

---

# Amazon Lex

## Developer Guide



## **Amazon Lex: Developer Guide**

Copyright © 2017 Amazon Web Services, Inc. and/or its affiliates. All rights reserved.

Amazon's trademarks and trade dress may not be used in connection with any product or service that is not Amazon's, in any manner that is likely to cause confusion among customers, or in any manner that disparages or discredits Amazon. All other trademarks not owned by Amazon are the property of their respective owners, who may or may not be affiliated with, connected to, or sponsored by Amazon.

# Table of Contents

What Is Amazon Lex?	1
Are You a First-time User of Amazon Lex?	2
How It Works	3
Programming Model	4
Model Building API Operations	5
Runtime API Operations	6
Lambda Functions as Code Hooks	6
Service Permissions	8
Creating Resource-Based Policies for AWS Lambda	8
Deleting Service-Linked Roles	8
Managing Messages (Prompts and Statements)	9
Types of Messages	10
Contexts for Configuring Messages	10
Supported Message Formats	13
Response Cards	14
Managing Conversation Context	18
Setting Session Timeout	18
Setting Session Attributes	18
Sharing Information Between Intents	19
Deployment Options	20
Built-in Intents and Slot Types	20
Built-in Intents	20
Built-in Slots	20
Getting Started	21
Step 1: Set Up an Account	21
Sign Up for AWS	21
Create an IAM User	22
Next Step	22
Step 2: Set Up the AWS CLI	23
Step 3: Getting Started (Console)	23
Exercise 1: Create a Bot Using a Blueprint	24
Exercise 2: Create a Custom Bot	48
Exercise 3: Publish a Version and Create an Alias	57
Step 4: Getting Started (AWS CLI)	58
Exercise 1: Create a Bot	58
Exercise 2: Add a New Utterance	69
Exercise 3: Add a Lambda Function	72
Exercise 4: Publish a Version	75
Exercise 5: Create an Alias	79
Exercise 6: Clean Up	79
Versioning and Aliases	81
Versioning	81
The \$LATEST Version	81
Publishing an Amazon Lex Resource Version	82
Updating an Amazon Lex Resource	82
Deleting an Amazon Lex Resource or Version	83
Aliases	83
Using Lambda Functions	85
Lambda Function Input Event and Response Format	85
Input Event Format	85
Response Format	88
Amazon Lex and AWS Lambda Blueprints	91
Deploying Bots	93

Deploying an Amazon Lex Bot on a Messaging Platform .....	93
Integrating with Facebook .....	94
Integrating with Twilio SMS .....	96
Integrating with Slack .....	98
Deploying an Amazon Lex Bot in Mobile Applications .....	102
Bot Examples .....	103
Example Bot: ScheduleAppointment .....	103
Overview of the Bot Blueprint (ScheduleAppointment) .....	105
Overview of the Lambda Function Blueprint (lex-make-appointment-python) .....	106
Step 1: Create an Amazon Lex Bot .....	106
Step 2: Create a Lambda Function .....	108
Step 3: Update the Intent: Configure a Code Hook .....	108
Step 4: Deploy the Bot on the Facebook Messenger Platform .....	109
Details of Information Flow .....	110
Example Bot: BookTrip .....	121
Step 1: Blueprint Review .....	123
Step 2: Create an Amazon Lex Bot .....	124
Step 3: Create a Lambda function .....	127
Step 4: Add the Lambda Function as a Code Hook .....	127
Details of the Information Flow .....	129
Example: Using a Response Card .....	143
Example: Updating Utterances .....	145
Example: Integrating with a Web site .....	146
Monitoring .....	148
Monitoring Amazon Lex with CloudWatch .....	148
Using CloudWatch Metrics for Amazon Lex .....	148
Viewing Metrics for Amazon Lex .....	149
Creating an Alarm .....	149
CloudWatch Metrics for Amazon Lex .....	149
Runtime Metrics for Amazon Lex .....	150
Channel Association Metrics for Amazon Lex .....	153
Guidelines and Limits .....	154
General Guidelines .....	154
Limits .....	156
General Limits .....	156
Runtime Service Limits .....	157
Model Building Limits .....	157
Authentication and Access Control .....	161
Authentication .....	161
Access Control .....	162
Overview of Managing Access .....	163
Amazon Lex Resources and Operations .....	163
Understanding Resource Ownership .....	163
Managing Access to Resources .....	164
Specifying Policy Elements: Actions, Effects, and Principals .....	165
Specifying Conditions in a Policy .....	165
Using Identity-Based Policies (IAM Policies) for Amazon Lex .....	166
Permissions Required to Use the Amazon Lex Console .....	167
AWS Managed (Predefined) Policies for Amazon Lex .....	169
Examples of Customer Managed Policies .....	170
Amazon Lex API Permissions Reference .....	171
API Reference .....	173
Actions .....	173
Amazon Lex Model Building Service .....	174
Amazon Lex Runtime Service .....	285
Data Types .....	298
Amazon Lex Model Building Service .....	298

Amazon Lex Runtime Service .....	325
Document History .....	330
AWS Glossary .....	331

# What Is Amazon Lex?

Amazon Lex is an AWS service for building conversational interfaces for any applications using voice and text. With Amazon Lex, the same conversational engine that powers Amazon Alexa is now available to any developer, enabling you to build sophisticated, natural language chatbots into your new and existing applications. Amazon Lex provides the deep functionality and flexibility of natural language understanding (NLU) and automatic speech recognition (ASR) so you can build highly engaging user experiences with lifelike, conversational interactions, and create new categories of products.

Amazon Lex enables any developer to build conversational chatbots quickly. With Amazon Lex, no deep learning expertise is necessary—to create a bot, you just specify the basic conversation flow in the Amazon Lex console. Amazon Lex manages the dialogue and dynamically adjusts the responses in the conversation. Using the console, you can build, test, and publish your text or voice chatbot. You can then add the conversational interfaces to bots on mobile devices, web applications, and chat platforms (for example, Facebook Messenger).

Amazon Lex provides pre-built integration with AWS Lambda, and you can easily integrate with many other services on the AWS platform, including Amazon Cognito, AWS Mobile Hub, Amazon CloudWatch, and Amazon DynamoDB. Integration with Lambda provides bots access to pre-built serverless enterprise connectors to link to data in SaaS applications, such as Salesforce, HubSpot, or Marketo.

Some of the benefits of using Amazon Lex include:

- **Simplicity** – Amazon Lex guides you through using the console to create your own chatbot in minutes. You supply just a few example phrases, and Amazon Lex builds a complete natural language model through which the bot can interact using voice and text to ask questions, get answers, and complete sophisticated tasks.
- **Democratized deep learning technologies** – Powered by the same technology as Alexa, Amazon Lex provides ASR and NLU technologies to create a Speech Language Understanding (SLU) system. Through SLU, Amazon Lex takes natural language speech and text input, understands the intent behind the input, and fulfills the user intent by invoking the appropriate business function.

Speech recognition and natural language understanding are some of the most challenging problems to solve in computer science, requiring sophisticated deep learning algorithms to be trained on massive amounts of data and infrastructure. Amazon Lex puts deep learning technologies within reach of all developers, powered by the same technology as Alexa. Amazon Lex chatbots convert incoming

speech to text and understand the user intent to generate an intelligent response, so you can focus on building your bots with differentiated value-add for your customers, to define entirely new categories of products made possible through conversational interfaces.

- **Seamless deployment and scaling** – With Amazon Lex, you can build, test, and deploy your chatbots directly from the Amazon Lex console. Amazon Lex enables you to easily publish your voice or text chatbots for use on mobile devices, web apps, and chat services (for example, Facebook Messenger). Amazon Lex scales automatically so you don't need to worry about provisioning hardware and managing infrastructure to power your bot experience.
- **Built-in integration with the AWS platform** – Amazon Lex has native interoperability with other AWS services, such as Amazon Cognito, AWS Lambda, Amazon CloudWatch, and AWS Mobile Hub. You can take advantage of the power of the AWS platform for security, monitoring, user authentication, business logic, storage, and mobile app development.
- **Cost-effectiveness** – With Amazon Lex, there are no upfront costs or minimum fees. You are charged only for the text or speech requests that are made. The pay-as-you-go pricing and the low cost per request make the service a cost-effective way to build conversational interfaces. With the Amazon Lex free tier, you can easily try Amazon Lex without any initial investment.

## Are You a First-time User of Amazon Lex?

If you are a first-time user of Amazon Lex, we recommend that you read the following sections in order:

1. [Getting Started with Amazon Lex \(p. 21\)](#) – In this section, you set up your account and test Amazon Lex.
2. [API Reference \(p. 173\)](#) – This section provides additional examples that you can use to explore Amazon Lex.

# Amazon Lex: How It Works

Amazon Lex enables you to build applications using a speech or text interface powered by the same technology that powers Amazon Alexa. Following are the typical steps you perform when working with Amazon Lex:

1. Create a bot and configure it with one or more intents that you want to support. You add the configuration so that the bot is able to understand the user's goal (intent), engage in conversation with the user to elicit information, and, after the user provides the necessary data, fulfill the user's intent.
2. Test the bot. You can use the test window client provided by the Amazon Lex console.
3. Publish a version and create an alias.
4. Deploy the bot. You can deploy the bot on platforms such as mobile applications or messaging platforms such as Facebook Messenger.

Before you get started, familiarize yourself with the following Amazon Lex core concepts and terminology:

- **Bot** – A bot performs automated tasks such as ordering a pizza, booking a hotel, ordering flowers, and so on. An Amazon Lex bot is powered by Automatic Speech Recognition (ASR) and Natural Language Understanding (NLU) capabilities, the same technology that powers Amazon Alexa.

Amazon Lex bots can understand user input provided with text or speech and converse in natural language. You can create Lambda functions and add them as code hook in your intent configuration to perform user data validation and fulfillment tasks.

- **Intent** – An intent represents an action that the user wants to perform. You create a bot to support one or more related intents. For example, you might create a bot that orders pizza and drinks. For each intent, you provide the following required information:
  - **Intent name** – A descriptive name for the intent. For example, `orderPizza`.
  - **Sample utterances** – How a user might convey the intent. For example, a user might say "Can I order a pizza please" or "I want to order a pizza".
  - **How to fulfill the intent** – How you want to fulfill the intent after the user provides the necessary information (for example, place order with a local pizza shop). We recommend that you create a Lambda function to fulfill the intent.



You can optionally configure the intent so Amazon Lex simply returns the information back to the client application to do the necessary fulfillment.

In addition to custom intents such as ordering a pizza, Amazon Lex also provides built-in intents to quickly set up your bot. For more information, see [Built-in Intents and Slot Types \(p. 20\)](#).

- **Slot** – An intent can require zero or more slots or parameters. You add slots as part of the intent configuration. At runtime, Amazon Lex prompts the user for specific slot values. The user must provide values for all *required* slots before Amazon Lex can fulfill the intent.

For example, the `OrderPizza` intent requires slots such as pizza size, crust type, and number of pizzas. In the intent configuration, you add these slots. For each slot, you provide slot type and a prompt for Amazon Lex to send to the client to elicit data from the user. A user can reply with a slot value that includes additional words, such as "large pizza please" or "let's stick with small." Amazon Lex can still understand the intended slot value.

- **Slot type** – Each slot has a type. You can create your custom slot types or use built-in slot types. For example, you might create and use the following slot types for the `OrderPizza` intent:
  - Size – With enumeration values `Small`, `Medium`, and `Large`.
  - Crust – With enumeration values `Thick` and `Thin`.

Amazon Lex also provides built-in slot types. For example, `AMAZON.NUMBER` is a built-in slot type that you can use for the number of pizzas ordered. For more information, see [Built-in Intents and Slot Types \(p. 20\)](#).

The following topics provide additional information. We recommend that you review them in order and then explore the [Getting Started with Amazon Lex \(p. 21\)](#) exercises.

#### Topics

- [Programming Model \(p. 4\)](#)
- [Service Permissions \(p. 8\)](#)
- [Managing Messages \(Prompts and Statements\) \(p. 9\)](#)
- [Managing Conversation Context \(p. 18\)](#)
- [Bot Deployment Options \(p. 20\)](#)
- [Built-in Intents and Slot Types \(p. 20\)](#)

## Programming Model

A *bot* is the primary resource type in Amazon Lex. The other resource types in Amazon Lex are *intent*, *slot type*, *alias*, and *bot channel association*.

You create a bot using the Amazon Lex console or the model building API. The console provides a graphical user interface that you use to build a production-ready bot for your application. If you prefer, you can use the model building API through the AWS CLI or your own custom program to create a bot.

After you create a bot, you deploy it on one of the [supported platforms](#) or integrate it into your own application. When a user interacts with the bot, the client application sends requests to the bot using the Amazon Lex runtime API. For example, when a user says "I want to order pizza," your client sends the user input to Amazon Lex using one of the runtime API operations. Users can provide speech or text input.

You can also create Lambda functions and use them in an intent. Use these Lambda function code hooks to perform runtime activities such as initialization, validation of user input, and intent fulfillment. The following sections provide additional information.

#### Topics

- [Model Building API Operations](#) (p. 5)
- [Runtime API Operations](#) (p. 6)
- [Lambda Functions as Code Hooks](#) (p. 6)

## Model Building API Operations

To programmatically create bots, intents, and slot types, use the model building API operations. You can also use the model building API to manage, update, and delete resources for your bot. The model building API operations include:

- [PutBot](#) (p. 261), [PutBotAlias](#) (p. 269), [PutIntent](#) (p. 273), and [PutSlotType](#) (p. 282) to create and update bots, bot aliases, intents, and slot types, respectively.
- [CreateBotVersion](#) (p. 176), [CreateIntentVersion](#) (p. 181), and [CreateSlotTypeVersion](#) (p. 187) to create and publish versions of your bots, intents, and slot types, respectively.
- [GetBot](#) (p. 209) and [GetBots](#) (p. 226) to get a specific bot or a list of bots that you have created, respectively.
- [GetIntent](#) (p. 238) and [GetIntents](#) (p. 243) to get a specific intent or a list of intents that you have created, respectively.
- [GetSlotType](#) (p. 249) and [GetSlotTypes](#) (p. 252) to get a specific slot type or a list of slot types that you have created, respectively.
- [GetBuiltinIntent](#) (p. 232), [GetBuiltinIntents](#) (p. 234), and [GetBuiltinSlotTypes](#) (p. 236) to get an Amazon Lex built-in intent, a list of Amazon Lex built-in intents, or a list of built-in slot types that you can use in your bot, respectively.
- [GetBotChannelAssociation](#) (p. 220) and [GetBotChannelAssociations](#) (p. 223) to get an association between your bot and a messaging platform or a list of the associations between your bot and messaging platforms, respectively.
- [DeleteBot](#) (p. 191), [DeleteBotAlias](#) (p. 193), [DeleteBotChannelAssociation](#) (p. 195), [DeleteIntent](#) (p. 199), and [DeleteSlotType](#) (p. 203) to remove unneeded resources in your account.

You can use the model building API to create custom tools to manage your Amazon Lex resources. For example, there is a limit of 100 versions each for bots, intents, and slot types. You could use the model building API to build a tool that automatically deletes old versions when your bot nears the limit.

To make sure that only one operation updates a resource at a time, Amazon Lex uses checksums. When you use a `Put` API operation—[PutBot](#) (p. 261), [PutBotAlias](#) (p. 269), [PutIntent](#) (p. 273), or [PutSlotType](#) (p. 282)—to update a resource, you must pass the current checksum of the resource in the request. If two tools try to update a resource at the same time, they both provide the same current checksum. The first request to reach Amazon Lex matches the current checksum of the resource.

By the time that the second request arrives, the checksum is different. The second tool receives a `PreconditionFailedException` exception and the update terminates.

The `Get` operations—[GetBot](#) (p. 209), [GetIntent](#) (p. 238), and [GetSlotType](#) (p. 249)—are eventually consistent. If you use a `Get` operation immediately after you create or modify a resource with one of the `Put` operations, the changes might not be returned. After a `Get` operation returns the most recent update, it always returns that updated resource until the resource is modified again. You can determine if an updated resource has been returned by looking at the checksum.

## Runtime API Operations

Client applications use the following runtime API operations to communicate with Amazon Lex:

- [PostContent](#) (p. 286) – Takes speech or text input and returns intent information and a text or speech message to convey to the user. Currently, Amazon Lex supports the following audio formats:

Input audio formats – LPCM and Opus

Output audio formats – MPEG, OGG, and PCM

The `PostContent` operation supports audio input at 8kHz and 16kHz. Applications where the end user speaks with Amazon Lex over the telephone, such as an automated call center, can pass 8kHz audio directly.

- [PostText](#) (p. 293) – Takes text as input and returns intent information and a text message to convey to the user.

Your client application uses the runtime API to call a specific Amazon Lex bot to process utterances — user text or voice input. For example, suppose that a user says "I want pizza." The client sends this user input to a bot using one of the Amazon Lex runtime API operations. From the user input, Amazon Lex recognizes that the user request is for the `OrderPizza` intent defined in the bot. Amazon Lex engages the user in a conversation to gather the required information, or slot data, such as pizza size, toppings, and number of pizzas. After the user provides all of the necessary slot data, Amazon Lex either invokes the Lambda function code hook to fulfill the intent, or returns the intent data to the client, according to the intent configuration.

Use the [PostContent](#) (p. 286) operation when your bot uses speech input. For example, an automated call center application can send speech to a Amazon Lex bot to address customer enquiries without needing to talk to an agent. You can use the 8kHz audio format to send audio directly from the telephone to Amazon Lex.

The test window in the Amazon Lex console uses the [PostContent](#) (p. 286) API to send text and speech requests to Amazon Lex. You use this test window in the [Getting Started with Amazon Lex](#) (p. 21) exercises.

## Lambda Functions as Code Hooks

You can configure your Amazon Lex bot to invoke a Lambda function as a code hook. The code hook can serve multiple purposes:

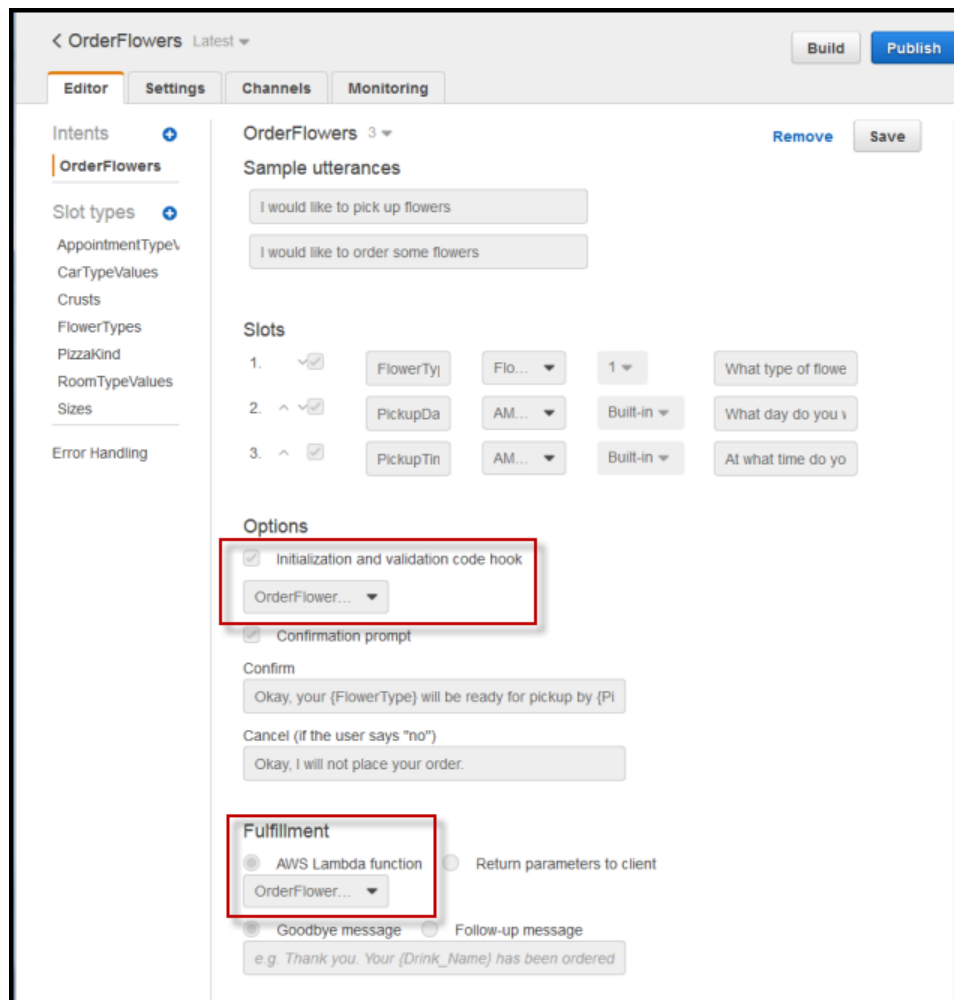
- Customize user interaction – For example, when Joe asks for available pizza toppings, you can use prior knowledge of Joe's choices to display a subset of toppings.

- Validate the user's input – Suppose that Jen wants to pick up flowers after hours. You can validate the time that Jen input and send an appropriate response.
- Fulfill the user's intent – After Joe provides all of the information for his pizza order, Amazon Lex can invoke a Lambda function to place the order with a local pizzeria.

When you configure an intent, you specify Lambda functions as code hooks in the following places:

- Dialog code hook for initialization and validation – This Lambda function is invoked on each user input, assuming Amazon Lex understood the user intent.
- Fulfillment code hook – This Lambda function is invoked after the user provides all of the slot data required to fulfill the intent.

You choose the intent and set the code hooks in the Amazon Lex console, as shown in the following screen shot:



You can also set the code hooks using the `dialogCodeHook` and `fulfillmentActivity` fields in the [PutIntent](#) (p. 273) operation.

One Lambda function can do initialization, validation, and fulfillment. The event data that the Lambda function receives has a field that identifies the caller as either a dialog or fulfillment code hook. You can use this information to execute the appropriate portion of your code.

You can use a Lambda function to build a bot that can navigate complex dialogs. You use the `dialogAction` field in the Lambda function response to direct Amazon Lex to take specific actions. For example, you can use the `ElicitSlot` dialog action to tell Amazon Lex to ask the user for a slot value that isn't required. You can use the `ElicitIntent` dialog action to elicit a new intent when the user is finished with the previous one.

For more information, see [Using Lambda Functions \(p. 85\)](#).

## Service Permissions

Amazon Lex uses AWS Identity and Access Management (IAM) [service-linked roles](#). Amazon Lex assumes these roles to call AWS services on behalf of your bots and bot channels. The roles exist within your account, but are linked to Amazon Lex use cases and have predefined permissions. Only Amazon Lex can assume these roles, and you can't modify their permissions. You can delete them after deleting their related resources using the Amazon Lex console. This protects your Amazon Lex resources because you can't inadvertently remove necessary permissions.

Amazon Lex uses two IAM service-linked roles:

- **AWSServiceRoleForLexBots** — Amazon Lex uses this service-linked role to invoke Amazon Polly to synthesize speech responses for your bot.
- **AWSServiceRoleForLexChannels** — Amazon Lex uses this service-linked role to post text to your bot when managing channels.

You don't need to manually create either of these roles. When you create your first bot using the console, Amazon Lex creates the **AWSServiceRoleForLexBots** role for you. When you first associate a bot with a messaging channel, Amazon Lex creates the **AWSServiceRoleForLexChannels** role for you.

## Creating Resource-Based Policies for AWS Lambda

When invoking Lambda functions, Amazon Lex uses resource-based policies. A *resource-based policy* is attached to a resource; it lets you specify who has access to the resource and which actions they can perform on it. This enables you to narrowly scope permissions between Lambda functions and the intents that you have created. It also allows you to see those permissions in a single policy when you manage Lambda functions that have many event sources.

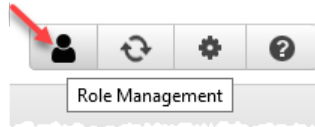
For more information, see [Using Resource-Based Policies for AWS Lambda \(Lambda Function Policies\)](#) in the *AWS Lambda Developer Guide*.

To create resource-based policies for intents that you associate with a Lambda function, you can use the Amazon Lex console. Or, you can use the AWS command line interface (AWS CLI). In the AWS CLI, use the Lambda [AddPermission](#) API with the `Principal` field set to `lex.amazonaws.com` and the `SourceArn` set to the ARN of the intent that is allowed to invoke the function.

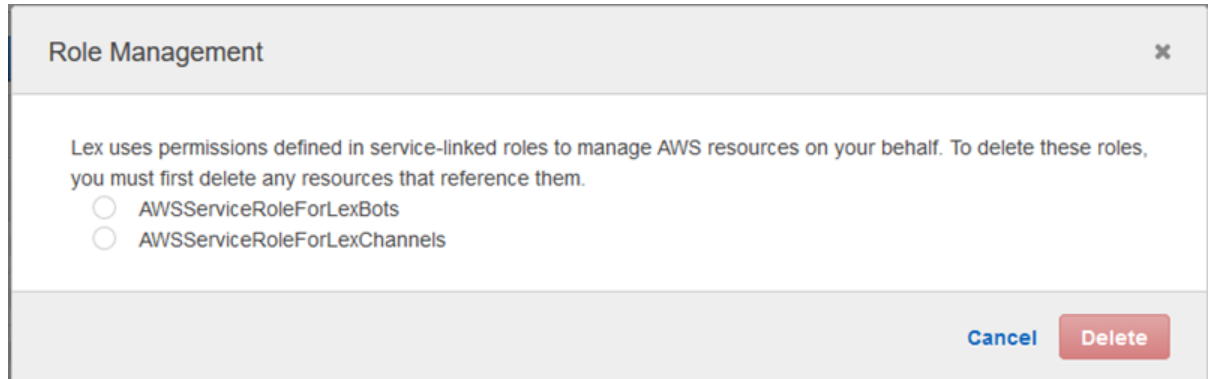
## Deleting Service-Linked Roles

To delete the service-linked roles from your account, use the Role Management tool. Before you can delete a role, you must delete all of the bots or bot channel associations that use the service-linked role.

1. Sign in to the AWS Management Console and open the Amazon Lex console at <https://console.aws.amazon.com/lex/>.
2. On the page that lists bots, choose the **Role Management** tool from the toolbar in the upper-right corner.



3. In the dialog box, choose the service-linked role that you want to delete, and then choose **Delete**.



## Managing Messages (Prompts and Statements)

### Topics

- [Types of Messages \(p. 10\)](#)
- [Contexts for Configuring Messages \(p. 10\)](#)
- [Supported Message Formats \(p. 13\)](#)
- [Response Cards \(p. 14\)](#)

You configure messages that you want a bot to send when you create the bot. Consider the following examples:

- You can configure your bot with the following clarification prompt:

```
I don't understand. What would you like to do?
```

Amazon Lex sends this message to the client if it doesn't understand the user's intent.

- Suppose that you create a bot to support an intent called `orderPizza`. For a pizza order, you want users to provide information such as pizza size, toppings, and crust type. For example, you can configure prompts such as the following:

```
What size pizza you want?  
What toppings you want on the pizza?  
Do you want thick or thin crust?
```

After Amazon Lex determines the user's intent to order pizza, it sends these messages to the client to elicit data from the user.

This section explains designing user interactions in your bot configuration.

## Types of Messages

You can classify messages as follows:

- Prompt – A prompt expects a user response, typically a question.
- Statement – A statement does not expect a response.

The messages you configure can have dynamic components:

- Messages can use the following syntax to refer to slot values of the intent that Amazon Lex is currently aware of:

```
{SlotName}
```

- Messages can use the following syntax to refer to session attributes:

```
[AttributeName]
```

You can have messages that include both slots and session attributes.

At runtime, Amazon Lex substitutes these references with actual values. For example, suppose that you configure the following message in the OrderPizza intent of your bot:

```
"Hey [FirstName], your {PizzaTopping} pizza will arrive in [DeliveryTime] minutes"
```

This message refers to both slot (`PizzaTopping`) and session attributes (`FirstName` and `DeliveryTime`). At runtime, Amazon Lex replaces these placeholders with values and returns the following message to the client:

```
"Hey John, your cheese pizza will arrive in 30 minutes"
```

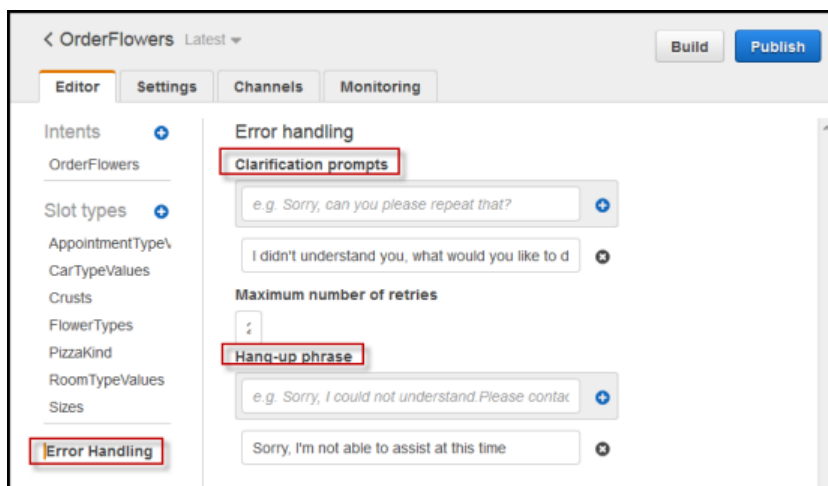
For information about session attributes, see the runtime API operations [PostText \(p. 293\)](#), and [PostContent \(p. 286\)](#). For an example, see [Example Bot: BookTrip \(p. 121\)](#).

If you add code hooks using Lambda functions in your intent configuration, you can create messages dynamically. Lambda functions can generate messages and return them to Amazon Lex to send to the user. By providing the messages while configuring your bot, you can eliminate the need to construct a prompt in code hooks.

## Contexts for Configuring Messages

You can add messages in the following contexts. Use the Amazon Lex console or build-time API to configure your bot:

- Bot-level messages – You can configure your bot with clarification prompts and hang-up messages. At runtime, Amazon Lex uses the clarification prompts if it does not understand the user's intent. You can also configure the number of times that Amazon Lex requests clarification before hanging up with a hang-up message. You configure these bot-level messages with the [PutBot \(p. 261\)](#) operation, or in the **Error Handling** section in the Amazon Lex console, as shown in the following screen shot:



### Note

- If you have a Lambda function configured as a code hook for an intent, the Lambda function might return a response directing Amazon Lex to elicit user intent. If the Lambda function does not provide a message to convey to the user, then Amazon Lex uses the clarification prompt you configured.
- Amazon Lex uses the hang-up statement whenever the user doesn't respond with an appropriate answer for a prompt within the maximum permissible attempts. This includes responses to intent elicitations, slot elicitations, follow-up prompts, and intent confirmations. To configure the maximum permissible attempts, use the [PutBot \(p. 261\)](#) operation, or, in the console, specify it in the **Error Handling** section.
- Intent-level messages – You can configure the intent-level messages such as confirmation prompts, cancel statements, goodbye message, and prompts that Amazon Lex can use to elicit slot values, as shown in the following screenshot:



The screenshot shows the Amazon Lex console for an intent named 'OrderFlowers'. The interface includes tabs for Editor, Settings, Channels, and Monitoring. The Editor tab is active, showing sample utterances, slots, options, and fulfillment settings. Red boxes highlight the 'Options' section (Confirmation prompt and Confirm/Cancel messages) and the 'Fulfillment' section (Goodbye message).

**Sample utterances:**

- I would like to pick up flowers
- I would like to order some flowers

**Slots:**

Slot	Type	Value	Order
1	FlowerType	Flower...	1
2	PickupDate	AMAZ...	Built-in
3	PickupTime	AMAZ...	Built-in

**Options:**

- ☒ Initialization and validation code hook
- OrderFlowersCod...
- ☒ Confirmation prompt
- Confirm:**  
Okay, your {FlowerType} will be ready for pickup by {PickupTime}
- Cancel (if the user says "no"):**  
Okay, I will not place your order.

**Fulfillment:**

- ☒ AWS Lambda function
- ☐ Return parameters to client
- OrderFlowersCod...
- ☒ Goodbye message
- ☐ Follow-up message
- e.g. Thank you. Your {Drink\_Name} has been ordered.

- Confirmation prompts and cancel statements – After a user provides all of the required data, Amazon Lex asks the user for confirmation using the specified message before fulfilling the intent. If the user replies "No" to a confirmation prompt, Amazon Lex returns the cancel statement to the client.
- Goodbye message or follow-up prompts – If you add a Lambda function as a code hook to fulfill the intent, you can configure one of these messages as backup messages. If the Lambda function succeeds but does not provide a message to send to the user, Amazon Lex sends the message that you configured.
- The following is an example of a goodbye message. The example assumes that the application maintains the `DeliveryTime` session attribute.

"I have placed your order for pizza. It will arrive in [DeliveryTime] minutes."

- The following is an example of a follow-up prompt:

"I have placed your order for pizza. Do you want me to do anything else?"

If you configure a follow-up prompt, you must also configure a cancel statement. If the user's reply to a follow-up prompt is a "Yes," Amazon Lex recognizes the user's confirmation and also recognizes the user's intent (`OrderDrink`), and then follows up accordingly. For example:

```
"Yes, I also want to order a drink."
```

If the user says "No," Amazon Lex sends the cancel statement. For example:

```
"Alright. Let me know if you need anything else."
```

- Prompts to elicit value slot values – You must specify at least one prompt message for each of the required slots in an intent. At runtime, Amazon Lex uses one of these messages to prompt the user to provide value for this slot. For example, for a `cityName` slot, the following is a valid prompt:

```
"Which city would you like to fly to?"
```

#### Note

In a Lambda function that is a code hook for an intent, you can override any of the messages that you configured at build time.

You can configure more than one message for a specific context. At runtime, Amazon Lex picks the message with the maximum possible substitutions. For example, to elicit a value for `crust type` in the `OrderPizza` intent, you can configure multiple messages, as follows:

```
Hey [FirstName], what topping would you like for your {PizzaSize} pizza?  
Hey [FirstName], what topping would you like for your pizza?  
What topping would you like?  
Tell me the topping you would like on your pizza.
```

Then, Amazon Lex uses the following order of selection:

- If both the `FirstName` session attribute and the `PizzaSize` slot value are available, Amazon Lex uses the first prompt.
- If the `FirstName` session attribute is available, but the `PizzaSize` slot value isn't, Amazon Lex uses the second prompt.
- If both the session attribute and the slot value aren't available, Amazon Lex randomly chooses the third or fourth prompt.

At runtime, Amazon Lex disregards messages with references to unresolved slot values. If all of the messages for a given context have unresolved references, Amazon Lex throws a `BadRequestException` error. We recommend that you have at least one message without references.

## Supported Message Formats

Amazon Lex supports messages in the following formats: plain text and Speech Synthesis Markup Language (SSML).

If the output mode is text, such as when a client sends requests using the `PostText` API operation or the `PostContent` API operation with the `Accept` HTTP header set to `text/plain; charset=utf-8`, Amazon Lex selects only plain text messages. It disregards SSML messages.

#### Note

- If you configure your bot with only SSML messages and a text client communicates with your bot, Amazon Lex returns a `BadRequestException` error. We recommend that you provide at least one `PlainText` message for each context.

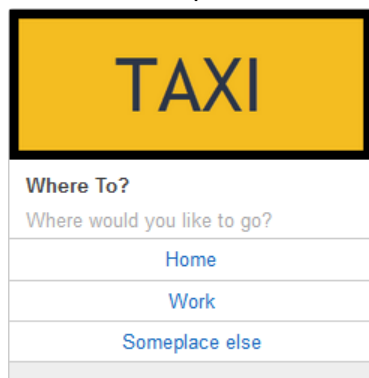
- If `outputDialogMode` in the incoming event is `text`, you must return a `PlainText` message from your AWS Lambda function. For more information, see [Lambda Function Input Event and Response Format \(p. 85\)](#).

Amazon Lex also supports synthesizing audio from SSML. For more information, see [Using SSML](#) in the *Amazon Polly Developer Guide*.

## Response Cards

Use response cards to simplify interactions for your users and increase your bot's accuracy by reducing typographical errors in text interactions. A *response card* contains a set of appropriate responses that a user can select to respond to a prompt. You can send a response card for each prompt that Amazon Lex sends to your client application. You can use response cards with Facebook Messenger, Slack, and Twilio as well as your own client applications.

For example, in a taxi application, you can configure an option in the response card for "Home" and set the value to the user's home address. When the user selects this option, Amazon Lex receives the entire address as the input text.



You can define a response card for the following prompts:

- Conclusion statement
- Confirmation prompt
- Follow-up prompt
- Rejection statement
- Slot type utterances

You can define only one response card for each prompt.

You configure response cards when you create an intent. You can define a static response card at build time using the console or the [PutIntent \(p. 273\)](#) operation. Or you can define a dynamic response card at runtime in a Lambda function. If you define both static and dynamic response cards, the dynamic response card takes precedence.

Amazon Lex sends response cards in the format that the client understands. It transforms response cards for Facebook Messenger, Slack, and Twilio. For other clients, Amazon Lex sends a JSON structure in the [PostText \(p. 293\)](#) response. For example, if the client is Facebook Messenger, Amazon Lex transforms the response card to a generic template. For more information about Facebook Messenger generic templates, see [Generic Template](#) on the Facebook website. For an example of the JSON structure, see [Generating Response Cards Dynamically \(p. 17\)](#).

You can use response cards only with the [PostText \(p. 293\)](#) operation. You can't use response cards with the [PostContent \(p. 286\)](#) operation.

## Defining Static Response Cards

Define static response cards with the [PutBot \(p. 261\)](#) operation or the Amazon Lex console when you create an intent. A static response card is defined at the same time as the intent. Use a static response card when the responses are fixed. Suppose that you are creating a bot with an intent that has a slot for flavor. When defining the flavor slot, you specify prompts, as shown in the following console screenshot:

▼ Slots ⓘ						
Priority	Required	Name	Slot type		Prompt	
		<input type="text"/>	e.g. AMA...		e.g. What city?	⚙️ +
1.	▼ <input checked="" type="checkbox"/>	teaSize	teaSize	1 ▼	What size iced tea wc	⚙️ ✖️
2.	^ <input checked="" type="checkbox"/>	teaFlavor	teaFlavor	1 ▼	Would you like a flavo	⚙️ ✖️

When defining prompts, you can optionally associate a response card and define details with the [PutBot \(p. 261\)](#) operation, or, in the Amazon Lex console, as shown in the following example:

teaFlavor Prompts

Maximum number of retries

2

Corresponding utterances

e.g. I would like to go to {toCity}

+

Prompt response cards

0

Card image

Card title

What Flavor?

