B-L4S5I-IOT01A X-CUBE-AWS Alexa Skill Getting Started

Getting started X-CUBE-AWS and Alexa on B-L4S5I-IOT01A

Introduction

This document describes how to get started with Alexa skills using X-CUBE-AWS on the B-L4S5I-IOT01A Discovery Board.

The reference is Amazon AWS FreeRTOS web documentation. This document just points to the web pages and does not reproduce them. It adds a few hints and tips to use the instructions from Amazon.

Contents

1 Presentation

This document entails how to connect the B-L4S5I-IOT01A with Alexa and run a custom Alexa application. It explains the process of creating the application and the AWS lambda function for the application.

2 Setting up development environment

2.1 Hardware

2.1.1 Required hardware

The following hardware is needed for this project:

1. B-L4S5I-IOT01A Discovery Board
2. A development PC
3. Micro-USB cable to connect the STM32 board to the development PC
4. A mobile device (phone or tablet) compatible with BLE 5.0 with Android or iOS operating system
5. Amazon Alexa enabled smart speaker

Notes:

- The role of the Amazon Alexa speaker can be fulfilled with the either the PC or the Mobile phone but is included in this demo for instructional purposes

2.2 Software tools

2.2.1 STM32CubeIDE

STM32CubeIDE (>=1.4.0) is required to compile the bootloader, kernel, drivers, and all applications either with the GUI or with cmake command-line tool.

Install STM32CubeIDE 1.4.0 from ST site:

https://www.st.com/content/st_com/en/products/development-tools/software-developmenttools/stm32-software-development-tools/stm32-ides/stm32cubeide.html

Once STM32CubeIDE is installed, allow it to update to latest version (1.4.2).

2.2.2 STM32CubeProgrammer

STM32CubeProgrammer (at least 2.4.0) is required to post-process the SBSFU secure bootloader and to program the Flash memory of P-NUCLEO-WB55.Nucleo board.

Install STM32CubeProgrammer from ST site:

https://www.st.com/content/st_com/en/products/development-tools/software-development-tools/stm32-software-development-tools/stm32-programmers/stm32cubeprog.html

Note: Java Runtime Environment is needed for STM32CubeProgrammer. See STM32CubeProgrammer installation instructions.

2.2.3 Terminal emulator

A serial terminal software like Tera Term is required to monitor serial communication between the B-L4S5I-IOT01A and the development PC.

2.2.4 OpenSSL

If you want to test Over The Air update, OpenSSL is needed for certificate creation.

If you have Git for Windows installed, OpenSSL is included and available from the Bash prompt.

2.2.5 AWS CLI

In case of OTA update, AWS CLI 2.0 is needed to import OTA certificate in AWS.

2.2.6 Development Scripts

The Dev Scripts will automate a lot of the configuration steps.

2.2.7 Python and Pyserial

The latest version of Python will be used in combination with the development scripts in order to automate configuration steps. Once python is installed ensure that it is added to path, and in command line run the command:

```
pip install pyserial
```

to install the required pyserial libraries.

2.3 AWS configuration

2.3.1 AWS IoT and CLI

There are several AWS operations to perform before compiling the applications.

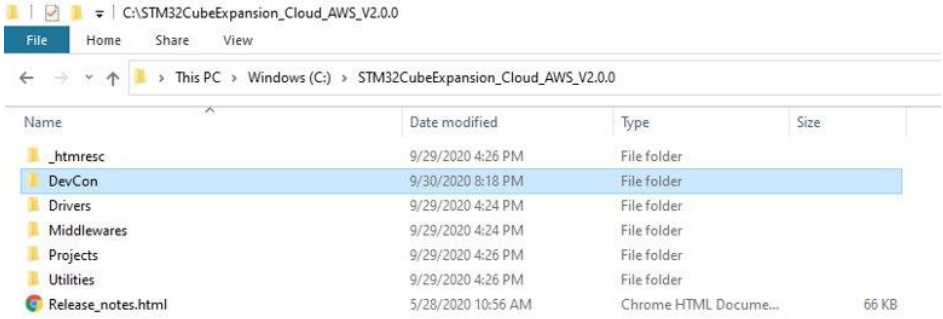
They are required for all applications.

