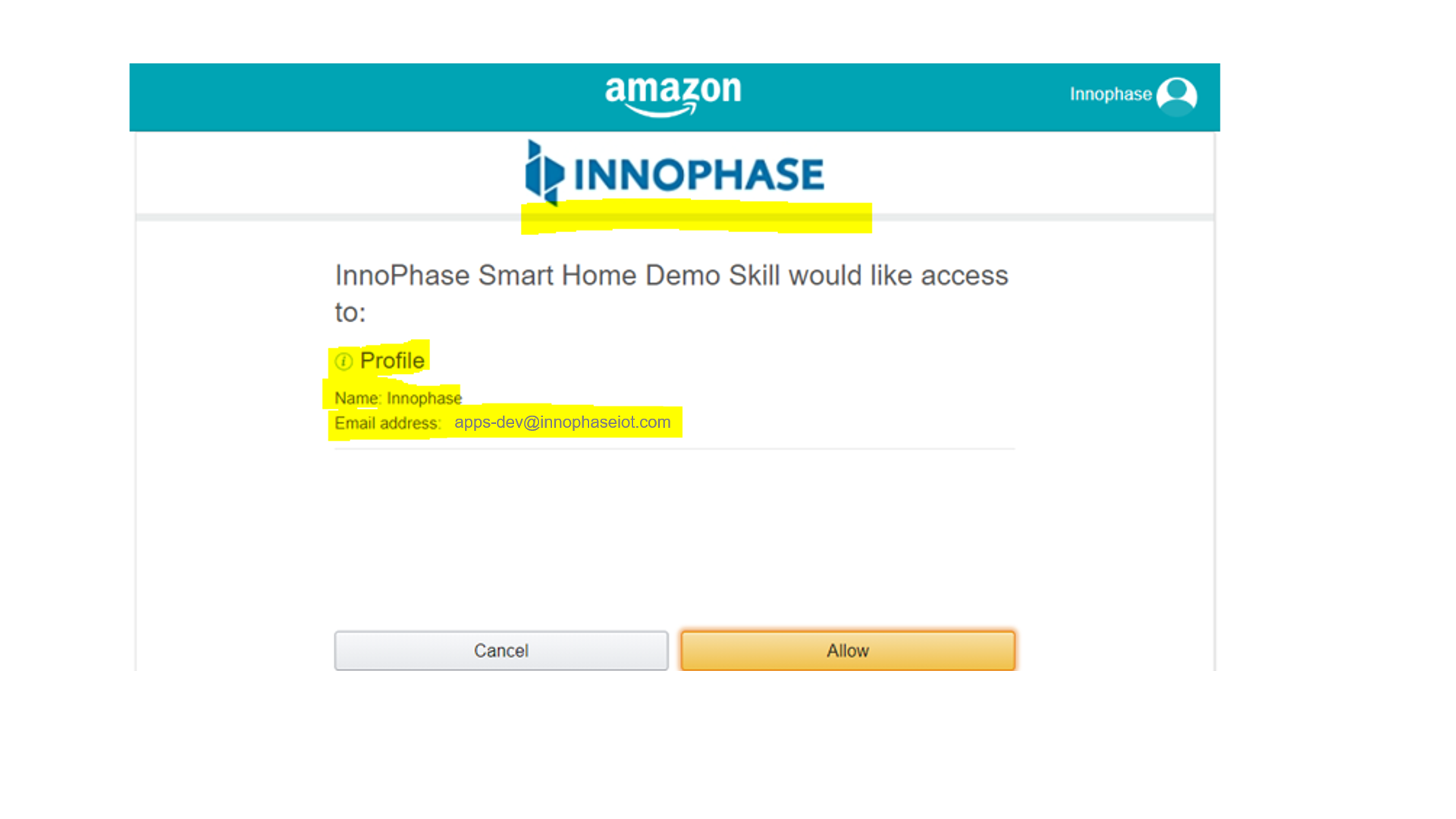
Description automatically generated with medium confidence

Figure : Permissions asked to access profile for Account Linking

**Note**: This permission is asked only for the first time of account linking. Disabling the Skill and enabling it again with account linking later might not pop-up a prompt looking exactly as shown in this figure. Instead, the user might see a variant of this prompt or this prompt might be skipped altogether.

This will lead to successfully linking your Amazon Account to InnoPhase Smart Home Demo Skill and a confirmation screen like Figure 7 would appear. This completes the Alexa skill setup in your Amazon account.

