Games SDK for Alexa Documentation

Release November 2019

Austin Wilson

Dec 04, 2019

GENERAL

In our demo, we attach the `AmazonDynamoDBFullAccess` policy to both the Lambda function and the Cognito Identity Pool. In a full deployment of a game, we should restrict this to only allow specific access to resources in AWS. Instead of adding `AmazonDynamoDBFullAccess`, follow the steps below to create policies for your Lambda Function and Cognito Identity Pool.

CHAPTER 1

Cognito Identity Pool Policy

1. Navigate to the Policies section in IAM.
2. Click **Create Policy**
3. Click **JSON**
4. Copy and paste the following:

```
{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Sid": "SpecificTable",
            "Effect": "Allow",
            "Action": [
                "dynamodb:GetItem",
                "dynamodb:UpdateItem",
                "dynamodb:PutItem"
            ],
            "Resource": "arn:aws:dynamodb:*:*:table/AlexaPlusUnityTest"
        }
    ]
}
```

Note: Replace `AlexaPlusUnityTest` with the name of your table.

5. Click **Review Policy**
6. Give the Policy a name and click **Create Policy**

Attach this policy to the UnAuth IAM Role instead of the `AmazonDynamoDBFullAccess` policy.

CHAPTER 2

Lambda Policy

1. Navigate to the Policies section in IAM.
2. Click **Create Policy**
3. Click **JSON**
4. Copy and paste the following:

```
{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Sid": "SpecificTable",
            "Effect": "Allow",
            "Action": [
                "dynamodb:CreateTable",
                "dynamodb:GetItem",
                "dynamodb:UpdateItem",
                "dynamodb:PutItem"
            ],
            "Resource": "arn:aws:dynamodb:*:*:table/AlexaPlusUnityTest"
        }
    ]
}
```

Note: Replace `AlexaPlusUnityTest` with the name of your table.

5. Click **Review Policy**
6. Give the Policy a name and click **Create Policy**

Attach this policy to thr Alexa Skill's Lambda IAM Role instead of the `AmazonDynamoDBFullAccess` policy.

CHAPTER 3

Frequently Asked Questions

3.1 GetSessionAttributes: The service returned an error with HTTP Body: Cannot resolve destination host

If you get this error, please try the following:

- Make sure you are connected to the internet and the Unity Project can access the internet.
- Make sure the DynamoDB table name matches in both the Alexa Skill and Unity Project.
- Run the Alexa Skill and make sure the DynamoDB table was created.

CHAPTER 4

Configuration

4.1 AWS Configuration

For Games SDK for Alexa to fuction, we need to configure AWS to allow the Unity project to communicate to DynamoDB (The Alexa Skill's persistant attributes).

4.1.1 Prerequisites

- An AWS Account.

4.1.2 Obtain an Identity Pool ID using Amazon Cognito

1. Log in to the Amazon Cognito Console and click **Create new identity pool**.
2. Enter a name for your Identity Pool and check the checkbox to enable access to unauthenticated identities. Click **Create Pool** to create your identity pool.
3. Click **Allow** to create the two default roles associated with your identity pool–one for unauthenticated users and one for authenticated users. These default roles provide your identity pool access to Cognito Sync and Mobile Analytics.

The next page displays code. Take note of the displayed **Identity Pool ID** and the **Region** you set up the Identity Pool in as you will need them when setting up Games SDK for Alexa.

4.1.3 Attach Polices to the Identity Pool default roles in AWS IAM

1. Log in to the AWS IAM Console and click **Roles** in the left navigation bar.
2. Find and click your **Unauthenticated** Identity Pool role. It should look similar to `Cognito_[YOUR IDENTITY POOL]Unauth_Role`.
3. Click **Attach Policies**.

4. Find and check the **AmazonDynamoDBFullAccess** policy.
5. Click **Attach Policy**.

Your Identity Pool is now configured to use the required AWS services for Games SDK for Alexa to function.

4.2 Pubnub Configuration

For Games SDK for Alexa to fuction, we need to configure Pubnub to allow the Unity project to communicate with the Alexa Skill.

4.2.1 Prerequisites

- A Pubnub Account.

4.2.2 Create a New App on Pubnub

1. Log in to the Pubnub Admin Console and click **Create new app**.
2. Enter a name for your new app. Click **Create** to create your new app.
3. Click on your new app in the admin console.

The next page displays your keysets. You can create as many as you keysets as you wish, but for our purposes, we can just use the Demo Keyset.

4. Click the Demo Keyset.
5. Make note of both the publish and subscribe keys as you will need them when setting up Games SDK for Alexa.
6. Enable the **Stream Controller** and the **Storage and Playback** Application add-ons.

Your Pubnub App is now configured for Games SDK for Alexa.