1. You must have an AWS account. See AWS create account
2. It is recommended to not use the main root user account for standard operations. Create a user without permissions.
3. Note the AWS Access Key, AWS Secret Access Key, and region to Configure AWS CLI Profile

2.4 Amazon FreeRTOS applications

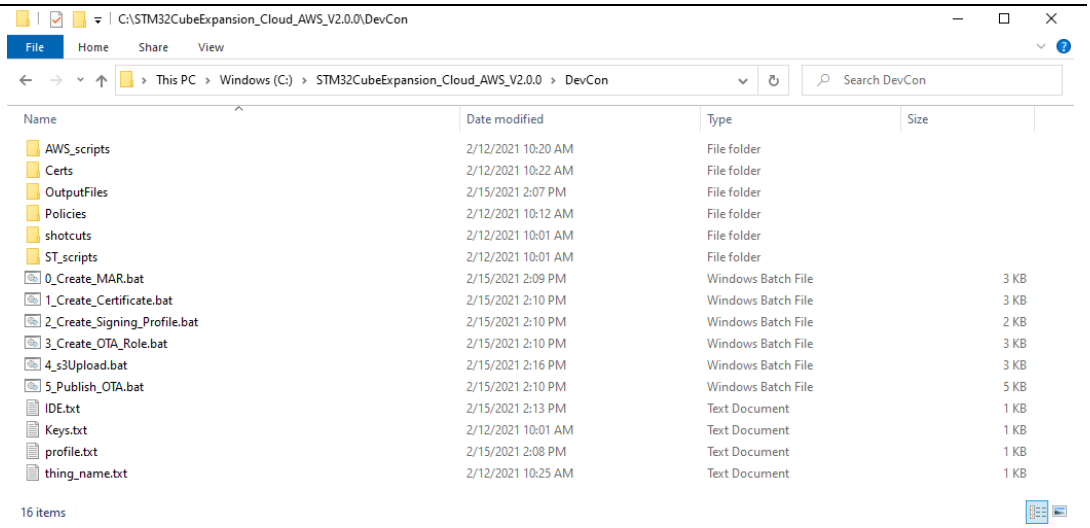
2.4.1 Sources

The X-CUBE-AWS should be placed directly on the C: drive of your development PC, and the Development scripts should be placed in the main directory of the X-CUBE-AWS package



2.4.2 Development Scripts Description

The Development Scripts are useful for automating the configuration steps and procedure required to connect the B-L4S5I-IOT01A to your AWS console.



AWS_scripts

This folder contains all the python and batch scripts used to interact with the AWS IoT Console.

Certs

This folder will serve as the storage location for all the certificates required for registering and updating your B-L4S5I-IOT01A with AWS.

OutputFiles

All of the configuration event logs, and text output will be saved in this directory for your review.

Policies

This directory will store the various security policies that are created during the B-L4S5I-IOT01A setup/configuration.

Shortcuts

This folder contains some handy shortcuts to documents that will need to be modified throughout the setup.

ST_scripts

This folder contains all the python and batch scripts used to interact with the ST software tools and hardware.

0_Create_MAR.bat

Creates a thing, policy and certificate and attaches the policy and certificate to the thing.

1_Create_Certificate.bat

Creates an ECDSA code-signing private key, code-signing certificate and certificate chain into AWS Certificate Manager.

2_Create_Signing_Profile.bat

Creates a signing profile in AWS

3_Create_OTA_Role

Creates an OTA Role in AWS.

4_s3Upload.bat

Creates an S3 bucket in AWS and saves the new firmware image to the bucket

5_Publish_OTA.bat

Publishes an OTA in AWS after verifying that all the necessary entities to do the update exist.

IDE.txt

This text file can be modified by the user to specify which IDE they have built their X-CUBE-AWS package with.

Keys.txt

This text file is for the user to save their AWS account credentials in a handy place, as they are difficult to recover if lost.

profile.txt

This text file can be modified by the user to specify which AWS CLI profile they intend to utilize.

thing_name.txt

This folder contains all the python and batch scripts used to interact with the ST software tools and hardware.