Card subtitle

What flavor tea would

Preview

Facebook

Button value

lemon

Button title

Lemon

raspberry

Raspberry

plain

Plain

None

e.g. Button title

None

e.g. Button title

Delete card

What Flavor?

What flavor tea would you like?

Lemon

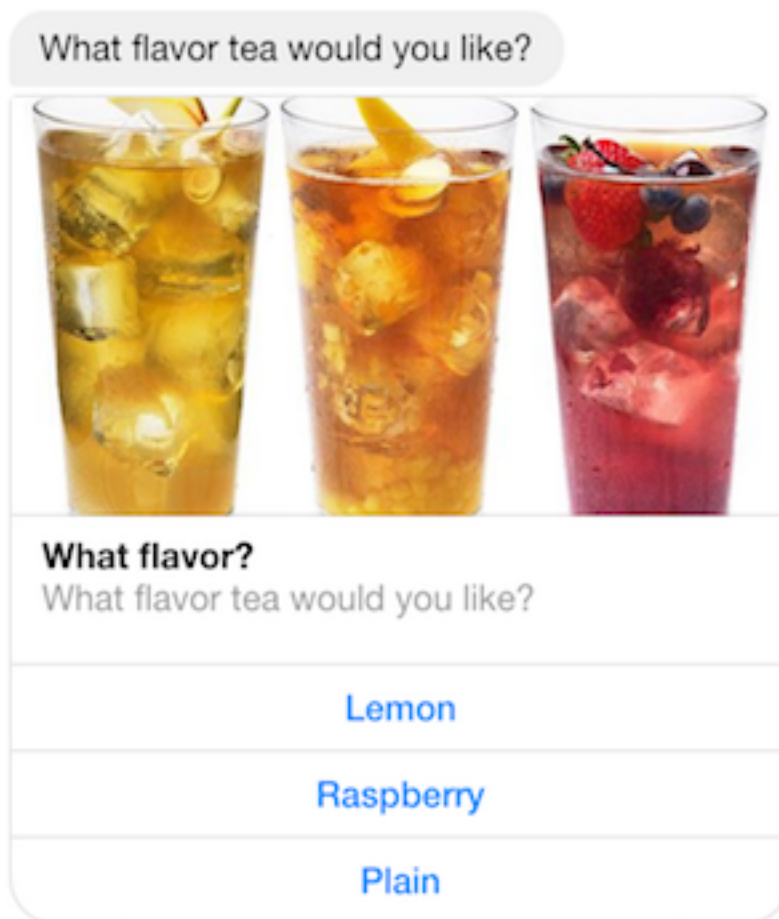
Raspberry

Plain

Cancel

Save

Now suppose that you've integrated your bot with Facebook Messenger. The user can click the buttons to choose a flavor, as shown in the following illustration:



To customize the content of a response card, you can refer to session attributes. At runtime, Amazon Lex substitutes these references with appropriate values from the session attributes. For more information, see [Setting Session Attributes \(p. 18\)](#). For an example, see [Example: Using a Response Card \(p. 143\)](#).

## Generating Response Cards Dynamically

To generate response cards dynamically at runtime, use the initialization and validation Lambda function for the intent. Use a dynamic response card when the responses are determined at runtime in the Lambda function. In response to user input, the Lambda function generates a response card and returns the it in the `dialogAction` section of the response. For more information, see [Response Format \(p. 88\)](#).

The following is a partial response from a Lambda function that shows the `responseCard` element. It generates a user experience similar to the one shown in the preceding section.

```
responseCard: {
  "version": 1,
  "contentType": "application/vnd.amazonaws.card.generic",
  "genericAttachments": [
    {
      "title": "What Flavor?",
      "subtitle": "What flavor do you want?",
      "imageUrl": "Link to image",
      "attachmentLinkUrl": "Link to attachment",
    }
  ]
}
```

```
    "buttons": [
      {
        "text": "Lemon",
        "value": "lemon"
      },
      {
        "text": "Raspberry",
        "value": "raspberry"
      },
      {
        "text": "Plain",
        "value": "plain"
      }
    ]
  }
}
```

For an example, see [Example Bot: ScheduleAppointment \(p. 103\)](#).

## Managing Conversation Context

*Conversation context* is the information that a user shares with Amazon Lex to fulfill an intent. This can be slot data that your user provides and session attributes that are created by the client application and Lambda functions. This topic explains the following:

- [Setting Session Timeout \(p. 18\)](#) — How long Amazon Lex maintains context information for an in-progress intent activity.
- [Setting Session Attributes \(p. 18\)](#) — How to manage application-specific information.
- [Sharing Information Between Intents \(p. 19\)](#) — How you can share context information across intents.

### Setting Session Timeout

For each Amazon Lex bot, you can configure the *session timeout*, the length of time that a conversation session lasts. For each in-progress conversation, Amazon Lex retains context information, slot data and session attributes, until the session ends. By default, the session duration is five minutes, but you can specify any duration between 0 and 1,440 minutes (24 hours).

For example, suppose that you create a `ShoeOrdering` bot that supports intents such as `OrderShoes` and `GetOrderStatus`. When Amazon Lex detects that the user's intent is to order shoes, it asks the user for slot data. For example, it would ask for shoe size, color, brand, etc. If the user provides some of the slot data but doesn't complete the shoe purchase, Amazon Lex remembers all of the slot data and session attributes for the duration of the session. If the user returns before the session expires, he or she can provide the remaining slot data, and complete the purchase.

Set the session timeout when you create a bot using the Amazon Lex console. Or, set it in the AWS command line interface (AWS CLI) when you create or update a bot with the [PutBot \(p. 261\)](#) operation by setting the `idleSessionTTLInSeconds` field.

### Setting Session Attributes

*Session attributes* are application-specific information passed between Amazon Lex and a client application using the `sessionAttributes` field in the [PostContent \(p. 286\)](#) or the [PostText \(p. 293\)](#)

request and response. Amazon Lex passes session attributes to all Lambda functions configured for a bot. If a Lambda function adds or updates session attributes, Amazon Lex passes the new information back to the client application. For example:

- In [Create an Amazon Lex Bot Using a Blueprint](#), the example bot uses the `price` session attribute to maintain the price of flowers ordered. The Lambda function sets this attribute based on the type of flowers ordered. For more information, see [Review the Details of the Information Flow](#).
- In [BookTrip](#), the example bot uses the `currentReservation` session attribute to maintain a copy of the slot type data during the conversation to book a hotel or to book a rental car. For more information, see [Details of the Information Flow \(p. 129\)](#).

Use session attributes in your Lambda functions to initialize and customize prompts and response cards. For example:

- Initialization — In a pizza ordering bot, the client application passes the user's location as a session attribute in the first call to the [PostContent \(p. 286\)](#) or [PostText \(p. 293\)](#) operation. For example, `"Location": "111 Maple Street"`. The Lambda function uses this information to find the closest pizzeria to place the order.
- Personalize prompts — Configure prompts to refer to session attributes, for example, "Hey [FirstName], what toppings would you like?" If you pass the user's first name as a session attribute (`"{FirstName": "Jo"}`), Amazon Lex substitutes the session attribute for the placeholder to send a personalized prompt to the user, "Hey Jo, which toppings would you like?"

Session attributes persist for the duration of the session. Amazon Lex stores them in an encrypted data store until the session ends. The client can create session attributes in a request by calling either the [PostContent \(p. 286\)](#) or the [PostText \(p. 293\)](#) operation with the `sessionAttributes` field set to a value. A Lambda function can create a session attribute in a response. After the client or a Lambda function establishes a session attribute, the attribute is used any time that the `sessionAttribute` field is not included in a request to Amazon Lex.

For example, suppose session attributes `{"x": 1, "y": 2}` have been created for a session. If the client makes a subsequent call to the `PostContent` or `PostText` operation without specifying the `sessionAttributes` field, Amazon Lex calls the Lambda function with the stored session attributes (`{"x": 1, "y": 2}`). If the Lambda function doesn't return session attributes, Amazon Lex returns the stored session attributes back to the client application.

If either the client application or a Lambda function passes session attributes, Amazon Lex updates the stored session attributes. Passing an existing value, such as `{"x": 2}`, updates the stored value. When an empty map, `{}`, is passed, stored values are erased.

## Sharing Information Between Intents

Amazon Lex supports sharing information between intents. To share across intents, you use session attributes.

For example, a user of the `ShoeOrdering` bot starts by ordering shoes. Amazon Lex engages in a conversation with the user, gathering slot data, such as shoe size, color, and brand. When the user places an order, the Lambda function that fulfills the order sets the `orderNumber` session attribute, which contains the order number. To get the status of the order, the user uses the `GetOrderStatus` intent. Amazon Lex can ask the user for slot data, such as order number and order date. Once Amazon Lex has the required information, it can return the status of the order.

However, if you think that your users might switch intents during the same session, you can design your bot to return the status of the latest order. Instead of asking the user for order information again, you can use the `orderNumber` session attribute to share information across intents and fulfill the `GetOrderStatus` intent by returning the status of the last order that the user placed.



For an example of cross-intent information sharing, see [Example Bot: BookTrip \(p. 121\)](#).

## Bot Deployment Options

Currently, Amazon Lex provides the following bot deployment options:

- [AWS Mobile SDK](#) – You can build mobile applications that communicate with Amazon Lex using the AWS Mobile SDKs.
- Facebook Messenger – You can integrate your Facebook Messenger page with your Amazon Lex bot so that end users on Facebook can communicate with the bot. In the current implementation, this integration supports only text input messages.
- Slack – You can integrate your Amazon Lex bot with a Slack messaging application.
- Twilio – You can integrate your Amazon Lex bot with the Twilio Simple Messaging Service (SMS).

For examples, see [Deploying Amazon Lex Bots on Various Platforms \(p. 93\)](#).

## Built-in Intents and Slot Types

Topics

- [Built-in Intents \(p. 20\)](#)
- [Built-in Slots \(p. 20\)](#)

### Built-in Intents

Amazon Lex supports the Alexa standard built-in intents library for common actions. To create an intent from a built-in intent, select one of the existing built-in intents in the console, and add a custom name.

The base intent configuration, such as sample utterances and slots, is available to the intent you are creating.

For a list of built-in intents, see [Standard Built-in Intents](#) in the *Alexa Skills Kit*.

#### Note

- Amazon Lex doesn't support `AMAZON.YesIntent` and `AMAZON.NoIntent`.
- Amazon Lex doesn't support the intents in the [Built-in Intent Library](#) in the *Alexa Skills Kit*.
- In the current implementation, you can't add or remove sample utterances or slots from the base intent. Also, you cannot configure slots for built-in intents.

### Built-in Slots

Amazon Lex supports several built-in slot types from the *Alexa Skills Kit*. You can create slots of these types in your intents. They eliminate the need to create enumeration values for commonly used slot data such as date, time, and location. The built-in slot types do not have versions.

For a list of available built-in slot types, see [Slot Type Reference](#) in the *Alexa Skills Kit*. Amazon Lex supports all of the built-in slot types except the ones marked "Developer Preview."

#### Note

Amazon Lex doesn't support the `AMAZON.LITERAL` built-in slot type.

# Getting Started with Amazon Lex

Learn to use Amazon Lex using the console or the AWS CLI.

Amazon Lex provides API operations that you can integrate with your existing applications. For a list of supported operations, see the [API Reference \(p. 173\)](#). You can use any of the following options:

- **AWS SDK** — When using the SDKs your requests to Amazon Lex are automatically signed and authenticated using the credentials that you provide. This is the recommended choice for building your applications.
- **AWS CLI** — You can use the AWS CLI to access any Amazon Lex feature without having to write any code.
- **AWS Console** — The console is the easiest way to get started testing and using Amazon Lex

If you are new to Amazon Lex, we recommend that you read [Amazon Lex: How It Works \(p. 3\)](#) first.

## Topics

- [Step 1: Set Up an AWS Account and Create an Administrator User \(p. 21\)](#)
- [Step 2: Set Up the AWS Command Line Interface \(p. 23\)](#)
- [Step 3: Getting Started \(Console\) \(p. 23\)](#)
- [Step 4: Getting Started \(AWS CLI\) \(p. 58\)](#)

## Step 1: Set Up an AWS Account and Create an Administrator User

Before you use Amazon Lex for the first time, complete the following tasks:

1. [Sign Up for AWS \(p. 21\)](#)
2. [Create an IAM User \(p. 22\)](#)

## Sign Up for AWS

If you already have an AWS account, skip this task.

When you sign up for Amazon Web Services (AWS), your AWS account is automatically signed up for all services in AWS, including Amazon Lex. You are charged only for the services that you use.

With Amazon Lex, you pay only for the resources that you use. If you are a new AWS customer, you can get started with Amazon Lex for free. For more information, see [AWS Free Usage Tier](#).

If you already have an AWS account, skip to the next task. If you don't have an AWS account, use the following procedure to create one.

### To create an AWS account

1. Open <https://aws.amazon.com/>, and then choose **Create an AWS Account**.
2. Follow the online instructions.

Part of the sign-up procedure involves receiving a phone call and entering a PIN using the phone keypad.

Write down your AWS account ID because you'll need it for the next task.

## Create an IAM User

Services in AWS, such as Amazon Lex, require that you provide credentials when you access them so that the service can determine whether you have permissions to access the resources owned by that service. The console requires your password. You can create access keys for your AWS account to access the AWS CLI or API.

However, we don't recommend that you access AWS using the credentials for your AWS account. Instead, we recommend that you:

- Use AWS Identity and Access Management (IAM) to create an IAM user
- Add the user to an IAM group with administrative permissions
- Grant administrative permissions to the IAM user that you created.

You can then access AWS using a special URL and the IAM user's credentials.

The Getting Started exercises in this guide assume that you have a user (`adminuser`) with administrator privileges. Follow the procedure to create `adminuser` in your account.

### To create an administrator user and sign in to the console

1. Create an administrator user called `adminuser` in your AWS account. For instructions, see [Creating Your First IAM User and Administrators Group](#) in the *IAM User Guide*.
2. As a user, you can sign in to the AWS Management Console using a special URL. For more information, [How Users Sign In to Your Account](#) in the *IAM User Guide*.

For more information about IAM, see the following:

- [Identity and Access Management \(IAM\)](#)
- [Getting Started](#)
- [IAM User Guide](#)

## Next Step

[Step 2: Set Up the AWS Command Line Interface \(p. 23\)](#)

## Step 2: Set Up the AWS Command Line Interface

If you prefer to use Amazon Lex with the AWS Command Line Interface (AWS CLI), download and configure it.

### Important

You don't need the AWS CLI to perform the steps in the Getting Started exercises. However, some of the later exercises in this guide use the AWS CLI. If you prefer to start by using the console, skip this step and go to [Step 3: Getting Started \(Console\) \(p. 23\)](#). Later, when you need the AWS CLI, return here to set it up.

### To set up the AWS CLI

1. Download and configure the AWS CLI. For instructions, see the following topics in the *AWS Command Line Interface User Guide*:
  - [Getting Set Up with the AWS Command Line Interface](#)
  - [Configuring the AWS Command Line Interface](#)
2. Add a named profile for the administrator user to the end of the AWS CLI config file. You use this profile when executing AWS CLI commands. For more information about named profiles, see [Named Profiles](#) in the *AWS Command Line Interface User Guide*.

```
[profile adminuser]
aws_access_key_id = adminuser access key ID
aws_secret_access_key = adminuser secret access key
region = aws-region
```

For a list of available AWS Regions, see [Regions and Endpoints](#) in the *Amazon Web Services General Reference*.

3. Verify the setup by typing the Help command at the command prompt:

```
aws help
```

[Step 3: Getting Started \(Console\) \(p. 23\)](#)

## Step 3: Getting Started (Console)

The easiest way to learn how to use Amazon Lex is by using the console. To get you started, we created the following exercises, all of which use the console:

- Exercise 1 — Create an Amazon Lex bot using a blueprint, a predefined bot that provides all of the necessary bot configuration. You do only a minimum of work to test the end-to-end setup.

In addition, you use the Lambda function blueprint, provided by AWS Lambda, to create a Lambda function. The function is a code hook that uses predefined code that is compatible with your bot.

- Exercise 2 — Create a custom bot by manually creating and configuring a bot. You also create a Lambda function as a code hook. Sample code is provided.
- Exercise 3 — Publish a bot, and then create a new version of it. As part of this exercise you create an alias that points to the bot version.

Topics

- [Exercise 1: Create an Amazon Lex Bot Using a Blueprint \(Console\) \(p. 24\)](#)
- [Exercise 2: Create a Custom Amazon Lex Bot \(p. 48\)](#)
- [Exercise 3: Publish a Version and Create an Alias \(p. 57\)](#)

## Exercise 1: Create an Amazon Lex Bot Using a Blueprint (Console)

In this exercise, you do the following:

- Create your first Amazon Lex bot, and test it in the Amazon Lex console.

For this exercise, you use the **OrderFlowers** blueprint. For information about blueprints, see [Amazon Lex and AWS Lambda Blueprints \(p. 91\)](#).

- Create an AWS Lambda function and test it in the Lambda console. While processing a request, your bot calls this Lambda function. For this exercise, you use a Lambda blueprint (**lex-order-flowers-python**) provided in the AWS Lambda console to create your Lambda function. The blueprint code illustrates how you can use the same Lambda function to perform initialization and validation, and to fulfill the `OrderFlowers` intent.
- Update the bot to add the Lambda function as the code hook to fulfill the intent. Test the end-to-end experience.

The following sections explain what the blueprints do.

### Amazon Lex Bot: Blueprint Overview

You use the **OrderFlowers** blueprint to create an Amazon Lex bot. For more information about the structure of a bot, see [Amazon Lex: How It Works \(p. 3\)](#). The bot is preconfigured as follows:

- **Intent** – `OrderFlowers`
- **Slot types** – One custom slot type called `FlowerTypes` with enumeration values: `roses`, `lilies`, and `tulips`.
- **Slots** – The intent requires the following information (that is, slots) before the bot can fulfill the intent.
  - `PickupTime` (AMAZON.TIME built-in type)
  - `FlowerType` (`FlowerTypes` custom type)
  - `PickupDate` (AMAZON.DATE built-in type)
- **Utterance** – The following sample utterances indicate the user's intent:
  - "I would like to pick up flowers."
  - "I would like to order some flowers."
- **Prompts** – After the bot identifies the intent, it uses the following prompts to fill the slots:
  - Prompt for the `FlowerType` slot – "What type of flowers would you like to order?"
  - Prompt for the `PickupDate` slot – "What day do you want the {FlowerType} to be picked up?"
  - Prompt for the `PickupTime` slot – "At what time do you want the {FlowerType} to be picked up?"
  - Confirmation statement – "Okay, your {FlowerType} will be ready for pickup by {PickupTime} on {PickupDate}. Does this sound okay?"

## AWS Lambda Function: Blueprint Summary

The Lambda function in this exercise performs both initialization and validation and fulfillment tasks. Therefore, after creating the Lambda function, you update the intent configuration by specifying the same Lambda function as a code hook to handle both the initialization and validation and fulfillment tasks.

- As an initialization and validation code hook, the Lambda function performs basic validation. For example, if the user provides a time for pickup that is outside of normal business hours, the Lambda function directs Amazon Lex to re-prompt the user for the time.
- As part of the fulfillment code hook, the Lambda function returns a summary message indicating that the flower order has been placed (that is, the intent is fulfilled).

### Next Step

[Step 1: Create an Amazon Lex Bot \(Console\) \(p. 25\)](#)

## Step 1: Create an Amazon Lex Bot (Console)

For this exercise, create a bot for ordering flowers, called OrderFlowersBot.

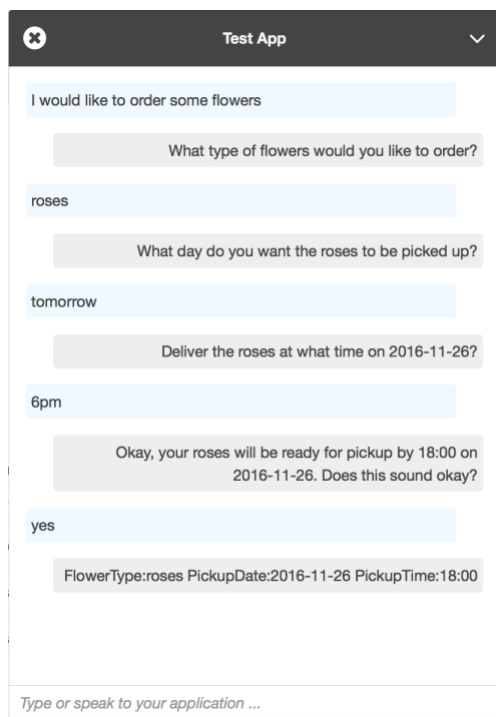
To create an Amazon Lex bot (console)

1. Sign in to the AWS Management Console and open the Amazon Lex console at <https://console.aws.amazon.com/lex/>.
2. On the **Bots** page, choose **Create**.
3. On the **Create your Lex bot** page, provide the following information, and then choose **Create**.
  - Choose the **OrderFlowers** blueprint.
  - Leave the default bot name (OrderFlowers).
4. Choose **Create**. The console makes the necessary requests to Amazon Lex to save the configuration. The console then displays the bot editor window.
5. To build the bot, choose **Build** twice.
6. Test the bot.

### Note

You can test the bot by typing text into the test window, or, for compatible browsers, by choosing the microphone button in the test window and speaking.

Use the following example text to engage in conversation with the bot to order flowers:



From this input, the bot infers the `OrderFlowers` intent and prompts for slot data. When you provide all of the required slot data, the bot fulfills the intent (`OrderFlowers`) by returning all of the information to the client application (in this case, the console). The console shows the information in the test window.

Specifically:

- In the statement "What day do you want the roses to be picked up?,"the term "roses" appears because the prompt for the `pickupDate` slot is configured using substitutions, `{FlowerType}`. Verify this in the console.
- The "Okay, your roses will be ready..." statement is the confirmation prompt that you configured.
- The last statement ("`FlowerType:roses...`") is just the slot data that is returned to the client, in this case, in the test window. In the next exercise, you use a Lambda function to fulfill the intent, in which case you get a message indicating that the order is fulfilled.

### Next Step

[Step 2 \(Optional\): Review the Details of Information Flow \(Console\) \(p. 26\)](#)

## Step 2 (Optional): Review the Details of Information Flow (Console)

This section explains the flow of information between a client and Amazon Lex for each user input in our example conversation.

To see the flow of information for spoken or typed content, choose the appropriate topic.

### Topics

- [Step 2a \(Optional\): Review the Details of the Spoken Information Flow \(Console\) \(p. 27\)](#)
- [Step 2b \(Optional\): Review the Details of the Typed Information Flow \(Console\) \(p. 31\)](#)

## Step 2a (Optional): Review the Details of the Spoken Information Flow (Console)

This section explains the flow of information between the client and Amazon Lex when the client uses speech to send requests. For more information, see [PostContent](#) (p. 286).

1. The user says: I would like to order some flowers.
  - a. The client (console) sends the following [PostContent](#) (p. 286) request to Amazon Lex:

```
POST /bot/OrderFlowers/alias/$LATEST/user/4o9wwdhx6nlheferh6a73fujd3118f5w/content
HTTP/1.1
x-amz-lex-session-attributes: "e30="
Content-Type: "audio/x-l16; sample-rate=16000; channel-count=1"
Accept: "audio/mpeg"

Request body
input stream
```

Both the request URI and the body provide information to Amazon Lex:

- Request URI – Provides the bot name (*OrderFlowers*), bot alias (*\$LATEST*), and the user name (a random string that identifies the user). *content* indicates that this is a *PostContent* API request (not a *PostText* request).
- Request headers
  - *x-amz-lex-session-attributes* – The base64-encoded value represents "{}". When the client makes the first request, there are no session attributes.
  - *Content-Type* – Reflects the audio format.
- Request body – The user input audio stream ("I would like to order some flowers").

### Note

If the user chooses to send text ("I would like to order some flowers") to the *PostContent* API instead of speaking, the request body is the user input. The *Content-Type* header is set accordingly:

```
POST /bot/OrderFlowers/alias/$LATEST/user/4o9wwdhx6nlheferh6a73fujd3118f5w/
content HTTP/1.1
x-amz-lex-session-attributes: "e30="
Content-Type: "text/plain; charset=utf-8"
Accept: accept

Request body
input stream
```

- b. From the input stream, Amazon Lex detects the intent (*OrderFlowers*). It then chooses one of the intent's slots (in this case, the *FlowerType*) and one of its value elicitation prompts, and then sends a response with the following headers:

```
x-amz-lex-dialog-state:ElicitSlot
x-amz-lex-input-transcript:I would like to order some flowers.
x-amz-lex-intent-name:OrderFlowers
x-amz-lex-message:What type of flowers would you like to order?
x-amz-lex-session-attributes:e30=
x-amz-lex-slot-to-elicite:FlowerType
x-amz-lex-
slots:eyJQaWNrdXBuZW11IjpudWxsLCJGbg93ZXJueXB1IjpudWxsLCJQaWNrdXBeyXR1IjpudWxsZmQ==
```



The header values provide the following information:

- `x-amz-lex-input-transcript` – Provides the transcript of the audio (user input) from the request
- `x-amz-lex-message` – Provides the transcript of the audio Amazon Lex returned in the response
- `x-amz-lex-slots` – The base64 encoded version of the slots and values:

```
{ "PickupTime": null, "FlowerType": null, "PickupDate": null }
```

- `x-amz-lex-session-attributes` – The base64-encoded version of the session attributes ({})

The client plays the audio in the response body.

2. The user says: roses

- a. The client (console) sends the following [PostContent](#) (p. 286) request to Amazon Lex:

```
POST /bot/OrderFlowers/alias/$LATEST/user/4o9wwdhx6nlheferh6a73fujd3118f5w/content
HTTP/1.1
x-amz-lex-session-attributes: "e30="
Content-Type: "audio/x-l16; sample-rate=16000; channel-count=1"
Accept: "audio/mpeg"

Request body

```

The request body is the user input audio stream (roses). The `sessionAttributes` remains empty.

- b. Amazon Lex interprets the input stream in the context of the current intent (it remembers that it had asked this user for information pertaining to the `FlowerType` slot). Amazon Lex first updates the slot value for the current intent. It then chooses another slot (`PickupDate`), along with one of its prompt messages (When do you want to pick up the roses?), and returns a response with the following headers:

```
x-amz-lex-dialog-state:ElicitSlot
x-amz-lex-input-transcript:roses
x-amz-lex-intent-name:OrderFlowers
x-amz-lex-message:When do you want to pick up the roses?
x-amz-lex-session-attributes:e30=
x-amz-lex-slot-to-elicite:PickupDate
x-amz-lex-slots:eyJQaWNrdXBuaw11IjpuclWxsLCJGbgG93ZXJueXB1Ijoicm9zaSdzIiwiaUGlja3VwRGF0ZSI6bnVsbH0=
```

The header values provide the following information:

- `x-amz-lex-slots` – The base64-encoded version of the slots and values:

```
{ "PickupTime": null, "FlowerType": "roses", "PickupDate": null }
```

- `x-amz-lex-session-attributes` – The base64-encoded version of the session attributes ({})

The client plays the audio in the response body.

3. The user says: tomorrow

- a. The client (console) sends the following [PostContent](#) (p. 286) request to Amazon Lex:

```
POST /bot/OrderFlowers/alias/$LATEST/user/4o9wwdhx6nlheferh6a73fujd3118f5w/content
HTTP/1.1
x-amz-lex-session-attributes: "e30="
Content-Type: "audio/x-l16; sample-rate=16000; channel-count=1"
Accept: "audio/mpeg"

Request body

```

The request body is the user input audio stream ("tomorrow"). The `sessionAttributes` remains empty.

- b. Amazon Lex interprets the input stream in the context of the current intent (it remembers that it had asked this user for information pertaining to the `PickupDate` slot). Amazon Lex updates the slot (`PickupDate`) value for the current intent. It then chooses another slot to elicit value for (`PickupTime`) and one of the value elicitation prompts (When do you want to pick up the roses on 2017-03-18?), and returns a response with the following headers:

```
x-amz-lex-dialog-state:ElicitSlot
x-amz-lex-input-transcript:tomorrow
x-amz-lex-intent-name:OrderFlowers
x-amz-lex-message:When do you want to pick up the roses on 2017-03-18?
x-amz-lex-session-attributes:e30=
x-amz-lex-slot-to-elicite:PickupTime
x-amz-lex-
slots:eyJQaWNrdXBuaW11IjpudWxsLCJGbg93ZXJueXB1Ijoicm9zaSdzIiwUGlja3VwRGF0ZSI6IjIwMTctMDMtMTg1fQ
x-amzn-RequestId:3a205b70-0b69-11e7-b447-eb69face3e6f
```

The header values provide the following information:

- `x-amz-lex-slots` – The base64-encoded version of the slots and values:

```
{"PickupTime":null,"FlowerType":"roses","PickupDate":"2017-03-18"}
```

- `x-amz-lex-session-attributes` – The base64-encoded version of the session attributes ({}).

The client plays the audio in the response body.

4. The user says: 6 pm
- a. The client (console) sends the following [PostContent](#) (p. 286) request to Amazon Lex:

```
POST /bot/OrderFlowers/alias/$LATEST/user/4o9wwdhx6nlheferh6a73fujd3118f5w/content
HTTP/1.1
x-amz-lex-session-attributes: "e30="
Content-Type: "text/plain; charset=utf-8"
Accept: "audio/mpeg"

Request body

```

The request body is the user input audio stream ("6 pm"). The `sessionAttributes` remains empty.

- b. Amazon Lex interprets the input stream in the context of the current intent (it remembers that it had asked this user for information pertaining to the `PickupTime` slot). It first updates the slot value for the current intent.

Now Amazon Lex detects that it has information for all of the slots. However, the `OrderFlowers` intent is configured with a confirmation message. Therefore, Amazon Lex needs an explicit confirmation from the user before it can proceed to fulfill the intent. It sends a response with the following headers requesting confirmation before ordering the flowers:

```
x-amz-lex-dialog-state:ConfirmIntent
x-amz-lex-input-transcript:six p. m.
x-amz-lex-intent-name:OrderFlowers
x-amz-lex-message:Okay, your roses will be ready for pickup by 18:00 on 2017-03-18.
  Does this sound okay?
x-amz-lex-session-attributes:e30=
x-amz-lex-
slots:eyJQaWNrdXBuaW11IjoIMTg6MDAiLCJGbG93ZXJueXB1Ijoicm9zaSdzIiwUGlja3VwRGF0ZSI6IjIwMTctMDMtM
x-amzn-RequestId:083ca360-0b6a-11e7-b447-eb69face3e6f
```

The header values provide the following information:

- `x-amz-lex-slots` – The base64-encoded version of the slots and values:

```
{"PickupTime":"18:00","FlowerType":"roses","PickupDate":"2017-03-18"}
```

- `x-amz-lex-session-attributes` – The base64-encoded version of the session attributes ({}).

The client plays the audio in the response body.

5. The user says: Yes
  - a. The client (console) sends the following [PostContent](#) (p. 286) request to Amazon Lex:

```
POST /bot/OrderFlowers/alias/$LATEST/user/4o9wwdhx6nlheferh6a73fujd3118f5w/content
HTTP/1.1
x-amz-lex-session-attributes: "e30="
Content-Type: "audio/x-l16; sample-rate=16000; channel-count=1"
Accept: "audio/mpeg"

Request body

```

The request body is the user input audio stream ("Yes"). The `sessionAttributes` remains empty.

- b. Amazon Lex interprets the input stream and understands that the user wants to proceed with the order. The `OrderFlowers` intent is configured with `ReturnIntent` as the fulfillment activity. This directs Amazon Lex to return all of the intent data to the client. Amazon Lex returns a response with the following:

```
x-amz-lex-dialog-state:ReadyForFulfillment
x-amz-lex-input-transcript:yes
x-amz-lex-intent-name:OrderFlowers
x-amz-lex-session-attributes:e30=
x-amz-lex-
slots:eyJQaWNrdXBuaW11IjoIMTg6MDAiLCJGbG93ZXJueXB1Ijoicm9zaSdzIiwUGlja3VwRGF0ZSI6IjIwMTctMDMtM
```

The `x-amz-lex-dialog-state` response header is set to `ReadyForFulfillment`. The client can then fulfill the intent.

6. Now, retest the bot. To establish a new (user) context, choose the **Clear** link in the console. Provide data for the `OrderFlowers` intent, and include some invalid data. For example:

- Jasmine as the flower type (it is not one of the supported flower types)
- Yesterday as the day when you want to pick up the flowers

Notice that the bot accepts these values because you don't have any code to initialize and validate the user data. In the next section, you add a Lambda function to do this. Note the following about the Lambda function:

- It validates slot data after every user input. It fulfills the intent at the end. That is, the bot processes the flower order and returns a message to the user instead of simply returning slot data to the client. For more information, see [Using Lambda Functions \(p. 85\)](#).
- It also sets the session attributes. For more information about session attributes, see [PostText \(p. 293\)](#).

After you complete the Getting Started section, you can do the additional exercises ([Additional Examples: Creating Amazon Lex Bots \(p. 103\)](#)). [Example Bot: BookTrip \(p. 121\)](#) uses session attributes to share cross-intent information to engage in a dynamic conversation with the user.

## Next Step

[Step 3: Create a Lambda Function \(Console\) \(p. 35\)](#)

## Step 2b (Optional): Review the Details of the Typed Information Flow (Console)

This section explains flow of information between client and Amazon Lex in which the client uses the `PostText` API to send requests. For more information, see [PostText \(p. 293\)](#).

1. User types: I would like to order some flowers
  - a. The client (console) sends the following [PostText \(p. 293\)](#) request to Amazon Lex:

```
POST /bot/OrderFlowers/alias/$LATEST/user/4o9wwd hx6nlheferh6a73fujd3118f5w/text
"Content-Type": "application/json"
"Content-Encoding": "amz-1.0"

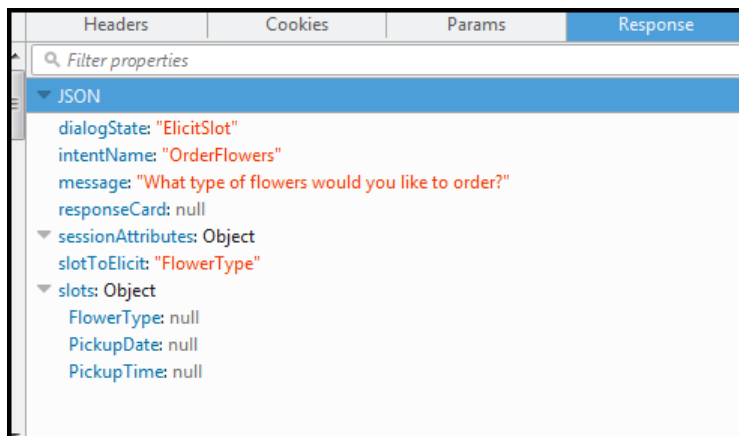
{
  "inputText": "I would like to order some flowers",
  "sessionAttributes": {}
}
```

Both the request URI and the body provide information to Amazon Lex:

- Request URI – Provides bot name (`OrderFlowers`), bot alias (`$LATEST`), and user name (a random string identifying the user). The trailing `text` indicates that it is a `PostText` API request (and not `PostContent`).
  - Request body – Includes the user input (`inputText`) and empty `sessionAttributes`. When the client makes the first request, there are no session attributes. The Lambda function initiates them later.
- b. From the `inputText`, Amazon Lex detects the intent (`OrderFlowers`). This intent does not have any code hooks (that is, the Lambda functions) for initialization and validation of user input or fulfillment.

Amazon Lex chooses one of the intent's slots (`FlowerType`) to elicit the value. It also selects one of the value-elicitation prompts for the slot (all part of the intent configuration), and then sends

the following response back to the client. The console displays the message in the response to the user.



The client displays the message in the response.

2. User types: roses

- a. The client (console) sends the following [PostText](#) (p. 293) request to Amazon Lex:

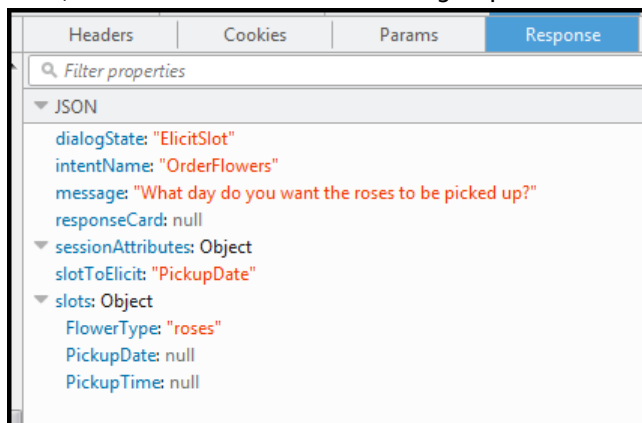
```
POST /bot/OrderFlowers/alias/$LATEST/user/4o9wwdhx6nlheferh6a73fujd3118f5w/text
"Content-Type": "application/json"
"Content-Encoding": "amz-1.0"

{
  "inputText": "roses",
  "sessionAttributes": {}
}
```

The `inputText` in the request body provides user input. The `sessionAttributes` remains empty.

- b. Amazon Lex first interprets the `inputText` in the context of the current intent—the service remembers that it had asked the specific user for information about the `FlowerType` slot. Amazon Lex first updates the slot value for the current intent and chooses another slot (`PickupDate`) along with one of its prompt messages—What day do you want the roses to be picked up?— for the slot.

Then, Amazon Lex returns the following response:



The client displays the message in the response.