CHAPTER 5

Setup for Unity

Let's setup the Games SDK for Alexa for Unity!

5.1 Prerequisites

- Unity version 4.x or above.
- Configure AWS and Pubnub as explained in configuration.

5.2 Integrating the Games SDK for Alexa into your Unity project

1. Download Games SDK for Alexa from the Unity Asset Store.
2. Add the asset package to your Unity project.
3. Make sure everything is checked and click **Import**.

That's it! However, in order to handle the communication to and from Alexa, you need to create your own script to initalize the manager. See a Tutorial for a more in-depth implementation.

CHAPTER 6

Setup for Alexa Skills

Now that we have configured our Unity workspace, it's time to set up the Alexa Skill!

6.1 Prerequisites

- A suitable Node.js development environment. The ASK SDK v2 for Node.js requires Node 4.3.2 or above.

6.2 Integrating the Games SDK for Alexa into your Alexa Skill

1. Navigate to your skill fuction in an command prompt or terminal
2. Install the AlexaPlusUnity package into yor skill's function: `npm install alexaplusunity`
3. Include the package in your skill with:

```
var alexaPlusUnityClass = require('alexaplusunity');
```

4. Create an instance of the class with:

```
var alexaPlusUnity = new alexaPlusUnityClass("<YOUR_PUBNUB_PUB_KEY>", "<YOUR_
↪PUBNUB_SUB_KEY>", true); //Third parameter enables verbose logging
```

That's it! However, in order to handle the communication to and from your game, you need to use the script's methods. See a Tutorial or the package page for more in-depth information.

CHAPTER 7

Introduction

Looking to learn how to use the the Games SDK for Alexa with an in-depth example? You've come to the right place!

In this tutorial, we will walk through the steps to create a basic game in which we'll be able to manipulate a light with both our keyboard and voice, all while having Alexa remain aware of the game's state.

CHAPTER 8

Creating the Unity Project

Let's setup the Unity Project!

If you get lost or want to see the final project, the light control demo scene is located in `Games SDK for Alexa\Examples`. You will need to create a project and add the Games SDK for Alexa asset package from the Unity Asset Store to gain access to this folder.

8.1 Prerequisites

- Unity version 4.x or above.
- Configure AWS and Pubnub as explained in configuration.

8.2 Creating the project

1. Open Unity.
2. Create a new 3D project.
3. Add a cube to the SampleScene (`GameObject -> 3D Object -> Cube`).
4. Add a Rididbody Component to the newly created cube (With the cube selected in the hierarchy, `Component -> Add -> Rididbody`. Uncheck `Use Gravity`)
5. Click on the Main Camera and change the Y position transform value to `0`.

8.3 Integrating the Games SDK for Alexa Package

1. Download Games SDK for Alexa from the Unity Asset Store.
2. Add the asset package to your Unity project.

3. Make sure everything is checked and click **Import**.

You should now see a new folder in you Assets folder, **Games SDK for Alexa**.

8.4 Adding the LightControl script into the Unity project

1. With your project **open**, create a new script (`Assets -> Create -> C# Script`) and name it **LightControl**.
2. Open your LightControl script by double-clicking it.
3. Delete everything in the script.
4. Copy and paste the code below into your LightControl script:

```
using Amazon;
using Amazon.DynamoDBv2.Model;
using AmazonsAlexa.Unity.AlexaCommunicationModule;
using System.Collections.Generic;
using UnityEngine;

public class LightControl : MonoBehaviour
{
    //Step 5: Add variables below


    // Use this for initialization
    void Start () {
        //Step 6: Add Games SDK for Alexa initialization

    }

    public void ConfirmSetup(GetSessionAttributesEventData eventData)
    {
        //Step 7: Notify the skill that setup has completed by updating the␣
↪skills persistant attributes (in DynamoDB)

    }

    void Update() {
        //Step 8: Add Keyboard event listener (For "gameplay")

    }

    public void HandleSpacePress()
    {
        //Step 9: Handle the keyboard input

    }

    //Callback for when a message is recieved
    public void OnAlexaMessage(HandleMessageEventData eventData)
    {
        //Step 10: Listen for new messages from the Alexa skill

    }
```

(continues on next page)

(continued from previous page)

```
    private void GetObjectInDirection(string type, string message)
    {
        //Step 11: Get the object in a specific direction (Note: For this demo,␣
↪there is only one object, the cube)

    }

    public void UpdateLight(string type, string value,␣
↪GetSessionAttributesEventData eventData)
    {
        //Step 12: Update the light based on the incoming message, then save the␣
↪state of the light through the skill's session attributes

    }

    public void SetAttributesCallback(SetSessionAttributesEventData eventData)
    {
        //Step 13: Callback for when session attributes have been updated

    }


    public void OnMessageSent(MessageSentEventData eventData)
    {
        //Step 14: Callback for when a message is sent

    }
}
```

The above code is our skeleton for our script. We will fill this skeleton step by step. The steps below corrospond to the step numbers in the skeleton. Place the code for each of the below steps under their step number in the skeleton.