2.4.3 Development Script Instructions

Note 1: Steps 1-4 are all that are required when Over The Air Updates (OTA) are not needed

Note 2: Scripts 1,2,3 should be run only once per AWS account, and output files should be saved for future use.

Note 3: Ensure that aws_clientcredentials.h, aws_demo_config.h, aws_ota_pal.c, and aws_application_version.h are not read only.

Connecting B-L4S5I-IOT01A Discovery Board to AWS:

1. Open all the projects in Cube IDE and build them in the following order:
   a. SECoreBin:
      C:\STM32CubeExpansion_Cloud_AWS_V2.0.0\Projects\B-L4S5I-IOT01A\Applications\BootLoader_STSAFE\2_Images_SECoreBin\<Tool-Chain>\(project file)
   b. SBSFU
      C:\STM32CubeExpansion_Cloud_AWS_V2.0.0\Projects\B-L4S5I-IOT01A\Applications\BootLoader_STSAFE\2_Images_SBSFU\<Tool-Chain>\(project file)
   c. STSafe Provisioning
      C:\STM32CubeExpansion_Cloud_AWS_V2.0.0\Projects\B-L4S5I-IOT01A\Applications\BootLoader_STSAFE\STSAFE_Provisioning\<Tool-Chain>\(project file)
2. Navigate to C:\STM32CubeExpansion_Cloud_AWS_V2.0.0\DevCon\shotcuts and Modify the folowing files
   a. aws_clientcredentials.h:
      - #define clientcredentialIOT_THING_NAME "[Insert Thing Name]"
      - #define clientcredentialWIFI_SSID "[Insert SSID]"
      - #define clientcredentialWIFI_PASSWORD "[Insert Password]"
      - #define clientcredentialMQTT_BROKER_ENDPOINT "[Insert Endpint]"
   b. Aws_demo_config.h:
      - Ensure '#define CONFIG_ST_CUSTOM_DEMO_ENABLED' is uncommented
3. Open and Build the aws_demos project in your IDE
   a. C:\Projects\STM32CubeExpansion_Cloud_AWS_V2.0.0\Projects\B-L4S5I-IOT01A\Applications\Cloud\aws_demos\<Tool-Chain>\(project file)
4. Run 0_Create_MAR.bat and your board will connect to AWS IoT Console and begin to publish Temperature/Humididty data and the current state of the LED

Updating B-L4S5I-IOT01A firmware with OTA:

5. Run 1_Create_Certificate.bat to generate ECDSA certs with OpenSSL

6. Navigate to C:\STM32CubeExpansion_Cloud_AWS_V2.0.0\DevCon\shotcuts and Modify the following files
    a. aws_ota_pal.c:
        - Copy the code signing certificate string to the pcClientCertificatePem[] static array in the OTA PAL
        - Follow this Form:

```
static const char pcClientCertificatePem[] =
                "-----BEGIN CERTIFICATE-----\n"
                "MIIBYTCCAQagAwIBAgIUN8BKrSVK8+3xjPaMw3gEWKpqVEwwCgYIKoZIzj0EAwIw\n"
                "HWAwwSZTIxMTEx1haWxAcHdfgdfgdmlkZXIuY29tMB4XDTIwMTExOTIwMTA0MloX\n"
                "DTEbMBkGA1UEOTIwMTlowHTEbMBkGA1UEAwSZW1haWxAcHJvdmlkZXIuY29tMFkw\n"
                "EwYHKoZIzj0CAQYIKoZIzj0DAQcDQgAEzx5nz7HyYAa5DqiXcjLoeonLAQvr3buM\n"
                "tuKvBH4TKg1N1vI+Ofk/eWezSuYazLUh/CvO7aBAJ25DAwF3ctzHFaMkMCIwCwYD\n"
                "VR0PBAQDAgeAMBMGA1UdJQQMMAoGCCsGAQUFBwMDMAoGCCqGSM49BAMCA0kAMEYC\n"
                "IQDMXXtH1ST0/uRznd+dZz1Fkz05wbLNok0kAWMY7dy3nAIhANGitLk4uw+7tP1J\n"
                "+2vJ81t4IFNUoeKibrMKgunO/qKD\n"
                "-----END CERTIFICATE-----\n";
```