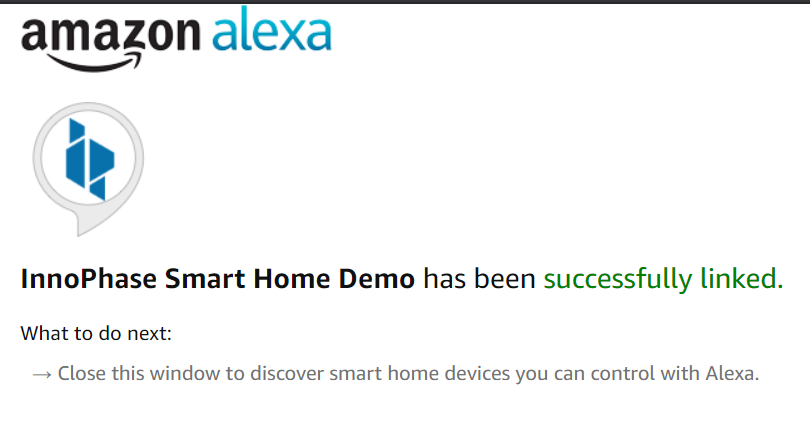


Figure : Amazon Alexa Account Successfully linked to the InnoPhase Smart Home Demo Alexa Skill

## Request for AWS IoT Thing creation & Certs created for the Thing

Please communicate the email-id associated with your Amazon Account to [apps-dev@innophaseiot.com](mailto:apps-dev@innophaseiot.com) with the subject line -- ‘Request for T2 Alexa End Point’.

In response, an email with the certificates, keys created for the thing and a ‘Thing Name’ similar to ‘INNO\_ENDPOINT\_ABCD1234’ will be provided. The certificates, keys and ‘Thing Name’ are unique to your account.

There are two ways in which the ELF is made available:

1. As part of FreeRTOS SDK package in the <freertos\_sdk>/binaries/eval/Alexa\_ready/bin path
2. In case you do not have the FreeRTOS SDK package, the ELF can be sent along with certs and Thing in response to the request email for Alexa Ready App.

**Note**: Make sure you can login to Alexa Smart Phone App with your Amazon Account.

# Programming Applications

## Programming Talaria TWO board with certificates

Program the ELFs, certificate and key onto Talaria TWO using the Download tool.

Launch the Download tool provided with InnoPhase Talaria TWO FreeRTOS SDK: (*freertos\_sdk\_x.y\pc\_tools\Download\_Tool\bin*).

**Note**: x and y in freertos\_sdk\_x.y refers to the FreeRTOS SDK release version.

### Show File System Contents

Click on Show File System Contents to see the current available files in the file system.

### Writing Files into File System

The user needs to add three files in file system:

1. aws\_device\_cert
2. aws\_root\_ca
3. aws\_device\_pkey

Users should rename certificates and key received in the mail with the above provided name.

For example: 5497cf0b16-private.pem.key must be renamed to aws\_device\_pkey.

To write files into Talaria TWO, user must create a folder with the name data and must create a sub folder (/data/certs/aws/alexa\_ready) which is the default sub-folder used and place all certificates, keys into it. Using the Download tool, files must be written to file system.

For more information on using the Download Tool, refer: UG\_Download\_Tool.pdf (*freertos\_sdk\_x.y\pc\_tools\Download\_Tool\doc).*

## Programming Talaria TWO board with ELF

There are two Talaria TWO applications available in the following location of the FreeRTOS SDK release package: *freertos\_sdk\_x.y\binaries\eval\alexa\_ready\bin*.

1. innoswitch.elf
2. innoswitch\_ble\_provisionig.elf

The difference amongst the two is, while using innoswitch\_ble\_provisioning.elf , the AP details (SSID, Passphrase) can be provisioned to the Talaria TWO application from a mobile application instead of passing it from Download Tool.

### Programming Talaria TWO board with innoswitch.elf

Program innoswitch.elf (*freertos\_sdk\_x.y\examples\watchdog\_timer\bin*) using the Download tool:

1. Launch the Download tool provided with InnoPhase Talaria TWO SDK.
2. In the GUI window:
   1. Boot Target: Select the appropriate EVK from the drop-down.
   2. ELF Input: Load the innoswitch.elf by clicking on Select ELF File.
   3. AP Options: Provide the appropriate SSID and Passphrase to connect to an Access Point.
   4. Boot Arguments: Pass the following boot arguments:

|  |
| --- |
| aws\_host=a3t0o11ohwlo2h-ats.iot.us-east-1.amazonaws.com, aws\_port=8883,suspend=1,no\_mcast=1,thing\_name=INNO\_ENDPOINT\_xxxxxxxx |

where,

1. aws\_host: Custom AWS location
2. aws\_port=8883
3. no\_mcast: (0 or 1) Flag for enabling/disabling reception of multicast frames.
4. suspend=1 for power save
5. thing\_name: Unique Thing name received. Replace xxxxxx with appropriate details.

