o Capture A and B or C Slots

Tips and Tutorials

d I have been leading a voice design workshop that focuses on voice-first nagement to simplify collecting a set of required slots that an intent needs to

get the user's address based on zip code **or** city and state. Can I do that with In that case, you're looking for the condition **A and B or C**, where **A** is the ou can mark all 3 slots required and then hook into the state machine and jump as been met by setting the **dialogState** to **COMPLETED** and returning the slots.

e dialog management state machine, I explained how to hook into the dialog state machine. Take a look at that post if you're not familiar with dialog management before diving into the rest of this post.

Setting Up Your Interaction Model

First you need to set up an **intent** and some **utterances** and **slots**. For this example, I've named the intent **GetLocationIntent** and created 3 slots: **zip**, **city**, and **state**. The **utterances** are:

record my location
I'm from {state}
I live in {city}
My zip code is {zip}
I live in {city} {state}
I live in {city} {state} {zip}

The interaction model has been set up to capture the slots in several ways:

No Slots

The record my location utterance will invoke the **GetLocationIntent** and since it provides no slots, dialog management will kick in and ask the user to fill all the required slots until all have been filled or our backend has identified that **A || B and C** has been met and pulls the rip cord.

Single Slot

I'm from {state}, I live in {city}, and My zip code is {zip} will collect a single slot and invoke the **GetLocationIntent**. Dialog management will then ask for the remaining slots in the order specified in your interaction model.

Multiple Slots

The I live in {city} {state} can capture multiple slots. If the user says, "I live in Seattle Washington," the **city** and **state** slots will be filled. Normally dialog management will follow up by asking for the missing **zip** slot, but since our condition **A || B && C** has been met (null || Seattle && Washington) our backend will jump out of dialog management and complete the task for the user.

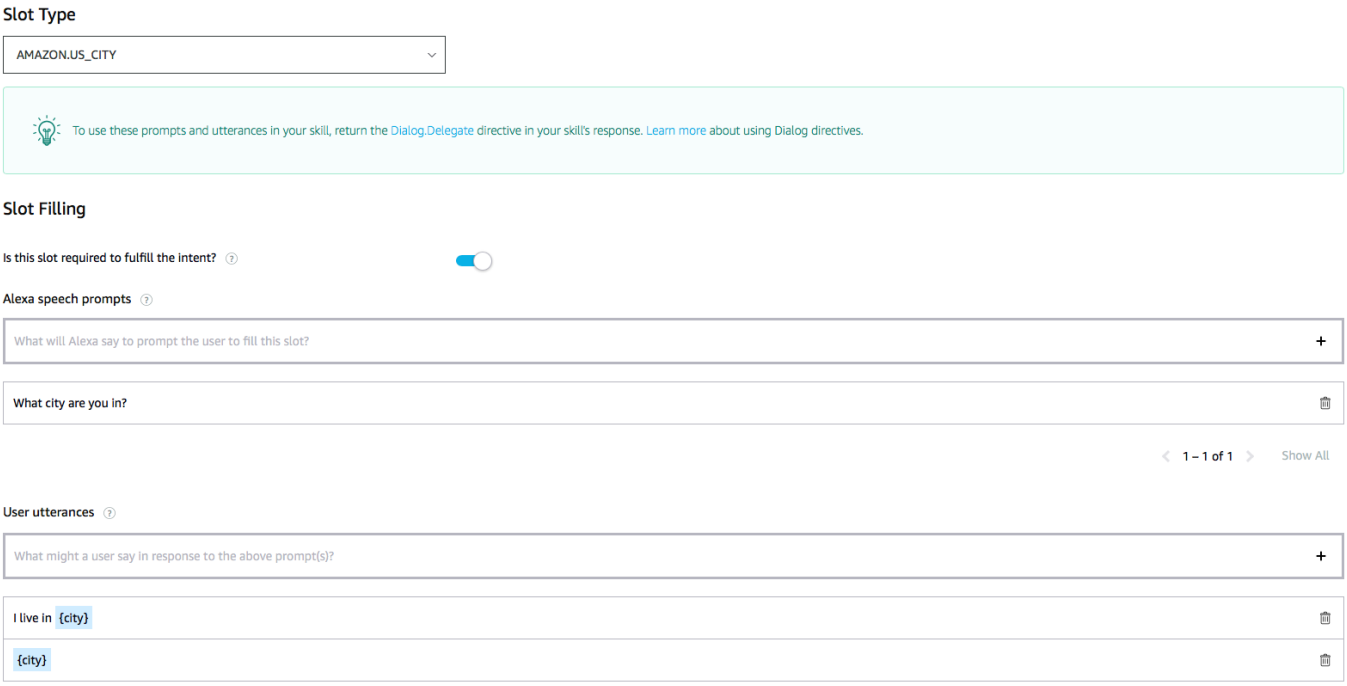
One-Shot Invocation

The I live in {city} {state} {zip} utterance is an example of a **One-shot invocation**, where all the information is provided in one breath. For example, if I were to say, "I live in Seattle Washington 98121," the **dialogState** will be **STARTED** but **A || B && C** (98121 || Seattle && Washington) will evaluate to true. At this point, we have two options, delegate back to Alexa, which will then set the **dialogState** to **COMPLETED,** or manually set it ourselves and continue on with our business. In the sample below, I'll be taking the latter approach.