7. Build aws_demos in your IDE and flash it to your board
8. Navigate to C:\STM32CubeExpansion_Cloud_AWS_V2.0.0\DevCon\shotcuts and Modify the following files
    a. aws_application_version.h:
        - Increment ‘APP_VERSION_MAJOR’
9. Build aws_demos in your IDE but **do not** flash it
10. Run 2_Create_Signing_Profile.bat to create your AWS code signing profile
11. Run 3_Create_Role.bat to create your AWS OTA role
12. Run 4_s3Upload.bat to upload the new firmware version to an s3 bucket
13. Run 5_Publish_OTA.bat to publish an OTA job

3 IoT Analytics Setup

3.1 Setting up ST boards with AWS IoT

1. Make sure you are logged into an AWS account with root privileges
2. Make sure you are on the default AWS region for your account
3. At the AWS services page, navigate to AWS IoT Core.
4. Make sure your sensor(s) show up in Manage -> things. If you do not have one, click create thing. It does not need any special parameters, just a name. Make sure that you have a thing, certificate and policy created. The certificate and policy have to be attached to the thing.
5. In the sidebar click “act” -> “rules” and then create a new rule eg: alexa_demo_rule.
    a. In the rule you are creating, edit the SQL statement to select all of the topics your data is being published to. For eg: SELECT * FROM 'dt/<your thing name>/sensor-data'. You can learn more about MQTT topics here if your query has to be different.

b. Scroll down and select add a new action, then pick “send message to IoT analytics”. Select the option to quick create resources. For example, you can give the prefix ‘alexa_skill_demo_’, no other parameters are needed.
c. After the rule is created, click “Actions -> Enable” in the top right drop down. In case it is already enabled, skip this step.
d. Go back to the AWS console home page, navigate to AWS IoT Analytics. Click on the dataset that was just created, and edit the query to be the following- SELECT board_id, from_unixtime(timestamp) AS timestamp_converted, temp, hum FROM stdemo_datastore
e. If your datastore is named something other than stdemo_datastore then modify the SQL query to use your name.
f. Click ‘Actions’, ‘Run Now’. You should be able to see all of your MQTT messages that have been sent from the board formatted in a table.
g. To get the data to update automatically, scroll to where it says schedule, and set up a schedule to run this query. I set it to run every hour.

3.2 Quicksight Setup (Optional)

Note: The data can be downloaded from AWS IoT Analytics -> Data sets ->(your dataset)>Content->(.csv file) and plotted using Matlab/Excel or any other graphical visualization tool. In addition, it can also be plotted in QuickSight as described below. In the AWS services page, navigate to QuickSight.

1. Create a QuickSight account.
2. In the sidebar, go to the datsets page, and create a new dataset. Select the option that links your data with AWS IoT Analytics. If the dataset does not show up, it may be because your QuickSight region is not the same as the region you set up AWS IoT in.
3. Once you have added the dataset, click on it. Here you can refresh it manually, set up a schedule to auto-refresh every hour, and you can click edit data to view your dataset.
4. Once you have finished configuring the dataset, click the analysis sidebar. Click the plus in the top left to add a visual or use the one that is already there. Drag the values from the sidebar onto the field wells. To view the field wells, you may need to click the drop-down arrows in the top right. You should now be able to see the IOT data in graph form.
5. To add a filter by board ID dropdown, first select parameters in the left sidebar. Add a new parameter, and then select dynamic default. Select your dataset, board id, and board id for the three drop-down menus. Now go to filters in the sidebar create a new filter. Select custom filter, equals, and then “use parameter”. Select the parameter you just made. The dropdown should now appear in the controls section above your graph.

3.3 Creating a dataset

1. Navigate to IOT Analytics
2. Create a new SQL dataset with a name (Eg: alexa_demo_dataset)
3. Give the dataset the following query
4. SELECT board_id, from_unixtime(timestamp) AS timestamp_converted, temp, hum FROM stdemo_datastore ORDER BY timestamp DESC LIMIT 1
5. In the schedule section, set the query to run as frequently as possible.