3. User types: tomorrow

- a. The client (console) sends the following [PostText \(p. 293\)](#) request to Amazon Lex:

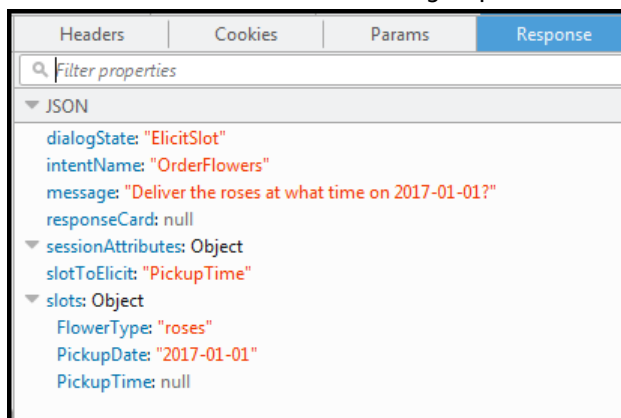
```
POST /bot/OrderFlowers/alias/$LATEST/user/4o9wwdhx6nlheferh6a73fujd3118f5w/text
"Content-Type": "application/json"
"Content-Encoding": "amz-1.0"

{
  "inputText": "tomorrow",
  "sessionAttributes": {}
}
```

The `inputText` in the request body provides user input. The `sessionAttributes` remains empty.

- b. Amazon Lex first interprets the `inputText` in the context of the current intent—the service remembers that it had asked the specific user for information about the `PickupDate` slot. Amazon Lex updates the slot (`PickupDate`) value for the current intent. It chooses another slot to elicit value for (`PickupTime`). It returns one of the value-elicitation prompts—Deliver the roses at what time on 2017-01-01?—to the client.

Amazon Lex then returns the following response:



The client displays the message in the response.

4. User types: 6 pm

- a. The client (console) sends the following [PostText \(p. 293\)](#) request to Amazon Lex:

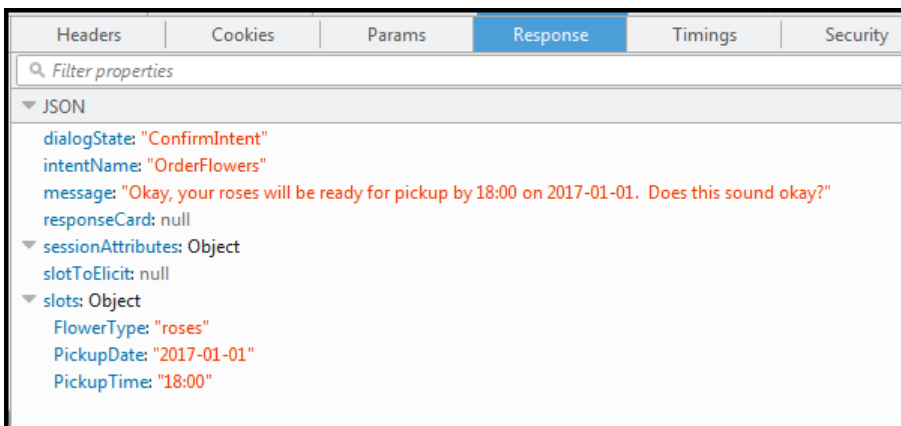
```
POST /bot/OrderFlowers/alias/$LATEST/user/4o9wwdhx6nlheferh6a73fujd3118f5w/text
"Content-Type": "application/json"
"Content-Encoding": "amz-1.0"

{
  "inputText": "6 pm",
  "sessionAttributes": {}
}
```

The `inputText` in the request body provides user input. The `sessionAttributes` remains empty.

- b. Amazon Lex first interprets the `inputText` in the context of the current intent—the service remembers that it had asked the specific user for information about the `PickupTime` slot. Amazon Lex first updates the slot value for the current intent. Now Amazon Lex detects that it has information for all the slots.

The `OrderFlowers` intent is configured with a confirmation message. Therefore, Amazon Lex needs an explicit confirmation from the user before it can proceed to fulfill the intent. Amazon Lex sends the following message to the client requesting confirmation before ordering the flowers:



The client displays the message in the response.

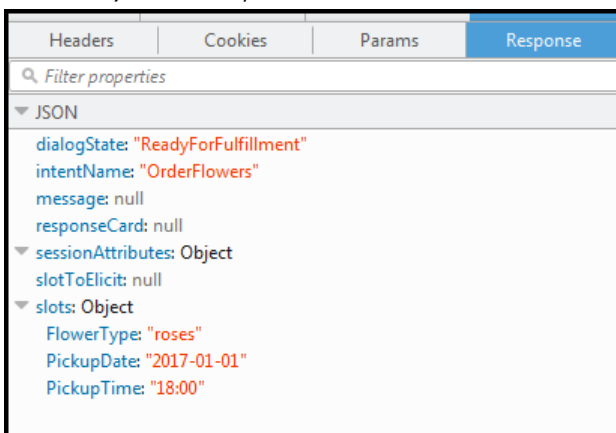
5. User types: Yes
  - a. The client (console) sends the following [PostText](#) (p. 293) request to Amazon Lex:

```
POST /bot/OrderFlowers/alias/$LATEST/user/4o9wwdxx6nlheferh6a73fujd3118f5w/text
"Content-Type": "application/json"
"Content-Encoding": "amz-1.0"

{
  "inputText": "Yes",
  "sessionAttributes": {}
}
```

The `inputText` in the request body provides user input. The `sessionAttributes` remains empty.

- b. Amazon Lex interprets the `inputText` in the context of confirming the current intent. It understands that the user wants to proceed with the order. The `OrderFlowers` intent is configured with `ReturnIntent` as the fulfillment activity (there is no Lambda function to fulfill the intent). Therefore, Amazon Lex returns the slot data to the client.



Amazon Lex set the `dialogState` to `ReadyForFulfillment`. The client can then fulfill the intent.

6. Now test the bot again. To do that, you must choose the **Clear** link in the console to establish a new (user) context. Now as you provide data for the order flowers intent, try to provide invalid data. For example:
  - Jasmine as the flower type (it is not one of the supported flower types).
  - Yesterday as the day when you want to pick up the flowers.

Notice that the bot accepts these values because you don't have any code to initialize/validate user data. In the next section, you add a Lambda function to do this. Note the following about the Lambda function:

- The Lambda function validates slot data after every user input. It fulfills the intent at the end. That is, the bot processes the flowers order and returns a message to the user instead of simply returning slot data to the client. For more information, see [Using Lambda Functions \(p. 85\)](#).
- The Lambda function also sets the session attributes. For more information about session attributes, see [PostText \(p. 293\)](#).

After you complete the Getting Started section, you can do the additional exercises ([Additional Examples: Creating Amazon Lex Bots \(p. 103\)](#)). [Example Bot: BookTrip \(p. 121\)](#) uses session attributes to share cross-intent information to engage in a dynamic conversation with the user.

### Next Step

[Step 3: Create a Lambda Function \(Console\) \(p. 35\)](#)

## Step 3: Create a Lambda Function (Console)

Create a Lambda function (using the **lex-order-flowers-python** blueprint) and perform test invocation using sample event data in the AWS Lambda console. This Lambda function is written in Node.js.

You return to the Amazon Lex console and add the Lambda function as the code hook to fulfill the `OrderFlowers` intent in the `OrderFlowersBot` that you created in the preceding section.

### To create the Lambda function (console)

1. Sign in to the AWS Management Console and open the AWS Lambda console at <https://console.aws.amazon.com/lambda/>.
2. Choose **Create a Lambda function**.
3. On **Select blueprint**, type `lex` in the filter text box to find the blueprint, choose the `lex-order-flowers-python` blueprint.

Lambda function blueprints are provided in both Node.js and Python. For this exercise, use the Python-based blueprint.

4. Choose **Next** on the **Configure Triggers** page.
5. On the **Configure function** page, do the following, and then choose **Next**.
  - Type a Lambda function name (`OrderFlowersCodeHook`).
  - For the IAM role, choose **Create a new role from template(s)**.
  - Type a role name.
  - Leave the other default values.
6. On the **Review** page, choose **Create function**.
7. Test the Lambda function.
  - a. Choose **Actions**, **Configure test event**.



- b. Choose **Lex-Order Flowers (preview)** from the **Sample event template** list. This sample event matches the Amazon Lex request/response model (see [Using Lambda Functions \(p. 85\)](#)).
- c. Choose **Save and test**.
- d. Verify that the Lambda function successfully executed. The response in this case matches the Amazon Lex response model.

### Next Step

[Step 4: Add the Lambda Function as Code Hook \(Console\) \(p. 36\)](#)

## Step 4: Add the Lambda Function as Code Hook (Console)

In this section, you update the configuration of the `OrderFlowers` intent to use the Lambda function as follows:

- First use the Lambda function as a code hook to perform fulfillment of the `OrderFlowers` intent. You test the bot and verify that you received a fulfillment message from the Lambda function. Amazon Lex invokes the Lambda function only after you provide data for all the required slots for ordering flowers.
- Configure the same Lambda function as a code hook to perform initialization and validation. You test and verify that the Lambda function performs validation (as you provide slot data).

### To add a Lambda function as a code hook (console)

1. In the Amazon Lex console, select the **OrderFlowers** bot. The console shows the **OrderFlowers** intent. Make sure that the intent version is set to `$LATEST` because this is the only version that we can modify.
2. Add the Lambda function as the fulfillment code hook and test it.
  - a. In the Editor, choose **AWS Lambda function as Fulfillment**, and select the Lambda function that you created in the preceding step (`OrderFlowersCodeHook`). Choose **OK** to give Amazon Lex permission to invoke the Lambda function.

You are configuring this Lambda function as a code hook to fulfill the intent. Amazon Lex invokes this function only after it has all the necessary slot data from the user to fulfill the intent.
  - b. Specify a **Goodbye message**.
  - c. Choose **Build**.
  - d. Test the bot using the previous conversation.

The last statement "Thanks, your order for roses....." is a response from the Lambda function that you configured as a code hook. In the preceding section, there was no Lambda function. Now you are using a Lambda function to actually fulfill the `OrderFlowers` intent.

3. Add the Lambda function as an initialization and validation code hook, and test.

The sample Lambda function code that you are using can both perform user input validation and fulfillment. The input event the Lambda function receives has a field (`invocationSource`) that the code uses to determine what portion of the code to execute. For more information, see [Lambda Function Input Event and Response Format \(p. 85\)](#).

- a. Select the `$LATEST` version of the `OrderFlowers` intent. That's is the only version that you can update.
- b. In the Editor, choose **Initialization and validation** in **Options**.
- c. Again, select the same Lambda function.

- d. Choose **Build**.
- e. Test the bot.

You are now ready to converse with Amazon Lex as follows. To test the validation portion, choose time 6 PM, and your Lambda function returns a response ("Our business hours are from 10 AM to 5 PM."), and prompts you again. After you provide all the valid slot data, the Lambda function fulfills the order.

The screenshot shows a chat interface titled 'Test App'. The conversation proceeds as follows:

- System: "What type of flowers would you like to order?"
- User: "roses"
- System: "What day do you want the roses to be picked up?"
- User: "tomorrow"
- System: "Deliver the roses at what time on 2016-11-26?"
- User: "6pm"
- System: "Our business hours are from ten a.m. to five p.m. Can you specify a time during this range?"
- User: "4pm"
- System: "Okay, your roses will be ready for pickup by 16:00 on 2016-11-26, and will cost 25 dollars. Does this sound okay?"
- User: "yes"
- System: "Thanks, your order for roses has been placed and will be ready for pickup by 16:00 on 2016-11-26"

At the bottom, there is a text input field with the placeholder text "Type or speak to your application ...".

### Next Step

[Step 5 \(Optional\): Review the Details of the Information Flow \(Console\) \(p. 37\)](#)

## Step 5 (Optional): Review the Details of the Information Flow (Console)

This section explains the flow of information between the client and Amazon Lex for each user input, including the integration of the Lambda function.

### Note

The section assumes that the client sends requests to Amazon Lex using the `PostText` runtime API and shows request and response details accordingly. For an example of the information flow between the client and Amazon Lex in which client uses the `PostContent` API, see [Step 2a \(Optional\): Review the Details of the Spoken Information Flow \(Console\) \(p. 27\)](#).

For more information about the `PostText` runtime API and additional details on the requests and responses shown in the following steps, see [PostText \(p. 293\)](#).

1. User: I would like to order some flowers.
  - a. The client (console) sends the following [PostText \(p. 293\)](#) request to Amazon Lex:

```
POST /bot/OrderFlowers/alias/$LATEST/user/ignw84y6seypre4xly5rimopuri2xwnd/text
```

```
"Content-Type": "application/json"
"Content-Encoding": "amz-1.0"

{
  "inputText": "I would like to order some flowers",
  "sessionAttributes": {}
}
```

Both the request URI and the body provide information to Amazon Lex:

- Request URI – Provides bot name (`OrderFlowers`), bot alias (`$LATEST`), and user name (a random string identifying the user). The trailing text indicates that it is a `PostText` API request (and not `PostContent`).
  - Request body – Includes the user input (`inputText`) and empty `sessionAttributes`. When the client makes the first request, there are no session attributes. The Lambda function initiates them later.
- b. From the `inputText`, Amazon Lex detects the intent (`OrderFlowers`). This intent is configured with a Lambda function as a code hook for user data initialization and validation. Therefore, Amazon Lex invokes that Lambda function by passing the following information as event data:

```
{
  "messageVersion": "1.0",
  "invocationSource": "DialogCodeHook",
  "userId": "ignw84y6seypre4xly5rimopuri2xwnd",
  "sessionAttributes": {},
  "bot": {
    "name": "OrderFlowers",
    "alias": null,
    "version": "$LATEST"
  },
  "outputDialogMode": "Text",
  "currentIntent": {
    "name": "OrderFlowers",
    "slots": {
      "PickupTime": null,
      "FlowerType": null,
      "PickupDate": null
    },
    "confirmationStatus": "None"
  }
}
```

For more information, see [Input Event Format \(p. 85\)](#).

In addition to the information that the client sent, Amazon Lex also includes the following additional data:

- `messageVersion` – Currently Amazon Lex supports only the 1.0 version.
  - `invocationSource` – Indicates the purpose of Lambda function invocation. In this case, it is to perform user data initialization and validation. At this time, Amazon Lex knows that the user has not provided all the slot data to fulfill the intent.
  - `currentIntent` information with all of the slot values set to null.
- c. At this time, all the slot values are null. There is nothing for the Lambda function to validate. The Lambda function returns the following response to Amazon Lex:

```
{
  "sessionAttributes": {},
  "dialogAction": {
    "type": "Delegate",
```

```
    "slots": {  
      "PickupTime": null,  
      "FlowerType": null,  
      "PickupDate": null  
    }  
  }  
}
```

For information about the response format, see [Response Format \(p. 88\)](#).

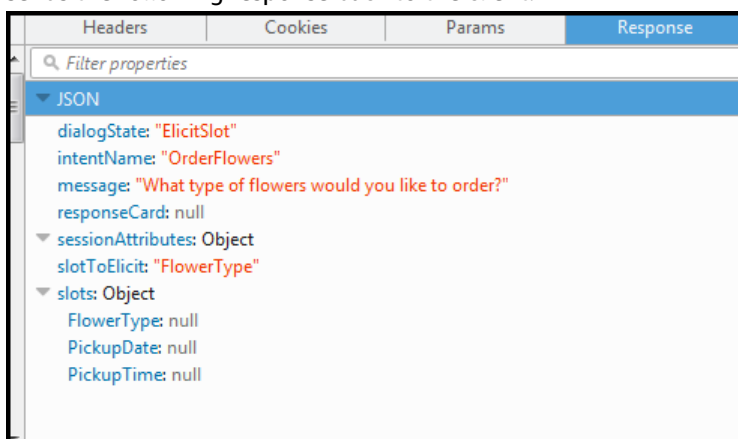
Note the following:

- `dialogAction.type` – By setting this value to `Delegate`, Lambda function delegates the responsibility of deciding the next course of action to Amazon Lex.

**Note**

If Lambda function detects anything in the user data validation, it instructs Amazon Lex what to do next, as shown in the next few steps.

- d. According to the `dialogAction.type`, Amazon Lex decides the next course of action. Because none of the slots are filled, it decides to elicit the value for the `FlowerType` slot. It selects one of the value elicitation prompts ("What type of flowers would you like to order?") for this slot and sends the following response back to the client:



The client displays the message in the response.

2. User: roses

- a. The client sends the following [PostText \(p. 293\)](#) request to Amazon Lex:

```
POST /bot/OrderFlowers/alias/$LATEST/user/ignw84y6seypre4xly5rimopuri2xwnd/text  
"Content-Type": "application/json"  
"Content-Encoding": "amz-1.0"  
  
{  
  "inputText": "roses",  
  "sessionAttributes": {}  
}
```

In the request body, the `inputText` provides user input. The `sessionAttributes` remains empty.

- b. Amazon Lex first interprets the `inputText` in the context of the current intent. The service remembers that it had asked the specific user for information about the `FlowerType` slot. It updates the slot value in the current intent and invokes the Lambda function with the following event data:

```
{
  "messageVersion": "1.0",
  "invocationSource": "DialogCodeHook",
  "userId": "ignw84y6seypre4xly5rimopuri2xwnd",
  "sessionAttributes": {},
  "bot": {
    "name": "OrderFlowers",
    "alias": null,
    "version": "$LATEST"
  },
  "outputDialogMode": "Text",
  "currentIntent": {
    "name": "OrderFlowers",
    "slots": {
      "PickupTime": null,
      "FlowerType": "roses",
      "PickupDate": null
    },
    "confirmationStatus": "None"
  }
}
```

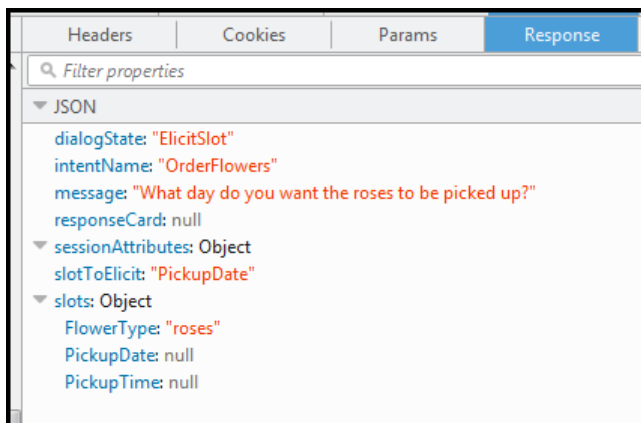
Note the following:

- `invocationSource` – continues to be `DialogCodeHook` (we are simply validating user data).
  - `currentIntent.slots` – Amazon Lex has updated the `FlowerType` slot to `roses`.
- c. According to the `invocationSource` value of `DialogCodeHook`, the Lambda function performs user data validation. It recognizes `roses` as a valid slot value (and sets `Price` as a session attribute) and returns the following response to Amazon Lex.

```
{
  "sessionAttributes": {
    "Price": 25
  },
  "dialogAction": {
    "type": "Delegate",
    "slots": {
      "PickupTime": null,
      "FlowerType": "roses",
      "PickupDate": null
    }
  }
}
```

Note the following:

- `sessionAttributes` – Lambda function has added `Price` (of the roses) as a session attribute.
  - `dialogAction.type` – is set to `Delegate`. The user data was valid so the Lambda function directs Amazon Lex to choose the next course of action.
- d. According to the `dialogAction.type`, Amazon Lex chooses the next course of action. Amazon Lex knows it needs more slot data so it picks the next unfilled slot (`PickupDate`) with the highest priority according to the intent configuration. Amazon Lex selects one of the value-elicitation prompt messages—"What day do you want the roses to be picked up?"—for this slot according to the intent configuration, and then sends the following response back to the client:



The client simply displays the message in the response – "What day do you want the roses to be picked up?."

3. User: tomorrow
  - a. The client sends the following [PostText](#) (p. 293) request to Amazon Lex:

```
POST /bot/OrderFlowers/alias/$LATEST/user/ignw84y6seypre4xly5rimopuri2xwnd/text
"Content-Type": "application/json"
"Content-Encoding": "amz-1.0"

{
  "inputText": "tomorrow",
  "sessionAttributes": {
    "Price": "25"
  }
}
```

In the request body, `inputText` provides user input and the client passes the session attributes back to the service.

- b. Amazon Lex remembers the context—that it was eliciting data for the `PickupDate` slot. In this context, it knows the `inputText` value is for the `PickupDate` slot. Amazon Lex then invokes the Lambda function by sending the following event:

```
{
  "messageVersion": "1.0",
  "invocationSource": "DialogCodeHook",
  "userId": "ignw84y6seypre4xly5rimopuri2xwnd",
  "sessionAttributes": {
    "Price": "25"
  },
  "bot": {
    "name": "OrderFlowersCustomWithRespCard",
    "alias": null,
    "version": "$LATEST"
  },
  "outputDialogMode": "Text",
  "currentIntent": {
    "name": "OrderFlowers",
    "slots": {
      "PickupTime": null,
      "FlowerType": "roses",
      "PickupDate": "2017-01-05"
    }
  },
  "confirmationStatus": "None"
}
```

```
}  
}
```

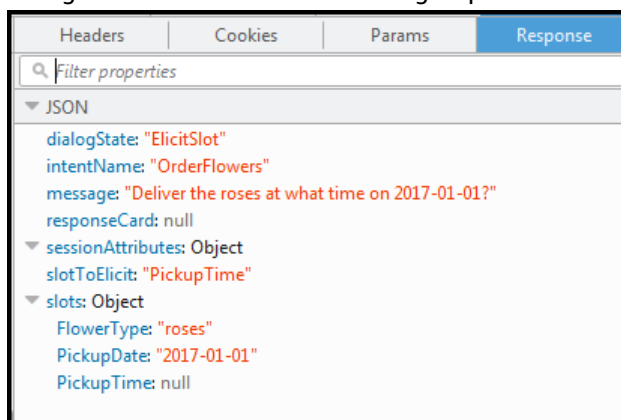
Amazon Lex has updated the `currentIntent.slots` by setting the `PickupDate` value. Also note that the service passes the `sessionAttributes` as it is to the Lambda function.

- c. As per `invocationSource` value of `DialogCodeHook`, the Lambda function performs user data validation. It recognizes `PickupDate` slot value is valid and returns the following response to Amazon Lex:

```
{  
  "sessionAttributes": {  
    "Price": 25  
  },  
  "dialogAction": {  
    "type": "Delegate",  
    "slots": {  
      "PickupTime": null,  
      "FlowerType": "roses",  
      "PickupDate": "2017-01-05"  
    }  
  }  
}
```

Note the following:

- `sessionAttributes` – No change.
  - `dialogAction.type` – is set to `Delegate`. The user data was valid, and the Lambda function directs Amazon Lex to choose the next course of action.
- d. According to the `dialogAction.type`, Amazon Lex chooses the next course of action. Amazon Lex knows it needs more slot data so it picks the next unfilled slot (`PickupTime`) with the highest priority according to the intent configuration. Amazon Lex selects one of the prompt messages ("Deliver the roses at what time on 2017-01-01?") for this slot according to the intent configuration and sends the following response back to the client:



The client displays the message in the response – "Deliver the roses at what time on 2017-01-01?"

4. User: 4 pm
- a. The client sends the following [PostText \(p. 293\)](#) request to Amazon Lex:

```
POST /bot/OrderFlowers/alias/$LATEST/user/ignw84y6seypre4xly5rimopuri2xwnd/text  
"Content-Type": "application/json"  
"Content-Encoding": "amz-1.0"
```

```
{
  "inputText": "4 pm",
  "sessionAttributes": {
    "Price": "25"
  }
}
```

In the request body, `inputText` provides user input. The client passes the `sessionAttributes` in the request.

- b. Amazon Lex understands context. It understands that it was eliciting data for the `PickupTime` slot. In this context, it knows that the `inputText` value is for the `PickupTime` slot. Amazon Lex then invokes the Lambda function by sending the following event:

```
{
  "messageVersion": "1.0",
  "invocationSource": "DialogCodeHook",
  "userId": "ignw84y6seypre4xly5rimopuri2xwnd",
  "sessionAttributes": {
    "Price": "25"
  },
  "bot": {
    "name": "OrderFlowersCustomWithRespCard",
    "alias": null,
    "version": "$LATEST"
  },
  "outputDialogMode": "Text",
  "currentIntent": {
    "name": "OrderFlowers",
    "slots": {
      "PickupTime": "16:00",
      "FlowerType": "roses",
      "PickupDate": "2017-01-05"
    },
    "confirmationStatus": "None"
  }
}
```

Amazon Lex has updated the `currentIntent.slots` by setting the `PickupTime` value.

- c. According to the `invocationSource` value of `DialogCodeHook`, the Lambda function performs user data validation. It recognizes `PickupDate` slot value is valid and returns the following response to Amazon Lex.

```
{
  "sessionAttributes": {
    "Price": 25
  },
  "dialogAction": {
    "type": "Delegate",
    "slots": {
      "PickupTime": "16:00",
      "FlowerType": "roses",
      "PickupDate": "2017-01-05"
    }
  }
}
```

Note the following:

- `sessionAttributes` – No change in session attribute.



- `dialogAction.type` – is set to `Delegate`. The user data was valid so the Lambda function directs Amazon Lex to choose the next course of action.
- d. At this time Amazon Lex knows it has all the slot data. This intent is configured with a confirmation prompt. Therefore, Amazon Lex sends the following response to the user asking for confirmation before fulfilling the intent:

Headers	Cookies	Params	Response	Timings
Filter properties				
JSON				
<pre>dialogState: "ConfirmIntent" intentName: "OrderFlowers" message: "Okay, your roses will be ready for pickup by 16:00 on 2017-01-05, and will cost 25 dollars. Does this sound okay?" responseCard: null sessionAttributes: Object   Price: "25"   slotToElicit: null slots: Object   FlowerType: "roses"   PickupDate: "2017-01-05"   PickupTime: "16:00"</pre>				

The client simply displays the message in the response and waits for the user response.

5. User: Yes
- a. The client sends the following [PostText](#) (p. 293) request to Amazon Lex:

```
POST /bot/OrderFlowers/alias/$LATEST/user/ignw84y6seypre4xly5rimopuri2xwnd/text
"Content-Type": "application/json"
"Content-Encoding": "amz-1.0"

{
  "inputText": "yes",
  "sessionAttributes": {
    "Price": "25"
  }
}
```

- b. Amazon Lex interprets the `inputText` in the context of confirming the current intent. Amazon Lex understands that the user wants to proceed with the order. This time Amazon Lex invokes the Lambda function to fulfill the intent by sending the following event, which sets the `invocationSource` to `FulfillmentCodeHook` in the event it sends to the Lambda function. Amazon Lex also sets the `confirmationStatus` to `Confirmed`.

```
{
  "messageVersion": "1.0",
  "invocationSource": "FulfillmentCodeHook",
  "userId": "ignw84y6seypre4xly5rimopuri2xwnd",
  "sessionAttributes": {
    "Price": "25"
  },
  "bot": {
    "name": "OrderFlowersCustomWithRespCard",
    "alias": null,
    "version": "$LATEST"
  },
  "outputDialogMode": "Text",
  "currentIntent": {
    "name": "OrderFlowers",
    "slots": {
```

```
        "PickupTime": "16:00",  
        "FlowerType": "roses",  
        "PickupDate": "2017-01-05"  
      },  
      "confirmationStatus": "Confirmed"  
    }  
  }  
}
```

Note the following:

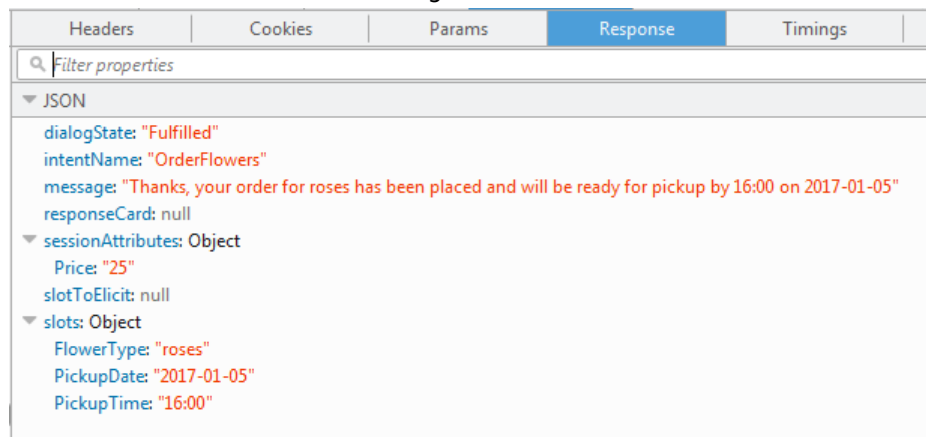
- `invocationSource` – This time Amazon Lex set this value to `FulfillmentCodeHook`, directing the Lambda function to fulfill the intent.
  - `confirmationStatus` – is set to `Confirmed`.
- c. This time, the Lambda function fulfills the `OrderFlowers` intent, and returns the following response:

```
{  
  "sessionAttributes": {  
    "Price": "25"  
  },  
  "dialogAction": {  
    "type": "Close",  
    "fulfillmentState": "Fulfilled",  
    "message": {  
      "contentType": "PlainText",  
      "content": "Thanks, your order for roses has been placed and will be  
ready for pickup by 16:00 on 2017-01-05"  
    }  
  }  
}
```

Note the following:

- Sets the `dialogAction.type` – The Lambda function sets this value to `Close`, directing Amazon Lex to not expect a user response.
  - `dialogAction.fulfillmentState` – is set to `Fulfilled` and includes an appropriate message to convey to the user.
- d. Amazon Lex reviews the `fulfillmentState` and sends the following response back to the client.

Amazon Lex then returns the following to the client:



Note that:

- `dialogState` – Amazon Lex sets this value to `fulfilled`.
- `message` – is the same message that the Lambda function provided.

The client displays the message.

6. Now test the bot again. To establish a new (user) context, choose the **Clear** link in the test window. Now provide invalid slot data for the `OrderFlowers` intent. This time the Lambda function performs the data validation, resets invalid slot data value to null, and asks Amazon Lex to prompt the user for valid data. For example, try the following:
  - Jasmine as the flower type (it is not one of the supported flower types).
  - Yesterday as the day when you want to pick up the flowers.
  - After placing your order, enter another flower type instead of replying "yes" to confirm the order. In response, the Lambda function updates the `Price` in the session attribute, keeping a running total of flower orders.

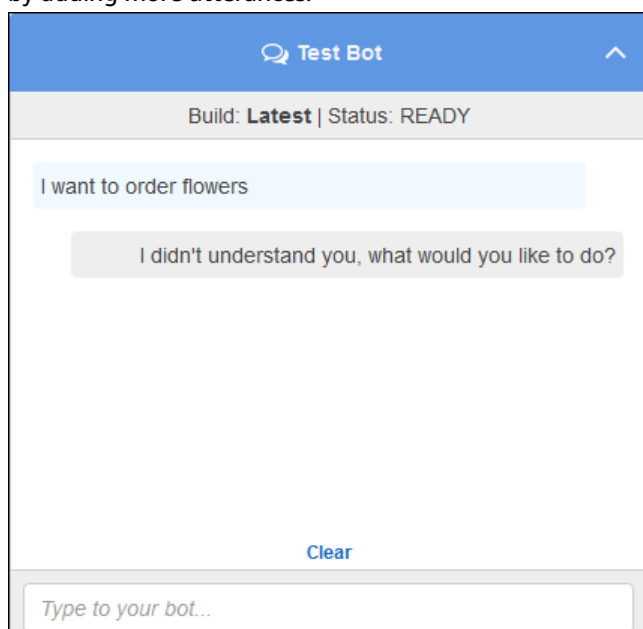
The Lambda function also performs the fulfillment activity.

### Next Step

[Step 6: Update the Intent Configuration to Add an Utterance \(Console\) \(p. 46\)](#)

## Step 6: Update the Intent Configuration to Add an Utterance (Console)

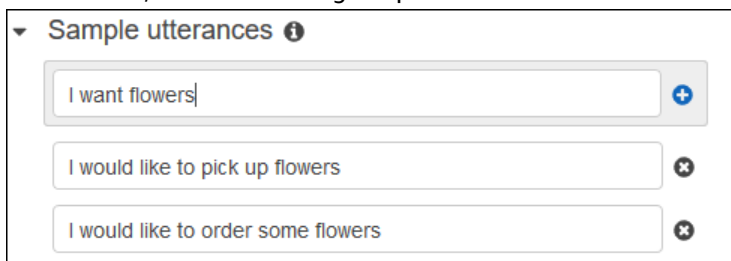
The `OrderFlowers` bot is configured with only two utterances. This provides limited information for Amazon Lex to build a machine learning model that recognizes and responds to the user's intent. Try typing "I want to order flowers" in the test window. Amazon Lex doesn't recognize the text, and responds with "I didn't understand you, what would you like to do?" You can improve the machine learning model by adding more utterances.



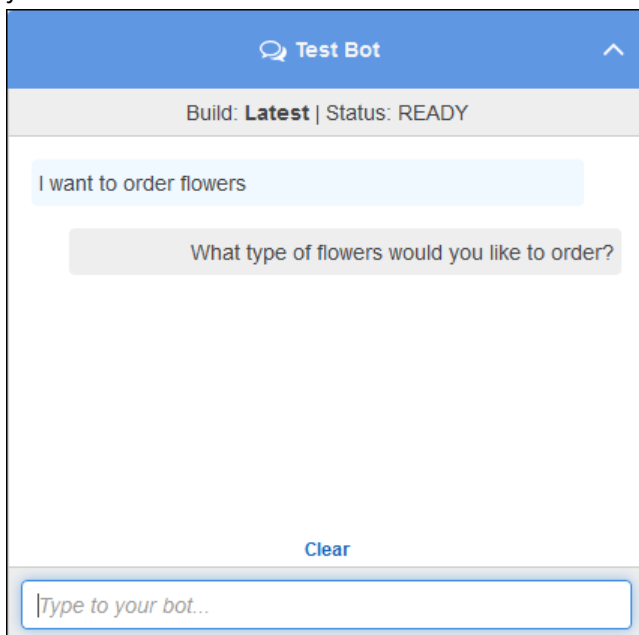
Each utterance that you add provides Amazon Lex with more information about how to respond to your users. You don't need to add an exact utterance, Amazon Lex generalizes from the samples that you provide to recognize both exact matches and similar input.

### To add an utterance (console)

1. Add the utterance "I want flowers" to the intent by typing it in the **Sample utterances** section of the intent editor, and then clicking the plus icon next to the new utterance.



2. Build your bot to pick up the change. Choose **Build**, and then choose **Build** again.
3. Test your bot to confirm that it recognized the new utterance. In the test window, type "I want to order flowers." Amazon Lex recognizes the phrase and responds with "What type of flowers would you like to order?".



### Next Step

[Step 7 \(Optional\): Clean Up \(Console\) \(p. 47\)](#)

## Step 7 (Optional): Clean Up (Console)

Now, delete the resources that you created and clean up your account.

You can delete only resources that are not in use. In general, you should delete resources in the following order:

- Delete bots to free up intent resources.

- Delete intents to free up slot type resources.
- Delete slot types last.

### To clean up your account (console)

1. Sign in to the AWS Management Console and open the Amazon Lex console at <https://console.aws.amazon.com/lex/>.
2. From the list of bots, choose the check box next to **OrderFlowers**.
3. To delete the bot, choose **Delete**, and then choose **Continue** in the confirmation dialog box.
4. In the left pane, choose **Intents**.
5. In the list of intents, choose **OrderFlowersIntent**.
6. To delete the intent, choose **Delete**, and then choose **Continue** in the confirmation dialog box.
7. In the left pane, choose **Slot types**.
8. In the list of slot types, choose **Flowers**.
9. To delete the slot type, choose **Delete**, and then choose **Continue** in the confirmation dialog box.

You have removed all of the resources that you created and cleaned up your account.

## Exercise 2: Create a Custom Amazon Lex Bot

Use the Amazon Lex console to create a custom bot.

In this exercise, you use the Amazon Lex console to create a custom bot that orders pizza (`OrderPizzaBot`). You configure the bot by adding a custom intent (`OrderPizza`), defining custom slot types, and defining the slots required to fulfill a pizza order (pizza crust, size, and so on). For more information about slot types and slots, see [Amazon Lex: How It Works \(p. 3\)](#).

### Topics

- [Step 1: Create a Lambda Function \(p. 48\)](#)
- [Step 2: Create a Bot \(p. 50\)](#)
- [Step 3: Build and Test the Bot \(p. 55\)](#)
- [Step 4 \(Optional\): Clean up \(p. 56\)](#)

## Step 1: Create a Lambda Function

First, create a Lambda function which fulfills a pizza order. You specify this function in your Amazon Lex bot, which you create in the next section.

### To create a Lambda function

1. Sign in to the AWS Management Console and open the AWS Lambda console at <https://console.aws.amazon.com/lambda/>.
2. Choose the US East (N. Virginia) Region (us-east-1).
3. Choose **Create a Lambda function**.
4. On the **Select blueprint** page, choose **Blank function**.

Because you are using custom code provided to you in this exercise to create a Lambda function, you choose the `Blank` function option.

5. On the **Configure triggers** page, choose **Next**.

6. On the **Configure function** page, do the following:
  - a. Type the name (`PizzaOrderProcessor`) and choose `Node.js.4.3` as the **runtime**.
  - b. In the **Lambda function code** section, choose **Edit code inline**, and then copy the following Node.js function code and paste it in the window.

```
'use strict';

// Close dialog with the customer, reporting fulfillmentState of Failed or
// Fulfilled ("Thanks, your pizza will arrive in 20 minutes")
function close(sessionAttributes, fulfillmentState, message) {
    return {
        sessionAttributes,
        dialogAction: {
            type: 'Close',
            fulfillmentState,
            message,
        },
    };
}

// ----- Events -----

function dispatch(intentRequest, callback) {
    console.log('request received for userId=${intentRequest.userId}, intentName=
    ${intentRequest.currentIntent.intentName}');
    const sessionAttributes = intentRequest.sessionAttributes;
    const slots = intentRequest.currentIntent.slots;
    const crust = slots.crust;
    const size = slots.size;
    const pizzaKind = slots.pizzaKind;

    callback(close(sessionAttributes, 'Fulfilled',
        {'contentType': 'PlainText', 'content': `Okay, I have ordered your ${size}
        ${pizzaKind} pizza on ${crust} crust`}));
}

// ----- Main handler -----

// Route the incoming request based on intent.
// The JSON body of the request is provided in the event slot.
exports.handler = (event, context, callback) => {
    try {
        dispatch(event,
            (response) => {
                callback(null, response);
            });
    } catch (err) {
        callback(err);
    }
};
```

- c. In the **Lambda function handler and role** section, choose **Choose a new role from template(s)** for the **Role**, and then type a role name.
  - d. Choose **Next**.
7. On the **Review** page, choose **Create function**.