Note: There may be IDE errors as we continue, but those will be resolved at the end when the skeleton is complete.

5. Define the class variables:

```
public string publishKey;
public string subscribeKey;
public string channel;
public string tableName;
public string identityPoolId;
public string AWSRegion = RegionEndpoint.USEast1.SystemName;
public bool debug = false;
public GameObject lightCube;
public GameObject camera;

private Dictionary<string, AttributeValue> attributes;
private AmazonAlexaManager alexaManager;
```

These variables are necessary to preform initialization and enable reusablity of the Alexa Manager within our Light-Control script.

6. Initialize the Alexa Manager:

```
UnityInitializer.AttachToGameObject(gameObject);
AWSConfigs.HttpClient = AWSConfigs.HttpClientOption.UnityWebRequest;
alexaManager = new AmazonAlexaManager(publishKey, subscribeKey, channel,␣
↪tableName, identityPoolId, AWSRegion, this.gameObject, OnAlexaMessage, null,␣
↪debug); //Initialize the Alexa Manager
```

7. Tell the skill that the game has completed setup and is ready to play:

```
attributes = eventData.Values;
attributes["SETUP_STATE"] = new AttributeValue { S = "COMPLETED" }; //Set SETUP_
↪STATE attribute to a string, COMPLETED
alexaManager.SetSessionAttributes(attributes, SetAttributesCallback);
```

8. Listen for a spacebar keypress:

```
if (Input.GetKeyDown(KeyCode.Space))
{
    Debug.Log("Space pressed");
    HandleSpacePress();
}
```

9. Update the light to blue when the spacebar is pressed:

```
if (!PlayerPrefs.HasKey("alexaUserDynamoKey")) //If the AlexaUserId has not been␣
↪recieved from Alexa (If the user has not opened the skill)
{
    Debug.LogError("'alexaUserDynamoKey' not found in PlayerPrefs. We must␣
↪establish connection from Alexa to set this. Please open the skill to set the
↪'AlexaUserId' PlayerPref.");
} else {
    alexaManager.GetSessionAttributes((result) =>
    {
        if (result.IsError)
            Debug.LogError(result.Exception.Message);
        UpdateLight("Color", "blue", result);
    });
}
```

10. Listen for new messages from the Alexa skill:

```
Debug.Log("OnAlexaMessage");

Dictionary<string, object> message = eventData.Message;

//Get Session Attributes with in-line defined callback
switch (message["type"] as string)
{
    case "AlexaUserId":
        Debug.Log("AlexaUserId: " + message["message"]);
        alexaManager.alexaUserDynamoKey = message["message"] as string;
        break;
}

alexaManager.GetSessionAttributes((result) =>
{
    if (result.IsError)
        Debug.LogError(eventData.Exception.Message);
```

(continues on next page)

(continued from previous page)

```
    switch (message["type"] as string)
    {
        case "AlexaUserId":
            ConfirmSetup(result);
            break;
        case "Color":
            Debug.Log("Requested Light Color: " + message["message"]);
            UpdateLight(message["type"] as string, message["message"] as string,␣
↪result);
            break;
        case "State":
            Debug.Log("Requested Light State: " + message["message"]);
            UpdateLight(message["type"] as string, message["message"] as string,␣
↪result);
            break;
        case "GetObject":
            Debug.Log("Requested object direction: " + message["message"]);
            GetObjectInDirection(message["type"] as string, message["message"] as␣
↪string);
            break;
        default:
            break;
    }
});
```

11. Get object in a direction:

```
RaycastHit hit;
Dictionary<string, string> messageToAlexa = new Dictionary<string, string>();
Vector3 forward = camera.transform.forward * 10;
Vector3 backward = camera.transform.forward * -10;
Vector3 right = camera.transform.right * 10;
Vector3 left = camera.transform.right * -10;
        Vector3 up = camera.transform.up * 10;
        Vector3 down = camera.transform.up * -10;

        Vector3 direction = forward;

        switch(message) {
                case "forward":
                        direction = forward;
                        break;
                case "backward":
                        direction = backward;
                        break;
                case "right":
                        direction = right;
                        break;
                case "left":
                        direction = left;
                        break;
                case "up":
                        direction = up;
                        break;
                case "down":
                        direction = down;
```

(continues on next page)