4. Alexa Setup

4.1 Creating a Lambda function in AWS

1. Navigate to AWS lambda
2. Create function button in top right
3. Select ‘Author from scratch’
4. Provide a name to the function (Eg: stdemo_lambda)
5. Select python 3.8 runtime
6. Under ‘Permissions’, select ‘Create new role with basic Lambda permissions’
7. Once your function has been created, you need to attach a policy to the role that was generated. Go into the permissions tab, and click the link to your role (execution role, role name)
   i. Attach a policy that grants the appropriate permissions. It would be better to choose an AdminAccess policy, granted security wise, it might not be the best idea.
   ii. The following inline policy can be added if needed:

```
{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Effect": "Allow",
            "Action": "*",
            "Resource": "*"
        }
    ]
}
```

8. Navigate back to the configuration tab of your lambda function and paste the code provided at Appendix. Make sure to change `C201300028`(Thing name) in the code below to your Thing name and the name of the SQL data set to yours (change `stdemo_latest_row` to the name of your data set).
9. Now you can deploy your function by clicking deploy in the top right.
10. For debugging the lambda function, you can go to Monitoring > View logs in CloudWatch, which will let you see the logs of the python script.

4.2 Creating an Alexa Skill

1. Open Alexa Developer Console(https://developer.amazon.com/alexa/console/ask). You may have to create an account to sign in.
2. To create a new skill, click on ‘Create Skill’ and provide a ‘Skill name’ (Eg: alexa_demo_skill).
3. Select the option to use a ‘Custom’ skill.
4. Select ‘Provision your own’ under ‘Choose a method to host your skill’s backend resources’
5. Click on ‘Create’.
6. Next, choose ‘Start from Scratch’, and then wait while your skill is being created after clicking ‘Choose’ on the top right-hand corner.

7. In the sidebar of your new skill, click invocation, and give your skill a name (Eg: temphum lambda/led lambda).
8. In the sidebar, click interaction model > intents, and then add the following intents. For each intent you must type out 5-10 ways that the user might invoke it. You must spell both L.E.D. and LED a couple of times.
    - GetTempHumIntent (what is the temperature? tell me the humidity)
    - GetLEDStateIntent (what does the l.e.d. read? Is the LED on?)
    - SetLEDOnIntent (turn on the LED. Turn on the L.E.D.)
    - SetLEDOffIntent (turn off the light. Shut off the L.E.D.)
9. Click endpoint on the sidebar. In the default region field, paste the ARN of the lambda function you created.
    a. To find this arn: Navigate to your lambda function in AWS and it will be displayed in the top right-hand corner.
10. Make sure to copy Your Skill ID to clipboard.
11. Click build model to finalize your changes.
12. In the AWS Lambda Console, click on ‘Add Trigger’. In the trigger configuration, click on ‘Alexa Skills Kit’. Under ‘Skill ID’, paste what you copied to the clipboard.
13. Make sure that the board is connected and the STM32CubeExpansion_Cloud_AWS_V2.0.0_CustomDemo is flashed to the board with CONFIG_ST_CUSTOM_DEMO_ENABLED in aws_demo_config.h. (Hint: Check that the board is publishing on TeraTerm and also in IoT Core->Things->Shadow, ensure that the Shadow is being published).
14. Now you can go to the ‘Test’ tab in the Alexa Development Console and see if everything is working. Type ‘run <insert Skill Invocation name>’, and then try and invoke one of the intents you just specified (Eg: “turn on the LED”).

5. Debugging

1. If the ‘Test’ tab in the Alexa Development Console does not provide the expected response, copy the code that gets printed under ‘JSON’ and in AWS ->Lambda ->(your lambda function), click on the drop down under ‘Test’ under ‘Function code’ and select ‘Create new test event’. Provide a name and click on ‘Save’. Finally, hit ‘Test’. The errors will be displayed which should enable you to debug.