**Note**:

1. Ensure correct boot parameters are supplied to your Wi-Fi network and the information from the device/thing created previously on AWS.
2. In addition to the above-mentioned boot arguments, high DTIM specific tests can be run by providing the following boot arguments and extending the timeouts as required:

These boot arguments are used to configure the timeouts in AWS IoT SDK.

|  |
| --- |
| aws\_keepAliveIntervalInSec=120,aws\_tlsHandshakeTimeout\_ms=60000,aws\_mqttCommandTimeout\_ms=60000,aws\_sslRead\_timeout\_ms=10,aws\_sslReadRetry\_timeout\_ms=20,aws\_sslWriteRetry\_timeout\_ms=20,aws\_shaddowActionAckTimeout\_sec=60,aws\_mqttPingResponseTimeout\_sec=60 |

* 1. Programming: Prog RAM or Prog Flash as per requirement.

Console log after programming:

|  |
| --- |
| Y-BOOT 208ef13 2019-07-22 12:26:54 -0500 790da1-b-7  ROM yoda-h0-rom-16-0-gd5a8e586  FLASH:PNWWWWWAEBuild $Id: git-b664be2af $  aws\_host=a3t0o11ohwlo2h-ats.iot.us-east-1.amazonaws.com aws\_port=8883 suspend=1 no\_mcast=< 1 thing\_name=INNO\_ENDPOINT\_6GBTSRZ8 np\_conf\_path=/sys/nprofile.json ssid=InnoPhase passphrase=43083191  $App:git-38ca4ab7  SDK Ver: FREERTOS\_SDK\_1.0  Innoswitch Demo App  Mounting file system  read\_certs() success  addr e0:69:3a:00:2c:3e  added network profile successfully, will try connecting..  [2.769,774] CONNECT:d2:01:2a:d2:4a:2d Channel:11 rssi:-24 dBm  wcm\_notify\_cb to App Layer - WCM\_NOTIFY\_MSG\_LINK\_UP  wcm\_notify\_cb to App Layer - WCM\_NOTIFY\_MSG\_ADDRESS  [4.805,318] MYIP 192.168.224.237  [4.805,402] IPv6 [fe80::e269:3aff:fe00:2c3e]-link  wcm\_notify\_cb to App Layer - WCM\_NOTIFY\_MSG\_CONNECTED  Shadow Connect  Root Done[0]Loading the client cert. and key. size TLSDataParams:2080  Loading the client cert done.... ret[0]  Client pkey loaded[0]  . Connecting to a3t0o11ohwlo2h-ats.iot.us-east-1.amazonaws.com/8883... ok  . Setting up the SSL/TLS structure... This certificate has no flags  This certificate has no flags  This certificate has no flags  SSL/TLS handshake. DONE ..ret:0  ok  [ Protocol is TLSv1.2 ]  [ Ciphersuite is TLS-ECDHE-RSA-WITH-AES-128-GCM-SHA256 ]  [ Record expansion is 29 ]  . Verifying peer X.509 certificate...  ok  Shadow Connected  init\_and\_connect\_aws\_iot. ret:0  LED Off  Update Shadow: {"state":{"reported":{"powerState":"OFF"}}, "clientToken":"INNO\_ENDPOINT\_6GBTSRZ8-0"}  Delta - Switch state changed to ON  LED On  Update Shadow: {"state":{"reported":{"powerState":"ON"}}, "clientToken":"INNO\_ENDPOINT\_6GBTSRZ8-1"}  Update Accepted !!  Update Accepted !!  Delta - Switch state changed to OFF  LED Off  Update Shadow: {"state":{"reported":{"powerState":"OFF"}}, "clientToken":"INNO\_ENDPOINT\_6GBTSRZ8-2"}  Update Accepted !!  Delta - Switch state changed to ON  LED On  Update Shadow: {"state":{"reported":{"powerState":"ON"}}, "clientToken":"INNO\_ENDPOINT\_6GBTSRZ8-3"}  Update Accepted !!  Delta - Switch state changed to OFF  LED Off  Update Shadow: {"state":{"reported":{"powerState":"OFF"}}, "clientToken":"INNO\_ENDPOINT\_6GBTSRZ8-4"}  Update Accepted !!  Delta - Switch state changed to ON  LED On  Update Shadow: {"state":{"reported":{"powerState":"ON"}}, "clientToken":"INNO\_ENDPOINT\_6GBTSRZ8-5"}  Update Accepted !!  Delta - Switch state changed to OFF  LED Off  Update Shadow: {"state":{"reported":{"powerState":"OFF"}}, "clientToken":"INNO\_ENDPOINT\_6GBTSRZ8-6"}  Update Accepted !!  Delta - Switch state changed to ON  LED On  Update Shadow: {"state":{"reported":{"powerState":"ON"}}, "clientToken":"INNO\_ENDPOINT\_6GBTSRZ8-7"}  Update Accepted !!  Delta - Switch state changed to OFF  LED Off  Update Shadow: {"state":{"reported":{"powerState":"OFF"}}, "clientToken":"INNO\_ENDPOINT\_6GBTSRZ8-8"}  Update Accepted !!  Delta - Switch state changed to ON  LED On  Update Shadow: {"state":{"reported":{"powerState":"ON"}}, "clientToken":"INNO\_ENDPOINT\_6GBTSRZ8-9"}  Update Accepted !!  Delta - Switch state changed to OFF  LED Off  Update Shadow: {"state":{"reported":{"powerState":"OFF"}}, "clientToken":"INNO\_ENDPOINT\_6GBTSRZ8-10"}  Update Accepted !! |