(continued from previous page)

```
                        break;
        }

messageToAlexa.Add("object", "nothing");

if (Physics.Raycast(camera.transform.position, direction, out hit, (float)15.0))
{
    if (hit.rigidbody)
    {
        messageToAlexa.Remove("object");
        messageToAlexa.Add("object", hit.rigidbody.name);
    }
}

alexaManager.SendToAlexaSkill(messageToAlexa, OnMessageSent);
```

12. Update the light:

```
attributes = eventData.Values;
if (type == "Color")
{
    attributes["color"] = new AttributeValue { S = value }; //Set color attribute␣
↪to a string value
}
else if (type == "State")
{
    attributes["state"] = new AttributeValue { S = value }; //Set state attribute␣
↪to a string value
}

switch (value)
{
    case "white":
        lightCube.GetComponent<Renderer>().material.color = Color.white;
        break;
    case "red":
        lightCube.GetComponent<Renderer>().material.color = Color.red;
        break;
    case "green":
        lightCube.GetComponent<Renderer>().material.color = Color.green;
        break;
    case "yellow":
        lightCube.GetComponent<Renderer>().material.color = Color.yellow;
        break;
    case "blue":
        lightCube.GetComponent<Renderer>().material.color = Color.blue;
        break;
    case "on":
        lightCube.GetComponent<Renderer>().enabled = true;
        break;
    case "off":
        lightCube.GetComponent<Renderer>().enabled = false;
        break;
}
alexaManager.SetSessionAttributes(attributes, SetAttributesCallback);  //Save␣
↪Attributes for Alexa to use
```

13. Let's be notified when there is a error setting the attributes:

```
Debug.Log("OnSetAttributes");
if (eventData.IsError)
    Debug.LogError(eventData.Exception.Message);
```

14. Let's be notified when there is a error sending a message:

```
Debug.Log("OnMessageSent");
if (eventData.IsError)
    Debug.LogError(eventData.Exception.Message);
```

15. Be sure to save this file!

8.5 Adding the Alexa Manager GameObject in Unity

1. Create a new **Empty GameObject** (`GameObject -> Create Empty`) and name it **Amazon Alexa**.
2. With your new GameObject selected, click **Add Component**, type **LightControl** and select the LightControl script.
3. Fill the `Publish Key` with the Pubnub publish key you made note of during configuration.
4. Fill the `Subscribe Key` with the Pubnub subscribe key you made note of during configuration.
5. Fill the `Channel` with the code sent from the Alexa skill when it launches.

Note: You will have to fill this in later, as we have not set up the Alexa skill yet.

---

6. Fill the `Table Name` with **AlexaPlusUnityTest**.
7. Fill the `Identity Pool Id` with the one you created during configuration.
8. Fill the `AWS Region` with the one you made note of during configuration.
9. Check the box next to `Debug` to enable detailed logging.
10. Drag the **Cube** from the hierarchy into the box next to `Light Cube`.
11. Drag the **Main Camera** from the hierarchy into the box next to `Camera`.

8.6 Wrapping Up

Aside from a few minor updates, we have finished the Unity project! Next Step: The Alexa Skill!

CHAPTER 9

Creating the Alexa Skill

Let's setup the Alexa Skill!

If you get lost, stuck, or just want a working demo right now, you can clone the complete sample project from here.

9.1 Prerequisites

- Node.js (v4.5 or above).
- Have an AWS Account.
- Have an Amazon Developer Account.
- Install the ASK CLI.

9.2 Creating the Skill

1. Clone the example skill templete: `git clone https://github.com/AustinMathuw/AlexaPlusUnityExampleSkillTemplate.git`
2. Open the template in a editor such as Visual Studio code
3. Open a command prompt or terminal and navigate to `<Template Location>/lambda/custom/`
4. In the command prompt or terminal, run `npm install`

9.3 Example Skill Template Overview

Our Example Skill has the following Intents:

- FlipSwitchIntent
  - Handles turning our light on and off

- ChangeColorIntent
  - Handles changing the color of our light
- GetColorIntent
  - Returns our lights current color
- AMAZON.HelpIntent
  - Returns help message to guide the user
- AMAZON.CancelIntent
  - Closes the skill
- AMAZON.StopIntent
  - Closes the skill

9.4 Adding Games SDK for Alexa to the Skill

The steps below corrospond to the step numbers in the skeleton. Place the code for each of the below steps under their step number in the templete code.

Note: There may be IDE errors as we continue, but those will be resolved at the end when the skeleton is complete.