Activating Dialog Management

Once you've set up your **intent**, **utterances** and **slots**, you'll need to activate dialog management. To do that from the developer portal, you need to make each slot required by selecting them one-by-one and turning on **Is this slot required to fulfill the intent?**.

Intents / GetLocationIntent / city



Then you will need to provide a set of**prompts** and **utterances**. The prompts are what Alexa will say to ask the user to fill the slot while in dialog management. For example, for the **city** the **prompt** is "What city do you live in?" and the**utterance** is I live in {city}. Once you've created **prompts** and **utterances** for each **required** slot, you'll need to set up your backend code.

Checking for A or B and C

Now that the frontend has been set up, now it's time to set up the backend so we can hook into the state machine and check if **A || B && C** is true and manually set **dialogState** to **COMPLETED**.

First, we'll set up our **intent,** which we named **GetLocationIntent**.

GetLocationIntent

```
'GetLocationIntent': function () {

    // delegate to Alexa to collect all the required slots
    let filledSlots = delegateSlotCollection.call(this, function(event) {
        let result = false;
        let slots = event.request.intent.slots;

        if(slots.zip.value || slots.city.value && slots.state.value) {
            result = true;
        }
        return result;
    });

    // delegateSlotCollection may make an asynchronous call, so there
    // is a chance that filledSlots is null. If it's null we need to
    // stop GetLocationIntent and on the next runtime tick,
    // this.emit(':delegate') which was called from
    // delegateSlotCollection will execute.
    if (!filledSlots) {
        return;
    }

    // at this point, we know that all required slots are filled.
    let slotValues = getSlotValues(filledSlots);

    console.log(JSON.stringify(slotValues));
```

```
    let speechOutput = "";
    if (slotValues.zip.resolved) {
        speechOutput = `Your zip code is <say-as interpret-as="digits">${slotValues.zip.resolved}</say-as>.`;
    } else {
        speechOutput = `Your shipping address is ${slotValues.city.resolved}, ${slotValues.state.resolved}`;
    }

    this.response.speak(speechOutput);
    this.emit(':responseReady');
}
```

The first line of the function calls delegateSlotCollection, which we will deep dive into later, but take note of the anonymous function that we passed to it. This function will be executed from within delegateSlotCollection and checks our condition **A || B && C** and returns **true** if met.

You might be asking to yourself why are we checking if(!filledSlots) and returning **null** if its undefined immediately after we call delegateSlotCollection. In delegateSlotCollection we make an asynchronous call to this.emit(':delegate') when the **dialogState** is not **COMPELETED**, which will put the dialog delegation to Alexa onto the next runtime tick and continue executing delegateSlotCollection and once it returns GetLocationIntent will continue. Once GetLocationIntent finishes executing, the next runtime tick will occur and this.emit(':delegate') will execute. Since the delegation to Alexa for slot collection doesn't run until the next tick, filledSlots will be null, so we must stop GetLocationIntent from executing or else we will encounter errors.

Once **dialogState** is **COMPLETED**, we receive the required slots and unpack them with getSlotValues which returns a simplified object of slots that include the spoken value, resolved synonym and a Boolean that indicates whether or not the slot has been confirmed via dialog management.

```
function getSlotValues (filledSlots) {
    //given event.request.intent.slots, a slots values object so you have
    //what synonym the person said - .synonym
    //what that resolved to - .resolved
    //and if it's a word that is in your slot values - .isValidated
    let slotValues = {};

    console.log('The filled slots: ' + JSON.stringify(filledSlots));
    Object.keys(filledSlots).forEach(function(item) {

    var name = filledSlots[item].name;

    if(filledSlots[item]&&
        filledSlots[item].resolutions &&
        filledSlots[item].resolutions.resolutionsPerAuthority[0] &&
        filledSlots[item].resolutions.resolutionsPerAuthority[0].status &&
        filledSlots[item].resolutions.resolutionsPerAuthority[0].status.code ) {

        switch (filledSlots[item].resolutions.resolutionsPerAuthority[0].status.code) {
            case "ER_SUCCESS_MATCH":
                slotValues[name] = {
                    "synonym": filledSlots[item].value,
                    "resolved": filledSlots[item].resolutions.resolutionsPerAuthority[0].values[0].value.name,
                    "isValidated": true
                };
                break;
            case "ER_SUCCESS_NO_MATCH":
                slotValues[name] = {
                    "synonym": filledSlots[item].value,
                    "resolved": filledSlots[item].value,
                    "isValidated":false
                };
```

```
                };
                break;
            }
        } else {
            slotValues[name] = {
                "synonym": filledSlots[item].value,
                "resolved": filledSlots[item].value,
                "isValidated": false
            };
        }
    },this);
    return slotValues;
}
```