## Test the Lambda Function Using Sample Event Data

In the console, test the Lambda function by using sample event data to manually invoke it.

### To test the Lambda function:

1. Sign in to the AWS Management Console and open the AWS Lambda console at <https://console.aws.amazon.com/lambda/>.
2. On the **Lambda function** page, choose **Actions**, and then choose **Configure test event**.
3. Choose the Lambda function.
4. On the **Input test event** page, copy the following Amazon Lex event into the window.

```
{
  "messageVersion": "1.0",
  "invocationSource": "FulfillmentCodeHook",
  "userId": "user-1",
  "sessionAttributes": {},
  "bot": {
    "name": "PizzaOrderingApp",
    "alias": "$LATEST",
    "version": "$LATEST"
  },
  "outputDialogMode": "Text",
  "currentIntent": {
    "name": "OrderPizza",
    "slots": {
      "size": "large",
      "pizzaKind": "meat",
      "crust": "thin"
    },
    "confirmationStatus": "None"
  }
}
```

5. Choose **Save and test**.

AWS Lambda executes your Lambda function and displays the following output in the **Execution result** pane.

```
{
  "sessionAttributes": {},
  "dialogAction": {
    "type": "Close",
    "fulfillmentState": "Fulfilled",
    "message": {
      "contentType": "PlainText",
      "content": "Okay, I have ordered your large meat pizza on thin crust"
    }
  }
}
```

## Next Step

[Step 2: Create a Bot \(p. 50\)](#)

## Step 2: Create a Bot

In this step, you create a bot to handle pizza orders.

### Topics

- [Create the Bot \(p. 51\)](#)
- [Create an Intent \(p. 51\)](#)
- [Create Slot Types \(p. 52\)](#)

- [Configure the Intent \(p. 53\)](#)
- [Configure the Bot \(p. 54\)](#)

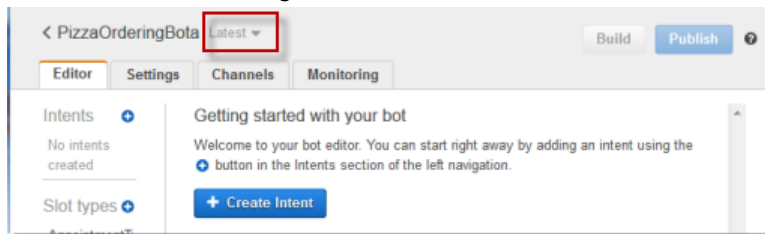
## Create the Bot

Create the `PizzaOrderingBot` bot with the minimum information needed. You add an intent, an action that the user wants to perform, for the bot later.

### To create the bot

1. Sign in to the AWS Management Console and open the Amazon Lex console at <https://console.aws.amazon.com/lex/>.
2. Create a bot.
  - a. If you are creating your first bot, choose **Get Started**. Otherwise, choose **Bots**, and then choose **Create**.
  - b. On the **Create your Lex bot** page, choose **Custom bot** and provide the following information:
    - **App name:** `PizzaOrderingBot`
    - **Output voice:** `Salli`
    - **Session timeout :** 5 minutes.
    - **Child-Directed:** Choose the appropriate response.
  - c. Choose **Create**.

The console sends Amazon Lex a request to create a new bot. Amazon Lex sets the bot version to `$LATEST`. After creating the bot, Amazon Lex shows the bot **Editor** tab:



- The bot version, **Latest**, appears next to the bot name in the console. New Amazon Lex resources have `$LATEST` as the version. For more information, see [Versioning and Aliases \(p. 81\)](#).
- Because you haven't created any intents or slots types, none are listed.
- **Build** and **Publish** are bot-level activities. After you configure the entire bot, you'll learn more about these activities.

## Next Step

[Create an Intent \(p. 51\)](#)

## Create an Intent

Now, create the `OrderPizza` intent, an action that the user wants to perform, with the minimum information needed. You add slot types for the intent and then configure the intent later.

### To create an intent

1. In the Amazon Lex console, choose the plus sign (+) next to **Intents**, and then choose **Create new intent**.



2. In the **Create intent** dialog box, type the name of the intent (`OrderPizza`), and then choose **Add**.

The console sends a request to Amazon Lex to create the `OrderPizza` intent. You configure the intent after you create slot types.

### Next Step

[Create Slot Types \(p. 52\)](#)

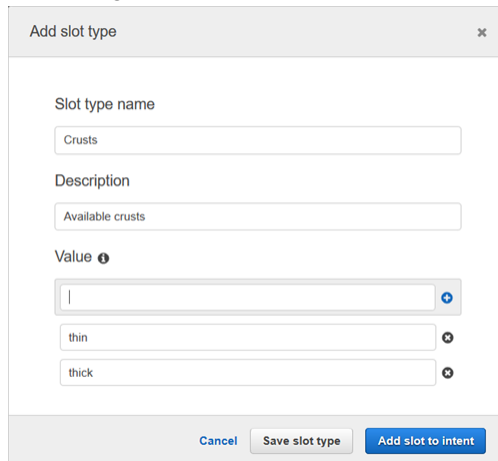
## Create Slot Types

Create the slot types, or parameter values, that the `OrderPizza` intent uses.

### To create slot types

1. In the left menu, choose the plus sign (+) next to **Slot types**.
2. In the **Add slot type** dialog box, add the following:
  - **Slot type name** – Crusts
  - **Description** – Available crusts
  - **Value** – Type `thin` and then choose the plus sign (+). Type `thick` and then choose the plus sign (+) again.

The dialog should look like this:



3. Choose **Add slot to intent**.
4. On the **Intent** page, choose **Required**. Change the name of the slot from `slotOne` to `crust`. Change the prompt to `What kind of crust would you like?`
5. Repeat [Step 1](#) through [Step 4](#) using the values in the following table:

Name	Description	Values	Slot name	Prompt
Sizes	Available sizes	small, medium, large	size	What size pizza?
PizzaKind	Available pizzas	veg, cheese	pizzaKind	Do you want a veg or cheese pizza?

## Next Step

[Configure the Intent \(p. 53\)](#)

## Configure the Intent

Configure the `OrderPizza` intent to fulfill a user's request to order a pizza.

### To configure an intent

- On the **OrderPizza** configuration page, configure the intent as follows:
  - **Sample utterances** – Type the following strings. The curly braces `{}` enclose slot names.
    - I want to order pizza please
    - I want to order a pizza
    - I want to order a `{pizzaKind}` pizza
    - I want to order a `{size}` `{pizzaKind}` pizza
    - I want a `{size}` `{crust}` crust `{pizzaKind}` pizza
    - Can I get a pizza please
    - Can I get a `{pizzaKind}` pizza
    - Can I get a `{size}` `{pizzaKind}` pizza
  - **Lambda initialization and validation** – Leave the default setting.
  - **Confirmation prompt** – Leave the default setting.
  - **Fulfillment** – Perform the following tasks:
    - Choose **AWS Lambda function**.
    - Choose `PizzaOrderProcessor`.
    - If the **Add permission to Lambda function** dialog box is shown, choose **OK** to give the `OrderPizza` intent permission to call the `PizzaOrderProcessor` Lambda function.
    - Leave **None** selected.

The intent should look like the following:

OrderPizza Latest ▾

▼ Sample utterances ⓘ

+

×

×

×

×

×

×

×

×

▶ Lambda initialization and validation ⓘ

▼ Slots ⓘ

Priority	Required	Name	Slot type	Prompt
		<input type="text" value="e.g. Location"/>	<input type="text" value="e.g. AMAZO..."/>	<input type="text" value="e.g. What city?"/> ⚙️ +
1. ▾	<input checked="" type="checkbox"/>	<input type="text" value="crust"/>	Crusts ▾	1 ▾
		<input type="text" value="What kind of crust would you"/> ⚙️ ×		
2. ^ ▾	<input checked="" type="checkbox"/>	<input type="text" value="size"/>	Sizes ▾	1 ▾
		<input type="text" value="What size pizza"/> ⚙️ ×		
3. ^	<input checked="" type="checkbox"/>	<input type="text" value="pizzaKind"/>	PizzaKind ▾	1 ▾
		<input type="text" value="Do you want a veg or chees"/> ⚙️ ×		

▶ Confirmation prompt ⓘ

▼ Fulfillment ⓘ

☒ AWS Lambda function ☐ Return parameters to client

▾

☐ Goodbye message ☐ Follow-up message ☒ None

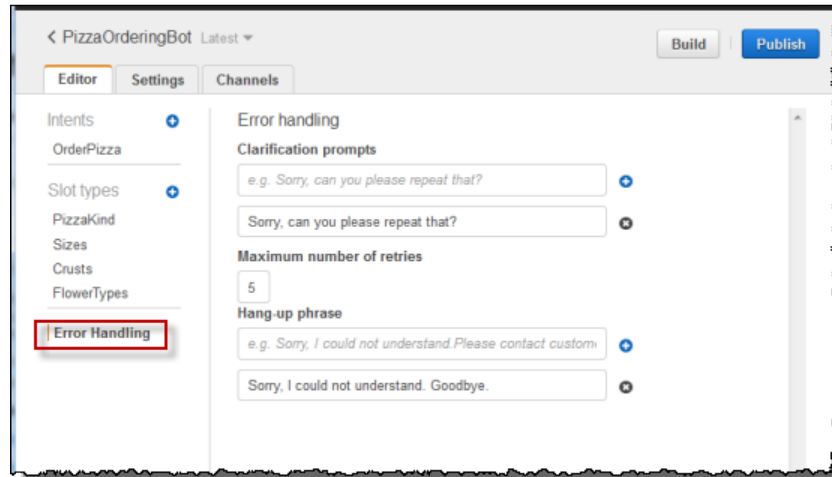
## Next Step

[Configure the Bot \(p. 54\)](#)

## Configure the Bot

Configure error handling for the `PizzaOrderingBot` bot.

1. Navigate to the `PizzaOrderingBot` bot. Choose **Editor**, and then choose **Error Handling**.



2. Use the **Editor** tab to configure bot error handling.

- Information you provide in **Clarification Prompts** maps to the bot's [clarificationPrompt](#) configuration.

When Amazon Lex can't determine the user intent, the service returns a response with this message

- Information that you provide in the **Hang-up** phrase maps to the bot's [abortStatement](#) configuration.

If the service can't determine the user's intent after a set number of consecutive requests, Amazon Lex returns a response with this message.

Leave the defaults.

## Next Step

[Step 3: Build and Test the Bot \(p. 55\)](#)

## Step 3: Build and Test the Bot

Make sure the bot works, by building and testing it.

### To build and test the bot

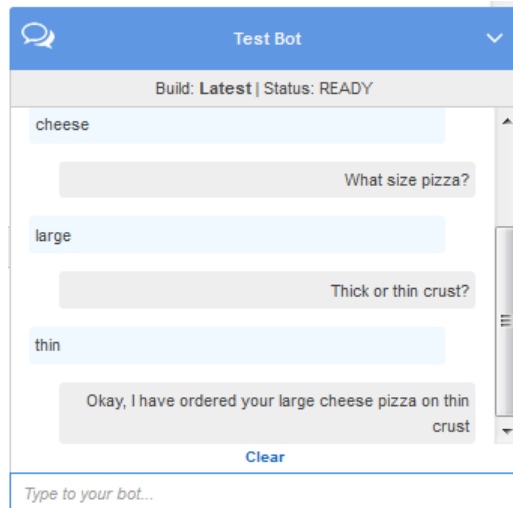
1. To build the `PizzaOrderingBot` bot, choose **Build**.

Amazon Lex builds a machine learning model for the bot. When you test the bot, the console uses the runtime API to send the user input back to Amazon Lex. Amazon Lex then uses the machine learning model to interpret the user input.

It can take some time to complete the build.

2. To test the bot, in the **Test Bot** window, start communicating with your Amazon Lex bot.

- For example, you might say or type:



- Use the sample utterances that you configured in the `OrderPizza` intent to test the bot. For example, the following is one of the sample utterances that you configured for the `PizzaOrder` intent:

```
I want a {size} {crust} crust {pizzaKind} pizza
```

To test it, type the following:

```
I want a large thin crust cheese pizza
```

When you type "I want to order a pizza," Amazon Lex detects the intent (`OrderPizza`). Then, Amazon Lex asks for slot information.

After you provide all of the slot information, Amazon Lex invokes the code hook (that is, the Lambda function that you configured for the intent).

The Lambda function returns a message ("Okay, I have ordered your ...") to Amazon Lex, which Amazon Lex returns to you..

## Next Step

[Step 4 \(Optional\): Clean up \(p. 56\)](#)

## Step 4 (Optional): Clean up

Delete the resources that you created and clean up your account to avoid incurring more charges for the resources you created.

You can delete only resources that are not in use. For example, you cannot delete a slot type that is referenced by an intent. You cannot delete an intent that is referenced by a bot.

Delete resources in the following order:

- Delete bots to free up intent resources.
- Delete intents to free up slot type resources.
- Delete slot types last.

### To clean up your account

1. Sign in to the AWS Management Console and open the Amazon Lex console at <https://console.aws.amazon.com/lex/>.
2. From the list of bots, choose **PizzaOrderingBot**.
3. To delete the bot, choose **Delete**, and then choose **Continue**.
4. In the left pane, choose **Intents**.
5. In the list of intents, choose **OrderPizza**.
6. To delete the intent, choose **Delete**, and then choose **Continue**.
7. In the left menu, choose **Slot types**.
8. In the list of slot types, choose **Crusts**.
9. To delete the slot type, choose **Delete**, and then choose **Continue**.
10. Repeat [Step 8](#) and [Step 9](#) for the `Sizes` and `PizzaKind` slot types.

You have removed all of the resources that you created and cleaned up your account.

### Next Steps

- [Publish a Version and Create an Alias](#)
- [Create an Amazon Lex bot with the AWS Command Line Interface](#)

## Exercise 3: Publish a Version and Create an Alias

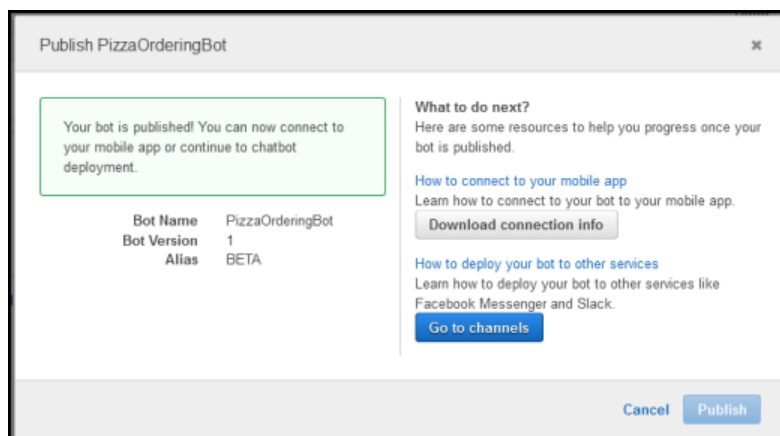
In Getting Started Exercises 1 and 2, you created a bot and tested it. In this exercise, you do the following:

- Publish a new version of the bot. Amazon Lex takes a snapshot copy of the `$LATEST` version to publish a new version.
- Create an alias that points to the new version.

For more information about versioning and aliases, see [Versioning and Aliases \(p. 81\)](#).

Do the following to publish a version of a bot you created for this exercise:

1. In the Amazon Lex console, choose one of the bots you created.  
  
Verify that the console shows the `$LATEST` as the bot version next to the bot name.
2. Choose **Publish**.
3. On the **Publish *botname*** wizard, specify an alias (BETA), and then choose **Publish**.
4. Verify that the Amazon Lex console shows the new version next to the bot name.



Now that you have a working bot with published version and an alias, you can deploy the bot (in your mobile application or integrate the bot with Facebook Messenger). For an example, see [Integrating an Amazon Lex Bot with Facebook Messenger \(p. 94\)](#).

## Step 4: Getting Started (AWS CLI)

In this step, you use the AWS CLI to create, test, and modify an Amazon Lex bot. To complete these exercises, you need to be familiar with using the CLI and have a text editor. For more information, see [Step 2: Set Up the AWS Command Line Interface \(p. 23\)](#)

- Exercise 1 — Create and test an Amazon Lex bot. The exercise provides all of the JSON objects that you need to create a custom slot type, an intent, and a bot. For more information, see [Amazon Lex: How It Works \(p. 3\)](#)
- Exercise 2 — Update the bot that you created in Exercise 1 to add an additional sample utterance. Amazon Lex uses sample utterances to build the machine learning model for your bot.
- Exercise 3 — Update the bot that you created in Exercise 1 to add a Lambda function to validate user input and to fulfill the intent.
- Exercise 4 — Publish a version of the slot type, intent, and bot resources that you created in Exercise 1. A version is a snapshot of a resource that can't be changed.
- Exercise 5 — Create an alias for the bot that you created in Exercise 1.
- Exercise 6 — Clean up your account by deleting the slot type, intent, and bot that you created in Exercise 1, and the alias that you created in Exercise 5.

### Topics

- [Exercise 1: Create an Amazon Lex Bot \(AWS CLI\) \(p. 58\)](#)
- [Exercise 2: Add a New Utterance \(AWS CLI\) \(p. 69\)](#)
- [Exercise 3: Add a Lambda Function \(AWS CLI\) \(p. 72\)](#)
- [Exercise 4: Publish a Version \(AWS CLI\) \(p. 75\)](#)
- [Exercise 5: Create an Alias \(AWS CLI\) \(p. 79\)](#)
- [Exercise 6: Clean Up \(AWS CLI\) \(p. 79\)](#)

## Exercise 1: Create an Amazon Lex Bot (AWS CLI)

In general, when you create bots, you:

1. Create slot types to define the information that your bot will be working with.
2. Create intents that define the user actions that your bot supports. Use the custom slot types that you created earlier to define the slots, or parameters, that your intent requires.
3. Create a bot that uses the intents that you defined.

In this exercise you create and test a new Amazon Lex bot using the CLI. Use the JSON structures that we provide to create the bot.

#### Topics

- [Step 1: Create a Service-Linked Role \(AWS CLI\) \(p. 59\)](#)
- [Step 2: Create a Custom Slot Type \(AWS CLI\) \(p. 60\)](#)
- [Step 3: Create an Intent \(AWS CLI\) \(p. 61\)](#)
- [Step 4: Create a Bot \(AWS CLI\) \(p. 64\)](#)
- [Step 5: Test a Bot \(AWS CLI\) \(p. 66\)](#)

## Step 1: Create a Service-Linked Role (AWS CLI)

Amazon Lex assumes AWS Identity and Access Management service-linked roles to call AWS services on behalf of your bots. The roles, which are in your account, are linked to Amazon Lex use cases and have predefined permissions. For more information, see [Service Permissions \(p. 8\)](#).

If you've already created an Amazon Lex bot using the console, the service-linked role was created automatically. Skip to [Step 2: Create a Custom Slot Type \(AWS CLI\) \(p. 60\)](#).

### To create a service-linked role (AWS CLI)

1. In the AWS CLI, type the following command:

```
aws iam create-service-linked-role --aws-service-name lex.amazonaws.com
```

2. Check the policy using the following command:

```
aws iam get-role --role-name AWSServiceRoleForLexBots
```

The response is:

```
{
  "Role": {
    "AssumeRolePolicyDocument": {
      "Version": "2012-10-17",
      "Statement": [
        {
          "Action": "sts:AssumeRole",
          "Effect": "Allow",
          "Principal": {
            "Service": "lex.amazonaws.com"
          }
        }
      ]
    },
    "RoleName": "AWSServiceRoleForLexBots",
    "Path": "/aws-service-role/lex.amazonaws.com/",
    "Arn": "arn:aws:iam::account-id:role/aws-service-role/lex.amazonaws.com/AWSServiceRoleForLexBots"
  }
}
```



## Next Step

[Step 2: Create a Custom Slot Type \(AWS CLI\) \(p. 60\)](#)

## Step 2: Create a Custom Slot Type (AWS CLI)

Create a custom slot type with enumeration values for the flowers that can be ordered. You use this type in the next step when you create the `OrderFlowers` intent. A *slot type* defines the possible values for a slot, or parameter, of the intent.

### To create a custom slot type (AWS CLI)

1. Create a text file named `FlowerTypes.json`. Copy the JSON code from [FlowerTypes.json \(p. 60\)](#) into the text file.
2. Call the [PutSlotType \(p. 282\)](#) operation using the AWS CLI to create the slot type:

```
aws lex-models put-slot-type --name FlowerTypes --cli-input-json file://
FlowerTypes.json
```

The response from the server is:

```
{
  "enumerationValues": [
    {
      "value": "tulips"
    },
    {
      "value": "lilies"
    },
    {
      "value": "roses"
    }
  ],
  "name": "FlowerTypes",
  "checksum": "checksum",
  "version": "$LATEST",
  "lastUpdatedDate": timestamp,
  "createdDate": timestamp,
  "description": "Types of flowers to pick up"
}
```

## Next Step

[Step 3: Create an Intent \(AWS CLI\) \(p. 61\)](#)

### FlowerTypes.json

The following code is the JSON data required to create the `FlowerTypes` custom slot type:

```
{
  "enumerationValues": [
    {
      "value": "tulips"
    },
    {
      "value": "lilies"
    },
    {
      "value": "roses"
    }
  ]
}
```

```
    }  
  ],  
  "name": "FlowerTypes",  
  "description": "Types of flowers to pick up"  
}
```

### Step 3: Create an Intent (AWS CLI)

Create an intent for the `OrderFlowersBot` bot and provide three slots, or parameters. The slots allow the bot to fulfill the intent:

- `FlowerType` is a custom slot type that specifies which types of flowers can be ordered.
- `AMAZON.DATE` and `AMAZON.TIME` are built-in slot types used for getting the date and time to deliver the flowers from the user.

#### To create the `orderFlowers` intent (AWS CLI)

1. Create a text file named `orderFlowers.json`. Copy the JSON code from [OrderFlowers.json \(p. 62\)](#) into the text file.
2. In the AWS CLI, call the [PutIntent \(p. 273\)](#) operation to create the intent:

```
aws lex-models put-intent --name OrderFlowers --cli-input-json file://OrderFlowers.json
```

The server responds with the following:

```
{  
  "confirmationPrompt": {  
    "maxAttempts": 2,  
    "messages": [  
      {  
        "content": "Okay, your {FlowerType} will be ready for pickup by  
{PickupTime} on {PickupDate}. Does this sound okay?",  
        "contentType": "PlainText"  
      }  
    ]  
  },  
  "name": "OrderFlowers",  
  "checksum": "checksum",  
  "version": "$LATEST",  
  "rejectionStatement": {  
    "messages": [  
      {  
        "content": "Okay, I will not place your order.",  
        "contentType": "PlainText"  
      }  
    ]  
  },  
  "createdDate": timestamp,  
  "lastUpdatedDate": timestamp,  
  "sampleUtterances": [  
    "I would like to pick up flowers",  
    "I would like to order some flowers"  
  ],  
  "slots": [  
    {  
      "slotType": "AMAZON.TIME",  
      "name": "PickupTime",  
      "slotConstraint": "Required",  
      "valueElicitationPrompt": {
```

```

        "maxAttempts": 2,
        "messages": [
            {
                "content": "Pick up the {FlowerType} at what time on
{PickupDate}?",
                "contentType": "PlainText"
            }
        ]
    },
    "priority": 3,
    "description": "The time to pick up the flowers"
},
{
    "slotType": "FlowerTypes",
    "name": "FlowerType",
    "slotConstraint": "Required",
    "valueElicitationPrompt": {
        "maxAttempts": 2,
        "messages": [
            {
                "content": "What type of flowers would you like to order?",
                "contentType": "PlainText"
            }
        ]
    },
    "priority": 1,
    "slotTypeVersion": "$LATEST",
    "sampleUtterances": [
        "I would like to order {FlowerType}"
    ],
    "description": "The type of flowers to pick up"
},
{
    "slotType": "AMAZON.DATE",
    "name": "PickupDate",
    "slotConstraint": "Required",
    "valueElicitationPrompt": {
        "maxAttempts": 2,
        "messages": [
            {
                "content": "What day do you want the {FlowerType} to be picked
up?",
                "contentType": "PlainText"
            }
        ]
    },
    "priority": 2,
    "description": "The date to pick up the flowers"
}
],
"fulfillmentActivity": {
    "type": "ReturnIntent"
},
"description": "Intent to order a bouquet of flowers for pick up"
}

```

## Next Step

[Step 4: Create a Bot \(AWS CLI\) \(p. 64\)](#)

## OrderFlowers.json

The following code is the JSON data required to create the `OrderFlowers` intent:

```
{
  "confirmationPrompt": {
    "maxAttempts": 2,
    "messages": [
      {
        "content": "Okay, your {FlowerType} will be ready for pickup by {PickupTime} on {PickupDate}. Does this sound okay?",
        "contentType": "PlainText"
      }
    ]
  },
  "name": "OrderFlowers",
  "rejectionStatement": {
    "messages": [
      {
        "content": "Okay, I will not place your order.",
        "contentType": "PlainText"
      }
    ]
  },
  "sampleUtterances": [
    "I would like to pick up flowers",
    "I would like to order some flowers"
  ],
  "slots": [
    {
      "slotType": "FlowerTypes",
      "name": "FlowerType",
      "slotConstraint": "Required",
      "valueElicitationPrompt": {
        "maxAttempts": 2,
        "messages": [
          {
            "content": "What type of flowers would you like to order?",
            "contentType": "PlainText"
          }
        ]
      },
      "priority": 1,
      "slotTypeVersion": "$LATEST",
      "sampleUtterances": [
        "I would like to order {FlowerType}"
      ],
      "description": "The type of flowers to pick up"
    },
    {
      "slotType": "AMAZON.DATE",
      "name": "PickupDate",
      "slotConstraint": "Required",
      "valueElicitationPrompt": {
        "maxAttempts": 2,
        "messages": [
          {
            "content": "What day do you want the {FlowerType} to be picked up?",
            "contentType": "PlainText"
          }
        ]
      },
      "priority": 2,
      "description": "The date to pick up the flowers"
    },
    {
      "slotType": "AMAZON.TIME",
      "name": "PickupTime",

```

```
        "slotConstraint": "Required",
        "valueElicitationPrompt": {
            "maxAttempts": 2,
            "messages": [
                {
                    "content": "Pick up the {FlowerType} at what time on
{PickupDate}?",
                    "contentType": "PlainText"
                }
            ]
        },
        "priority": 3,
        "description": "The time to pick up the flowers"
    }
],
"fulfillmentActivity": {
    "type": "ReturnIntent"
},
"description": "Intent to order a bouquet of flowers for pick up"
}
```

## Step 4: Create a Bot (AWS CLI)

The OrderFlowersBot bot has one intent, the OrderFlowers intent that you created in the previous step.

### Note

The following AWS CLI example is formatted for Unix, Linux, and macOS. For Windows, change `"\${LATEST}"` to `$LATEST`.

### To create the orderFlowersBot bot (AWS CLI)

1. Create a text file named `OrderFlowersBot.json`. Copy the JSON code from [OrderFlowersBot.json \(p. 65\)](#) into the text file.
2. In the AWS CLI, call the [PutBot \(p. 261\)](#) operation to create the bot:

```
aws lex-models put-bot --name OrderFlowersBot --cli-input-json file://
OrderFlowersBot.json
```

The response from the server follows. When you create or update bot, the `status` field is set to `BUILDING`. This indicates that the bot isn't ready to use. To determine when the bot is ready for use, use the [GetBot \(p. 209\)](#) operation in the next step.

```
{
  "status": "BUILDING",
  "intents": [
    {
      "intentVersion": "${LATEST}",
      "intentName": "OrderFlowers"
    }
  ],
  "name": "OrderFlowersBot",
  "locale": "en-US",
  "checksum": "checksum",
  "abortStatement": {
    "messages": [
      {
        "content": "Sorry, I'm not able to assist at this time",
        "contentType": "PlainText"
      }
    ]
  },
  "version": "${LATEST}",
```

```
"lastUpdatedDate": timestamp,
"createdDate": timestamp,
"clarificationPrompt": {
  "maxAttempts": 2,
  "messages": [
    {
      "content": "I didn't understand you, what would you like to do?",
      "contentType": "PlainText"
    }
  ]
},
"voiceId": "Salli",
"childDirected": false,
"idleSessionTTLInSeconds": 600,
"processBehavior": "BUILD",
"description": "Bot to order flowers on the behalf of a user"
}
```

3. To determine if your new bot is ready for use, run the following command. Repeat this command until the `status` field returns `READY`:

```
aws lex-models get-bot --name OrderFlowersBot --version-or-alias "$LATEST"
```

Look for the `status` field in the response:

```
{
  "status": "READY",
  ...
}
```

## Next Step

[Step 5: Test a Bot \(AWS CLI\) \(p. 66\)](#)

## OrderFlowersBot.json

The following code provides the JSON data required to build the `OrderFlowers` Amazon Lex bot:

```
{
  "intents": [
    {
      "intentVersion": "$LATEST",
      "intentName": "OrderFlowers"
    }
  ],
  "name": "OrderFlowersBot",
  "locale": "en-US",
  "abortStatement": {
    "messages": [
      {
        "content": "Sorry, I'm not able to assist at this time",
        "contentType": "PlainText"
      }
    ]
  },
  "clarificationPrompt": {
    "maxAttempts": 2,
    "messages": [
```

```
        {
          "content": "I didn't understand you, what would you like to do?",
          "contentType": "PlainText"
        }
      ]
    },
    "voiceId": "Salli",
    "childDirected": false,
    "idleSessionTTLInSeconds": 600,
    "description": "Bot to order flowers on the behalf of a user"
  }
}
```

## Step 5: Test a Bot (AWS CLI)

To test the bot, you can use either a text-based or a speech-based test.

### Topics

- [Test the Bot Using Text Input \(AWS CLI\) \(p. 66\)](#)
- [Test the Bot Using Speech Input \(AWS CLI\) \(p. 67\)](#)

## Test the Bot Using Text Input (AWS CLI)

To verify that the bot works correctly with text input, use the [PostText \(p. 293\)](#) operation.

### Note

The following AWS CLI example is formatted for Unix, Linux, and macOS. For Windows, change `"\${LATEST}"` to `$LATEST`.

### To use text to test the bot (AWS CLI)

1. In the AWS CLI, start a conversation with the `OrderFlowersBot` bot:

```
aws lex-runtime post-text --bot-name OrderFlowersBot --bot-alias "${LATEST}" --user-id
UserOne --input-text "i would like to order flowers"
```

Amazon Lex recognizes the user's intent and starts a conversation by returning the following response:

```
{
  "slotToElicit": "FlowerType",
  "slots": {
    "PickupDate": null,
    "PickupTime": null,
    "FlowerType": null
  },
  "dialogState": "ElicitSlot",
  "message": "What type of flowers would you like to order?",
  "intentName": "OrderFlowers"
}
```

2. Run the following commands to finish the conversation with the bot.

```
aws lex-runtime post-text --bot-name OrderFlowersBot --bot-alias "${LATEST}" --user-id
UserOne --input-text "roses"
```

```
aws lex-runtime post-text --bot-name OrderFlowersBot --bot-alias "${LATEST}" --user-id
UserOne --input-text "tuesday"
```

```
aws lex-runtime post-text --bot-name OrderFlowersBot --bot-alias "\$LATEST" --user-id UserOne --input-text "10:00 a.m."
```

```
aws lex-runtime post-text --bot-name OrderFlowersBot --bot-alias "\$LATEST" --user-id UserOne --input-text "yes"
```

After you confirm the order, Amazon Lex sends a fulfillment response to complete the conversation:

```
{
  "slots": {
    "PickupDate": "2017-05-16",
    "PickupTime": "10:00",
    "FlowerType": "roses"
  },
  "dialogState": "ReadyForFulfillment",
  "intentName": "OrderFlowers"
}
```

## Next Step

[Test the Bot Using Speech Input \(AWS CLI\) \(p. 67\)](#)

## Test the Bot Using Speech Input (AWS CLI)

To test the bot using audio files, use the [PostContent \(p. 286\)](#) operation. You generate the audio files using Amazon Polly text-to-speech operations.

### Note

The following AWS CLI example is formatted for Unix, Linux, and macOS. For Windows, change `"\$LATEST"` to `$LATEST`.

### To use a speech input to test the bot (AWS CLI)

1. In the AWS CLI, create an audio file using Amazon Polly:

```
aws polly synthesize-speech --output-format pcm --text "i would like to order flowers" --voice-id "Kendra" IntentSpeech.mpg
```

2. To send the audio file to Amazon Lex, run the following command. Amazon Lex saves the audio from the response in the specified output file.

```
aws lex-runtime post-content --bot-name OrderFlowersBot --bot-alias "\$LATEST" --user-id UserOne --content-type "audio/l16; rate=16000; channels=1" --input-stream IntentSpeech.mpg IntentOutputSpeech.mpg
```

Amazon Lex responds with a request for the first slot. It saves the audio response in the specified output file.

```
{
  "contentType": "audio/mpeg",
  "slotToElicit": "FlowerType",
  "dialogState": "ElicitSlot",
  "intentName": "OrderFlowers",
  "inputTranscript": "i would like to order some flowers",
  "slots": {
    "PickupDate": null,
```



```
        "PickupTime": null,  
        "FlowerType": null  
    },  
    "message": "What type of flowers would you like to order?"  
}
```

3. To set the appointment type, create the following audio file and send it to Amazon Lex:

```
aws polly synthesize-speech --output-format pcm --text "roses" --voice-id "Kendra"  
FlowerTypeSpeech.mpg
```

```
aws lex-runtime post-content --bot-name OrderFlowersBot --bot-alias "$LATEST" --  
user-id UserOne --content-type "audio/l16; rate=16000; channels=1" --input-stream  
FlowerTypeSpeech.mpg FlowerTypeOutputSpeech.mpg
```

4. To set the appointment date, create the following audio file and send it to Amazon Lex:

```
aws polly synthesize-speech --output-format pcm --text "tuesday" --voice-id "Kendra"  
DateSpeech.mpg
```

```
aws lex-runtime post-content --bot-name OrderFlowersBot --bot-alias "$LATEST" --  
user-id UserOne --content-type "audio/l16; rate=16000; channels=1" --input-stream  
DateSpeech.mpg DateOutputSpeech.mpg
```

5. To set the appointment time, create the following audio file and send it to Amazon Lex:

```
aws polly synthesize-speech --output-format pcm --text "10:00 a.m." --voice-id "Kendra"  
TimeSpeech.mpg
```

```
aws lex-runtime post-content --bot-name OrderFlowersBot --bot-alias "$LATEST" --  
user-id UserOne --content-type "audio/l16; rate=16000; channels=1" --input-stream  
TimeSpeech.mpg TimeOutputSpeech.mpg
```

6. To confirm the appointment, create the following audio file and send it to Amazon Lex:

```
aws polly synthesize-speech --output-format pcm --text "yes" --voice-id "Kendra"  
ConfirmSpeech.mpg
```

```
aws lex-runtime post-content --bot-name OrderFlowersBot --bot-alias "$LATEST" --  
user-id UserOne --content-type "audio/l16; rate=16000; channels=1" --input-stream  
ConfirmSpeech.mpg ConfirmOutputSpeech.mpg
```

After you confirm the appointment, Amazon Lex sends a response that confirms fulfillment of the intent:

```
{  
  "contentType": "text/plain;charset=utf-8",  
  "dialogState": "ReadyForFulfillment",  
  "intentName": "OrderFlowers",  
  "inputTranscript": "yes",  
  "slots": {  
    "PickupDate": "2017-05-16",  
    "PickupTime": "10:00",  
    "FlowerType": "roses"  
  }  
}
```

## Next Step

[Exercise 2: Add a New Utterance \(AWS CLI\)](#) (p. 69)

# Exercise 2: Add a New Utterance (AWS CLI)

To improve the machine learning model that Amazon Lex uses to recognize requests from your users, add another sample utterance to the bot.

Adding a new utterance is a four-step process.

1. Use the [GetIntent](#) (p. 238) operation to get an intent from Amazon Lex.
2. Update the intent.
3. Use the [PutIntent](#) (p. 273) operation to send the updated intent back to Amazon Lex.
4. Use the [GetBot](#) (p. 209) and [PutBot](#) (p. 261) operations to rebuild any bot that uses the intent.

The response from the `GetIntent` operation contains a field called `checksum` that identifies a specific revision of the intent. You must provide the checksum value when you use the [PutIntent](#) (p. 273) operation to update an intent. If you don't, you'll get the following error message:

```
An error occurred (PreconditionFailedException) when calling
the PutIntent operation: Intent intent name already exists.
If you are trying to update intent name you must specify the
checksum.
```

### Note

The following AWS CLI example is formatted for Unix, Linux, and macOS. For Windows, change `"\$LATEST"` to `$LATEST`.

## To update the `OrderFlowers` intent (AWS CLI)

1. In the AWS CLI, get the intent from Amazon Lex. Amazon Lex sends the output to a file called `OrderFlowers-V2.json`.

```
aws lex-models get-intent --name OrderFlowers --intent-version "\$LATEST" >
OrderFlowers-V2.json
```

2. Open `OrderFlowers-V2.json` in a text editor.

1. Find and delete the `createdDate`, `lastUpdatedDate`, and `version` fields.
2. Add the following to the `sampleUtterances` field:

```
I want to order flowers
```

3. Save the file.
3. Send the updated intent to Amazon Lex with the following command:

```
aws lex-models put-intent --name OrderFlowers --cli-input-json file://OrderFlowers-
V2.json
```

Amazon Lex sends the following response:

```
{
  "confirmationPrompt": {
```

```
        "maxAttempts": 2,
        "messages": [
            {
                "content": "Okay, your {FlowerType} will be ready for pickup by {PickupTime} on {PickupDate}. Does this sound okay?",
                "contentType": "PlainText"
            }
        ]
    },
    "name": "OrderFlowers",
    "checksum": "checksum",
    "version": "$LATEST",
    "rejectionStatement": {
        "messages": [
            {
                "content": "Okay, I will not place your order.",
                "contentType": "PlainText"
            }
        ]
    },
    "createdDate": timestamp,
    "lastUpdatedDate": timestamp,
    "sampleUtterances": [
        "I would like to pick up flowers",
        "I would like to order some flowers",
        "I want to order flowers"
    ],
    "slots": [
        {
            "slotType": "AMAZON.TIME",
            "name": "PickupTime",
            "slotConstraint": "Required",
            "valueElicitationPrompt": {
                "maxAttempts": 2,
                "messages": [
                    {
                        "content": "Pick up the {FlowerType} at what time on {PickupDate}?",
                        "contentType": "PlainText"
                    }
                ]
            },
            "priority": 3,
            "description": "The time to pick up the flowers"
        },
        {
            "slotType": "FlowerTypes",
            "name": "FlowerType",
            "slotConstraint": "Required",
            "valueElicitationPrompt": {
                "maxAttempts": 2,
                "messages": [
                    {
                        "content": "What type of flowers would you like to order?",
                        "contentType": "PlainText"
                    }
                ]
            },
            "priority": 1,
            "slotTypeVersion": "$LATEST",
            "sampleUtterances": [
                "I would like to order {FlowerType}"
            ],
            "description": "The type of flowers to pick up"
        }
    ],
    "description": "Order flowers for pickup"
```

```
        "slotType": "AMAZON.DATE",
        "name": "PickupDate",
        "slotConstraint": "Required",
        "valueElicitationPrompt": {
            "maxAttempts": 2,
            "messages": [
                {
                    "content": "What day do you want the {FlowerType} to be picked
up?",
                    "contentType": "PlainText"
                }
            ]
        },
        "priority": 2,
        "description": "The date to pick up the flowers"
    }
],
"fulfillmentActivity": {
    "type": "ReturnIntent"
},
"description": "Intent to order a bouquet of flowers for pick up"
}
```

Now that you have updated the intent, rebuild any bot that uses it.