1. Our cloned templete already has the `alexaplusunity` node module, but we still need to include it. Open `index.js` under `lambda/custom/` and add the following:

```
var alexaPlusUnityClass = require('alexaplusunity');
var alexaPlusUnity = new alexaPlusUnityClass("<YOUR_PUBNUB_PUB_KEY>", "<YOUR_
↪PUBNUB_SUB_KEY>", true); //Third parameter enables verbose logging
```

2. In our example skill, we will use state management to confirm that the user has connected to our game. Insert the code below inside of the LaunchRequestHandler:

```
if(attributes.SETUP_STATE == "STARTED") {
    var launchSetUpResult = await launchSetUp(speechText, reprompt, handlerInput,␣
↪attributes);
    attributes = launchSetUpResult.attributes;
    response = launchSetUpResult.response;
}
```

In the code block above, we are checking to see if we are still in the `SETUP_STATE`. If we are, then run the method `launchSetUp()` and build our response. If we are not in the `SETUP_STATE`, then use the response we already built.

3. In the next steps, we will add Games SDK for Alexa to the FlipSwitchIntent (CompletetedFlipSwitchIntentHandler). We will start by creating a payload object for the intent:

```
var payloadObj = {
    type: "State",
    message: state
};
```

4. Then we will send the payload to our game and reply with a success or error if the payload failed to send:

```
var response = await alexaPlusUnity.publishMessage(payloadObj, attributes.PUBNUB_
↪CHANNEL).then((data) => {
    return handlerInput.responseBuilder
        .speak(speechText + reprompt)
        .reprompt(reprompt)
        .getResponse();
}).catch((err) => {
    return ErrorHandler.handle(handlerInput, err);
});
```

5. Now, we will add Games SDK for Alexa to the ChangeColorIntent (CompletedChangeColorIntentHandler). We will create our payload object:

```
var payloadObj = {
    type: "Color",
    message: color
};
```

6. Send the payload to our game and reply with a success or error if the payload failed to send:

```
var response = await alexaPlusUnity.publishMessage(payloadObj, attributes.PUBNUB_
↪CHANNEL).then((data) => {
    return handlerInput.responseBuilder
        .speak(speechText + reprompt)
        .reprompt(reprompt)
        .getResponse();
}).catch((err) => {
    return ErrorHandler.handle(handlerInput, err);
});
```

7. Add Games SDK for Alexa to the GetObjectInDirectionIntent (CompletedGetObjectInDirectionIntentHandler). We will create our payload object:

```
var payloadObj = {
    type: "GetObject",
    message: direction_id
};
```

8. Send a the payload to our game and reply with a success or error if the payload failed to send:

```
var response = await alexaPlusUnity.publishMessageAndListenToResponse(payloadObj,␣
↪attributes.PUBNUB_CHANNEL, 4000).then((data) => {
   speechText = 'Currently, ' + data.message.object + ' is ' + direction + ' of␣
↪you!';

   return handlerInput.responseBuilder
      .speak(speechText + reprompt)
      .reprompt(reprompt)
      .getResponse();
}).catch((err) => {
   return ErrorHandler.handle(handlerInput, err);
});
```

9. Create the user's unique channel:

```
var response = await alexaPlusUnity.addChannelToGroup(attributes.PUBNUB_CHANNEL,
↪"AlexaPlusUnityTest").then(async (data) => {
```

(continues on next page)

(continued from previous page)

```
var responseToReturn = responseBuilder
    .speak(speechText)
    .reprompt(reprompt)
    .withSimpleCard('Games SDK for Alexa', "Here is your Player ID: " +␣
↪attributes.PUBNUB_CHANNEL)
    .getResponse();

var userId = handlerInput.requestEnvelope.session.user.userId;
return await sendUserId(userId, attributes, handlerInput, responseToReturn);
}).catch((err) => {
    return ErrorHandler.handle(handlerInput, err);
});
```

10. Now, we need to send the game the user's Alexa ID so we can access their persistant session attributes. Create our payload object:

```
var payloadObj = {
    type: "AlexaUserId",
    message: userId
};
```

11. Send the payload to the game:

```
return await alexaPlusUnity.publishMessage(payloadObj, attributes.PUBNUB_CHANNEL).
↪then((data) => {
    return response;
}).catch((err) => {
    return ErrorHandler.handle(handlerInput, err);
});
```

12. Lastly, we need to initialize the skills attributes

```
attributes.SETUP_STATE = "STARTED";
var newChannel = await alexaPlusUnity.uniqueQueueGenerator("AlexaPlusUnityTest");

if(newChannel != null) {
    attributes.PUBNUB_CHANNEL = newChannel;
} else {
    return null;
}
```

9.5 Deploying the Skill

1. Open a command prompt or terminal and navigate to <Template Location>
2. Type `ask deploy` to deploy the skill.
3. In a browser, navigate to your newly created Lambda Function
4. Scroll down to the **Execution Role** and click on `View the ask-lambda-Unity-Light-Control-AlexaPlusUnityT role`. This takes you to the IAM role in IAM.
5. Click **Attach Policies**.
6. Find and check the **AmazonDynamoDBFullAccess** policy.
7. Click **Attach Policy**.