### Programming Talaria TWO board with innoswitch\_ble\_provisioning.elf

Write the part.json file to the data folder using Write Files in the Download Tool. Once done, program innoswitch\_with\_bleProvisioning.elf (refer steps from section: *Programming Talaria TWO board with innoswitch.elf* to program the ELF onto Talaria TWO).

**Note**:

1. To reprovision the Talaria TWO module, write the part.json file onto Talaria TWO filesystem using Write Files.
2. For connecting the Talaria TWO to an AP, the SSID and Passphrase are provisioned to Talaria TWO through BLE from a mobile application as mentioned in the following section (section: *Using InnoPhase Talaria TWO Smart Home Application*). Ensure to keep the SSID and Passphrase fields in the Download Tool empty.

#### Using InnoPhase Talaria TWO Smart Home Application

To test this sample application (innoswitch\_ble\_provisionig.elf ), the companion Innophase T2 Smart Home Android application can be used from either an Android or iOS device.

1. To install, open the provided .apk file (*freertos\_sdk\_x.y\apps\ble\_provisioning\mobile\_app*) from the phone (Android or iOS).

**Note**: x and y in *freertos\_*sdk\_x.y refer to the FreeRTOS SDK release version.

1. To connect to the Talaria TWO BLE Server, wait for the application to complete the scanning and look for Inno\_Ble\_WiFiProvisioning and click on it.



Figure : Android - Scanning for Talaria TWO BLE Server for Wi-Fi Provisioning

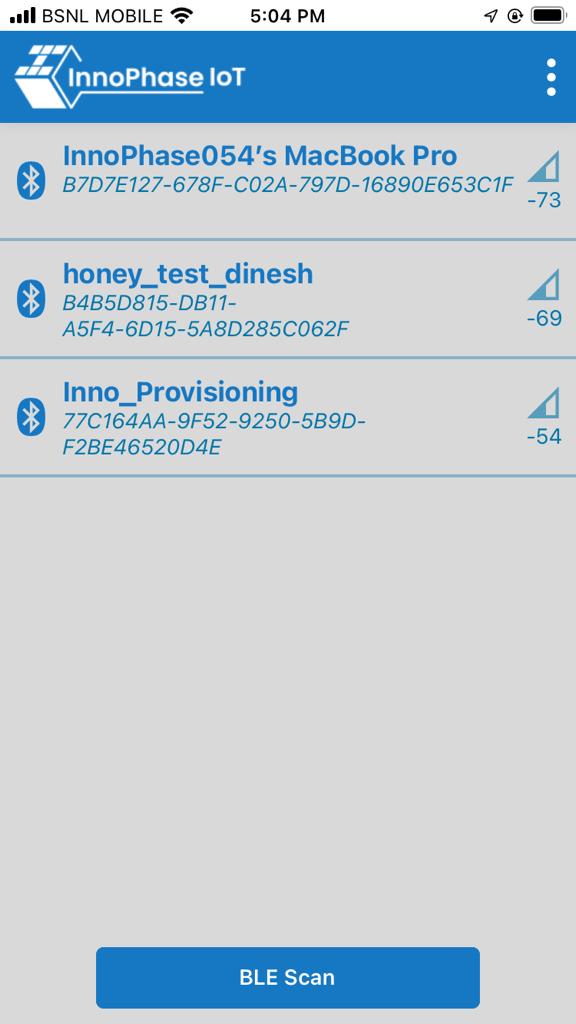


Figure : iOS - Scanning for Talaria TWO BLE Server for Wi-Fi Provisioning

Android phone connects as a BLE Client to Talaria TWO device at this stage.