#### To rebuild the `OrderFlowersBot` bot (AWS CLI)

1. In the AWS CLI, get the definition of the `OrderFlowersBot` bot and save it to a file with the following command:

```
aws lex-models get-bot --name OrderFlowersBot --version-or-alias "$LATEST" >
OrderFlowersBot-V2.json
```

2. In a text editor, open `OrderFlowersBot-V2.json`. Remove the `createdDate`, `lastUpdatedDate`, `status` and `version` fields.
3. In a text editor, add the following line to the bot definition:

```
"processBehavior": "BUILD",
```

4. In the AWS CLI, build a new revision of the bot by running the following command to :

```
aws lex-models put-bot --name OrderFlowersBot --cli-input-json file://OrderFlowersBot-
V2.json
```

The response from the server is:

```
{
  "status": "BUILDING",
  "intents": [
    {
      "intentVersion": "$LATEST",
      "intentName": "OrderFlowers"
    }
  ],
  "name": "OrderFlowersBot",
  "locale": "en-US",
  "checksum": "checksum",
  "abortStatement": {
    "messages": [
      {

```

```
        "content": "Sorry, I'm not able to assist at this time",  
        "contentType": "PlainText"  
    }  
]  
},  
"version": "$LATEST",  
"lastUpdatedDate": timestamp,  
"createdDate": timestamp  
"clarificationPrompt": {  
    "maxAttempts": 2,  
    "messages": [  
        {  
            "content": "I didn't understand you, what would you like to do?",  
            "contentType": "PlainText"  
        }  
    ]  
},  
"voiceId": "Salli",  
"childDirected": false,  
"idleSessionTTLInSeconds": 600,  
"description": "Bot to order flowers on the behalf of a user"  
}
```

## Next Step

[Exercise 3: Add a Lambda Function \(AWS CLI\)](#) (p. 72)

## Exercise 3: Add a Lambda Function (AWS CLI)

Add a Lambda function that validates user input and fulfills the user's intent to the bot.

Adding a Lambda expression is a five-step process.

1. Use the Lambda [AddPermission](#) function to enable the `OrderFlowers` intent to call the Lambda [Invoke](#) operation.
2. Use the [GetIntent](#) (p. 238) operation to get the intent from Amazon Lex.
3. Update the intent to add the Lambda function.
4. Use the [PutIntent](#) (p. 273) operation to send the updated intent back to Amazon Lex.
5. Use the [GetBot](#) (p. 209) and [GetBot](#) (p. 209) operations to rebuild any bot that uses the intent.

If you add a Lambda function to an intent before you add the `InvokeFunction` permission, you get the following error message:

```
An error occurred (BadRequestException) when calling the  
PutIntent operation: Lex is unable to access the Lambda  
function Lambda function ARN in the context of intent  
intent ARN. Please check the resource-based policy on  
the function.
```

The response from the `GetIntent` operation contains a field called `checksum` that identifies a specific revision of the intent. When you use the [PutIntent](#) (p. 273) operation to update an intent, you must provide the checksum value. If you don't, you get the following error message:

```
An error occurred (PreconditionFailedException) when calling
```

the PutIntent operation: Intent `intent name` already exists.  
If you are trying to update `intent name` you must specify the checksum.

This exercise uses the Lambda function from [Example Bot: ScheduleAppointment \(p. 103\)](#). For instructions to create the Lambda function, see [Step 2: Create a Lambda Function \(p. 108\)](#).

**Note**

The following AWS CLI example is formatted for Unix, Linux, and macOS. For Windows, change `"\${LATEST}"` to `$LATEST`.

**To add a Lambda function to an intent**

1. In the AWS CLI, add the `InvokeFunction` permission for the `OrderFlowers` intent:

```
aws lambda add-permission --function-name OrderFlowersCodeHook --statement-id  
LexGettingStarted-OrderFlowersBot --action lambda:InvokeFunction --principal  
lex.amazonaws.com --source-arn "arn:aws:lex:region:account ID:intent:OrderFlowers:*
```

Lambda sends the following response:

```
{  
  "Statement": "{\n\"Sid\": \"LexGettingStarted-OrderFlowersBot\",  
  \"Resource\": \"arn:aws:lambda:region:account ID:function:OrderFlowersCodeHook\",  
  \"Effect\": \"Allow\",  
  \"Principal\": {\n\"Service\": \"lex.amazonaws.com\"},  
  \"Action\": [\"lambda:InvokeFunction\"],  
  \"Condition\": {\n\"ArnLike\":  
    {\n\"AWS:SourceArn\":  
      \"arn:aws:lex:region:account ID:intent:OrderFlowers:*\"}}}"
```

2. Get the intent from Amazon Lex. Amazon Lex sends the output to a file called `orderFlowers-V3.json`.

```
aws lex-models get-intent --name OrderFlowers --intent-version "\${LATEST}" >  
OrderFlowers-V3.json
```

3. In a text editor, open the `OrderFlowers-V3.json`.

1. Find and delete the `createdDate`, `lastUpdatedDate`, and `version` fields.
2. Update the `fulfillmentActivity` field:

```
"fulfillmentActivity": {  
  "type": "CodeHook",  
  "codeHook": {  
    "uri": "arn:aws:lambda:region:account ID:function:OrderFlowersCodeHook",  
    "messageVersion": "1.0"  
  },  
}
```

3. Save the file.
4. In the AWS CLI, send the updated intent to Amazon Lex:

```
aws lex-models put-intent --name OrderFlowers --cli-input-json file://OrderFlowers-  
V3.json
```

Now that you have updated the intent, rebuild the bot.

### To rebuild the `OrderFlowersBot` bot

1. In the AWS CLI, get the definition of the `OrderFlowersBot` bot and save it to a file:

```
aws lex-models get-bot --name OrderFlowersBot --version-or-alias "$LATEST" >
OrderFlowersBot-V3.json
```

2. In a text editor, open `OrderFlowersBot-V3.json`. Remove the `createdDate`, `lastUpdatedDate`, `status`, and `version` fields.
3. In the text editor, add the following line to the definition of the bot:

```
"processBehavior": "BUILD",
```

4. In the AWS CLI, build a new revision of the bot:

```
aws lex-models put-bot --name OrderFlowersBot --cli-input-json file://OrderFlowersBot-
V3.json
```

The response from the server is:

```
{
  "status": "READY",
  "intents": [
    {
      "intentVersion": "$LATEST",
      "intentName": "OrderFlowers"
    }
  ],
  "name": "OrderFlowersBot",
  "locale": "en-US",
  "checksum": "checksum",
  "abortStatement": {
    "messages": [
      {
        "content": "Sorry, I'm not able to assist at this time",
        "contentType": "PlainText"
      }
    ]
  },
  "version": "$LATEST",
  "lastUpdatedDate": timestamp,
  "createdDate": timestamp,
  "clarificationPrompt": {
    "maxAttempts": 2,
    "messages": [
      {
        "content": "I didn't understand you, what would you like to do?",
        "contentType": "PlainText"
      }
    ]
  },
  "voiceId": "Salli",
  "childDirected": false,
  "idleSessionTTLInSeconds": 600,
  "description": "Bot to order flowers on the behalf of a user"
}
```

## Next Step

[Exercise 4: Publish a Version \(AWS CLI\) \(p. 75\)](#)

## Exercise 4: Publish a Version (AWS CLI)

Now, create a version of the bot that you created in Exercise 1. A *version* is a snapshot of the bot. After you create a version, you can't change it. The only version of a bot that you can update is the `$LATEST` version. For more information about versions, see [Versioning and Aliases \(p. 81\)](#).

Before you can publish a version of a bot, you must publish the intents that it uses. Likewise, you must publish the slot types that those intents refer to. In general, to publish a version of a bot, you do the following:

1. Publish a version of a slot type with the [CreateSlotTypeVersion \(p. 187\)](#) operation.
2. Publish a version of an intent with the [CreateIntentVersion \(p. 181\)](#) operation.
3. Publish a version of a bot with the [CreateBotVersion \(p. 176\)](#) operation.

### Topics

- [Step 1: Publish the Slot Type \(AWS CLI\) \(p. 75\)](#)
- [Step 2: Publish the Intent \(AWS CLI\) \(p. 76\)](#)
- [Step 3: Publish the Bot \(AWS CLI\) \(p. 77\)](#)

## Step 1: Publish the Slot Type (AWS CLI)

Before you can publish a version of any intents that use a slot type, you must publish a version of that slot type. In this case, you publish the `FlowerTypes` slot type.

### Note

The following AWS CLI example is formatted for Unix, Linux, and macOS. For Windows, change `"\${LATEST}"` to `$LATEST`.

### To publish a slot type (AWS CLI)

1. In the AWS CLI, get the latest version of the slot type:

```
aws lex-models get-slot-type --name FlowerTypes --slot-type-version "${LATEST}"
```

The response from Amazon Lex follows. Record the checksum for the current revision of the `$LATEST` version.

```
{
  "enumerationValues": [
    {
      "value": "tulips"
    },
    {
      "value": "lilies"
    },
    {
      "value": "roses"
    }
  ],
}
```



```
{
  "name": "FlowerTypes",
  "checksum": "checksum",
  "version": "$LATEST",
  "lastUpdatedDate": timestamp,
  "createdDate": timestamp,
  "description": "Types of flowers to pick up"
}
```

2. Publish a version of the slot type. Use the checksum that you recorded in the previous step.

```
aws lex-models create-slot-type-version --name FlowerTypes --checksum "checksum"
```

The response from Amazon Lex follows. Record the version number for the next step.

```
{
  "version": "1",
  "enumerationValues": [
    {
      "value": "tulips"
    },
    {
      "value": "lilies"
    },
    {
      "value": "roses"
    }
  ],
  "name": "FlowerTypes",
  "createdDate": timestamp,
  "lastUpdatedDate": timestamp,
  "description": "Types of flowers to pick up"
}
```

## Next Step

[Step 2: Publish the Intent \(AWS CLI\) \(p. 76\)](#)

## Step 2: Publish the Intent (AWS CLI)

Before you can publish an intent, you have to publish all of the slot types referred to by the intent. The slot types must be numbered versions, not the `$LATEST` version.

First, update the `OrderFlowers` intent to use the version of the `FlowerTypes` slot type that you published in the previous step. Then publish a new version of the `OrderFlowers` intent.

### Note

The following AWS CLI example is formatted for Unix, Linux, and macOS. For Windows, change `"\${LATEST}"` to `$LATEST`.

### To publish a version of an intent (AWS CLI)

1. In the AWS CLI, get the `$LATEST` version of the `OrderFlowers` intent and save it to a file:

```
aws lex-models get-intent --name OrderFlowers --intent-version "${LATEST}" >
OrderFlowers_V4.json
```

2. In a text editor, open the `OrderFlowers_V4.json` file. Delete the `createdDate`, `lastUpdatedDate`, and `version` fields. Find the `FlowerTypes` slot type and change the version to the version number that

you recorded in the previous step. The following fragment of the `OrderFlowers_V4.json` file shows the location of the change:

```
{
  "slotType": "FlowerTypes",
  "name": "FlowerType",
  "slotConstraint": "Required",
  "valueElicitationPrompt": {
    "maxAttempts": 2,
    "messages": [
      {
        "content": "What type of flowers?",
        "contentType": "PlainText"
      }
    ]
  },
  "priority": 1,
  "slotTypeVersion": "version",
  "sampleUtterances": []
},
```

3. In the AWS CLI, save the revision of the intent:

```
aws lex-models put-intent --name OrderFlowers --cli-input-json file://
OrderFlowers_V4.json
```

4. Get the checksum of the latest revision of the intent:

```
aws lex-models get-intent --name OrderFlowers --intent-version "$LATEST" >
OrderFlowers_V4a.json
```

The following fragment of the response shows the checksum of the intent. Record this for the next step.

```
"name": "OrderFlowers",
"checksum": "checksum",
"version": "$LATEST",
```

5. Publish a new version of the intent:

```
aws lex-models create-intent-version --name OrderFlowers --checksum "checksum"
```

The following fragment of the response shows the new version of the intent. Record the version number for the next step.

```
"name": "OrderFlowers",
"checksum": "checksum",
"version": "version",
```

## Next Step

[Step 3: Publish the Bot \(AWS CLI\) \(p. 77\)](#)

## Step 3: Publish the Bot (AWS CLI)

After you have published all of the slot types and intents that are used by your bot, you can publish the bot.

Update the `OrderFlowersBot` bot to use the `OrderFlowers` intent that you updated in the previous step. Then, publish a new version of the `OrderFlowersBot` bot.

**Note**

The following AWS CLI example is formatted for Unix, Linux, and macOS. For Windows, change `"\${LATEST}"` to `$LATEST`.

**To publish a version of a bot (AWS CLI)**

1. In the AWS CLI, get the `$LATEST` version of the `OrderFlowersBot` bot and save it to a file:

```
aws lex-models get-bot --name OrderFlowersBot --version-or-alias "${LATEST}" >
OrderFlowersBot_V4.json
```

2. In a text editor, open the `orderFlowersBot_V4.json` file. Delete the `createdDate`, `lastUpdatedDate`, `status` and `version` fields. Find the `OrderFlowers` intent and change the version to the version number that you recorded in the previous step. The following fragment of `orderFlowersBot_V4.json` shows the location of the change.

```
"intents": [
  {
    "intentVersion": "version",
    "intentName": "OrderFlowers"
  }
]
```

3. In the AWS CLI, save the new revision of the bot:

```
aws lex-models put-bot --name OrderFlowersBot --cli-input-json file://
OrderFlowersBot_V4.json
```

4. Get the checksum of the latest revision of the bot:

```
aws lex-models get-bot --name OrderFlowersBot > OrderFlowersBot_V4a.json
```

The following fragment of the response shows the checksum of the bot. Record this for the next step.

```
"name": "OrderFlowersBot",
"locale": "en-US",
"checksum": "checksum",
```

5. Publish a new version of the bot:

```
aws lex-models create-bot-version --name OrderFlowersBot --checksum "checksum"
```

The following fragment of the response shows the new version of the bot.

```
"checksum": "checksum",
"abortStatement": {
  ...
},
"version": "1",
"lastUpdatedDate": timestamp,
```

## Next Step

[Exercise 5: Create an Alias \(AWS CLI\) \(p. 79\)](#)

## Exercise 5: Create an Alias (AWS CLI)

An alias is a pointer to a specific version of a bot. With an alias you can easily update the version that your client applications are using. For more information, see [Versioning and Aliases \(p. 81\)](#)

### To create an alias (AWS CLI)

1. In the AWS CLI, get the version of the `OrderFlowersBot` bot that you created in [Exercise 4: Publish a Version \(AWS CLI\) \(p. 75\)](#).

```
aws lex-models get-bot --name OrderFlowersBot --version-or-alias version >  
OrderFlowersBot_v5.json
```

2. In a text editor, open `OrderFlowersBot_v5.json`. Find and record the version number.
3. In the AWS CLI, create the bot alias:

```
aws lex-models put-bot-alias --name PROD --bot-name OrderFlowersBot --bot-  
version version
```

The following is the response from the server:

```
{  
  "name": "PROD",  
  "createdDate": timestamp,  
  "checksum": "checksum",  
  "lastUpdatedDate": timestamp,  
  "botName": "OrderFlowersBot",  
  "botVersion": "1"  
}}
```

## Next Step

[Exercise 6: Clean Up \(AWS CLI\) \(p. 79\)](#)

## Exercise 6: Clean Up (AWS CLI)

Delete the resources that you created and clean up your account.

You can delete only resources that are not in use. In general, you should delete resources in the following order.

1. Delete aliases to free up bot resources.
2. Delete bots to free up intent resources.
3. Delete intents to free up slot type resources.
4. Delete slot types.

### To clean up your account (AWS CLI)

1. In the AWS CLI command line, delete the alias:

```
aws lex-models delete-bot-alias --name PROD --bot-name OrderFlowersBot
```

2. In the AWS CLI command line, delete the bot:

```
aws lex-models delete-bot --name OrderFlowersBot
```

3. In the AWS CLI command line, delete the intent:

```
aws lex-models delete-intent --name OrderFlowers
```

4. From the AWS CLI command line, delete the slot type:

```
aws lex-models delete-slot-type --name FlowerTypes
```

You have removed all of the resources that you created and cleaned up your account.

Amazon Lex supports publishing versions of bots, intents, and slot types so that you can control the implementation that your client applications use. A *version* is a numbered snapshot of your work that you can publish for use in different parts of your workflow, such as development, beta deployment, and production.

Amazon Lex bots also support aliases. An *alias* is a pointer to a specific version of a bot. With an alias, you can easily update the version that your client applications are using. For example, you can point an alias to version 1 of your bot. When you are ready to update the bot, you publish version 2 and change the alias to point to the new version. Because your applications use the alias instead of a specific version, all of your clients get the new functionality without needing to be updated.

- Versioning (p. 81)
- Aliases (p. 83)

When you version an Amazon Lex resource you create a snapshot of the resource so that you can use the resource as it existed when the version was made. Once you've created a version it will stay the same while you continue to work on your application.

When you create an Amazon Lex bot, intent, or slot type there is only one version, the `$LATEST` version.



Only the `$LATEST` version of a resource can use the `$LATEST` version of another resource. For example, the `$LATEST` version of a bot can use the `$LATEST` version of an intent, and the `$LATEST` version of an intent can use the `$LATEST` version of a slot type.

When you publish a resource, Amazon Lex makes a copy of the `$LATEST` version and saves it as a numbered version. The published version can't be changed.



Before you can publish a bot, you must point it to a numbered version of any intent that it uses. If you try to publish a new version of a bot that uses the `$LATEST` version of an intent, Amazon Lex returns an HTTP 400 Bad Request exception. Before you can publish a numbered version of the intent, you must point the intent to a numbered version of any slot type that it uses. Otherwise you will get an HTTP 400 Bad Request exception.



Amazon Lex publishes a new version only if the last published version is different from the `$LATEST` version. If you try to publish the `$LATEST` version without modifying it, Amazon Lex doesn't create or publish a new version.

You can update only the `$LATEST` version of an Amazon Lex bot, intent, or slot type. Published versions can't be changed. You can publish a new version any time after you update a resource.

in the console or with the [CreateBotVersion](#) (p. 176), the [CreateIntentVersion](#) (p. 181) or the [CreateSlotTypeVersion](#) (p. 187) operations.

## Deleting an Amazon Lex Resource or Version

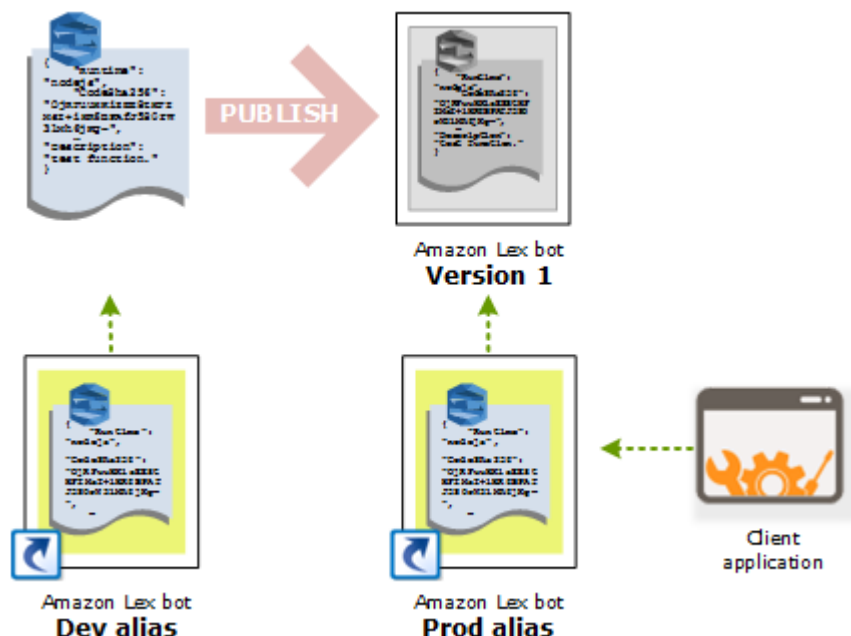
Amazon Lex supports deleting a resource or version using the console or one of the API operations:

- [DeleteBot](#) (p. 191)
- [DeleteBotVersion](#) (p. 197)
- [DeleteBotAlias](#) (p. 193)
- [DeleteBotChannelAssociation](#) (p. 195)
- [DeleteIntent](#) (p. 199)
- [DeleteIntentVersion](#) (p. 201)
- [DeleteSlotType](#) (p. 203)
- [DeleteSlotTypeVersion](#) (p. 205)

## Aliases

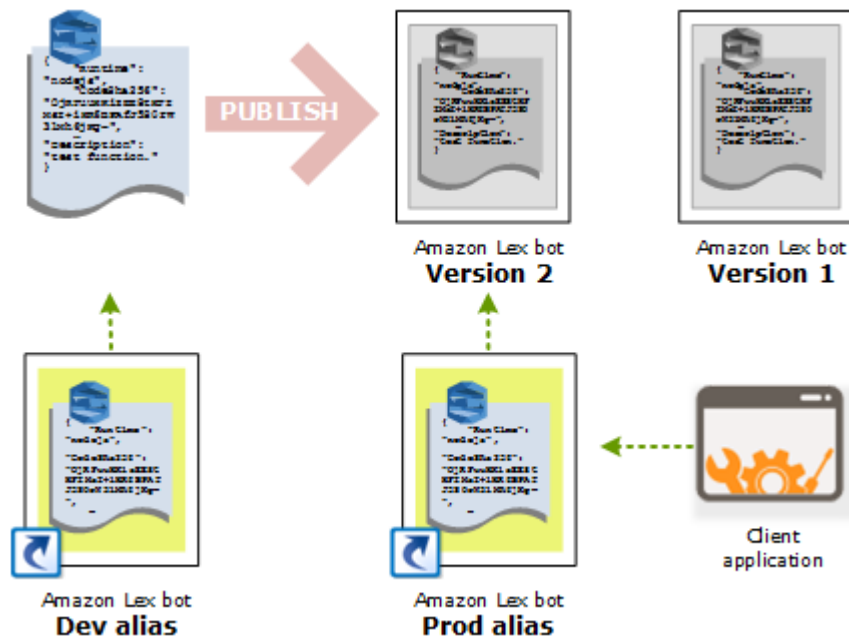
An alias is a pointer to a specific version of an Amazon Lex bot. Use an alias to allow client applications to use a specific version of the bot without requiring the application to track which version that is.

The following example shows two versions of an Amazon Lex bot, version `$LATEST` and version 1. Each of these bot versions has an associated alias, DEV and PROD, respectively. Client applications use the PROD alias to access the bot.



When you create a second version of the bot, you can update the alias to point to the new version of the bot using the console or the [PutBot](#) (p. 261) operation. When you change the alias, all of your client applications use the new version. If there is a problem with the new version, you can roll back to the previous version by simply changing the alias to point to that version.





#### Note

Although you can test the `$LATEST` version of a bot in the console, we recommend that when you integrate a bot with your client application, you first publish a version and create an alias that points to that version. Use the alias in your client application for the reasons explained in this section. When you update an alias, it might take a few minutes for the change to propagate.

# Using Lambda Functions

You can create AWS Lambda functions to use as code hooks for your Amazon Lex bot. You can identify Lambda functions to perform initialization and validation, fulfillment, or both in your intent configuration.

We recommend that you use a Lambda function as a code hook for your bot. Without a Lambda function, your bot returns the intent information to the client application for fulfillment.

## Topics

- [Lambda Function Input Event and Response Format \(p. 85\)](#)
- [Amazon Lex and AWS Lambda Blueprints \(p. 91\)](#)

## Lambda Function Input Event and Response Format

This section describes the structure of the event data that Amazon Lex provides to a Lambda function. Use this information to parse the input in your Lambda code. It also explains the format of the response that Amazon Lex expects your Lambda function to return.

## Topics

- [Input Event Format \(p. 85\)](#)
- [Response Format \(p. 88\)](#)

## Input Event Format

The following shows the general format of an Amazon Lex event that is passed to a Lambda function. Use this information when you are writing your Lambda function.

### Note

The input format may change without a corresponding change in the `messageVersion`. Your code should not throw an error if new fields are present.

```
{
  "currentIntent": {
    "name": "intent-name",
    "slots": {
```

```
    "slot-name": "value",
    "slot-name": "value",
    "slot-name": "value"
  },
  "confirmationStatus": "None, Confirmed, or Denied (intent confirmation, if
configured)",
},
"bot": {
  "name": "bot-name",
  "alias": "bot-alias",
  "version": "bot-version"
},
"userId": "User ID specified in the POST request to Amazon Lex.",
"inputTranscript": "Text used to process the request",
"invocationSource": "FulfillmentCodeHook or DialogCodeHook",
"outputDialogMode": "Text or Voice, based on ContentType request header in runtime API
request",
"messageVersion": "1.0",
"sessionAttributes": {
  "key1": "value1",
  "key2": "value2"
}
}
```

Note the following additional information about the event fields:

- **currentIntent** – Provides the intent name, slots, and `confirmationStatus` fields.

`slots` provides a list of slots that are configured for the intent and values that are recognized by Amazon Lex in the user conversation. Otherwise, the values are null.

`slots` is a map of slot names, configured for the intent, to slot values that Amazon Lex has recognized in the user conversation. A slot value remains null until the user provides a value.

A slot value may not match one of the slot values configured for the slot. For example, if the user responds to the prompt "What color car would you like?" with "pizza," Amazon Lex will return "pizza" as the slot type value. Your function should validate the values to make sure that they make sense in context.

`confirmationStatus` provides the user response to a confirmation prompt, if there is one. For example, if Amazon Lex asks "Do you want to order a large cheese pizza?," depending on the user response, the value of this field can be `Confirmed` or `Denied`. Otherwise, this value of this field is `None`.

If the user confirms the intent, Amazon Lex sets this field to `Confirmed`. If the user denies the intent, Amazon Lex sets this value to `Denied`.

In the confirmation response, a user utterance might provide slot updates. For example, the user might say "yes, change size to medium." In this case, the subsequent Lambda event has the updated slot value, `PizzaSize` set to `medium`. Amazon Lex sets the `confirmationStatus` to `None`, because the user modified some slot data, requiring the Lambda function to perform user data validation.

- **bot** – Information about the bot that processed the request.
  - **name** – The name of the bot that processed the request.
  - **alias** – The alias of the bot version that processed the request.
  - **version** – The version of the bot that processed the request.
- **userId** – This value is provided by the client application. Amazon Lex passes it to the Lambda function.
- **inputTranscript** – The text used to process the request.

If the input was text, the `inputTranscript` field contains the text that was input by the user.

If the input was an audio stream, the `inputTranscript` field contains the text extracted from the audio stream. This is the text that is actually processed to recognize intents and slot values.

- **invocationSource** – To indicate why Amazon Lex is invoking the Lambda function, it sets this to one of the following values:
  - **DialogCodeHook** – Amazon Lex sets this value to direct the Lambda function to initialize the function and to validate the user's data input.

When the intent is configured to invoke a Lambda function as an initialization and validation code hook, Amazon Lex invokes the specified Lambda function on each user input (utterance) after Amazon Lex understands the intent.

**Note**

If the intent is not clear, Amazon Lex can't invoke the Lambda function.

- **FulfillmentCodeHook** – Amazon Lex sets this value to direct the Lambda function to fulfill an intent.

If the intent is configured to invoke a Lambda function as a fulfillment code hook, Amazon Lex sets the `invocationSource` to this value only after it has all the slot data to fulfill the intent.

In your intent configuration, you can have two separate Lambda functions to initialize and validate user data and to fulfill the intent. You can also use one Lambda function to do both. In that case, your Lambda function can use the `invocationSource` value to follow the correct code path.

- **outputDialogMode** – For each user input, the client sends the request to Amazon Lex using one of the runtime API operations, [PostContent \(p. 286\)](#) or [PostText \(p. 293\)](#). Amazon Lex use the request parameters, Amazon Lex to determine whether the response to the client is text or voice, and sets this field accordingly.

The Lambda function can use this information to generate an appropriate message. For example, if the client expects a voice response, your Lambda function could return Speech Synthesis Markup Language (SSML) instead of text.

- **messageVersion** – The version of the message that identifies the format of the event data going into the Lambda function and the expected format of the response from a Lambda function.

**Note**

You configure this value when you define an intent. In the current implementation, only message version 1.0 is supported. Therefore, the console assumes the default value of 1.0 and doesn't show the message version.

- **sessionAttributes** – Application-specific session attributes that the client sent in the request. If you want Amazon Lex to include them in the response to the client, your Lambda function should send these back to Amazon Lex in the response. For more information, see the runtime API operations, [PostContent \(p. 286\)](#) and [PostText \(p. 293\)](#).

## Response Format

Amazon Lex expects a response from a Lambda function in the following format:

```
{
  "sessionAttributes": {
    "key1": "value1",
    "key2": "value2"
    ...
  },
  "dialogAction": {
    "type": "ElicitIntent, ElicitSlot, ConfirmIntent, Delegate, or Close",
    Full structure based on the type field. See below for details.
  }
}
```

The response consists of two fields. The `sessionAttributes` field is optional, the `dialogAction` field is required. The contents of the `dialogAction` field depends on the value of the `type` field. For details, see [dialogAction \(p. 88\)](#).

### sessionAttributes

Optional. If you include the `sessionAttributes` field it can be empty. If you want Amazon Lex to include any session attributes in the response to the client application, your Lambda function must return them in this field. For more information, see the [PostContent \(p. 286\)](#) and [PostText \(p. 293\)](#) operations.

```
"sessionAttributes": {
  "key1": "value1",
  "key2": "value2"
}
```

### dialogAction

Required. The `dialogAction` field directs Amazon Lex to the next course of action, and describes what to expect from the user after Amazon Lex returns a response to the client.

The `type` field indicates the next course of action. It also determines the other fields that the Lambda function needs to provide as part of the `dialogAction` value.

- `close` — Informs Amazon Lex not to expect a response from the user. For example, "Your pizza order has been placed" does not require a response.

The `fulfillmentState` field is required. Amazon Lex uses this value to set the `dialogState` in the response to the client application. The `message` and `responseCard` fields are optional. If you don't specify a message, Amazon Lex uses the goodbye message or the follow-up message configured for the intent.

```
"dialogAction": {
  "type": "Close",
  "fulfillmentState": "Fulfilled or Failed",
  "message": {
    "contentType": "PlainText or SSML",
    "content": "Message to convey to the user. For example, Thanks, your pizza has been ordered."
  },
  "responseCard": {
    "version": integer-value,
    "contentType": "application/vnd.amazonaws.card.generic",
    "genericAttachments": [
      {
        "title": "card-title",
        "subTitle": "card-sub-title",
        "imageUrl": "URL of the image to be shown",
        "attachmentLinkUrl": "URL of the attachment to be associated with the card",
        "buttons": [
          {
            "text": "button-text",
            "value": "Value sent to server on button click"
          }
        ]
      }
    ]
  }
}
```

- **ConfirmIntent** — Informs Amazon Lex that the user is expected to give a yes or no answer to confirm or deny the current intent.

You must include the `intentName` and `slots` fields. The `slots` field must contain an entry for each of the slots configured for the specified intent. If the value of a slot is unknown, you must set it to null. The `message` and `responseCard` fields are optional.

```
"dialogAction": {
  "type": "ConfirmIntent",
  "message": {
    "contentType": "PlainText or SSML",
    "content": "Message to convey to the user. For example, Are you sure you want a large pizza?"
  },
  "intentName": "intent-name",
  "slots": {
    "slot-name": "value",
    "slot-name": "value",
    "slot-name": "value"
  },
  "responseCard": {
    "version": integer-value,
    "contentType": "application/vnd.amazonaws.card.generic",
    "genericAttachments": [
      {

```

```

        "title": "card-title",
        "subTitle": "card-sub-title",
        "imageUrl": "URL of the image to be shown",
        "attachmentLinkUrl": "URL of the attachment to be associated with the card",
        "buttons": [
            {
                "text": "button-text",
                "value": "Value sent to server on button click"
            }
        ]
    }
}

```

- **Delegate** — Directs Amazon Lex to choose the next course of action based on the bot configuration. The response must include any session attributes, and the `slots` field must include all of the slots specified for the requested intent. If the value of the field is unknown, you must set it to null. You will get a `DependencyFailedException` exception if your fulfillment function returns the `Delegate` dialog action without removing any slots.

```

"dialogAction": {
  "type": "Delegate",
  "slots": {
    "slot-name": "value",
    "slot-name": "value",
    "slot-name": "value"
  }
}

```

- **ElicitIntent** — Informs Amazon Lex that the user is expected to respond with an utterance that includes an intent. For example, "I want a large pizza," which indicates the `OrderPizzaIntent`. The utterance "large," on the other hand, is not sufficient for Amazon Lex to infer the user's intent.

The `message` and `responseCard` fields are optional. If you don't provide a message, Amazon Lex uses one of the bot's clarification prompts.

```

{
  "dialogAction": {
    "type": "ElicitIntent",
    "message": {
      "contentType": "PlainText or SSML",
      "content": "Message to convey to the user. For example, What can I help you with?"
    },
    "responseCard": {
      "version": integer-value,
      "contentType": "application/vnd.amazonaws.card.generic",
      "genericAttachments": [
        {
          "title": "card-title",
          "subTitle": "card-sub-title",
          "imageUrl": "URL of the image to be shown",
          "attachmentLinkUrl": "URL of the attachment to be associated with the card",
          "buttons": [
            {
              "text": "button-text",
              "value": "Value sent to server on button click"
            }
          ]
        }
      ]
    }
  }
}

```

```
    ]  
  }  
}
```

- `ElicitSlot` — Informs Amazon Lex that the user is expected to provide a slot value in the response.

The `intentName`, `slotToElicit`, and `slots` fields are required. The `slots` field must include all of the slots specified for the requested intent. The `message` and `responseCard` fields are optional. If you don't specify a message, Amazon Lex uses one of the slot elicitation prompts configured for the slot.

```
"dialogAction": {  
  "type": "ElicitSlot",  
  "message": {  
    "contentType": "PlainText or SSML",  
    "content": "Message to convey to the user. For example, What size pizza would you  
like?"  
  },  
  "intentName": "intent-name",  
  "slots": {  
    "slot-name": "value",  
    "slot-name": "value",  
    "slot-name": "value"  
  },  
  "slotToElicit": "slot-name",  
  "responseCard": {  
    "version": integer-value,  
    "contentType": "application/vnd.amazonaws.card.generic",  
    "genericAttachments": [  
      {  
        "title": "card-title",  
        "subTitle": "card-sub-title",  
        "imageUrl": "URL of the image to be shown",  
        "attachmentLinkUrl": "URL of the attachment to be associated with the card",  
        "buttons": [  
          {  
            "text": "button-text",  
            "value": "Value sent to server on button click"  
          }  
        ]  
      }  
    ]  
  }  
}
```

## Amazon Lex and AWS Lambda Blueprints

The Amazon Lex console provides example bots (called bot blueprints) that are preconfigured so you can quickly create and test a bot in the console. For each of these bot blueprints, Lambda function blueprints are also provided. These blueprints provide sample code that works with their corresponding bots. You can use these blueprints to quickly create a bot that is configured with a Lambda function as a code hook, and test the end-to-end setup without having to write code.

You can use the following Amazon Lex bot blueprints and the corresponding AWS Lambda function blueprints as code hooks for bots:

- Amazon Lex blueprint — `OrderFlowers`



- AWS Lambda blueprints — `lex-order-flowers` (Node.js code) and `lex-order-flowers-python`
- Amazon Lex blueprint — `ScheduleAppointment`
- AWS Lambda blueprints — `lex-make-appointment` (Node.js code) and `lex-make-appointment-python`
- Amazon Lex blueprint — `BookTrip`
- AWS Lambda blueprints — `lex-book-trip` (Node.js code) and `lex-book-trip-python`

To create a bot using a blueprint and configure it to use a Lambda function as a code hook, see [Exercise 1: Create an Amazon Lex Bot Using a Blueprint \(Console\) \(p. 24\)](#). For an example of using other blueprints, see [Additional Examples: Creating Amazon Lex Bots \(p. 103\)](#).

# Deploying Amazon Lex Bots on Various Platforms

This section provides examples of deploying your Amazon Lex bot on various platforms.

## Topics

- [Deploying an Amazon Lex Bot on a Messaging Platform \(p. 93\)](#)
- [Deploying an Amazon Lex Bot in Mobile Applications \(p. 102\)](#)

## Deploying an Amazon Lex Bot on a Messaging Platform

This section provides examples of deploying your Amazon Lex bot on various platforms.

### Note

Amazon Lex uses AWS Key Management Service customer master keys (CMK) when storing your Facebook, Slack, or Twilio configurations. The first time that you integrate a messaging channel, Amazon Lex creates a default CMK (`aws/lex`). Alternatively, you can create your own CMK with AWS KMS. This gives you more flexibility, including the ability to create, rotate, and disable keys. You can also define access controls and audit the encryption keys used to protect your data. For more information, see the [AWS Key Management Service Developer Guide](#).