Note: This will only work if you set up the ASK CLI correctly!

9.6 Wrapping Up

At this point, you should be able to test the skill by saying, "Alexa, open unity light".

Note: You will likely get an error the first couple of times initially opening the skill. This is because the skill needs to create the DynamoDB table and it can take a couple of minutes to do so. See this issue for more information.

We have finished the Alexa Skill!

CHAPTER 10

Conclusion

Congratulations! You have completed the Games SDK for Alexa Light Control Demo!

10.1 Testing the Light Control Demo

1. Open the skill by saying "Alexa, open unity light".
2. Make note of your player ID.
3. Make sure your Unity project is open.
4. Click on the **Amazon Alexa** GameObject.
5. Fill the `Channel` field in the Light Control script with the player ID returned by the Alexa Skill.
6. Press the **Play Button** in Unity to enter **Play Mode**.
7. Once in **Play Mode**, re-launch your Alexa Skill.

You should notice that your skill's welcome message changed. This means that the game is successfully communicating to your Alexa Skill. You can now say commands like "Change light to blue", or "What is in front of me?" to interact with the game with your voice.

CHAPTER 11

Unity C# Technical Docs

11.1 Classes

11.2 AlexaBaseData

Syntax

```
public abstract class AlexaBaseData : BaseEventData
```

11.2.1 Constructors

AlexaBaseData(EventSystem)

Declaration

```
public AlexaBaseData(EventSystem eventSystem)
```

Parameters

Type	Name	Description
EventSystem	eventSystem	

11.2.2 Properties

Exception

Declaration

```
public Exception Exception { get; }
```

Property Value

Type	Description
System.Exception	

IsError

Declaration

```
public bool IsError { get; }
```

Property Value

Type	Description
System.Boolean	

11.2.3 Methods

BaseInitialize(Boolean, Exception)

Declaration

```
protected virtual void BaseInitialize(bool isError, Exception exception)
```

Parameters

Type	Name	Description
System.Boolean	isError	
System.Exception	exception	

11.3 AmazonAlexaManager

Syntax

```
public class AmazonAlexaManager
```

11.3.1 Constructors

AmazonAlexaManager(String, String, String, String, String, String, GameObject, Action<HandleMessageEventData>, Action<ConnectionStatusEventData>, Boolean)

AmazonAlexaManager Constructor.

Declaration

```
protected virtual void BaseInitialize(bool isError, Exception exception)
```

Parameters

Type	Name	Description
System.String	publishKey	Your Pubnub publish key.
System.String	subscribeKey	Your Pubnub subscribe key.
System.String	channel	Your player's channel. (Should be unique to the player)
System.String	tableName	Name of your skill's DynamoDB table where the persistant attributes are stored.
System.String	identityPoolId	Identifier of your AWS Cognito identity pool.
System.String	AWSRegion	The AWS Region where your DynamoDB table and Cognito identity pool are hosted.
GameObject	gameObject	The GameObject you are attaching this manager instance to.
System.Action<HandleMessageEventData>	messageCallback	The callback for when a message is recieved from your Alexa Skill.
System.Action<ConnectionStatusEventData>	connectionStatusCallback	
System.Boolean	debug	(Optional) True to debug.

11.3.2 Fields

handleConnectionStatusCallback

The connection status recieved callback.

Declaration

```
public Action<ConnectionStatusEventData> handleConnectionStatusCallback
```

Field Value

Type	Description
System.Action<ConnectionStatusEventData>	

handleMessageCallback

The message recieved callback.

Declaration

```
public Action<HandleMessageEventData> handleMessageCallback
```

Field Value

Type	Description
System.Action<HandleMessageEventData>	

11.3.3 Properties

alexaUserDynamoKey

Gets or Resets the player's DynanoDB table key.

Declaration

```
public string alexaUserDynamoKey { get; set; }
```

Property Value

Type	Description
System.String	The alexa user dynamo key.

channel

Resets your player's channel. (Should be unique to the player)

Declaration

```
public string channel { set; }
```

Property Value

Type	Description
System.String	The channel.

11.3.4 Methods

GetSessionAttributes(Action<GetSessionAttributesEventData>)

Gets the Skill's persistant session attributes from DynamoDB.

Declaration

```
public void GetSessionAttributes(Action<GetSessionAttributesEventData> callback)
```

Parameters

Type	Name	Description
System.Action<GetSessionAttributesEventData>	callback	The callback.

SendToAlexaSkill(Object, Action<MessageSentEventData>)

Sends a message to Alexa Skill. NOTE: Skill will only recieve the message if it is listening for a response.

Declaration

```
public void SendToAlexaSkill(object message, Action<MessageSentEventData> callback)
```

Parameters

Type	Name	Description
System.Object	message	The message.
System.Action<MessageSentEventData>	callback	The callback.