1. Android application scans for the nearby available Wi-Fi networks and displays them in a list view.

Graphical user interface, text, application, email

Description automatically generated

Figure : Android - Available Wi-Fi networks as scanned by Android Phone

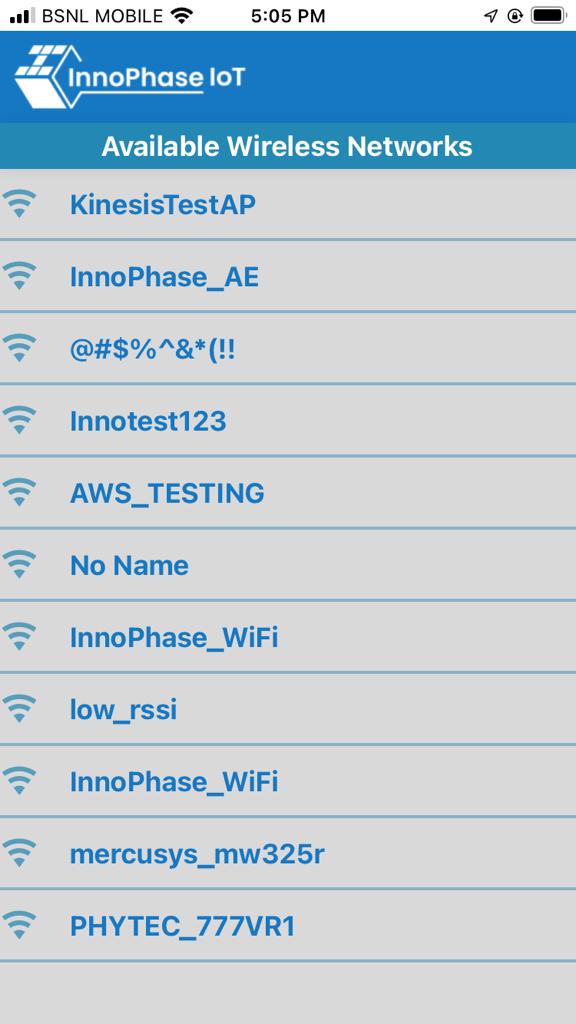


Figure : iOS - Available Wi-Fi networks as scanned by Android Phone

1. Select the SSID of the AP you want to connect to. A passphrase needs to be provided for the SSID.

Graphical user interface, text, application

Description automatically generated

Figure : Android - Providing the passphrase

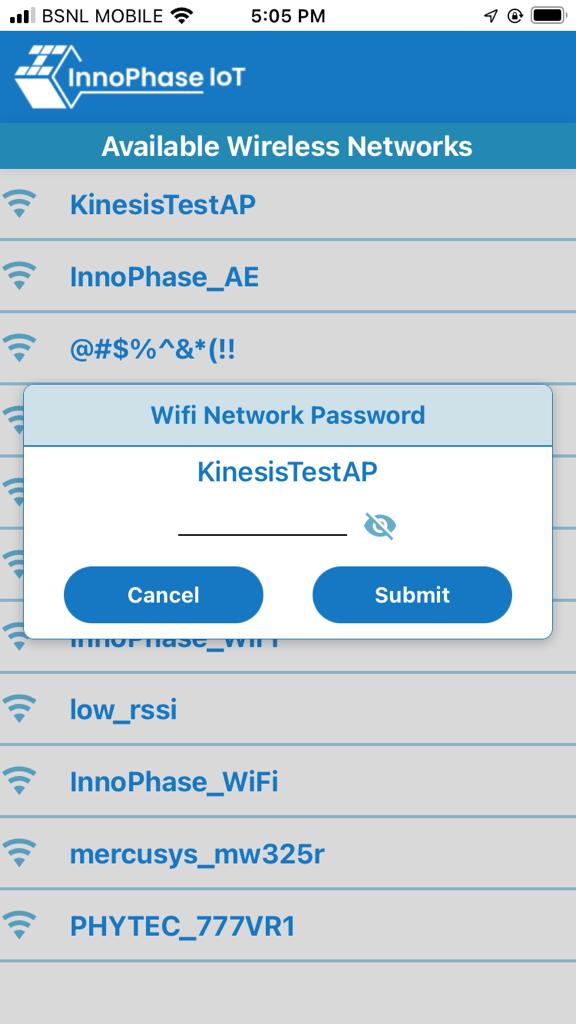


Figure : iOS - Providing the passphrase

1. Once the passphrase is entered, click on Done. If the provided passphrase is correct, connection is established successfully. If not, an error message is shown.