2. https://developer.amazon.com/en-US/blogs/alexa/alexa-skills-kit/2019/06/5-common-error-messages-for-custom-alexa-skills-and-how-to-troubleshoot-them

6. Appendix

Note: Make sure to change `C201300028`(Thing name) in the code below to your Thing name and the name of the SQL data set to yours(change `stdemo_latest_row` to the name of your data set).

```
import boto3
import urllib
import csv
import codecs
import json
```

```
import time


# --------------- Helpers that build all of the responses ----------------------

def build_speechlet_response(title, output, reprompt_text, should_end_session):
    return {
        'outputSpeech': {
            'type': 'PlainText',
            'text': output
        },
        'card': {
            'type': 'Simple',
            'title': "SessionSpeechlet - " + title,
            'content': "SessionSpeechlet - " + output
        },
        'reprompt': {
            'outputSpeech': {
                'type': 'PlainText',
                'text': reprompt_text
            }
        },
        'shouldEndSession': should_end_session
    }

def build_response(session_attributes, speechlet_response):
    return {
        'version': '1.0',
        'sessionAttributes': session_attributes,
        'response': speechlet_response
    }

# --------------- Functions that control the skill's behavior ------------------
def get_test_response():
    """ An example of a custom intent. Same structure as welcome message, just make
sure to add this intent
    in your alexa skill in order for it to work.
```

```
    """
    session_attributes = {}
    card_title = "Test"
    speech_output = "This is a test message"
    reprompt_text = "You never responded to the first test message. Sending another
one."
    should_end_session = False
    return build_response(session_attributes, build_speechlet_response(
        card_title, speech_output, reprompt_text, should_end_session))


def get_led_state():
    session_attributes = {}
    card_title = "Get LED State"


    client = boto3.client('iot-data')
    shad = client.get_thing_shadow(thingName='C201300028')
    state=int(json.loads(shad["payload"].read().decode("utf-
8"))['state']['reported']['LED'])


    if state:
        speech_output = "LED is currently on."
    else:
        speech_output = "LED is currently off."


    reprompt_text = "Anything else?"
    should_end_session = False
    return build_response9(session_attributes, build_speechlet_response(
        card_title, speech_output, reprompt_text, should_end_session))


def set_led_state(state):
    session_attributes = {}
    card_title = "Set LED State"
    r = {
        "state": {
            "desired": {
                "LED": state
                }
            },
```

```
        "clientToken": "027269"
    }
    print(json.dumps(r))

    client = boto3.client('iot-data')
    # response = client.update_thing_shadow(
    #     thingName='C201300028',
    #     payload= json.dumps(r)
    # )

    client.publish(topic='$aws/things/C201300028/shadow/update', qos=0, payload=
json.dumps(r) )

    if state:
        speech_output = "LED is now set to on."
    else:
        speech_output = "LED is now set to off."

    reprompt_text = "Anything else?"
    should_end_session = False
    return build_response(session_attributes, build_speechlet_response(
        card_title, speech_output, reprompt_text, should_end_session))

def get_temp_hum():
    session_attributes = {}
    card_title = "Get Temperature and Humidity"
    speech_output = "Failed to connect with IoT dataset."

    iota = boto3.client('iotanalytics')
    response =
iota.get_dataset_content(datasetName='alexa_demo_dataset',versionId='$LATEST',)
    contentState = response['status']['state']

    if (contentState == 'SUCCEEDED') :
        url = response['entries'][0]['dataURI']
        stream = urllib.request.urlopen(url)
        reader = csv.DictReader(codecs.iterdecode(stream, 'utf-8'))
```

```
        head = next(reader)
        speech_output = "Temperature is {}, humidity is
{}.".format(head['temp'],head['hum'])
    else:
        time.sleep(4)
        iota = boto3.client('iotanalytics')
        response =
iota.get_dataset_content(datasetName='alexa_demo_dataset',versionId='$LATEST',)

    reprompt_text = "Hello? {}".format(speech_output)
    should_end_session = False
    return build_response(session_attributes, build_speechlet_response(
        card_title, speech_output, reprompt_text, should_end_session))



def get_welcome_response():
    """ If we wanted to initialize the session to have some attributes we could
    add those here
    """
    session_attributes = {}
    card_title = "Welcome"
    speech_output = "Welcome to your custom alexa application!"
    # If the user either does not reply to the welcome message or says something
    # that is not understood, they will be prompted again with this text.
    reprompt_text = "I don't know if you heard me, welcome to your custom alexa
application!"
    should_end_session = False
    return build_response(session_attributes, build_speechlet_response(