## Topics

- [Integrating an Amazon Lex Bot with Facebook Messenger \(p. 94\)](#)
- [Integrating an Amazon Lex Bot with Twilio Programmable SMS \(p. 96\)](#)
- [Integrating an Amazon Lex Bot with Slack \(p. 98\)](#)

# Integrating an Amazon Lex Bot with Facebook Messenger

## Topics

- [Step 1: Create an Amazon Lex Bot \(p. 94\)](#)
- [Step 2: Create a Facebook Application \(p. 94\)](#)
- [Step 3: Integrate Facebook Messenger with the Amazon Lex Bot \(p. 94\)](#)
- [Step 4: Test the Integration \(p. 95\)](#)

This exercise shows how to integrate Facebook Messenger with your Amazon Lex bot. You perform the following steps:

- Create an Amazon Lex bot.
- Integrate Facebook Messenger with your Amazon Lex bot.

## Step 1: Create an Amazon Lex Bot

In this section, you create an Amazon Lex bot.

1. Create an Amazon Lex bot. For instructions, see [Getting Started with Amazon Lex \(p. 21\)](#).
2. Deploy the bot and create an alias. For instructions, see [Exercise 3: Publish a Version and Create an Alias \(p. 57\)](#).

## Step 2: Create a Facebook Application

On the Facebook developer portal, create a Facebook application and a Facebook page. For instructions, see [Quick Start](#) in the Facebook Messenger platform documentation. Write down the following:

- **App Secret** for the Facebook App.
- **Page Access Token** for the Facebook page.

## Step 3: Integrate Facebook Messenger with the Amazon Lex Bot

In this section, you integrate Facebook Messenger with your Amazon Lex bot.

1. Open the Amazon Lex console, and then associate Facebook Messenger with your Amazon Lex bot.

After you complete this step, the console provides a callback URL. Write down this URL.

- a. Choose your Amazon Lex bot.
- b. Choose the **Channels** tab.
- c. Choose **Facebook** under For **Chatbots**. The console displays the Facebook integration page.
- d. On the **Facebook** integration page, provide the following information:
  - Type a name: `BotFacebookAssociation`
  - Choose "aws/lex" from the **KMS key** drop-down.
  - Choose the bot alias from the drop-down.
  - Type the verify token. This can be any string you choose (for example, `ExampleToken`). You use this same token in the Facebook developer portal in the Webhook setup step.

- Type the page access token and the app secret key you obtained from Facebook in the preceding step.

The screenshot shows the Amazon Lex console interface for configuring a bot channel. The bot is named 'BookTrip'. The 'Channels' tab is selected, and the 'Facebook' channel is being set up. The form contains the following fields and values:

- Name:** BotFacebookAssociation
- Description:** Channel for associating Facebook
- IAM Role:** AWSRoleForLexChannels (Automatically created on your behalf)
- KMS key:** aws/lex
- Alias:** Beta
- Verify token:** ExampleToken
- Page access token:** (empty field)
- App secret key:** (empty field)

An 'Activate' button is located at the bottom of the form. A 'Test Bot' button is visible in the bottom right corner.

- e. Choose **Activate**.

The console creates the bot channel association and returns a callback URL. Write down this URL.

2. On the Facebook developer portal, choose your app. Then, select the **Messenger** product and choose **Setup webhooks** in the **Webhooks** section of the page.

For instructions, see [Quick Start](#) in the Facebook Messenger platform documentation.

On the webhook page of the subscription wizard, do the following:

- For **Callback URL**, type the callback URL provided in the Amazon Lex console in the preceding section.
  - For **Verify Token**, type the same token that you used in Amazon Lex.
  - Choose **Subscription Fields** (messages, messaging\_postbacks, and messaging\_optins).
  - Choose **Verify and Save**. This results in a handshake between Facebook and Amazon Lex.
3. Enable Webhooks integration. Choose the page you created, and then choose **subscribe**.

**Note**

If you update or recreate a webhook, you must unsubscribe and then subscribe to the page again.

## Step 4: Test the Integration

You can now start conversation from Facebook Messenger with your Amazon Lex bot.

1. Open your Facebook page and choose, **Message**.
2. In the Messenger window that opens, use the same test utterances provided in getting started with your Amazon Lex bot.

# Integrating an Amazon Lex Bot with Twilio Programmable SMS

## Topics

- [Step 1: Create an Amazon Lex Bot \(p. 96\)](#)
- [Step 2: Create a Twilio SMS Account \(p. 96\)](#)
- [Step 3: Integrate the Twilio Messaging Service Endpoint with the Amazon Lex Bot \(p. 96\)](#)
- [Step 4: Test the Integration \(p. 97\)](#)

This exercise provides instructions for integrating an Amazon Lex bot with the Twilio simple messaging service (SMS). You perform the following steps:

- Create an Amazon Lex bot.
- Integrate Twilio programmable SMS with your bot Amazon Lex.
- Engage in an interaction with the Amazon Lex bot by testing the setup using the SMS service on your mobile phone.

## Step 1: Create an Amazon Lex Bot

If you don't already have an Amazon Lex bot, create and deploy one. In this topic, we assume that you are using the bot that you created in Getting Started Exercise 1. However, you can use any of the example bots provided in this guide. For Getting Started Exercise 1, see [Exercise 1: Create an Amazon Lex Bot Using a Blueprint \(Console\) \(p. 24\)](#).

1. Create an Amazon Lex bot. For instructions, see [Exercise 1: Create an Amazon Lex Bot Using a Blueprint \(Console\) \(p. 24\)](#).
2. Deploy the bot and create an alias. For instructions, see [Exercise 3: Publish a Version and Create an Alias \(p. 57\)](#).

## Step 2: Create a Twilio SMS Account

Sign up for a Twilio account and have the following account information available:

- **ACCOUNT SID**
- **AUTH TOKEN**

For sign-up instructions, see <https://www.twilio.com/console>.

## Step 3: Integrate the Twilio Messaging Service Endpoint with the Amazon Lex Bot

### To integrate Twilio with your Amazon Lex bot

1. To associate the Amazon Lex bot with your Twilio programmable SMS endpoint, activate bot channel association in the Amazon Lex console. When the bot channel association has been activated, Amazon Lex returns a callback URL. Record this callback URL because you need it later.
  - a. Sign in to the AWS Management Console, and open the Amazon Lex console at <https://console.aws.amazon.com/lex/>.

- b. Choose the Amazon Lex bot that you created in Step 1.
- c. Choose the **Channels** tab.
- d. In the **Chatbots** section, choose **Twilio SMS**.
- e. On the **Twilio SMS** page, provide the following information:
  - Type a name. For example, BotTwilioAssociation.
  - Choose "aws/lex" from the **KMS key** drop-down.
  - For **Alias**, choose the bot alias.
  - For **Authentication Token**, type the AUTH TOKEN for your Twilio account.
  - For **Account SID**, type the ACCOUNT SID for your Twilio account.

The screenshot shows the 'Twilio SMS' configuration page in the Amazon Lex console. The page has a sidebar with 'Chatbots' and 'Twilio SMS' selected. The main area contains the following fields:

- Name:** BotTwilioAssociation
- Description:** Channel for Twilio
- IAM Role:** AWSRoleForLexChannels (Automatically created on your behalf)
- KMS key:** aws/lex
- Alias:** Beta
- Authentication Token:** Authentication Token
- Account SID:** Account SID

Below these fields is an **Activate** button. At the bottom right, there is a **Test Bot** button.

- f. Choose **Activate**.

The console creates the bot channel association and returns a callback URL. Record this URL.

2. On the Twilio console, connect the Twilio SMS endpoint to the Amazon Lex bot.
  - a. Sign in to the Twilio console at <https://www.twilio.com/console>.
  - b. If you don't have a Twilio SMS endpoint, create it.
  - c. Update the **Inbound Settings** configuration of the messaging service by setting the **REQUEST URL** value to the callback URL that Amazon Lex provided in the preceding step.

## Step 4: Test the Integration

Use your mobile phone to test the integration between Twilio SMS and your bot.

### To test integration

1. Sign in to the Twilio console at <https://www.twilio.com/console> and do the following:

- a. Verify that you have a Twilio number associated with the messaging service under **Manage Numbers**.

You send messages to this number, and engage in SMS interaction with the Amazon Lex bot, from your mobile phone.

- b. Verify that your mobile phone is whitelisted as **Verified Caller ID**.

If it isn't, follow instructions on the Twilio console to whitelist the mobile phone that you plan to use for testing.

Now you can use your mobile phone to send messages to the Twilio SMS endpoint, which is mapped to the Amazon Lex bot.

2. Using your mobile phone, send messages to the Twilio number.

The Amazon Lex bot responds. If you created the bot using Getting Started Exercise 1, you can use the example conversations provided in that exercise. For more information, see [Step 4: Add the Lambda Function as Code Hook \(Console\)](#) (p. 36).

## Integrating an Amazon Lex Bot with Slack

### Topics

- [Step 1: Create an Amazon Lex Bot](#) (p. 98)
- [Step 2: Sign Up for Slack and Create a Slack Team](#) (p. 99)
- [Step 3: Create a Slack Application](#) (p. 99)
- [Step 4: Integrate the Slack Application with the Amazon Lex Bot](#) (p. 100)
- [Step 5: Complete Slack Integration](#) (p. 101)
- [Step 6: Test the Integration](#) (p. 101)

This exercise provides instructions for integrating an Amazon Lex bot with the Slack messaging application. You perform the following steps:

- Create an Amazon Lex bot.
- Create a Slack messaging application and integrate it with your bot Amazon Lex.
- Test the integration by engaging in conversation with your Amazon Lex bot. You sending messages with the Slack application and test in a browser window.

### Step 1: Create an Amazon Lex Bot

If you don't already have an Amazon Lex bot, create and deploy one. In this topic, we assume that you are using the bot that you created in Getting Started Exercise 1. However, you can use any of the example bots provided in this guide. For Getting Started Exercise 1, see [Exercise 1: Create an Amazon Lex Bot Using a Blueprint \(Console\)](#) (p. 24).

1. Create an Amazon Lex bot. For instructions, see [Exercise 1: Create an Amazon Lex Bot Using a Blueprint \(Console\)](#) (p. 24).
2. Deploy the bot and create an alias. For instructions, see [Exercise 3: Publish a Version and Create an Alias](#) (p. 57).

### Next Step

[Step 2: Sign Up for Slack and Create a Slack Team \(p. 99\)](#)

## Step 2: Sign Up for Slack and Create a Slack Team

Sign up for a Slack account and create a Slack team. For instructions, see [Using Slack](#). In the next section, you create a Slack application, which any Slack team can install.

### Next Step

[Step 3: Create a Slack Application \(p. 99\)](#)

## Step 3: Create a Slack Application

In this section, you do the following:

- Create a Slack application on the Slack API Console.
- Configure the application to add the following features:
  - A bot user
  - Interactive messaging

At the end of this section, you get the application credentials (Client Id, Client Secret, and Verification Token). In the next section, you use this information to configure bot channel association in the Amazon Lex console.

1. Sign in to the Slack API Console at <http://api.slack.com>.
2. Create an application.

After you have successfully created the application, Slack displays the **Basic Information** page for the application.

3. Configure the application features as follows:
    - a. In the left menu, choose **Bot Users**.
      - Provide a user name.
      - For **Always Show My Bot as Online**, choose **On**.
      - Choose **Add Bot User** to save the changes.
    - b. Choose **Interactive Messages** from the left menu.
      - Choose **Enable Interactive Messages**.
      - Specify any valid URL in the **Request URL** box. For example, you can use <https://slack.com>.
- Note**  
For now, enter any valid URL so that you get the verification token that you need in the next step. You will update this URL after you add the bot channel association in the Amazon Lex console.
- Choose **Enable Interactive Messages**.
4. In the **Settings** section in the left menu, choose **Basic Information**. Record the following application credentials:
    - Client ID
    - Client Secret
    - Verification Token



## Next Step

Step 4: Integrate the Slack Application with the Amazon Lex Bot (p. 100)

## Step 4: Integrate the Slack Application with the Amazon Lex Bot

Now that you have Slack application credentials, you can integrate the application with your Amazon Lex bot. To associate the Slack application with your bot, you add a bot channel association in Amazon Lex.

### To integrate the Slack application with your Amazon Lex bot

In the Amazon Lex console, activate a bot channel association to associate the bot with your Slack application. When the bot channel association is activated, Amazon Lex returns two URLs (**Postback URL** and **OAuth URL**). Record these URLs because you need them later.

1. Sign in to the AWS Management Console, and open the Amazon Lex console at <https://console.aws.amazon.com/lex/>.
2. Choose the Amazon Lex bot that you created in Step 1.
3. Choose the **Channels** tab.
4. In the **Chatbots** section, choose **Slack**.
5. On the **Slack** page, provide the following:
  - Type a name. For example, BotSlackIntegration.
  - Choose "aws/lex" from the **KMS key** drop-down.
  - For **Alias**, choose the bot alias.
  - Type the **Client Id**, **Client secret**, and **Verification Token**, which you recorded in the preceding step. These are the credentials of the Slack application.

The screenshot shows the Amazon Lex console interface for configuring a Slack bot channel association. The 'Channels' tab is active, and the 'Slack' chatbot is selected. The form contains the following fields:

- Name:** BotSlackAssociation
- Description:** Channel for Slack
- IAM Role:** AWSServiceRoleForLexChannels (Automatically created on your behalf)
- KMS key:** aws/lex
- Alias:** Beta
- Client Id:** Client Id
- Client secret:** Client secret
- Verification Token:** Verification Token
- Success Page URL:** Success Page URL

At the bottom right, there is a 'Test Bot' button.

6. Choose **Activate**.

The console creates the bot channel association and returns two URLs (Postback URL and OAuth URL). Record them. In the next section, you update your Slack application configuration to use these endpoints as follows:

- The Postback URL is the Amazon Lex bot's endpoint that listens to Slack events. You use this URL:
  - As the request URL in the **Event Subscriptions** feature of the Slack application.
  - To replace the placeholder value for the request URL in the **Interactive Messages** feature of the Slack application.
- The OAuth URL is your Amazon Lex bot's endpoint for an OAuth handshake with Slack.

### Next Step

[Step 5: Complete Slack Integration \(p. 101\)](#)

## Step 5: Complete Slack Integration

In this section, use the Slack API console to complete integration of the Slack application.

1. Sign in to the Slack API console at <http://api.slack.com>.
  2. Update the **OAuth & Permissions** feature as follows:
    - a. In the **Redirect URLs** section, add the OAuth URL that Amazon Lex provided in the preceding step. Choose **Add a new Redirect URL**, and then choose **Save URLs**.
    - b. In the **Permission Scopes** section, choose two permissions in the **Select Permission Scopes** drop down. Filter the list with the following text:
      - `chat:write:bot`
      - `team:read`
- Choose **Save Changes**.
3. Update the **Interactive Messages** feature by updating the **Request URL** value to the Postback URL that Amazon Lex provided in the preceding step. Choose **Add**, and then choose **Save URLs**.
  4. Subscribe to the **Event Subscriptions** feature as follows:
    - Enable events by choosing the **On** option.
    - Set the **Request URL** value to the Postback URL that Amazon Lex provided in the preceding step.
    - Subscribe to the `message.im` bot event to enable direct messaging between the end user and the Slack bot.
    - Save the changes.

### Next Step

[Step 6: Test the Integration \(p. 101\)](#)

## Step 6: Test the Integration

Now use a browser window to test the integration of Slack with your Amazon Lex bot.

1. Choose **Manage Distribution** under **Settings**. Choose **Add to Slack** to install the application. Authorize the bot to respond to messages.
2. You are redirected to your Slack team. Choose your bot from the **Direct Messages** section in the left menu. If you don't see your bot, choose the plus icon (+) next to **Direct Messages** to search for your bot.

3. Engage in a chat with your Slack application, which is linked to the Amazon Lex bot. Your bot now responds to messages.

If you created the bot using Getting Started Exercise 1, you can use the example conversations provided in that exercise. For more information, see [Step 4: Add the Lambda Function as Code Hook \(Console\)](#) (p. 36).

## Deploying an Amazon Lex Bot in Mobile Applications

Using AWS SDKs, you can integrate your Amazon Lex bot with your mobile applications. For more information, see the following topics:

- Android SDK – [Getting Started with Amazon Lex Android SDK](#)
- iOS SDK – [Getting Started with Amazon Lex iOS SDK](#)

You can also use the AWS Mobile Hub to create a quickstart mobile app that demonstrates using the Amazon Lex SDK in iOS and Android mobile applications. For more information, see [AWS Mobile Hub Conversational Bots](#).

# Additional Examples: Creating Amazon Lex Bots

The following sections provide additional Amazon Lex exercises with step-by-step instructions.

## Topics

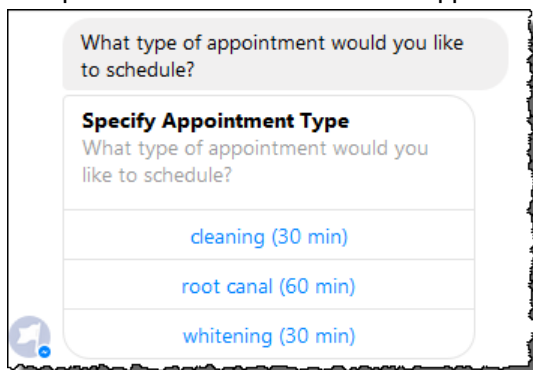
- [Example Bot: ScheduleAppointment \(p. 103\)](#)
- [Example Bot: BookTrip \(p. 121\)](#)
- [Example: Using a Response Card \(p. 143\)](#)
- [Example: Updating Utterances \(p. 145\)](#)
- [Example: Integrating with a Web site \(p. 146\)](#)

## Example Bot: ScheduleAppointment

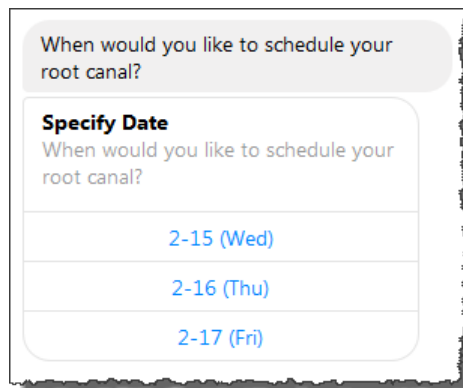
The example bot in this exercise schedules appointments for a dentist's office. The example also illustrates using response cards to obtain user input with buttons. Specifically, the example illustrates generating response cards dynamically at runtime.

You can configure response cards at build time (also referred to as static response cards) or generate them dynamically in an AWS Lambda function. In this example, the bot uses the following response cards:

- A response card that lists buttons for appointment type. For example:



- A response card that lists buttons for appointment date. For example:

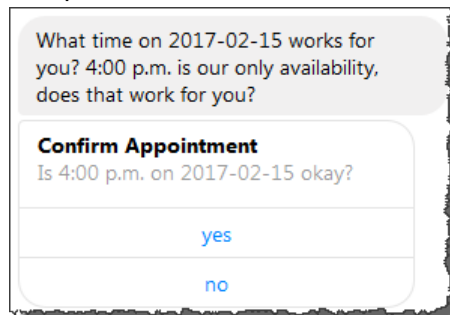


When would you like to schedule your root canal?

**Specify Date**  
When would you like to schedule your root canal?

2-15 (Wed)
2-16 (Thu)
2-17 (Fri)

- A response card that lists buttons to confirm a suggested appointment time. For example:



What time on 2017-02-15 works for you? 4:00 p.m. is our only availability, does that work for you?

**Confirm Appointment**  
Is 4:00 p.m. on 2017-02-15 okay?

yes
no

The available appointment dates and times vary, which requires you to generate response cards at runtime. You use an AWS Lambda function to generate these response cards dynamically. The Lambda function returns response cards in its response to Amazon Lex. Amazon Lex includes the response card in its response to the client.

If a client (for example, Facebook Messenger) supports response cards, the user can either choose from the list of buttons or type the response. Otherwise, the user simply types the response.

In addition to the button shown in the preceding example, you can also include images, attachments, and other useful information to display on response cards. For information about response cards, see [Response Cards \(p. 14\)](#).

In this exercise, you do the following:

- Create and test a bot (using the ScheduleAppointment blueprint). For this exercise, you use a bot blueprint to quickly set up and test the bot. For a list of available blueprints, see [Amazon Lex and AWS Lambda Blueprints \(p. 91\)](#). This bot is preconfigured with one intent (`MakeAppointment`).
- Create and test a Lambda function (using the `lex-make-appointment-python` blueprint provided by Lambda). You configure the `MakeAppointment` intent to use this Lambda function as a code hook to perform initialization, validation, and fulfillment tasks.

**Note**

The provided example Lambda function showcases a dynamic conversation based on the mocked-up availability of a dentist appointment. In a real application, you might use a real calendar to set an appointment.

- Update the `MakeAppointment` intent configuration to use the Lambda function as a code hook. Then, test the end-to-end experience.

- Publish the schedule appointment bot to Facebook Messenger so you can see the response cards in action (the client in the Amazon Lex console currently does not support response cards).

The following sections provide summary information about the blueprints you use in this exercise.

#### Topics

- [Overview of the Bot Blueprint \(ScheduleAppointment\) \(p. 105\)](#)
- [Overview of the Lambda Function Blueprint \(lex-make-appointment-python\) \(p. 106\)](#)
- [Step 1: Create an Amazon Lex Bot \(p. 106\)](#)
- [Step 2: Create a Lambda Function \(p. 108\)](#)
- [Step 3: Update the Intent: Configure a Code Hook \(p. 108\)](#)
- [Step 4: Deploy the Bot on the Facebook Messenger Platform \(p. 109\)](#)
- [Details of Information Flow \(p. 110\)](#)

## Overview of the Bot Blueprint (ScheduleAppointment)

The ScheduleAppointment blueprint that you use to create a bot for this exercise is preconfigured with the following:

- **Slot types** – One custom slot type called `AppointmentTypeValue`, with the enumeration values `root canal`, `cleaning`, and `whitening`.
- **Intent** – One intent (`MakeAppointment`), which is preconfigured as follows:
  - **Slots** – The intent is configured with the following slots:
    - Slot `AppointmentType`, of the `AppointmentTypes` custom type.
    - Slot `Date`, of the `AMAZON.DATE` built-in type.
    - Slot `Time`, of the `AMAZON.TIME` built-in type.
  - **Utterances** – The intent is preconfigured with the following utterances:
    - "I would like to book an appointment"
    - "Book an appointment"
    - "Book a {AppointmentType}"

If the user utters any of these, Amazon Lex determines that `MakeAppointment` is the intent, and then uses the prompts to elicit slot data.

- **Prompts** – The intent is preconfigured with the following prompts:
  - Prompt for the `AppointmentType` slot – "What type of appointment would you like to schedule?"
  - Prompt for the `Date` slot – "When should I schedule your {AppointmentType}?"
  - Prompt for the `Time` slot – "At what time do you want to schedule the {AppointmentType}?" and "At what time on {Date}?"
  - Confirmation prompt – "{Time} is available, should I go ahead and book your appointment?"
  - Cancel message– "Okay, I will not schedule an appointment."

## Overview of the Lambda Function Blueprint (lex-make-appointment-python)

The Lambda function blueprint (lex-make-appointment-python) is a code hook for bots that you create using the ScheduleAppointment bot blueprint.

This Lambda function blueprint code can perform both initialization/validation and fulfillment tasks.

- The Lambda function code showcases a dynamic conversation that is based on example availability for a dentist appointment (in real applications, you might use a calendar). For the day or date that the user specifies, the code is configured as follows:
  - If there are no appointments available, the Lambda function returns a response directing Amazon Lex to prompt the user for another day or date (by setting the `dialogAction` type to `ElicitSlot`). For more information, see [Response Format \(p. 88\)](#).
  - If there is only one appointment available on the specified day or date, the Lambda function suggests the available time in the response and directs Amazon Lex to obtain user confirmation by setting the `dialogAction` in the response to `ConfirmIntent`. This illustrates how you can improve the user experience by proactively suggesting the available time for an appointment.
  - If there are multiple appointments available, the Lambda function returns a list of available times in the response to Amazon Lex. Amazon Lex returns a response to the client with the message from the Lambda function.
- As the fulfillment code hook, the Lambda function returns a summary message indicating that an appointment is scheduled (that is, the intent is fulfilled).

### Note

In this example, we show how to use response cards. The Lambda function constructs and returns a response card to Amazon Lex. The response card lists available days and times as buttons to choose from. When testing the bot using the client provided by the Amazon Lex console, you cannot see the response card. To see it, you must integrate the bot with a messaging platform, such as Facebook Messenger. For instructions, see [Integrating an Amazon Lex Bot with Facebook Messenger \(p. 94\)](#). For more information about response cards, see [Managing Messages \(Prompts and Statements\) \(p. 9\)](#).

When Amazon Lex invokes the Lambda function, it passes event data as input. One of the event fields is `invocationSource`, which the Lambda function uses to choose between an input validation and fulfillment activity. For more information, see [Input Event Format \(p. 85\)](#).

### Next Step

[Step 1: Create an Amazon Lex Bot \(p. 106\)](#)

## Step 1: Create an Amazon Lex Bot

In this section, you create an Amazon Lex bot using the ScheduleAppointment blueprint, which is provided in the Amazon Lex console.

1. Sign in to the AWS Management Console and open the Amazon Lex console at <https://console.aws.amazon.com/lex/>.
2. On the **Bots** page, choose **Create**.
3. On the **Create your Lex bot** page, do the following:
  - Choose the **ScheduleAppointment** blueprint.
  - Leave the default bot name (ScheduleAppointment).

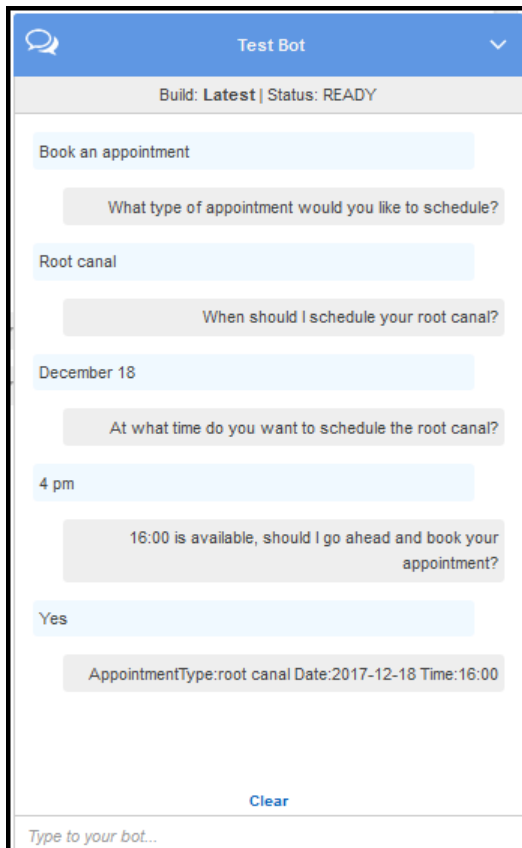
4. Choose **Create**.

This step saves and builds the bot. The console sends the following requests to Amazon Lex during the build process:

- Create a new version of the slot types (from the \$LATEST version). For information about slot types defined in this bot blueprint, see [Overview of the Bot Blueprint \(ScheduleAppointment\) \(p. 105\)](#).
- Create a version of the `MakeAppointment` intent (from the \$LATEST version). In some cases, the console sends a request for the `update` API operation before creating a new version.
- Update the \$LATEST version of the bot.

At this time, Amazon Lex builds a machine learning model for the bot. When you test the bot in the console, the console uses the runtime API to send user input back to Amazon Lex. Amazon Lex then uses the machine learning model to interpret the user input.

5. The console shows the `ScheduleAppointment` bot. On the **Editor** tab, review the preconfigured intent (`MakeAppointment`) details.
6. Test the bot in the test window. Use the following screen shot to engage in a test conversation with your bot:



Note the following:

- From the initial user input ("Book an appointment"), the bot infers the intent (`MakeAppointment`).
- The bot then uses the configured prompts to get slot data from the user.
- The bot blueprint has the `MakeAppointment` intent configured with the following confirmation prompt:



```
{Time} is available, should I go ahead and book your appointment?
```

After the user provides all of the slot data, Amazon Lex returns a response to the client with a confirmation prompt as the message. The client displays the message for the user:

```
16:00 is available, should I go ahead and book your appointment?
```

Notice that the bot accepts any appointment date and time values because you don't have any code to initialize or validate the user data. In the next section, you add a Lambda function to do this.

### Next Step

[Step 2: Create a Lambda Function \(p. 108\)](#)

## Step 2: Create a Lambda Function

In this section, you create a Lambda function using a blueprint (lex-make-appointment-python) that is provided in the Lambda console. You also test the Lambda function by invoking it using sample Amazon Lex event data that is provided by the console.

1. Sign in to the AWS Management Console and open the AWS Lambda console at <https://console.aws.amazon.com/lambda/>.
2. Choose **Create a Lambda function**.
3. For **Select blueprint**, type `lex` to find the blueprint, and then choose the **lex-make-appointment-python** blueprint.
4. Configure the Lambda function as follows, and then choose **Create Function**.
  - Type the Lambda function name (`MakeAppointmentCodeHook`).
  - For the role, choose **Create a new role from template(s)**, and then type a role name.
  - Leave other default values.
5. Test the Lambda function.
  - a. Choose **Actions**, and then choose **Configure test event**.
  - b. From the **Sample event template** list, choose **Lex-Make Appointment (preview)**. This sample event uses the Amazon Lex request/response model, with values set to match a request from your Amazon Lex bot. For information about the Amazon Lex request/response model, see [Using Lambda Functions \(p. 85\)](#).
  - c. Choose **Save and test**.
  - d. Verify that the Lambda function successfully executed. The response in this case matches the Amazon Lex response model.

### Next Step

[Step 3: Update the Intent: Configure a Code Hook \(p. 108\)](#)

## Step 3: Update the Intent: Configure a Code Hook

In this section, you update the configuration of the `MakeAppointment` intent to use the Lambda function as a code hook for the validation and fulfillment activities.

1. In the Amazon Lex console, select the ScheduleAppointment bot. The console shows the **MakeAppointment** intent. Modify the intent configuration as follows.

**Note**

You can update only the \$LATEST versions of any of the Amazon Lex resources, including the intents. Make sure that the intent version is set to \$LATEST. You have not published a version of your bot yet, so it should still be the \$LATEST version in the console.

- a. In the **Options** section, choose **Initialization and validation code hook**, and then choose the Lambda function from the list.
  - b. In the **Fulfillment** section, choose **AWS Lambda function**, and then choose the Lambda function from the list.
  - c. Choose **Goodbye message**, and type a message.
2. Choose **Save**, and then choose **Build**.
  3. Test the bot.

The screenshot shows the 'Test Bot' interface in the Amazon Lex console. At the top, it says 'Build: Latest | Status: READY'. The conversation starts with the user input 'Book appointment'. The bot responds with 'What type of appointment would you like to schedule?'. The user inputs 'root canal'. The bot responds with 'When would you like to schedule your root canal?'. The user inputs 'Tuesday'. The bot responds with 'We do not have any availability on that date, is there another day which works for you?'. The user inputs 'Wednesday'. The bot responds with 'What time on 2017-01-18 works for you? 4:00 p.m. is our only availability, does that work for you?'. The user inputs 'yes'. The bot responds with 'Okay, I have booked your appointment. We will see you at 4:00 p.m. on 2017-01-18'. At the bottom, there is a 'Clear' button and a text input field with the placeholder 'Type to your bot...'.

**Next Step**

[Step 4: Deploy the Bot on the Facebook Messenger Platform \(p. 109\)](#)

## Step 4: Deploy the Bot on the Facebook Messenger Platform

In the preceding section, you tested the ScheduleAppointment bot using the client in the Amazon Lex console. Currently, the Amazon Lex console does not support response cards. To test the dynamically

generated response cards that the bot supports, deploy the bot on the Facebook Messenger platform and test it.

For instructions, see [Integrating an Amazon Lex Bot with Facebook Messenger \(p. 94\)](#).

### Next Step

[Details of Information Flow \(p. 110\)](#)

## Details of Information Flow

The `ScheduleAppointment` bot blueprint primarily showcases the use of dynamically generated response cards. The Lambda function in this exercise includes response cards in its response to Amazon Lex. Amazon Lex includes the response cards in its reply to the client. This section explains both the following:

- Data flow between client and Amazon Lex.

The section assumes client sends requests to Amazon Lex using the `PostText` runtime API and shows request/response details accordingly. For more information about the `PostText` runtime API, see [PostText \(p. 293\)](#).

### Note

For an example of information flow between client and Amazon Lex in which client uses the `PostContent` API, see [Step 2a \(Optional\): Review the Details of the Spoken Information Flow \(Console\) \(p. 27\)](#).

- Data flow between Amazon Lex and the Lambda function. For more information, see [Lambda Function Input Event and Response Format \(p. 85\)](#).

### Note

The example assumes that you are using the Facebook Messenger client, which does not pass session attributes in the request to Amazon Lex. Accordingly, the example requests shown in this section show empty `sessionAttributes`. If you test the bot using the client provided in the Amazon Lex console, the client includes the session attributes.

This section describes what happens after each user input.

1. User: Types **Book an appointment**.
  - a. The client (console) sends the following [PostContent \(p. 286\)](#) request to Amazon Lex:

```
POST /bot/ScheduleAppointment/alias/$LATEST/user/bijt6rovckwecnzsbthrr1d7lv3ja3n/
text
"Content-Type":"application/json"
"Content-Encoding":"amz-1.0"

{
  "inputText":"book appointment",
  "sessionAttributes":{}
}
```

Both the request URI and the body provide information to Amazon Lex:

- Request URI – Provides the bot name (ScheduleAppointment), the bot alias (\$LATEST), and the user name ID. The trailing `text` indicates that it is a `PostText` (not `PostContent`) API request.
  - Request body – Includes the user input (`inputText`) and empty `sessionAttributes`.
- b. From the `inputText`, Amazon Lex detects the intent (`MakeAppointment`). The service invokes the Lambda function, which is configured as a code hook, to perform initialization and validation by passing the following event. For details, see [Input Event Format](#) (p. 85).

```
{
  "currentIntent": {
    "slots": {
      "AppointmentType": null,
      "Date": null,
      "Time": null
    },
    "name": "MakeAppointment",
    "confirmationStatus": "None"
  },
  "bot": {
    "alias": null,
    "version": "$LATEST",
    "name": "ScheduleAppointment"
  },
  "userId": "bijt6rovckwecnzeshrrld7lv3ja3n",
  "invocationSource": "DialogCodeHook",
  "outputDialogMode": "Text",
  "messageVersion": "1.0",
  "sessionAttributes": {}
}
```

In addition to the information sent by the client, Amazon Lex also includes the following data:

- `currentIntent` – Provides current intent information.
  - `invocationSource` – Indicates the purpose of the Lambda function invocation. In this case, the purpose is to perform user data initialization and validation. (Amazon Lex knows that the user has not provided all of the slot data to fulfill the intent yet.)
  - `messageVersion` – Currently Amazon Lex supports only the 1.0 version.
- c. At this time, all of the slot values are null (there is nothing to validate). The Lambda function returns the following response to Amazon Lex, directing the service to elicit information for the `AppointmentType` slot. For information about the response format, see [Response Format](#) (p. 88).

```
{
  "dialogAction": {
    "slotToElicit": "AppointmentType",
    "intentName": "MakeAppointment",
    "responseCard": {
      "genericAttachments": [
        {
          "buttons": [
            {
              "text": "cleaning (30 min)",
              "value": "cleaning"
            },
            {
              "text": "root canal (60 min)",
              "value": "root canal"
            },
            {
              "text": "whitening (30 min)",

```

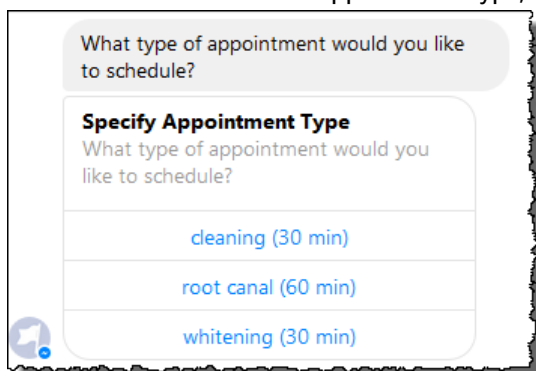
```

        "value": "whitening"
      }
    ],
    "subTitle": "What type of appointment would you like to
schedule?",
    "title": "Specify Appointment Type"
  }
],
"version": 1,
"contentType": "application/vnd.amazonaws.card.generic"
},
"slots": {
  "AppointmentType": null,
  "Date": null,
  "Time": null
},
"type": "ElicitSlot",
"message": {
  "content": "What type of appointment would you like to schedule?",
  "contentType": "PlainText"
}
},
"sessionAttributes": {}
}

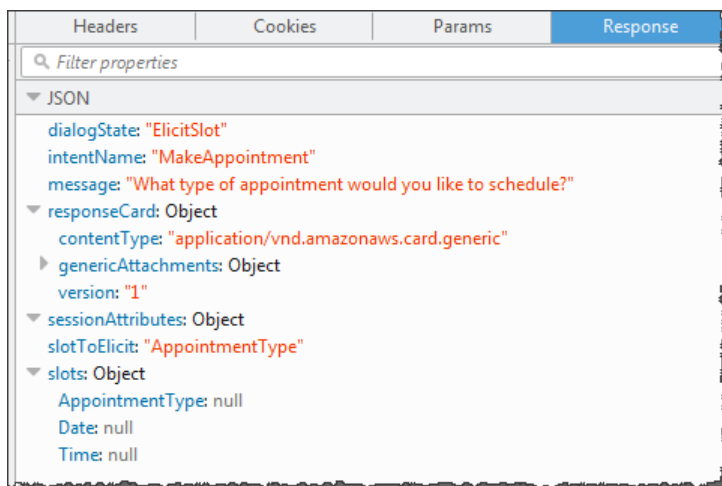
```

The response includes the `dialogAction` and `sessionAttributes` fields. Among other things, the `dialogAction` field returns the following fields:

- `type` – By setting this field to `ElicitSlot`, the Lambda function directs Amazon Lex to elicit the value for the slot specified in the `slotToElicit` field. The Lambda function also provides a message to convey to the user.
- `responseCard` – Identifies a list of possible values for the `AppointmentType` slot. A client that supports response cards (for example, the Facebook Messenger) displays a response card to allow the user to choose an appointment type, as follows:



- d. As indicated by the `dialogAction.type` in the response from the Lambda function, Amazon Lex sends the following response back to the client:



The client reads the response, and then displays the message: "What type of appointment would you like to schedule?" and the response card (if the client supports response cards).