Graphical user interface, application, PowerPoint

Description automatically generated

Figure : Android - Connecting successful

Graphical user interface, text, application, chat or text message

Description automatically generated

Figure : iOS - Connecting successful

Graphical user interface, application

Description automatically generated

Figure : Android - Error in connection

Graphical user interface, text, application, chat or text message

Description automatically generated

Figure : iOS - Error in connection

1. On establishing the connection successfully, the android application should transfer the Wi-Fi credentials using custom GATT Service and Characteristics we created.

Shape

Description automatically generated with low confidence

Figure : Android - Connection successful

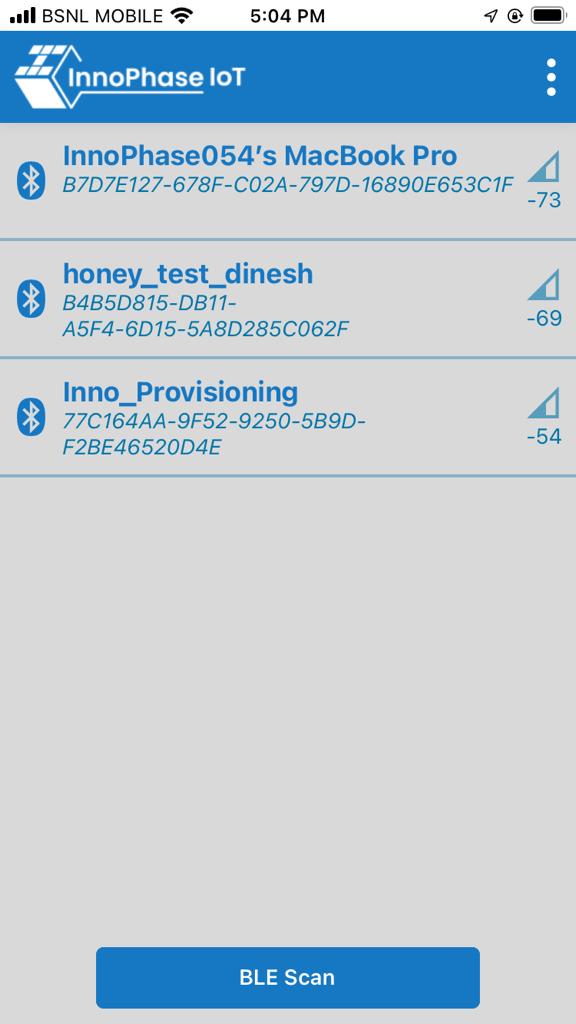


Figure : iOS - Connection successful

Talaria TWO will try to connect to the provisioned network and provide the following console output:

|  |
| --- |
| UART:SNWWWWWAEBuild $Id: git-65f6c1f46 $  aws\_host=a3t0o11ohwlo2h-ats.iot.us-east-1.amazonaws.com aws\_port=8883 suspend=1 no\_mcast=< 1 thing\_name=INNO\_ENDPOINT\_6GBTSRZ8  Inno\_Ble\_WiFiProvisioning started  [63.146,804] BT connect[0]: ia:60:4d:89:ec:f3:51 aa:05:04:03:02:01:00 phy2:0/0 phyC:00  Client connected  client reading status:waiting  WiFi Details SSID: InnoPhase, PASSWORD: 43083191  addr e0:69:3a:00:13:90  client reading status:waiting  client reading status:waiting  Connecting to WiFi...  added network successfully, will try connecting..  connecting to network status: 0  connection attempt timer started. current timein microseconds:[66337388]  [66.879,059] CONNECT:00:5f:67:cd:c5:a6 Channel:6 rssi:-32 dBm  wcm\_notify\_cb to App Layer - WCM\_NOTIFY\_MSG\_LINK\_UP  client reading status:waiting  wcm\_notify\_cb to App Layer - WCM\_NOTIFY\_MSG\_ADDRESS  [67.638,187] MYIP 192.168.0.104  [67.638,466] IPv6 [fe80::e269:3aff:fe00:1390]-link  wcm\_notify\_cb to App Layer - WCM\_NOTIFY\_MSG\_CONNECTED  Cancelling the connection timeout timer. current timein microseconds:[67639064]  client reading status : success  status sent to phone app, now calling common\_server\_destroy and bt\_gap\_destroy  $App:git-cd11dc34  SDK Ver: FREERTOS\_SDK\_1.0  Innoswitch Demo App  Mounting file system  read\_certs() success  Shadow Connect  Root Done[0]Loading the client cert. and key. size TLSDataParams:2080  Loading the client cert done.... ret[0]  Client pkey loaded[0]  . Connecting to a3t0o11ohwlo2h-ats.iot.us-east-1.amazonaws.com/8883... ok  . Setting up the SSL/TLS structure... This certificate has no flags  This certificate has no flags  This certificate has no flags  SSL/TLS handshake. DONE ..ret:0  ok  [ Protocol is TLSv1.2 ]  [ Ciphersuite is TLS-ECDHE-RSA-WITH-AES-128-GCM-SHA256 ]  [ Record expansion is 29 ]  . Verifying peer X.509 certificate...  ok  Shadow Connected  init\_and\_connect\_aws\_iot. ret:0  LED Off  Update Shadow: {"state":{"reported":{"powerState":"OFF"}}, "clientToken":"INNO\_ENDPOINT\_6GBTSRZ8-0"}  Delta - Switch state changed to ON  LED On  Update Shadow: {"state":{"reported":{"powerState":"ON"}}, "clientToken":"INNO\_ENDPOINT\_6GBTSRZ8-1"}  Update Accepted !!  Update Accepted !! |

## Jumper Setting on Talaria TWO EVB

This example uses GPIO 14 to toggle the LED D1. Ensure jumper J3 is installed which connects GPIO14 and LED.

## Interacting the Talaria TWO EVB with Alexa

Inno Switch can be controlled either using Amazon’s Alexa App installed in Android Phone or iOS Phone, or using the Alexa Voice Interactions with the Alexa speaker which is linked with User’s Alexa account.

For testing with a Phone, go to the Devices -> Switches and refresh the page if the Inno Switch is not found here.

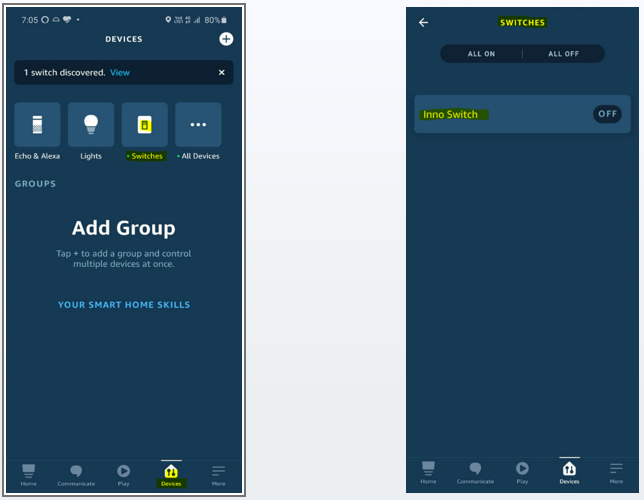


Figure : Devices – Switches -- InnoSwitch

Tap on Inno Switch to find power on/off control. The switch can be controlled from here by tapping on the Power Button, and the results will be reflected in LED status and Console of Talaria TWO EVB.

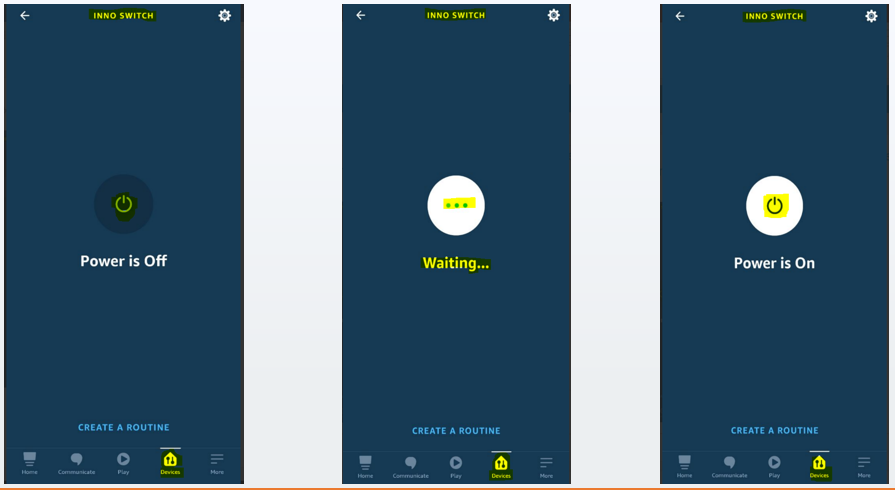


Figure : InnoSwitch power on-off control

Optionally, for controlling the device with Voice Interaction, you will need an Alexa enabled speaker (e.g., Amazon Echo) logged in with same Amazon Account.