        card_title, speech_output, reprompt_text, should_end_session))



def handle_unclear_intent():
    session_attributes = {}
    card_title = "Unclear Intent"
    speech_output = "Sorry, I'm not sure what you meant."
    reprompt_text = "You can try again or you can exit"
    should_end_session = False
    return build_response(session_attributes, build_speechlet_response(
```

```
        card_title, speech_output, reprompt_text, should_end_session))


def handle_session_end_request():
    card_title = "Session Ended"
    speech_output = "Thank you for trying the Alexa Skills Kit sample. " \
                    "Have a nice day! "
    # Setting this to true ends the session and exits the skill.
    should_end_session = True
    return build_response({}, build_speechlet_response(
        card_title, speech_output, None, should_end_session))

# --------------- Events ------------------

def on_session_started(session_started_request, session):
    """ Called when the session starts.
        One possible use of this function is to initialize specific
        variables from a previous state stored in an external database
    """
    # Add additional code here as needed
    pass



def on_launch(launch_request, session):
    """ Called when the user launches the skill without specifying what they
    want
    """
    # Dispatch to your skill's launch message
    return get_welcome_response()

def on_intent(intent_request, session):
    """ Called when the user specifies an intent for this skill """

    intent = intent_request['intent']
    intent_name = intent_request['intent']['name']
```

```
    # Dispatch to your skill's intent handlers
    if intent_name == "test":
        return get_test_response()
    elif intent_name == "SetLEDOnIntent":
        return set_led_state(1)
    elif intent_name == "SetLEDOffIntent":
        return set_led_state(0)
    elif intent_name == "GetLEDStateIntent":
        return get_led_state()
    elif intent_name == "GetTempHumIntent":
        return get_temp_hum()
    elif intent_name == "AMAZON.HelpIntent":
        return get_welcome_response()
    elif intent_name == "AMAZON.CancelIntent" or intent_name == "AMAZON.StopIntent":
        return handle_session_end_request()
    else:
        return handle_unclear_intent()
        #raise ValueError("Invalid intent")

def on_session_ended(session_ended_request, session):
    """ Called when the user ends the session.

    Is not called when the skill returns should_end_session=true
    """
    print("on_session_ended requestId=" + session_ended_request['requestId'] +
          ", sessionId=" + session['sessionId'])
    # add cleanup logic here

# --------------- Main handler ------------------

def lambda_handler(event, context):
    """ Route the incoming request based on type (LaunchRequest, IntentRequest,
    etc.) The JSON body of the request is provided in the event parameter.
    """
    print("Incoming request...")

    """
```

```
    Uncomment this if statement and populate with your skill's application ID to
    prevent someone else from configuring a skill that sends requests to this
    function.
    """
    # if (event['session']['application']['applicationId'] !=
    #         "amzn1.echo-sdk-ams.app.[unique-value-here]"):
    #     raise ValueError("Invalid Application ID")

    if event['session']['new']:
        on_session_started({'requestId': event['request']['requestId']},
                           event['session'])

    if event['request']['type'] == "LaunchRequest":
        return on_launch(event['request'], event['session'])
    elif event['request']['type'] == "IntentRequest":
        return on_intent(event['request'], event['session'])
    elif event['request']['type'] == "SessionEndedRequest":
        return on_session_ended(event['request'], event['session'])
```

IMPORTANT NOTICE – PLEASE READ CAREFULLY

STMicroelectronics NV and its subsidiaries (“ST”) reserve the right to make changes, corrections, enhancements, modifications, and improvements to ST products and/or to this document at any time without notice. Purchasers should obtain the latest relevant information on ST products before placing orders. ST products are sold pursuant to ST’s terms and conditions of sale in place at the time of order acknowledgement.

Purchasers are solely responsible for the choice, selection, and use of ST products and ST assumes no liability for application assistance or the design of Purchasers’ products.

No license, express or implied, to any intellectual property right is granted by ST herein.

Resale of ST products with provisions different from the information set forth herein shall void any warranty granted by ST for such product.

ST and the ST logo are trademarks of ST. All other product or service names are the property of their respective owners.

Information in this document supersedes and replaces information previously supplied in any prior versions of this document.

© 2020 STMicroelectronics – All rights reserved