Then we are checking if the**zip** has been provided and read back the zip code. If it's undefined at this point we know that we have our **city** and **state** since our condition has been met, so we read off the**city** and **state** values.

For your skill, you will most likely need to do something with the information that you just captured rather than read it out, like looking up restaurants nearby or creating a shipping invoice. Here in the code is where you would perform such a task.

Now that we know how our**GetLocationIntent** operates, let's take a closer look atdelegateSlotCollection.

delegateSlotCollection

```
function delegateSlotCollection(func) {
    console.log("in delegateSlotCollection");
    console.log("current dialogState: " + this.event.request.dialogState);

    if(func) {
        if (func(this.event)) {
            this.event.request.dialogState = "COMPLETED";
            return this.event.request.intent.slots;
        }
    }

    if (this.event.request.dialogState === "STARTED") {
        console.log("in STARTED");
        console.log(JSON.stringify(this.event));
        var updatedIntent = this.event.request.intent;
        // optionally pre-fill slots: update the intent object with slot values
        // for which you have defaults, then return Dialog.Delegate with this
        // updated intent in the updatedIntent property

        this.emit(":delegate", updatedIntent);
    } else if (this.event.request.dialogState !== "COMPLETED") {
        console.log("in not completed");
        //console.log(JSON.stringify(this.event));
        this.emit(":delegate", updatedIntent);
    } else {
        console.log("in completed");
        //console.log("returning: "+ JSON.stringify(this.event.request.intent));
        // Dialog is now complete and all required slots should be filled,
        // so call your normal intent handler.
        return this.event.request.intent.slots;
    }
    return null;
}
```

As I discussed the blog post ontaking control of the dialog management state machine, delegateSlotCollection taps into the state machine and delegates slot collection to Alexa when the **dialogState** is **STARTED** or **IN_PROGRESS**. Once it's **COMPLETED**, it returns the slots. To check our condition,**A || B && C**, we've added a new parameter called**func** which is an anonymous function. If the anonymous function returns **true**, then delegateSlotCollection will set the **dialogState** to **COMPLETED** and return the slots. Let's recall our anonymous function.

Anonymous Function to check A || B && C

```
function(event) {
    let result = false;
    let slots = event.request.intent.slots;

    if(slots.zip.value || slots.city.value && slots.state.value) {
        result = true;
    }
    return result;
});
```

As you can see, it checks our condition **A || B && C**, where **A** is **zip**, **B** is **city** and **C** is **state**, (**zip** || **city** && **state**). If the condition if(slots.zip.value || slots.city.value && slots.state.value) is met, then it returns true an delegateSlotCollection pulls the ripcord on dialog management and stops collecting slots.

Not only can dialog management help you pull a set of required slots from your user, with a just a little bit of extra code you can set it up to collect a subset of the required slots based on a condition. Passing an **anonymous** function that returns true when the condition is met allows you to reuse the function for whatever condition fits your needs.

If you end up using this approach in your skill, I want to hear about it. I want to know about the challenges you faced and how you overcame them, so please reach out to me on twitter @sleepydeveloper.

More Resources on Dialog Management

- Taking Control of the Dialog Management State Machine
- Alexa Skill Teardown: Decoding Dialog Management with the Pet Match Skill
- Pet Match Dialog Management Tutorial
- Technical Documentation for Dialog Management
- Plan My Trip Dialog Management Tutorial

Why Voice Design Matters: We Don’t Speak the Way We Write

Der Spiele-Klassiker Mau-Mau begeistert auch auf Alexa

Developer Tools to Help You Build Alexa Skills

Request Permission and Access Contact Information with the New Customer Profile API

Accelerate Skill Development Using the ASK Toolkit for Visual Studio Code (Beta)

Archive

Year Month GO

Back to top

Alexa Skills Kit >

Resources >

Alexa Voice Service >

Resources >

Connected Devices >

Agreements >

Blogs >

Support >

Devices >

Language English

Follow Us:

amazon alexa

amazon appstore

aws

amazon software + games

amazon dash services