Following voice commands are used to control the switch operation:

1. Alexa, turn on the Inno Switch
2. Alexa, turn off the Inno Switch

Console log while interacting with the device is as follows:

|  |
| --- |
| Y-BOOT 208ef13 2019-07-22 12:26:54 -0500 790da1-b-7  ROM yoda-h0-rom-16-0-gd5a8e586  FLASH:PNWWWWWAEBuild $Id: git-b664be2af $  aws\_host=a3t0o11ohwlo2h-ats.iot.us-east-1.amazonaws.com aws\_port=8883 suspend=1 no\_mcast=< 1 thing\_name=INNO\_ENDPOINT\_6GBTSRZ8  Inno\_Ble\_WiFiProvisioning started  [22.113,536] BT connect[0]: ia:7c:36:ff:b4:67:18 aa:05:04:03:02:01:00 phy2:0/0 phyC:00  Client connected  client reading status:waiting  WiFi Details SSID:InnoPhase, PASSWORD: 43083191  addr e0:69:3a:00:2c:3e  client reading status:waiting  client reading status:waiting  Connecting to WiFi...  added network successfully, will try connecting..  connecting to network status: 0  connection attempt timer started. current timein microseconds:[25293138]  [25.832,396] CONNECT:e8:48:b8:fb:35:70 Channel:6 rssi:-71 dBm  wcm\_notify\_cb to App Layer - WCM\_NOTIFY\_MSG\_LINK\_UP  client reading status:waiting  wcm\_notify\_cb to App Layer - WCM\_NOTIFY\_MSG\_ADDRESS  [26.705,681] MYIP 192.168.0.116  [26.705,845] IPv6 [fe80::e269:3aff:fe00:2c3e]-link  wcm\_notify\_cb to App Layer - WCM\_NOTIFY\_MSG\_CONNECTED  Cancelling the connection timeout timer. current timein microseconds:[26706029]  client reading status : success  status sent to phone app, now calling common\_server\_destroy and bt\_gap\_destroy  $App:git-38ca4ab7  SDK Ver: FREERTOS\_SDK\_1.0  Innoswitch Demo App  Mounting file system  read\_certs() success  Shadow Connect  Root Done[0]Loading the client cert. and key. size TLSDataParams:2080  Loading the client cert done.... ret[0]  Client pkey loaded[0]  . Connecting to a3t0o11ohwlo2h-ats.iot.us-east-1.amazonaws.com/8883... ok  . Setting up the SSL/TLS structure... This certificate has no flags  This certificate has no flags  This certificate has no flags  SSL/TLS handshake. DONE ..ret:0  ok  [ Protocol is TLSv1.2 ]  [ Ciphersuite is TLS-ECDHE-RSA-WITH-AES-128-GCM-SHA256 ]  [ Record expansion is 29 ]  . Verifying peer X.509 certificate...  ok  Shadow Connected  init\_and\_connect\_aws\_iot. ret:0  LED Off  Update Shadow: {"state":{"reported":{"powerState":"OFF"}}, "clientToken":"INNO\_ENDPOINT\_6GBTSRZ8-0"}  Update Accepted !!  Delta - Switch state changed to ON  LED On  Update Shadow: {"state":{"reported":{"powerState":"ON"}}, "clientToken":"INNO\_ENDPOINT\_6GBTSRZ8-1"}  Update Accepted !!  Delta - Switch state changed to OFF  LED Off  Update Shadow: {"state":{"reported":{"powerState":"OFF"}}, "clientToken":"INNO\_ENDPOINT\_6GBTSRZ8-2"}  Update Accepted !!  Delta - Switch state changed to ON  LED On  Update Shadow: {"state":{"reported":{"powerState":"ON"}}, "clientToken":"INNO\_ENDPOINT\_6GBTSRZ8-3"}  Update Accepted !!  Delta - Switch state changed to OFF  LED Off  Update Shadow: {"state":{"reported":{"powerState":"OFF"}}, "clientToken":"INNO\_ENDPOINT\_6GBTSRZ8-4"}  Update Accepted !! |