SetSessionAttributes(Dictionary<String, AttributeValue>, Action<SetSessionAttributesEventData>)

Sets the Skill's persistant session attributes in DynamoDB.

Declaration

```
public void SetSessionAttributes(Dictionary<string, AttributeValue> attributes, Action
↪<SetSessionAttributesEventData> callback)
```

Parameters

Type	Name	Description
System.Collections.Generic.Dictionary<System.String, AttributeValue>	attributes	The attributes to set.
System.Action<SetSessionAttributesEventData>	callback	The callback.

11.4 ConnectionStatusEventData

Syntax

```
public class ConnectionStatusEventData : AlexaBaseData
```

11.4.1 Constructors

ConnectionStatusEventData(EventSystem)

Declaration

```
public ConnectionStatusEventData(EventSystem eventSystem)
```

Parameters

Type	Name	Description
EventSystem	eventSystem	

11.4.2 Properties

Category

Declaration

```
public PNStatusCategory Category { get; }
```

Property Value

Type	Description
PNStatusCategory	

11.4.3 Methods

Initialize(Boolean, PNStatusCategory, Exception)

Declaration

```
public void Initialize(bool isError, PNStatusCategory category, Exception exception =␣
↪null)
```

Parameters

Type	Name	Description
System.Boolean	isError	
PNStatusCategory	category	
System.Exception	exception	

11.5 ErrorEventData

Syntax

```
public class ErrorEventData : AlexaBaseData
```

11.5.1 Constructors

ErrorEventData(EventSystem)

Declaration

```
public ErrorEventData(EventSystem eventSystem)
```

Parameters

Type	Name	Description
EventSystem	eventSystem	

11.5.2 Methods

Initialize(Boolean, PNStatusCategory, Exception)

Declaration

```
public void Initialize(Exception exception, bool isError = true)
```

Parameters

Type	Name	Description
System.Exception	exception	
System.Boolean	isError	

11.6 GetSessionAttributesEventData

Syntax

```
public class GetSessionAttributesEventData : AlexaBaseData
```

11.6.1 Constructors

GetSessionAttributesEventData(EventSystem)

Declaration

```
public GetSessionAttributesEventData(EventSystem eventSystem)
```

Parameters

Type	Name	Description
EventSystem	eventSystem	

11.6.2 Properties

Values

Declaration

```
public Dictionary<string, AttributeValue> Values { get; }
```

Property Value

Type	Description
System.Collections.Generic.Dictionary<System.String, AttributeValue>	

11.6.3 Methods

Initialize(Boolean, Dictionary<String, AttributeValue>, Exception)

Declaration

```
public void Initialize(bool isError, Dictionary<string, AttributeValue> values,␣
↪Exception exception = null)
```

Parameters

Type	Name	Description
System.Exception	exception	
System.Collections.Generic.Dictionary<System.String, AttributeValue>	values	
System.Boolean	isError	

11.7 HandleMessageEventData

Syntax

```
public class HandleMessageEventData : AlexaBaseData
```

11.7.1 Constructors

HandleMessageEventData(EventSystem)

Declaration

```
public HandleMessageEventData(EventSystem eventSystem)
```

Parameters

Type	Name	Description
EventSystem	eventSystem	

11.7.2 Properties

Message

Declaration

```
public Dictionary<string, object> Message { get; }
```

Property Value

Type	Description
System.Collections.Generic.Dictionary<System.String, System.Object>	

11.7.3 Methods

Initialize(Boolean, Dictionary<String, Object>, Exception)

Declaration

```
public void Initialize(bool isError, Dictionary<string, object> message, Exception
↪exception = null)
```

Parameters

Type	Name	Description
System.Boolean	isError	
System.Collections.Generic.Dictionary<System.String, System.Object>	message	
System.Exception	exception	

11.8 MessageSentEventData

Syntax

```
public class MessageSentEventData : AlexaBaseData
```

11.8.1 Constructors

MessageSentEventData(EventSystem)

Declaration

```
public MessageSentEventData(EventSystem eventSystem)
```

Parameters

Type	Name	Description
EventSystem	eventSystem	

11.8.2 Properties

Message

Declaration

```
public object Message { get; }
```

Property Value

Type	Description
System.Object	

11.8.3 Methods

Initialize(Boolean, Object, Exception)

Declaration

```
public void Initialize(bool isError, object message, Exception exception = null)
```

Parameters

Type	Name	Description
System.Boolean	isError	
System.Object	message	
System.Exception	exception	

11.9 SetSessionAttributesEventData

Syntax

```
public class SetSessionAttributesEventData : AlexaBaseData
```

11.9.1 Constructors

SetSessionAttributesEventData(EventSystem)