2. User: Depending on the client, the user has two options:
  - If the response card is shown, choose **root canal (60 min)** or type **root canal**.
  - If the client does not support response cards, type **root canal**.
- a. The client sends the following `PostText` request to Amazon Lex (line breaks have been added for readability):

```

POST /bot/BookTrip/alias/$LATEST/user/bijt6rovckwecnzesbthrrid7lv3ja3n/text
"Content-Type": "application/json"
"Content-Encoding": "amz-1.0"

{
  "inputText": "root canal",
  "sessionAttributes": {}
}

```

- b. Amazon Lex invokes the Lambda function for user data validation by sending the following event as a parameter:

```

{
  "currentIntent": {
    "slots": {
      "AppointmentType": "root canal",
      "Date": null,
      "Time": null
    },
    "name": "MakeAppointment",
    "confirmationStatus": "None"
  },
  "bot": {
    "alias": null,
    "version": "$LATEST",
    "name": "ScheduleAppointment"
  },
  "userId": "bijt6rovckwecnzesbthrrid7lv3ja3n",
  "invocationSource": "DialogCodeHook",
  "outputDialogMode": "Text",
  "messageVersion": "1.0",
}

```

```
    "sessionAttributes": {}  
  }
```

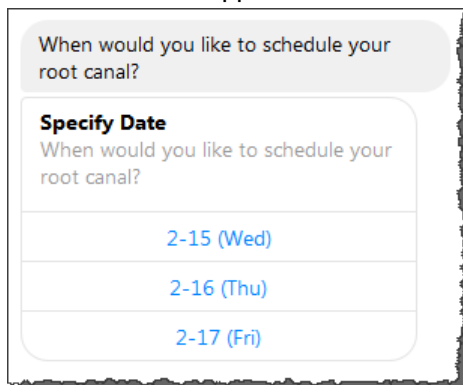
In the event data, note the following:

- `invocationSource` continues to be `DialogCodeHook`. In this step, we are just validating user data.
  - Amazon Lex sets the `AppointmentType` field in the `currentIntent.slots` slot to `root canal`.
  - Amazon Lex simply passes the `sessionAttributes` field between the client and the Lambda function.
- c. The Lambda function validates the user input and returns the following response to Amazon Lex, directing the service to elicit a value for the appointment date.

```
{  
  "dialogAction": {  
    "slotToElicit": "Date",  
    "intentName": "MakeAppointment",  
    "responseCard": {  
      "genericAttachments": [  
        {  
          "buttons": [  
            {  
              "text": "2-15 (Wed)",  
              "value": "Wednesday, February 15, 2017"  
            },  
            {  
              "text": "2-16 (Thu)",  
              "value": "Thursday, February 16, 2017"  
            },  
            {  
              "text": "2-17 (Fri)",  
              "value": "Friday, February 17, 2017"  
            },  
            {  
              "text": "2-20 (Mon)",  
              "value": "Monday, February 20, 2017"  
            },  
            {  
              "text": "2-21 (Tue)",  
              "value": "Tuesday, February 21, 2017"  
            }  
          ],  
          "subTitle": "When would you like to schedule your root canal?",  
          "title": "Specify Date"  
        }  
      ],  
      "version": 1,  
      "contentType": "application/vnd.amazonaws.card.generic"  
    },  
    "slots": {  
      "AppointmentType": "root canal",  
      "Date": null,  
      "Time": null  
    },  
    "type": "ElicitSlot",  
    "message": {  
      "content": "When would you like to schedule your root canal?",  
      "contentType": "PlainText"  
    }  
  },  
  "sessionAttributes": {}  
}
```

Again, the response includes the `dialogAction` and `sessionAttributes` fields. Among other things, the `dialogAction` field returns the following fields:

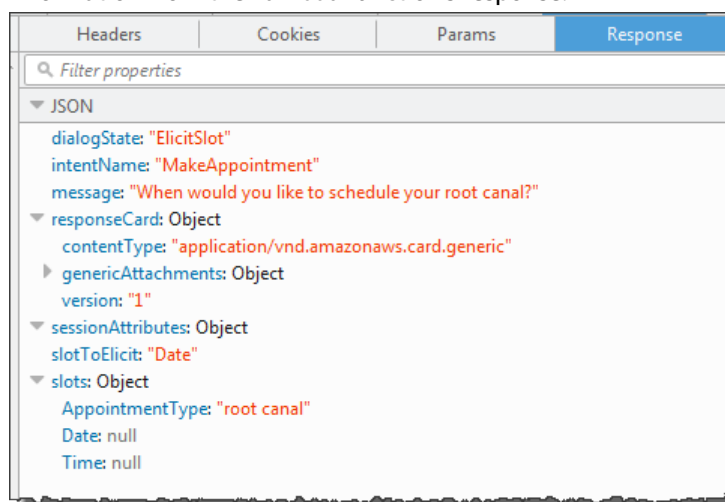
- `type` – By setting this field to `ElicitSlot`, the Lambda function directs Amazon Lex to elicit the value for the slot specified in the `slotToElicit` field. The Lambda function also provides a message to convey to the user.
- `responseCard` – Identifies a list of possible values for the `Date` slot. A client that supports response cards (for example, Facebook Messenger) displays a response card that allows the user to choose an appointment date:



Although the Lambda function returned five dates, the client (Facebook Messenger) has a limit of three buttons for a response card. Therefore, you see only the first three values in the screen shot.

These dates are hard coded in the Lambda function. In a production application, you might use a calendar to get available dates in real time. Because the dates are dynamic, you must generate the response card dynamically in the Lambda function.

- d. Amazon Lex notices the `dialogAction.type` and returns a response to the client that includes information from the Lambda function's response.



The client displays the message: **When would you like to schedule your root canal?** and the response card (if the client supports response cards).

3. User: Types **Thursday**.



- a. The client sends the following `PostText` request to Amazon Lex (line breaks have been added for readability):

```
POST /bot/BookTrip/alias/$LATEST/user/bijt6rovckwecnzeshrr1d7lv3ja3n/text
"Content-Type": "application/json"
"Content-Encoding": "amz-1.0"

{
  "inputText": "Thursday",
  "sessionAttributes": {}
}
```

- b. Amazon Lex invokes the Lambda function for user data validation by sending in the following event as a parameter:

```
{
  "currentIntent": {
    "slots": {
      "AppointmentType": "root canal",
      "Date": "2017-02-16",
      "Time": null
    },
    "name": "MakeAppointment",
    "confirmationStatus": "None"
  },
  "bot": {
    "alias": null,
    "version": "$LATEST",
    "name": "ScheduleAppointment"
  },
  "userId": "u3fpr9gghj02zts7y5tpq5mm4din2xqy",
  "invocationSource": "DialogCodeHook",
  "outputDialogMode": "Text",
  "messageVersion": "1.0",
  "sessionAttributes": {}
}
```

In the event data, note the following:

- `invocationSource` continues to be `DialogCodeHook`. In this step, we are just validating the user data.
  - Amazon Lex sets the `Date` field in the `currentIntent.slots` slot to `2017-02-16`.
  - Amazon Lex simply passes the `sessionAttributes` between the client and the Lambda function.
- c. The Lambda function validates the user input. This time the Lambda function determines that there are no appointments available on the specified date. It returns the following response to Amazon Lex, directing the service to again elicit a value for the appointment date.

```
{
  "dialogAction": {
    "slotToElicit": "Date",
    "intentName": "MakeAppointment",
    "responseCard": {
      "genericAttachments": [
        {
          "buttons": [
            {
              "text": "2-15 (Wed)",
              "value": "Wednesday, February 15, 2017"
            }
          ]
        }
      ]
    }
  }
}
```

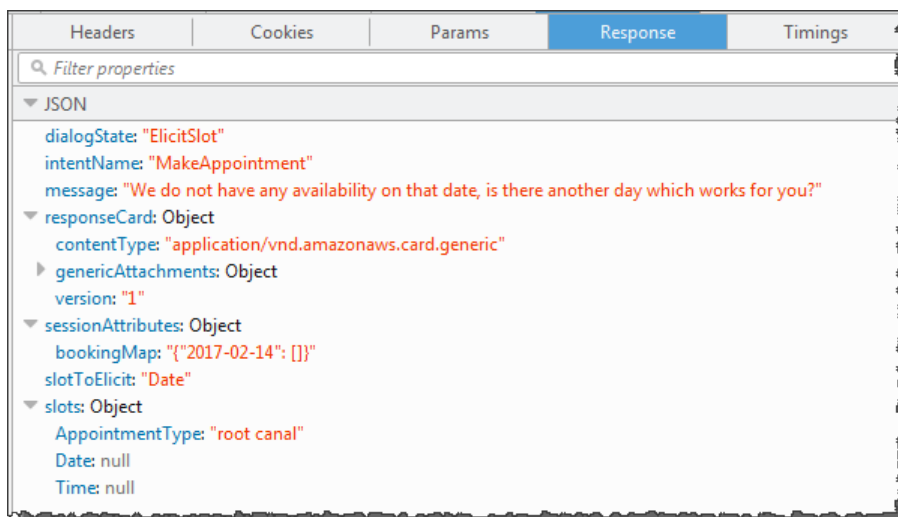
```

        {
            "text": "2-16 (Thu)",
            "value": "Thursday, February 16, 2017"
        },
        {
            "text": "2-17 (Fri)",
            "value": "Friday, February 17, 2017"
        },
        {
            "text": "2-20 (Mon)",
            "value": "Monday, February 20, 2017"
        },
        {
            "text": "2-21 (Tue)",
            "value": "Tuesday, February 21, 2017"
        }
    ],
    "subTitle": "When would you like to schedule your root canal?",
    "title": "Specify Date"
}
],
"version": 1,
"contentType": "application/vnd.amazonaws.card.generic"
},
"slots": {
    "AppointmentType": "root canal",
    "Date": null,
    "Time": null
},
"type": "ElicitSlot",
"message": {
    "content": "We do not have any availability on that date, is there
another day which works for you?",
    "contentType": "PlainText"
}
},
"sessionAttributes": {
    "bookingMap": "{\"2017-02-16\": []}"
}
}

```

Again, the response includes the `dialogAction` and `sessionAttributes` fields. Among other things, the `dialogAction` returns the following fields:

- `dialogAction` field:
    - `type` – The Lambda function sets this value to `ElicitSlot` and resets the `slotToElicit` field to `Date`. The Lambda function also provides an appropriate message to convey to the user.
    - `responseCard` – Returns a list of values for the `Date` slot.
  - `sessionAttributes` – This time the Lambda function includes the `bookingMap` session attribute. Its value is the requested date of the appointment and available appointments (an empty object indicates that no appointments are available).
- d. Amazon Lex notices the `dialogAction.type` and returns a response to the client that includes information from the Lambda function's response.



The client displays the message: **We do not have any availability on that date, is there another day which works for you?** and the response card (if the client supports response cards).

4. User: Depending on the client, the user has two options:
  - If the response card is shown, choose **2-15 (Wed)** or type **wednesday**.
  - If the client does not support response cards, type **wednesday**.
- a. The client sends the following `PostText` request to Amazon Lex:

```
POST /bot/BookTrip/alias/$LATEST/user/bijt6rovckwecnzesbthrr1d7lv3ja3n/text
"Content-Type": "application/json"
"Content-Encoding": "amz-1.0"

{
  "inputText": "Wednesday",
  "sessionAttributes": {
  }
}
```

### Note

The Facebook Messenger client does not set any session attributes. If you want to maintain session states between requests, you must do so in the Lambda function. In a real application, you might need to maintain these session attributes in a backend database.

- b. Amazon Lex invokes the Lambda function for user data validation by sending the following event as a parameter:

```
{
  "currentIntent": {
    "slots": {
      "AppointmentType": "root canal",
      "Date": "2017-02-15",
      "Time": null
    },
    "name": "MakeAppointment",
    "confirmationStatus": "None"
  },
}
```

```
"bot": {
  "alias": null,
  "version": "$LATEST",
  "name": "ScheduleAppointment"
},
"userId": "u3fpr9gghj02zts7y5tpq5mm4din2xqy",
"invocationSource": "DialogCodeHook",
"outputDialogMode": "Text",
"messageVersion": "1.0",
"sessionAttributes": {
}
}
```

Amazon Lex updated `currentIntent.slots` by setting the `Date` slot to 2017-02-15.

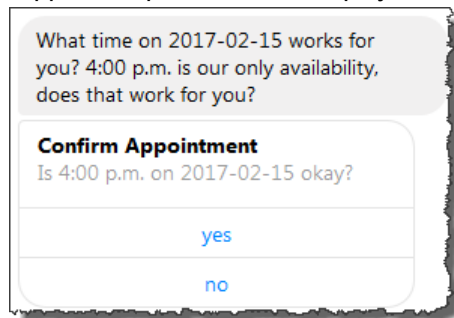
- c. The Lambda function validates the user input and returns the following response to Amazon Lex, directing it to elicit the value for the appointment time.

```
{
  "dialogAction": {
    "slots": {
      "AppointmentType": "root canal",
      "Date": "2017-02-15",
      "Time": "16:00"
    },
    "message": {
      "content": "What time on 2017-02-15 works for you? 4:00 p.m. is our  
only availability, does that work for you?",
      "contentType": "PlainText"
    },
    "type": "ConfirmIntent",
    "intentName": "MakeAppointment",
    "responseCard": {
      "genericAttachments": [
        {
          "buttons": [
            {
              "text": "yes",
              "value": "yes"
            },
            {
              "text": "no",
              "value": "no"
            }
          ]
        },
        {
          "subTitle": "Is 4:00 p.m. on 2017-02-15 okay?",
          "title": "Confirm Appointment"
        }
      ]
    },
    "version": 1,
    "contentType": "application/vnd.amazonaws.card.generic"
  },
  "sessionAttributes": {
    "bookingMap": "{\"2017-02-15\": [\"10:00\", \"16:00\", \"16:30\"]}"
  }
}
```

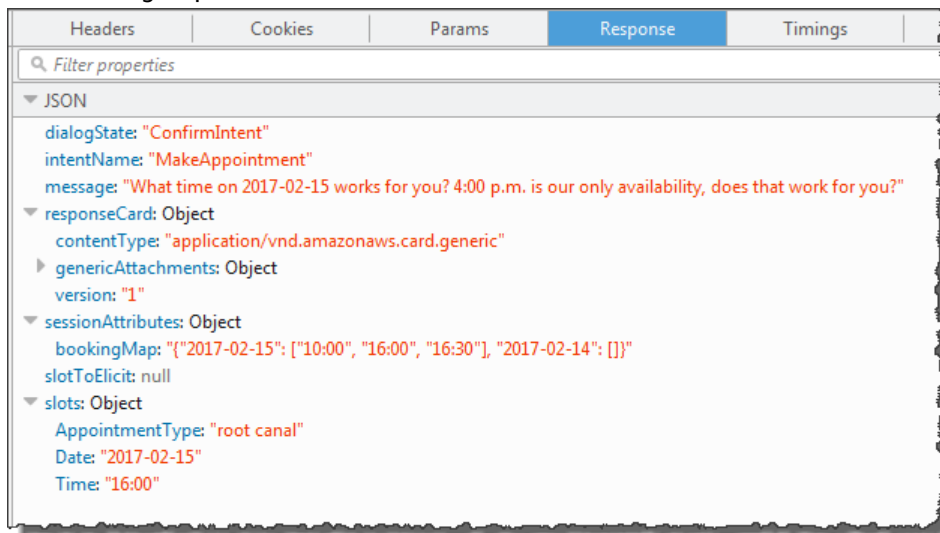
Again, the response includes the `dialogAction` and `sessionAttributes` fields. Among other things, the `dialogAction` returns the following fields:

- `dialogAction` field:

- `type` – The `Lambda` function sets this value to `ConfirmIntent`, directing Amazon Lex to obtain user confirmation of the appointment time suggested in the `message`.
- `responseCard` – Returns a list of yes/no values for the user to choose from. If the client supports response cards, it displays the response card, as shown in the following example:



- `sessionAttributes` – The `Lambda` function sets the `bookingMap` session attribute with its value set to the appointment date and available appointments on that date. In this example, these are 30-minute appointments. For a root canal that requires one hour, only 4 p.m. can be booked.
- d. As indicated in the `dialogAction.type` in the `Lambda` function's response, Amazon Lex returns the following response to the client:



The client displays the message: **What time on 2017-02-15 works for you? 4:00 p.m. is our only availability, does that work for you?**

##### 5. User: Types **yes**.

Amazon Lex invokes the `Lambda` function with the following event data. Because the user replied **yes**, Amazon Lex sets the `confirmationStatus` to `Confirmed`, and sets the `Time` field in `currentIntent.slots` to 4 p.m.

```
{
  "currentIntent": {
    "slots": {
      "AppointmentType": "root canal",
```

```
        "Date": "2017-02-15",
        "Time": "16:00"
      },
      "name": "MakeAppointment",
      "confirmationStatus": "Confirmed"
    },
    "bot": {
      "alias": null,
      "version": "$LATEST",
      "name": "ScheduleAppointment"
    },
    "userId": "u3fpr9gghj02zts7y5tpq5mm4din2xqy",
    "invocationSource": "FulfillmentCodeHook",
    "outputDialogMode": "Text",
    "messageVersion": "1.0",
    "sessionAttributes": {
  }
```

Because the `confirmationStatus` is confirmed, the Lambda function processes the intent (books a dental appointment) and returns the following response to Amazon Lex:

```
{
  "dialogAction": {
    "message": {
      "content": "Okay, I have booked your appointment. We will see you at 4:00 p.m. on 2017-02-15",
      "contentType": "PlainText"
    },
    "type": "Close",
    "fulfillmentState": "Fulfilled"
  },
  "sessionAttributes": {
    "formattedTime": "4:00 p.m.",
    "bookingMap": "{\"2017-02-15\": [\"10:00\"]}"
  }
}
```

Note the following:

- The Lambda function has updated the `sessionAttributes`.
- `dialogAction.type` is set to `close`, which directs Amazon Lex to not expect a user response.
- `dialogAction.fulfillmentState` is set to `Fulfilled`, indicating that the intent is successfully fulfilled.

The client displays the message: **Okay, I have booked your appointment. We will see you at 4:00 p.m. on 2017-02-15.**

## Example Bot: BookTrip

This example illustrates creating a bot that is configured to support multiple intents. The example also illustrates how you can use session attributes for cross-intent information sharing. After creating the bot, you use a test client in the Amazon Lex console to test the bot (BookTrip). The client uses the [PostText \(p. 293\)](#) runtime API operation to send requests to Amazon Lex for each user input.

The BookTrip bot in this example is configured with two intents (BookHotel and BookCar). For example, suppose a user first books a hotel. During the interaction, the user provides information such as check-in dates, location, and number of nights. After the intent is fulfilled, the client can persist this information using session attributes. For more information about session attributes, see [PostText \(p. 293\)](#).

Now suppose that the user continues to book a car. Using information that the user provided in the previous BookHotel intent (that is, destination city, and check-in and check-out dates), the code hook (Lambda function) you configured to initialize and validate the BookCar intent, initializes slot data for the BookCar intent (that is, destination, pick-up city, pick-up date, and return date). This illustrates how cross-intent information sharing enables you to build bots that can engage in dynamic conversation with the user.

In this example, we use the following session attributes. Only the client and the Lambda function can set and update session attributes. Amazon Lex only passes these between the client and the Lambda function. Amazon Lex doesn't maintain or modify any session attributes.

- **currentReservation** – Contains slot data for an in-progress reservation and other relevant information. For example, the following is a sample request from the client to Amazon Lex. It shows the **currentReservation** session attribute in the request body.

```
POST /bot/BookTrip/alias/$LATEST/user/wch89kjqcpkds8seny7dly5x3otq68j3/text
"Content-Type":"application/json"
"Content-Encoding":"amz-1.0"

{
  "inputText":"Chicago",
  "sessionAttributes":{
    "currentReservation":{"ReservationType":"Hotel",
                        "Location":"Moscow",
                        "RoomType":null,
                        "CheckInDate":null,
                        "Nights":null}
  }
}
```

- **lastConfirmedReservation** – Contains similar information for a previous intent, if any. For example, if the user booked a hotel and then is in process of booking a car, this session attribute stores slot data for the previous BookHotel intent.
- **confirmationContext** – The Lambda function sets this to **AutoPopulate** when it prepopulates some of the slot data based on slot data from the previous reservation (if there is one). This enables cross-intent information sharing. For example, if the user previously booked a hotel and now wants to book a car, Amazon Lex can prompt the user to confirm (or deny) that the car is being booked for the same city and dates as their hotel reservation

In this exercise you use blueprints to create an Amazon Lex bot and a Lambda function. For more information about blueprints, see [Amazon Lex and AWS Lambda Blueprints \(p. 91\)](#).

## Next Step

[Step 1: Review the Blueprints Used in this Exercise \(p. 123\)](#)

## Step 1: Review the Blueprints Used in this Exercise

### Topics

- [Overview of the Bot Blueprint \(BookTrip\) \(p. 123\)](#)
- [Overview of the Lambda Function Blueprint \(lex-book-trip-python\) \(p. 124\)](#)

## Overview of the Bot Blueprint (BookTrip)

The blueprint (**BookTrip**) you use to create a bot provides the following preconfiguration:

- **Slot types** – Two custom slot types:
  - `RoomTypes` with enumeration values: `king`, `queen`, and `deluxe`, for use in the `BookHotel` intent.
  - `CarTypes` with enumeration values: `economy`, `standard`, `midsize`, `full size`, `luxury`, and `minivan`, for use in the `CarTypes` intent.
- **Intent 1 (BookHotel)** – It is preconfigured as follows:
  - **Preconfigured slots**
    - `RoomType`, of the `RoomTypes` custom slot type
    - `Location`, of the `AMAZON.US_CITY` built-in slot type
    - `CheckInDate`, of the `AMAZON.DATE` built-in slot type
    - `Nights`, of the `AMAZON.NUMBER` built-in slot type
  - **Preconfigured utterances**
    - "Book a hotel"
    - "I want to make hotel reservations"
    - "Book a {Nights} stay in {Location}"

If the user utters any of these, Amazon Lex determines that `BookHotel` is the intent and then prompts the user for slot data.

- **Preconfigured prompts**
  - Prompt for the `Location` slot – "What city will you be staying in?"
  - Prompt for the `CheckInDate` slot – "What day do you want to check in?"
  - Prompt for the `Nights` slot – "How many nights will you be staying?"
  - Prompt for the `RoomType` slot – "What type of room would you like, queen, king, or deluxe?"
  - Confirmation statement – "Okay, I have you down for a {Nights} night stay in {Location} starting {CheckInDate}. Shall I book the reservation?"
  - Denial – "Okay, I have cancelled your reservation in progress."
- **Intent 2 (BookCar)** – It is preconfigured as follows:
  - **Preconfigured slots**
    - `PickUpCity`, of the `AMAZON.US_CITY` built-in type
    - `PickUpDate4`, of the `AMAZON.DATE` built-in type
    - `ReturnDate`, of the `AMAZON.DATE` built-in type
    - `DriverAge`, of the `AMAZON.NUMBER` built-in type
    - `CarType`, of the `CarTypes` custom type
  - **Preconfigured utterances**
    - "Book a car"



- "Reserve a car"
- "Make a car reservation"

If the user utters any of these, Amazon Lex determines BookCar is the intent and then prompts the user for slot data.

- **Preconfigured prompts**
  - Prompt for the `PickUpCity` slot – "In what city do you need to rent a car?"
  - Prompt for the `PickUpDate` slot – "What day do you want to start your rental?"
  - Prompt for the `ReturnDate` slot – "What day do you want to return this car?"
  - Prompt for the `DriverAge` slot – "How old is the driver for this rental?"
  - Prompt for the `CarType` slot – "What type of car would you like to rent? Our most popular options are economy, midsize, and luxury"
  - Confirmation statement – "Okay, I have you down for a {CarType} rental in {PickUpCity} from {PickUpDate} to {ReturnDate}. Should I book the reservation?"
  - Denial – "Okay, I have cancelled your reservation in progress."

## Overview of the Lambda Function Blueprint (lex-book-trip-python)

In addition to the bot blueprint, AWS Lambda provides a blueprint (**lex-book-trip-python**) that you can use as a code hook with the bot blueprint. For a list of bot blueprints and corresponding Lambda function blueprints, see [Amazon Lex and AWS Lambda Blueprints \(p. 91\)](#).

When you create a bot using the BookTrip blueprint, you update configuration of both the intents (BookCar and BookHotel) by adding this Lambda function as a code hook for both initialization/validation of user data input and fulfillment of the intents.

This Lambda function code provided showcases dynamic conversation using previously known information (persisted in session attributes) about a user to initialize slot values for an intent. For more information, see [Managing Conversation Context \(p. 18\)](#).

### Next Step

[Step 2: Create an Amazon Lex Bot \(p. 124\)](#)

## Step 2: Create an Amazon Lex Bot

In this section, you create an Amazon Lex bot (BookTrip).

1. Sign in to the AWS Management Console and open the Amazon Lex console at <https://console.aws.amazon.com/lex/>.
2. On the **Bots** page, choose **Create**.
3. On the **Create your Lex bot** page,
  - Choose **BookTrip** blueprint.
  - Leave the default bot name (BookTrip).
4. Choose **Create**. The console sends a series of requests to Amazon Lex to create the bot. Note the following:
5. The console shows the BookTrip bot. On the **Editor** tab, review the details of the preconfigured intents (BookCar and BookHotel).
6. Test the bot in the test window. Use the following to engage in a test conversation with your bot:

The screenshot shows a chat interface titled "Test Bot". The conversation is as follows:

- User: Book a hotel
- Bot: What city will you be staying in?
- User: Chicago
- Bot: What day do you want to check in?
- User: December 18th
- Bot: How many nights will you be staying?
- User: 4
- Bot: What type of room would you like, queen, king or deluxe?
- User: Queen
- Bot: Okay, I have you down for a 4 night stay in Chicago starting 2016-12-18. Shall I book the reservation?
- User: Yes
- Bot: CheckInDate:2016-12-18 Location:Chicago Nights:4 RoomType:queen

Below the bot's final message is a "Clear" button and a text input field with the placeholder "Type to your bot...".

From the initial user input ("Book a hotel"), Amazon Lex infers the intent (BookHotel). The bot then uses the prompts preconfigured in this intent to elicit slot data from the user. After user provide all of the slot data, Amazon Lex returns a response back to the client with a message that includes all the user input as a message. The client displays the message in the response as shown.

```
CheckInDate:2016-12-18 Location:Chicago Nights:4 RoomType:queen
```

Now you continue the conversation and try to book a car.

The screenshot shows the 'Test Bot' interface in the Amazon Lex console. The conversation is as follows:

- User: Also book a car
- Bot: In what city do you need to rent a car?
- User: Chicago
- Bot: What day do you want to start your rental?
- User: December 18th
- Bot: What day do you want to return the car?
- User: December 22nd
- Bot: How old is the driver for this rental?
- User: 35
- Bot: What type of car would you like to rent? Our most popular options are economy, midsize, and luxury
- User: economy
- Bot: The price of this economy rental in Chicago from 2016-12-18 to 2016-12-22 is 556 dollars. Shall I book the reservation?
- User: Yes
- Bot: CarType:economy DriverAge:35 PickUpCity:Chicago  
PickUpDate:2016-12-18 ReturnDate:2016-12-22

At the bottom, there is a 'Clear' button and a text input field with the placeholder 'Type to your bot...'.

Note that,

- There is no user data validation at this time. For example, you can provide any city to book a hotel.
- You are providing some of the same information again (destination, pick-up city, pick-up date, and return date) to book a car. In a dynamic conversation, your bot should initialize some of this information based on prior input user provided for booking hotel.

In this next section, you create a Lambda function to do some of the user data validation, and initialization using cross-intent information sharing via session attributes. Then you update the intent configuration by adding the Lambda function as code hook to perform initialization/ validation of user input and fulfill intent.

### Next Step

[Step 3: Create a Lambda function \(p. 127\)](#)

## Step 3: Create a Lambda function

In this section you create a Lambda function using a blueprint (**lex-book-trip-python**) provided in the Amazon Lex console. You also test the Lambda function by invoking it using sample event data provided by the console.

This Lambda function is written in Node.js.

1. Sign in to the AWS Management Console and open the AWS Lambda console at <https://console.aws.amazon.com/lambda/>.
2. Choose **Create a Lambda function**.
3. On **Select blueprint**, type **lex** to find the blueprint, choose the **lex-book-trip-python** blueprint.
4. Configure the Lambda function as follows and then choose **Create Function**.

- Type a Lambda function name (**BookTripCodeHook**).
- For the role, choose **Create a new role from template(s)** and then type a role name.
- Leave the other default values.

5. Test the Lambda function. You invoke the Lambda function twice, using sample data for both booking a car and booking a hotel.

- a. Choose **Actions, Configure test event**.
- b. Choose **Lex-Book Hotel (preview)** from the **Sample event template** list.

This sample event matches the Amazon Lex request/response model. For more information, see [Using Lambda Functions \(p. 85\)](#).

- c. Choose **Save and test**.
- d. Verify that the Lambda function successfully executed. The response in this case matches the Amazon Lex response model.
- e. Repeat the step. This time you choose the **Lex-Book Car (preview)** from the **Sample event template** list. The Lambda function processes the car reservation.

### Next Step

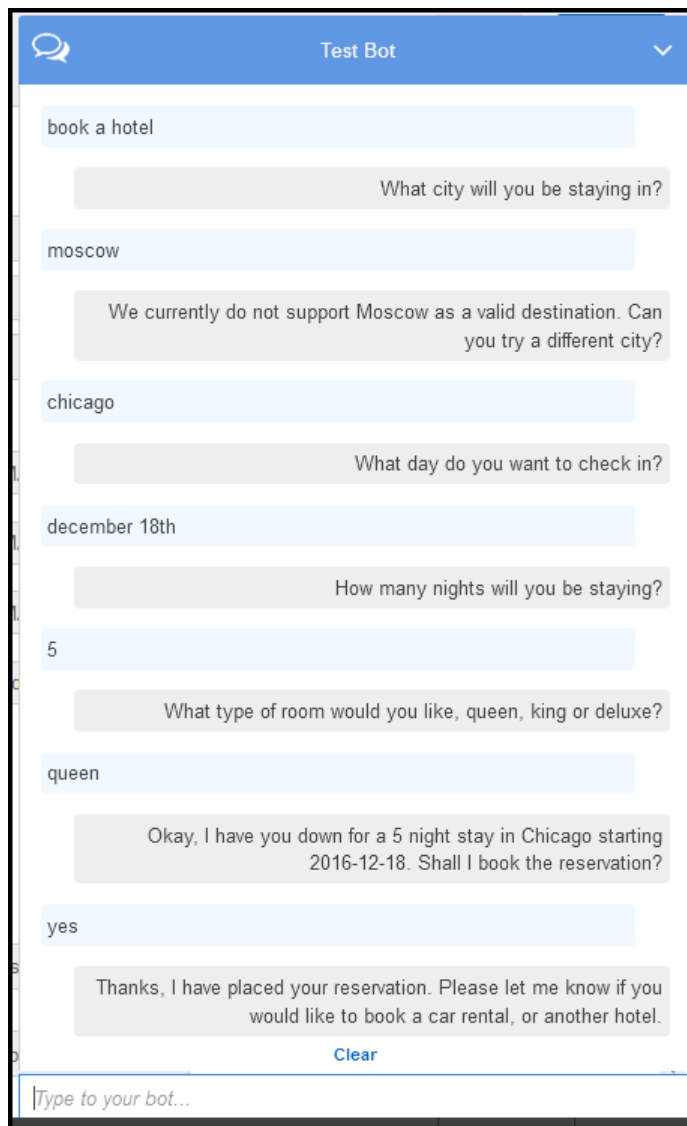
[Step 4: Add the Lambda Function as a Code Hook \(p. 127\)](#)

## Step 4: Add the Lambda Function as a Code Hook

In this section, you update the configurations of both the BookCar and BookHotel intents by adding the Lambda function as a code hook for initialization/validation and fulfillment activities. Make sure you choose the \$LATEST version of the intents because you can only update the \$LATEST version of your Amazon Lex resources.

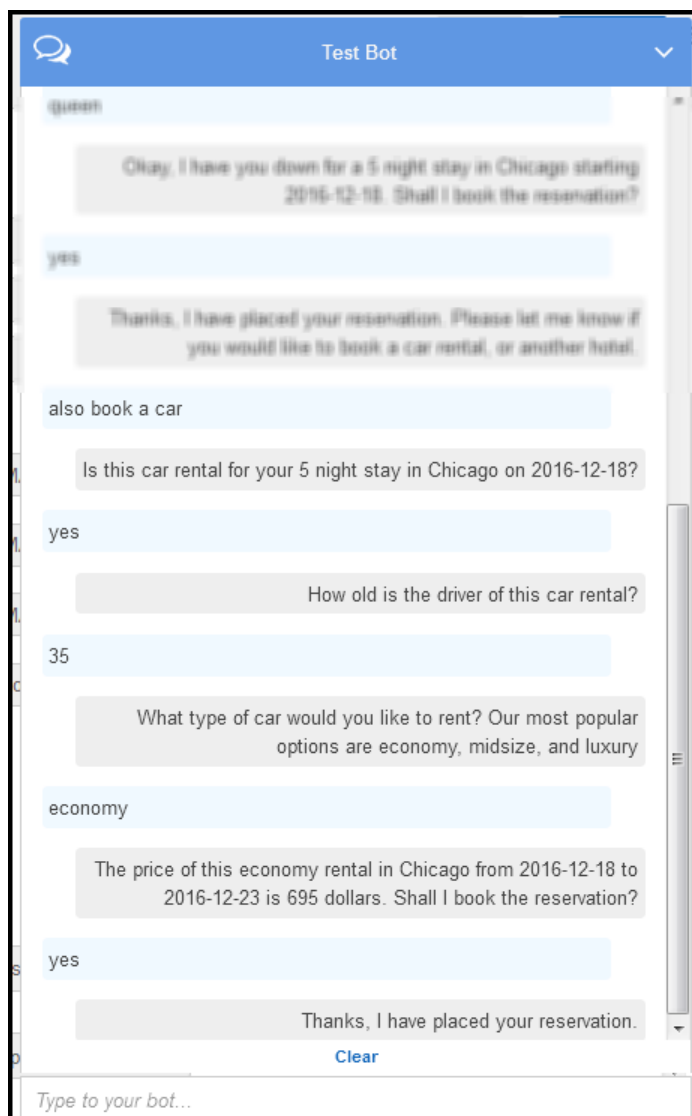
1. In the Amazon Lex console, choose the **BookTrip** bot.
2. On the **Editor** tab, choose the **BookHotel** intent. Update the intent configuration as follows:
  - a. Make sure the intent version (next to the intent name) is \$LATEST.
  - b. Add the Lambda function as an initialization and validation code hook as follows:
    - In **Options**, choose **Initialization and validation code hook**.
    - Choose your Lambda function from the list.

- c. Add the Lambda function as a fulfillment code hook as follows:
  - In **Fulfillment**, choose **AWS Lambda function**.
  - Choose your Lambda function from the list.
  - Choose **Goodbye message** and type a message.
- d. Choose **Save**.
3. On the **Editor** tab, choose the BookCar intent. Follow the preceding step to add your Lambda function as validation and fulfillment code hook.
4. Choose **Build**. The console sends a series of requests to Amazon Lex to save the configurations.
5. Test the bot. Now that you have a Lambda function performing the initialization, user data validation and fulfillment, you can see the difference in the user interaction.



For more information about the data flow from the client (console) to Amazon Lex, and from Amazon Lex to the Lambda function, see [Data Flow: Book Hotel Intent \(p. 130\)](#).

6. Continue the conversation and book a car as shown following:



When you choose to book a car, the client (console) sends a request to Amazon Lex that includes the session attributes (from the previous conversation, BookHotel). Amazon Lex passes this information to the Lambda function, which then initializes (that is, it prepopulates) some of the BookCar slot data (that is, PickUpDate, ReturnDate, and PickUpCity).

#### Note

This illustrates how session attributes can be used to maintain context across intents. The console client provides the **Clear** link in the test window that a user can use to clear any prior session attributes.

For more information about the data flow from the client (console) to Amazon Lex, and from Amazon Lex to the Lambda function, see [Data Flow: Book Car Intent \(p. 138\)](#).

## Details of the Information Flow

In this exercise, you engaged in a conversation with the Amazon Lex BookTrip bot using the test window client provided in the Amazon Lex console. This section explains the following:

- The data flow between the client and Amazon Lex.

The section assumes that the client sends requests to Amazon Lex using the `PostText` runtime API and shows request and response details accordingly. For more information about the `PostText` runtime API, see [PostText \(p. 293\)](#).

**Note**

For an example of the information flow between the client and Amazon Lex in which the client uses the `PostContent` API, see [Step 2a \(Optional\): Review the Details of the Spoken Information Flow \(Console\) \(p. 27\)](#).

- The data flow between Amazon Lex and the Lambda function. For more information, see [Lambda Function Input Event and Response Format \(p. 85\)](#).

Topics

- [Data Flow: Book Hotel Intent \(p. 130\)](#)
- [Data Flow: Book Car Intent \(p. 138\)](#)

## Data Flow: Book Hotel Intent

This section explains what happens after each user input.