Declaration

```
public SetSessionAttributesEventData(EventSystem eventSystem)
```

Parameters

Type	Name	Description
EventSystem	eventSystem	

11.9.2 Methods

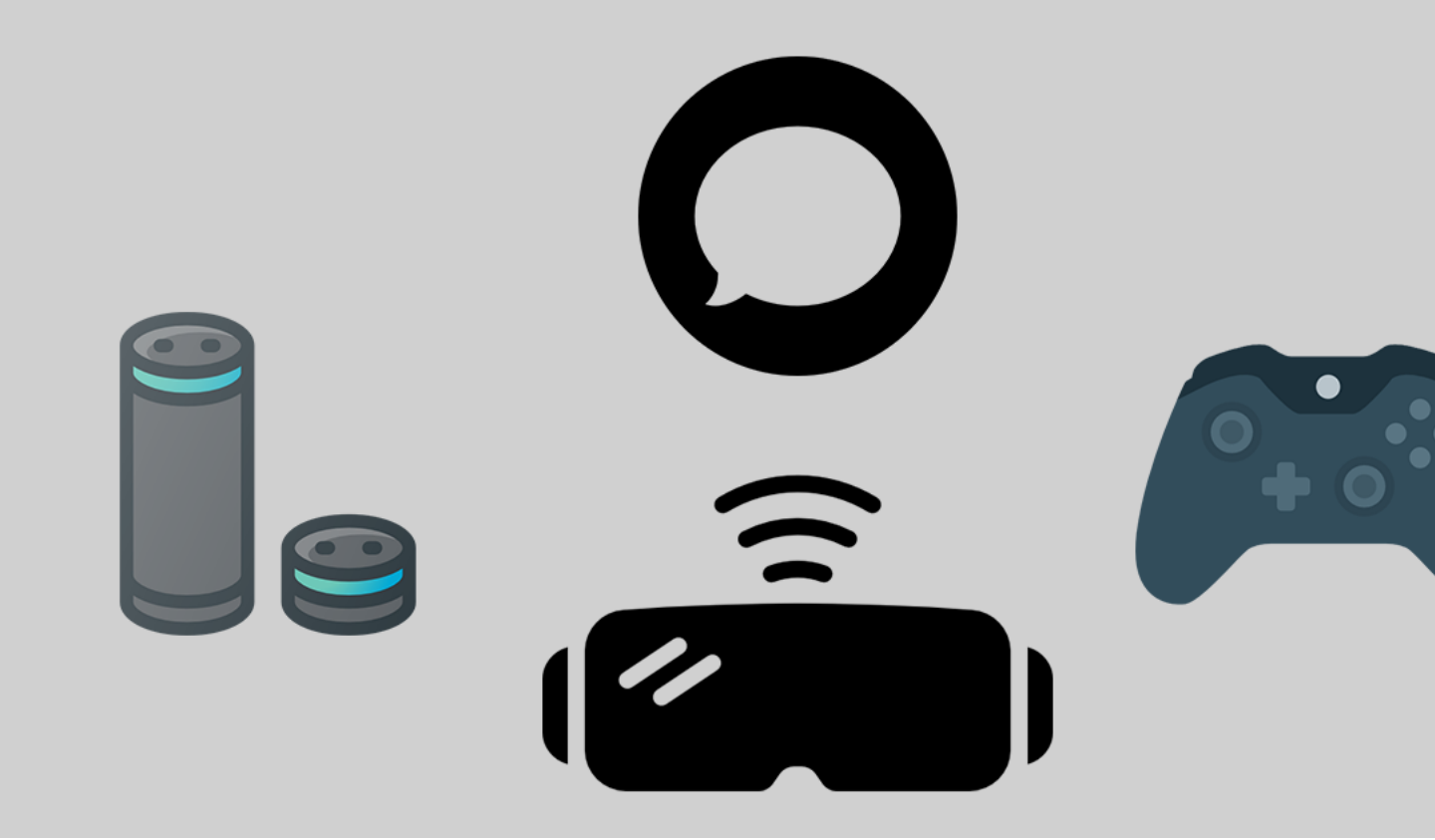
Initialize(Boolean, Exception)

Declaration

```
public void Initialize(bool isError, Exception exception = null)
```

Parameters

Type	Name	Description
System.Boolean	isError	
System.Exception	exception	



CHAPTER 12

Games SDK for Alexa

The Games SDK for Alexa was built to help with the expansion of Amazon's Alexa into video games made with the Unity game engine, bringing a whole new layer of immersion into gaming.

Ready to get started? Follow the guides in the navigation bar to learn how to use the platform!

12.1 Support

Need help? Send us an email at games-sdk-support@tauon.tech!

12.2 The Explorer Demo