1. User: "book a hotel"
  - a. The client (console) sends the following [PostText \(p. 293\)](#) request to Amazon Lex:

```
POST /bot/BookTrip/alias/$LATEST/user/wch89kjqcpkds8seny7dly5x3otq68j3/text
"Content-Type":"application/json"
"Content-Encoding":"amz-1.0"

{
  "inputText":"book a hotel",
  "sessionAttributes":{}
}
```

Both the request URI and the body provides information to Amazon Lex:

- Request URI – Provides bot name (*BookTrip*), bot alias (*\$LATEST*) and the user name. The trailing *text* indicates that it is a `PostText` API request (and not `PostContent`).
  - Request body – Includes the user input (*inputText*) and empty *sessionAttributes*. Initially, this is an empty object and the Lambda function first sets the session attributes.
- b. From the *inputText*, Amazon Lex detects the intent (*BookHotel*). This intent is configured with a Lambda function as a code hook for user data initialization/validation. Therefore, Amazon Lex invokes that Lambda function by passing the following information as the event parameter (see [Input Event Format \(p. 85\)](#)):

```
{
  "messageVersion":"1.0",
  "invocationSource":"DialogCodeHook",
  "userId":"wch89kjqcpkds8seny7dly5x3otq68j3",
  "sessionAttributes":{
  },
}
```

```
"bot":{
  "name":"BookTrip",
  "alias":null,
  "version":"$LATEST"
},
"outputDialogMode":"Text",
"currentIntent":{
  "name":"BookHotel",
  "slots":{
    "RoomType":null,
    "CheckInDate":null,
    "Nights":null,
    "Location":null
  },
  "confirmationStatus":"None"
}
}
```

In addition to the information sent by the client, Amazon Lex also includes the following additional data:

- `messageVersion` – Currently Amazon Lex supports only the 1.0 version.
  - `invocationSource` – Indicates the purpose of Lambda function invocation. In this case, it is to perform user data initialization and validation (at this time Amazon Lex knows that the user has not provided all the slot data to fulfill the intent).
  - `currentIntent` – All of the slot values are set to null.
- c. At this time, all the slot values are null. There is nothing for the Lambda function to validate. The Lambda function returns the following response to Amazon Lex. For information about response format, see [Response Format \(p. 88\)](#).

```
{
  "sessionAttributes":{
    "currentReservation":{"\"ReservationType\": \"Hotel\", \"Location\": null,
    \"RoomType\": null, \"CheckInDate\": null, \"Nights\": null}"
  },
  "dialogAction":{
    "type":"Delegate",
    "slots":{
      "RoomType":null,
      "CheckInDate":null,
      "Nights":null,
      "Location":null
    }
  }
}
```

### Note

- `currentReservation` – The Lambda function includes this session attribute. Its value is a copy of the current slot information and the reservation type.

Only the Lambda function and the client can update these session attributes. Amazon Lex simply passes these values.

- `dialogAction.type` – By setting this value to `Delegate`, the Lambda function delegates the responsibility for the next course of action to Amazon Lex.

If the Lambda function detected anything in the user data validation, it instructs Amazon Lex what to do next.



- d. As per the `dialogAction.type`, Amazon Lex decides the next course of action—elicit data from the user for the `Location` slot. It selects one of the prompt messages ("What city will you be staying in?") for this slot, according to the intent configuration, and then sends the following response to the user:



The session attributes are passed to the client.

The client reads the response and then displays the message: "What city will you be staying in?"

2. User: "Moscow"
- a. The client sends the following `PostText` request to Amazon Lex (line breaks added for readability):

```
POST /bot/BookTrip/alias/$LATEST/user/wch89kjqcpkds8seny7dly5x3otq68j3/text
"Content-Type": "application/json"
"Content-Encoding": "amz-1.0"

{
  "inputText": "Moscow",
  "sessionAttributes": {
    "currentReservation": {
      "ReservationType": "Hotel",
      "Location": null,
      "RoomType": null,
      "CheckInDate": null,
      "Nights": null
    }
  }
}
```

In addition to the `inputText`, the client includes the same `currentReservation` session attributes it received.

- b. Amazon Lex first interprets the `inputText` in the context of the current intent (the service remembers that it had asked the specific user for information about `Location` slot). It updates the slot value for the current intent and invokes the Lambda function using the following event:

```
{
  "messageVersion": "1.0",
  "invocationSource": "DialogCodeHook",
  "userId": "wch89kjqcpkds8seny7dly5x3otq68j3",
  "sessionAttributes": {
```

```
    "currentReservation": "{\\"ReservationType\\":\\"Hotel\\",\\"Location\\":null,\\"RoomType\\":null,\\"CheckInDate\\":null,\\"Nights\\":null}"
  },
  "bot": {
    "name": "BookTrip",
    "alias": null,
    "version": "$LATEST"
  },
  "outputDialogMode": "Text",
  "currentIntent": {
    "name": "BookHotel",
    "slots": {
      "RoomType": null,
      "CheckInDate": null,
      "Nights": null,
      "Location": "Moscow"
    },
    "confirmationStatus": "None"
  }
}
```

### Note

- `invocationSource` continues to be `DialogCodeHook`. In this step, we are just validating user data.
  - Amazon Lex is just passing the session attribute to the Lambda function.
  - For `currentIntent.slots`, Amazon Lex has updated the `Location` slot to `Moscow`.
- c. The Lambda function performs the user data validation and determines that `Moscow` is an invalid location.

### Note

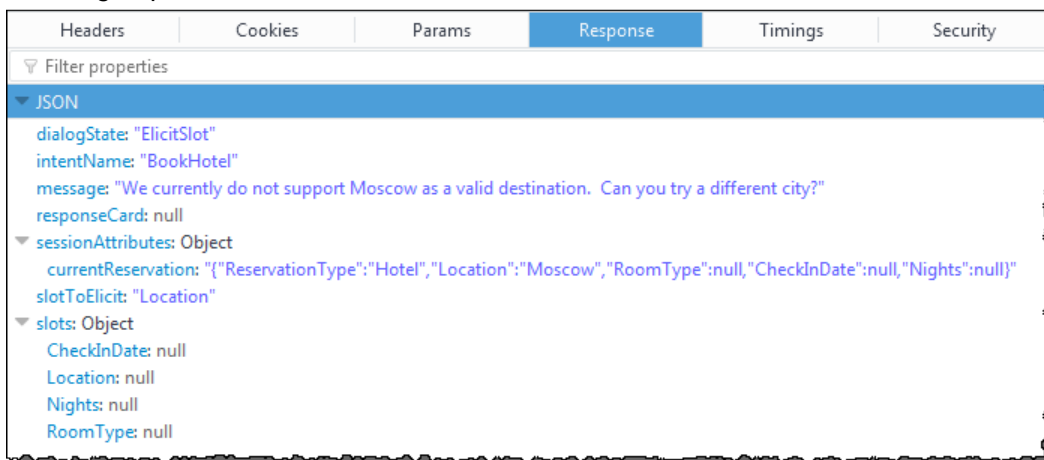
The Lambda function in this exercise has a simple list of valid cities and `Moscow` is not on the list. In a production application, you might use a back-end database to get this information.

It resets the slot value back to null and directs Amazon Lex to prompt the user again for another value by sending the following response:

```
{
  "sessionAttributes": {
    "currentReservation": "{\\"ReservationType\\":\\"Hotel\\",\\"Location\\":\\"Moscow\\",\\"RoomType\\":null,\\"CheckInDate\\":null,\\"Nights\\":null}"
  },
  "dialogAction": {
    "type": "ElicitSlot",
    "intentName": "BookHotel",
    "slots": {
      "RoomType": null,
      "CheckInDate": null,
      "Nights": null,
      "Location": null
    },
    "slotToElicit": "Location",
    "message": {
      "contentType": "PlainText",
      "content": "We currently do not support Moscow as a valid destination. Can you try a different city?"
    }
  }
}
```

## Note

- `currentIntent.slots.Location` is reset to null.
  - `dialogAction.type` is set to `ElicitSlot`, which directs Amazon Lex to prompt the user again by providing the following:
    - `dialogAction.slotToElicit` – slot for which to elicit data from the user.
    - `dialogAction.message` – a message to convey to the user.
- d. Amazon Lex notices the `dialogAction.type` and passes the information to the client in the following response:



The client simply displays the message: "We currently do not support Moscow as a valid destination. Can you try a different city?"

### 3. User: "Chicago"

- a. The client sends the following `PostText` request to Amazon Lex:

```
POST /bot/BookTrip/alias/$LATEST/user/wch89kjqcpkds8seny7dly5x3otq68j3/text
"Content-Type": "application/json"
"Content-Encoding": "amz-1.0"

{
  "inputText": "Chicago",
  "sessionAttributes": {
    "currentReservation": {
      "ReservationType": "Hotel",
      "Location": "Moscow",
      "RoomType": null,
      "CheckInDate": null,
      "Nights": null
    }
  }
}
```

- b. Amazon Lex knows the context, that it was eliciting data for the `Location` slot. In this context, it knows the `inputText` value is for the `Location` slot. It then invokes the Lambda function by sending the following event:

```
{
  "messageVersion": "1.0",
  "invocationSource": "DialogCodeHook",
  "userId": "wch89kjqcpkds8seny7dly5x3otq68j3",
  "sessionAttributes": {
```

```
    "currentReservation": "{\\"ReservationType\\":\\"Hotel\\",\\"Location\\":Moscow,\\n\\"RoomType\\":null,\\"CheckInDate\\":null,\\"Nights\\":null}"
  },
  "bot": {
    "name": "BookTrip",
    "alias": null,
    "version": "$LATEST"
  },
  "outputDialogMode": "Text",
  "currentIntent": {
    "name": "BookHotel",
    "slots": {
      "RoomType": null,
      "CheckInDate": null,
      "Nights": null,
      "Location": "Chicago"
    },
    "confirmationStatus": "None"
  }
}
```

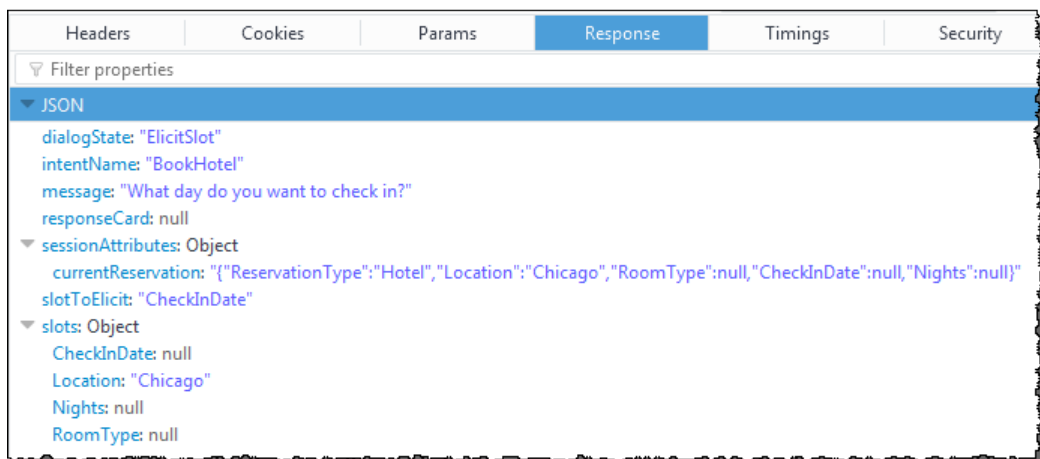
Amazon Lex updated the `currentIntent.slots` by setting the `Location` slot to `Chicago`.

- c. According to the `invocationSource` value of `DialogCodeHook`, the Lambda function performs user data validation. It recognizes `Chicago` as a valid slot value, updates the session attribute accordingly, and then returns the following response to Amazon Lex.

```
{
  "sessionAttributes": {
    "currentReservation": "{\\"ReservationType\\":\\"Hotel\\",\\"Location\\":\\n\\"Chicago\\",\\"RoomType\\":null,\\"CheckInDate\\":null,\\"Nights\\":null}"
  },
  "dialogAction": {
    "type": "Delegate",
    "slots": {
      "RoomType": null,
      "CheckInDate": null,
      "Nights": null,
      "Location": "Chicago"
    }
  }
}
```

### Note

- `currentReservation` – The Lambda function updates this session attribute by setting the `Location` to `Chicago`.
  - `dialogAction.type` – Is set to `Delegate`. User data was valid, and the Lambda function directs Amazon Lex to choose the next course of action.
- d. According to `dialogAction.type`, Amazon Lex chooses the next course of action. Amazon Lex knows that it needs more slot data and picks the next unfilled slot (`CheckInDate`) with the highest priority according to the intent configuration. It selects one of the prompt messages ("What day do you want to check in?") for this slot according to the intent configuration and then sends the following response back to the client:



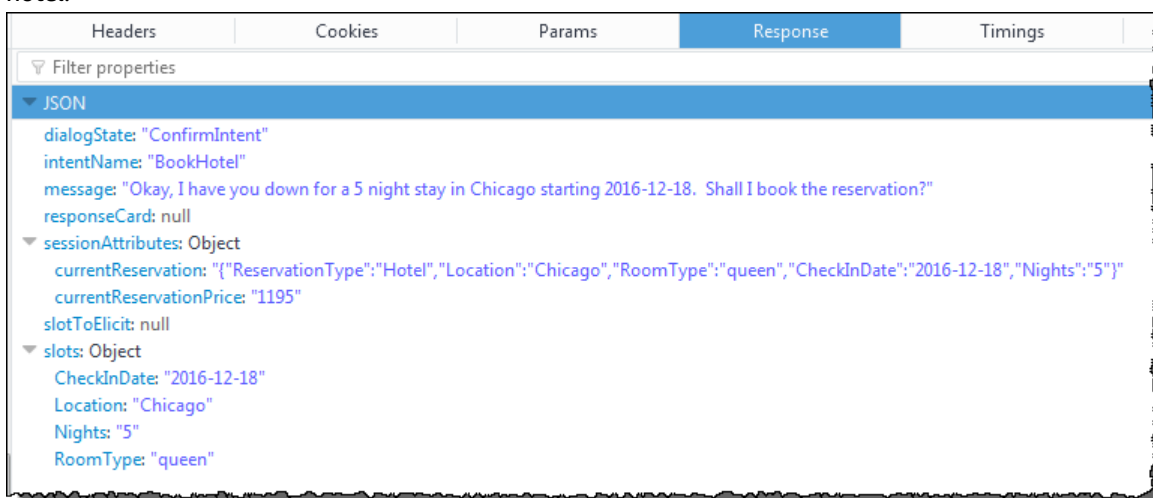
The client displays the message: "What day do you want to check in?"

4. The user interaction continues—the user provides data, the Lambda function validates data, and then delegates the next course of action to Amazon Lex. Eventually the user provides all of the slot data, the Lambda function validates all of the user input, and then Amazon Lex recognizes it has all the slot data.

**Note**

In this exercise, after the user provides all of the slot data, the Lambda function computes the price of the hotel reservation and returns it as another session attribute (`currentReservationPrice`).

At this point, the intent is ready to be fulfilled, but the `BookHotel` intent is configured with a confirmation prompt requiring user confirmation before Amazon Lex can fulfill the intent. Therefore, Amazon Lex sends the following message to the client requesting confirmation before booking the hotel:



The client display the message: "Okay, I have you down for a 5 night in Chicago starting 2016-12-18. Shall I book the reservation?"

5. User: "yes"
  - a. The client sends the following `PostText` request to Amazon Lex:

```
POST /bot/BookTrip/alias/$LATEST/user/wch89kjqcpkds8seny7dly5x3otq68j3/text
"Content-Type":"application/json"
"Content-Encoding":"amz-1.0"

{
  "inputText":"Yes",
  "sessionAttributes":{
    "currentReservation":{"ReservationType":"Hotel",
      "Location":"Chicago",
      "RoomType":"queen",
      "CheckInDate":"2016-12-18",
      "Nights":"5"},
    "currentReservationPrice":"1195"
  }
}
```

- b. Amazon Lex interprets the `inputText` in the context of confirming the current intent. Amazon Lex understands that the user wants to proceed with the reservation. This time Amazon Lex invokes the Lambda function to fulfill the intent by sending the following event. By setting the `invocationSource` to `FulfillmentCodeHook` in the event, it sends to the Lambda function. Amazon Lex also sets the `confirmationStatus` to `Confirmed`.

```
{
  "messageVersion": "1.0",
  "invocationSource": "FulfillmentCodeHook",
  "userId": "wch89kjqcpkds8seny7dly5x3otq68j3",
  "sessionAttributes": {
    "currentReservation": {"ReservationType":"Hotel","Location":
    \"Chicago\", \"RoomType\":\"queen\", \"CheckInDate\":\"2016-12-18\", \"Nights\":
    \"4\""},
    "currentReservationPrice": "956"
  },
  "bot": {
    "name": "BookTrip",
    "alias": null,
    "version": "$LATEST"
  },
  "outputDialogMode": "Text",
  "currentIntent": {
    "name": "BookHotel",
    "slots": {
      "RoomType": "queen",
      "CheckInDate": "2016-12-18",
      "Nights": "4",
      "Location": "Chicago"
    },
    "confirmationStatus": "Confirmed"
  }
}
```

### Note

- `invocationSource` – This time, Amazon Lex set this value to `FulfillmentCodeHook`, directing the Lambda function to fulfill the intent.
  - `confirmationStatus` – Is set to `Confirmed`.
- c. This time, the Lambda function fulfills the `BookHotel` intent, Amazon Lex completes the reservation, and then it returns the following response:

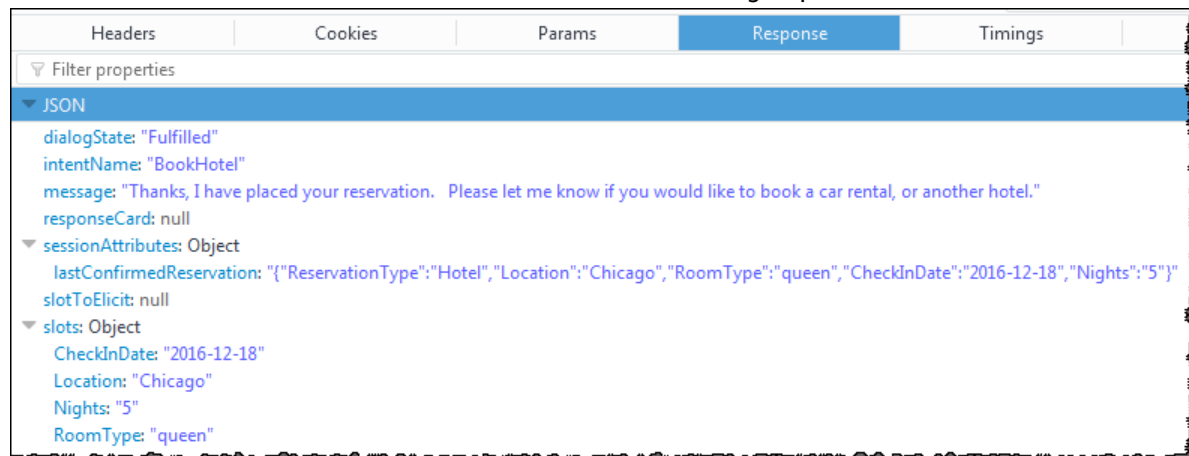
```
{
  "sessionAttributes": {
```

```
      "lastConfirmedReservation": "{\\"ReservationType\\":\\"Hotel\\",\\"Location\\":\\"Chicago\\",\\"RoomType\\":\\"queen\\",\\"CheckInDate\\":\\"2016-12-18\\",\\"Nights\\":\\"4\\"}",
    },
    "dialogAction": {
      "type": "Close",
      "fulfillmentState": "Fulfilled",
      "message": {
        "contentType": "PlainText",
        "content": "Thanks, I have placed your reservation.  Please let me know if you would like to book a car rental, or another hotel."
      }
    }
  }
}
```

### Note

- `lastConfirmedReservation` – Is a new session attribute that the Lambda function added (instead of the `currentReservation`, `currentReservationPrice`).
- `dialogAction.type` – The Lambda function sets this value to `close`, indicating that Amazon Lex to not expect a user response.
- `dialogAction.fulfillmentState` – Is set to `Fulfilled` and includes an appropriate message to convey to the user.

- d. Amazon Lex reviews the `fulfillmentState` and sends the following response to the client:



### Note

- `dialogState` – Amazon Lex sets this value to `Fulfilled`.
- `message` – Is the same message that the Lambda function provided.

The client displays the message.

## Data Flow: Book Car Intent

The BookTrip bot in this exercise supports two intents (BookHotel and BookCar). After booking a hotel, the user can continue the conversation to book a car. As long as the session hasn't timed out, in each subsequent request the client continues to send the session attributes (in this example, the

`lastConfirmedReservation`). The Lambda function can use this information to initialize slot data for the BookCar intent. This shows how you can use session attributes in cross-intent data sharing.

Specifically, when the user chooses the BookCar intent, the Lambda function uses relevant information in the session attribute to prepopulate slots (PickUpDate, ReturnDate, and PickUpCity) for the BookCar intent.

**Note**

The Amazon Lex console provides the **Clear** link that you can use to clear any prior session attributes.

Follow the steps in this procedure to continue the conversation.

1. User: "also book a car"
  - a. The client sends the following `PostText` request to Amazon Lex.

```
POST /bot/BookTrip/alias/$LATEST/user/wch89kjqcpkds8seny7dly5x3otq68j3/text
"Content-Type":"application/json"
"Content-Encoding":"amz-1.0"

{
  "inputText":"also book a car",
  "sessionAttributes":{"
    "lastConfirmedReservation":{"\ReservationType\":"Hotel\","
                                \Location\":"Chicago\","
                                \RoomType\":"queen\","
                                \CheckInDate\":"2016-12-18\","
                                \Nights\":"5\"}
  }
}
```

The client includes the `lastConfirmedReservation` session attribute.

- b. Amazon Lex detects the intent (BookCar) from the `inputText`. This intent is also configured to invoke the Lambda function to perform the initialization and validation of the user data. Amazon Lex invokes the Lambda function with the following event:

```
{
  "messageVersion": "1.0",
  "invocationSource": "DialogCodeHook",
  "userId": "wch89kjqcpkds8seny7dly5x3otq68j3",
  "sessionAttributes": {
    "lastConfirmedReservation": "{\ReservationType\":"Hotel\","Location
\":"Chicago\","RoomType\":"queen\","CheckInDate\":"2016-12-18\","Nights\":"4\"}
  },
  "bot": {
    "name": "BookTrip",
    "alias": null,
    "version": "$LATEST"
  },
  "outputDialogMode": "Text",
  "currentIntent": {
    "name": "BookCar",
    "slots": {
      "PickUpDate": null,
      "ReturnDate": null,
      "DriverAge": null,
      "CarType": null,
      "PickUpCity": null
    }
  },
  "confirmationStatus": "None"
}
```



```
}  
}
```

### Note

- `messageVersion` – Currently Amazon Lex supports the 1.0 version only.
  - `invocationSource` – Indicates the purpose of invocation is to perform initialization and user data validation.
  - `currentIntent` – It includes the intent name and the slots. At this time, all slot values are null.
- c. The Lambda function notices all null slot values with nothing to validate. However, it uses session attributes to initialize some of the slot values (`PickUpDate`, `ReturnDate`, and `PickUpCity`), and then returns the following response:

```
{  
  "sessionAttributes": {  
    "lastConfirmedReservation": "{\"ReservationType\": \"Hotel\", \"Location\": \"Chicago\", \"RoomType\": \"queen\", \"CheckInDate\": \"2016-12-18\", \"Nights\": \"4\"}\",  
    "currentReservation": "{\"ReservationType\": \"Car\", \"PickUpCity\": null, \"PickUpDate\": null, \"ReturnDate\": null, \"CarType\": null}\",  
    "confirmationContext": "AutoPopulate"  
  },  
  "dialogAction": {  
    "type": "ConfirmIntent",  
    "intentName": "BookCar",  
    "slots": {  
      "PickUpCity": "Chicago",  
      "PickUpDate": "2016-12-18",  
      "ReturnDate": "2016-12-22",  
      "CarType": null,  
      "DriverAge": null  
    },  
    "message": {  
      "contentType": "PlainText",  
      "content": "Is this car rental for your 4 night stay in Chicago on 2016-12-18?"  
    }  
  }  
}
```

### Note

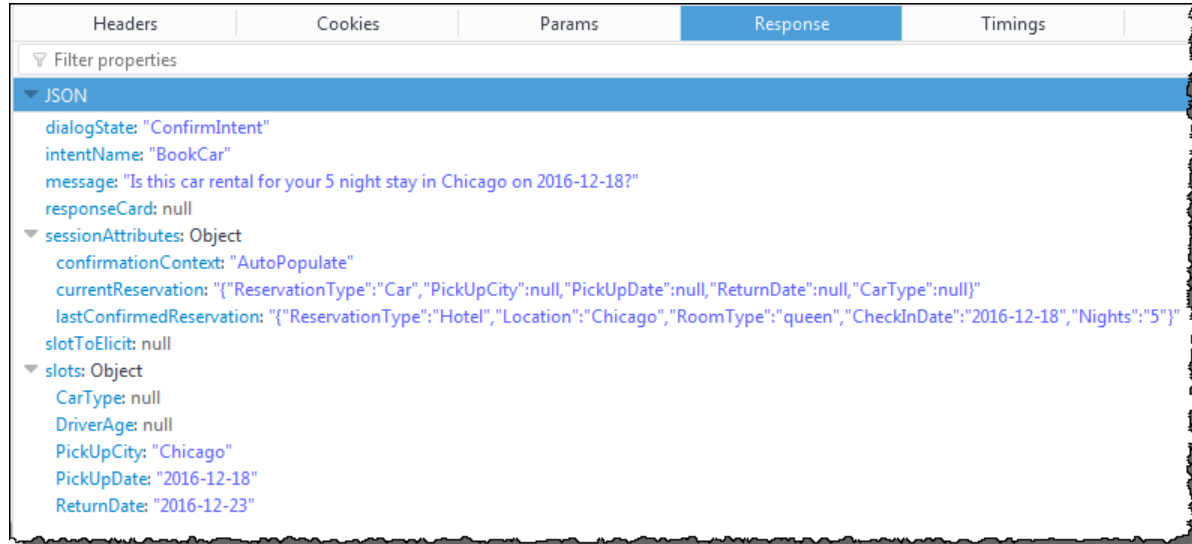
- In addition to the `lastConfirmedReservation`, the Lambda function includes more session attributes (`currentReservation` and `confirmationContext`).
- `dialogAction.type` is set to `ConfirmIntent`, which informs Amazon Lex that a yes, no reply is expected from the user (the `confirmationContext` set to `AutoPopulate`, the Lambda function knows that the yes/no user reply is to obtain user confirmation of the initialization the Lambda function performed (auto populated slot data).

The Lambda function also includes in the response an informative message in the `dialogAction.message` for Amazon Lex to return to the client.

### Note

The term `ConfirmIntent` (value of the `dialogAction.type`) is not related to any bot intent. In the example, Lambda function uses this term to direct Amazon Lex to get a yes/no reply from the user.

- d. According to the `dialogAction.type`, Amazon Lex returns the following response to the client:



The client displays the message: "Is this car rental for your 5 night stay in Chicago on 2016-12-18?"

2. User: "yes"

- a. The client sends the following `PostText` request to Amazon Lex.

```
POST /bot/BookTrip/alias/$LATEST/user/wch89kjcpkds8seny7dly5x3otq68j3/text
"Content-Type": "application/json"
"Content-Encoding": "amz-1.0"

{
  "inputText": "yes",
  "sessionAttributes": {
    "confirmationContext": "AutoPopulate",
    "currentReservation": {
      "ReservationType": "Car",
      "PickUpCity": null,
      "PickUpDate": null,
      "ReturnDate": null,
      "CarType": null
    },
    "lastConfirmedReservation": {
      "ReservationType": "Hotel",
      "Location": "Chicago",
      "RoomType": "queen",
      "CheckInDate": "2016-12-18",
      "Nights": "5"
    }
  }
}
```

- b. Amazon Lex reads the `inputText` and it knows the context (asked the user to confirm the auto population). Amazon Lex invokes the Lambda function by sending the following event:

```
{
  "messageVersion": "1.0",
  "invocationSource": "DialogCodeHook",
}
```

```
"userId": "wch89kjqcpkds8seny7dly5x3otq68j3",
"sessionAttributes": {
  "confirmationContext": "AutoPopulate",
  "currentReservation": "{\"ReservationType\":\"Car\", \"PickUpCity\":null,
  \"PickUpDate\":null, \"ReturnDate\":null, \"CarType\":null}\",
  "lastConfirmedReservation": "{\"ReservationType\":\"Hotel\", \"Location
  \": \"Chicago\", \"RoomType\": \"queen\", \"CheckInDate\": \"2016-12-18\", \"Nights\":
  \"4\"}"
},
"bot": {
  "name": "BookTrip",
  "alias": null,
  "version": "$LATEST"
},
"outputDialogMode": "Text",
"currentIntent": {
  "name": "BookCar",
  "slots": {
    "PickUpDate": "2016-12-18",
    "ReturnDate": "2016-12-22",
    "DriverAge": null,
    "CarType": null,
    "PickUpCity": "Chicago"
  },
  "confirmationStatus": "Confirmed"
}
}
```

Because the user replied Yes, Amazon Lex sets the `confirmationStatus` to `Confirmed`.

- c. From the `confirmationStatus`, the Lambda function knows that the prepopulated values are correct. The Lambda function does the following:
- Updates the `currentReservation` session attribute to slot value it had prepopulated.
  - Sets the `dialogAction.type` to `ElicitSlot`
  - Sets the `slotToElicit` value to `DriverAge`.

The following response is sent:

```
{
  "sessionAttributes": {
    "currentReservation": "{\"ReservationType\":\"Car\", \"PickUpCity\":
    \"Chicago\", \"PickUpDate\": \"2016-12-18\", \"ReturnDate\": \"2016-12-22\", \"CarType
    \":null}\",
    "lastConfirmedReservation": "{\"ReservationType\":\"Hotel\", \"Location
    \": \"Chicago\", \"RoomType\": \"queen\", \"CheckInDate\": \"2016-12-18\", \"Nights\":
    \"4\"}"
  },
  "dialogAction": {
    "type": "ElicitSlot",
    "intentName": "BookCar",
    "slots": {
      "PickUpDate": "2016-12-18",
      "ReturnDate": "2016-12-22",
      "DriverAge": null,
      "CarType": null,
      "PickUpCity": "Chicago"
    },
    "slotToElicit": "DriverAge",
    "message": {
      "contentType": "PlainText",
      "content": "How old is the driver of this car rental?"
    }
  }
}
```

```
}  
  }  
}
```

- d. Amazon Lex returns following response:

Headers	Cookies	Params	Response	Timings	Se
Filter properties					
JSON					
dialogState: "ElicitSlot"					
intentName: "BookCar"					
message: "How old is the driver of this car rental?"					
responseCard: null					
sessionAttributes: Object					
currentReservation: {"ReservationType": "Car", "PickUpCity": "Chicago", "PickUpDate": "2016-12-18", "ReturnDate": "2016-12-23", "CarType": null}					
lastConfirmedReservation: {"ReservationType": "Hotel", "Location": "Chicago", "RoomType": "queen", "CheckInDate": "2016-12-18", "Nights": "5"}					
slotToElicit: "DriverAge"					
slots: Object					
CarType: null					
DriverAge: null					
PickUpCity: "Chicago"					
PickUpDate: "2016-12-18"					
ReturnDate: "2016-12-23"					

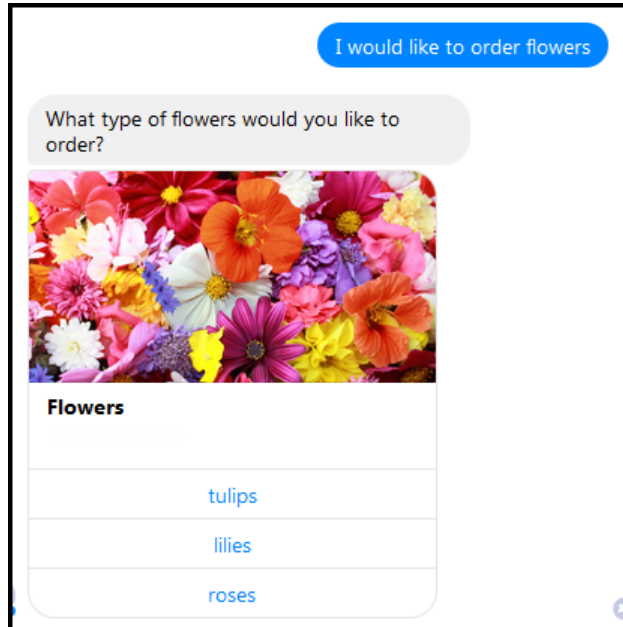
The client displays the message "How old is the driver of this car rental?" and the conversation continues.

## Example: Using a Response Card

In this exercise, you extend Getting Started Exercise 1 by adding a response card. You create a bot that supports the OrderFlowers intent, and then update the intent by adding a response card for the `FlowerType` slot. In addition to the following prompt for the `FlowerType` slot, the user can choose the type of flowers from the response card:

What type of flowers would you like to order?

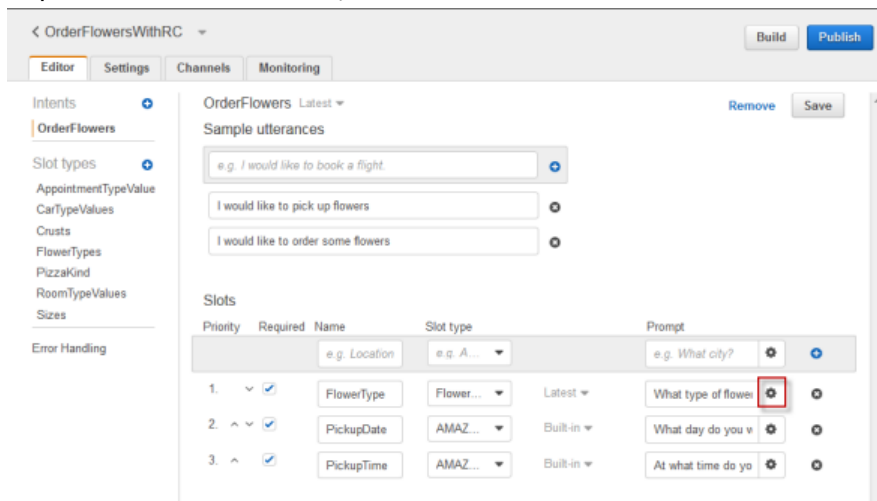
The following is the response card:



The bot user can either type the text or choose from the list of flower types. This response card is configured with an image, which appears in the client as shown. For more information about response cards, see [Response Cards \(p. 14\)](#).

To create and test a bot with a response card:

1. Follow Getting Started Exercise 1 to create and test an OrderFlowers bot. You must complete steps 1, 2, and 3. You don't need to add a Lambda function to test the response card. For instructions, see [Exercise 1: Create an Amazon Lex Bot Using a Blueprint \(Console\) \(p. 24\)](#).
2. Update the bot by adding the response card, and then publish a version. When you publish a version, specify an alias (BETA) to point to it.
  - a. In the Amazon Lex console, choose your bot.
  - b. Choose the OrderFlowers intent.
  - c. Choose the settings gear icon next to the "What type of flowers" **Prompt** to configure a response card for the FlowerType.



- d. Give the card a title and configure three buttons as shown in the following screen shot. You can optionally add an image to the response card, provided you have an image URL.

The screenshot shows the 'FlowerType Prompts' configuration page in the Amazon Lex console. The page is divided into several sections: 'Prompts', 'Corresponding utterances', and 'Prompt response cards'. The 'Prompt response cards' section is highlighted with a red box, showing a card with a title 'What type of flowers?' and three buttons labeled 'Tulips', 'Lilies', and 'Roses'. The 'Preview' button is also highlighted with a red box. The card has a subtitle 'e.g. Card subtitle text' and a 'Facebook' platform selected. The buttons have values 'Tulips', 'Lilies', and 'Roses' and titles 'e.g. Button title'.

- e. Verify the button values in the **Preview**, and then choose **Save**.
- f. On the **Editor** tab, choose **Save** to save the intent configuration.
- g. To build the bot, choose **Build**.
- h. To publish a bot version, choose **Publish**. Specify BETA as an alias that points to the bot version. For information about versioning, see [Versioning and Aliases \(p. 81\)](#).
3. Deploy the bot on the Facebook Messenger platform and test the integration. For instructions, see [Integrating an Amazon Lex Bot with Facebook Messenger \(p. 94\)](#).

When you order flowers, the message window shows the response card so you can choose a flower type.

## Example: Updating Utterances

In this exercise, you add additional utterances to those you created in Getting Started Exercise 1. You use the **Monitoring** tab in the Amazon Lex console to view utterances that your bot did not recognize. To improve the experience for your users, you add those utterances to the bot.

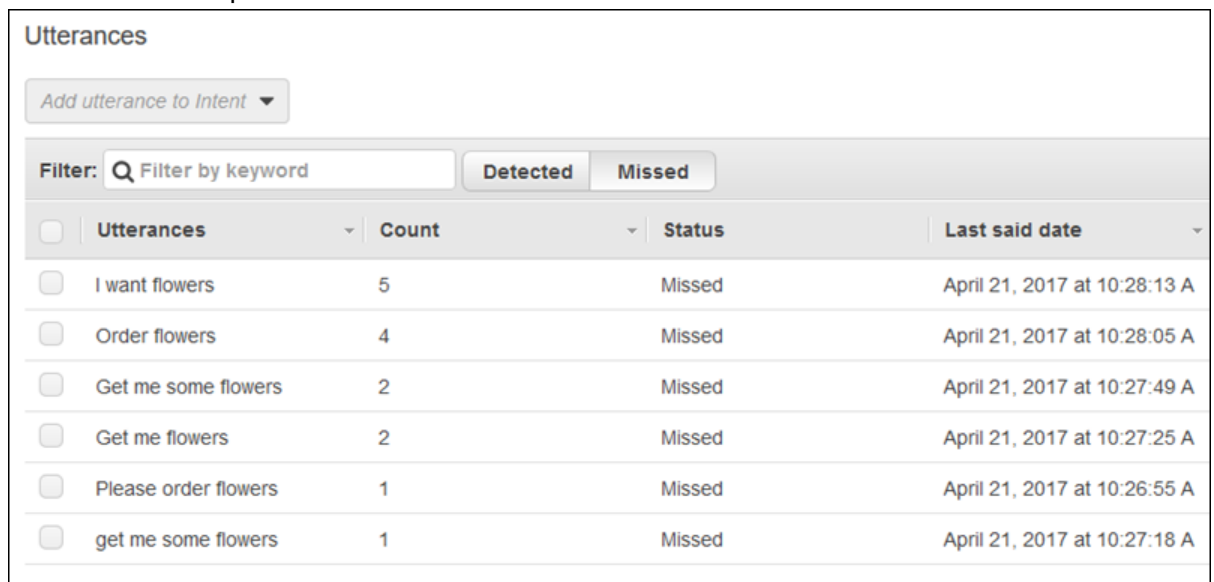
### Note

Utterance statistics are generated once a day, generally in the evening. You can see the utterance that was not recognized, how many times it was heard, and the last date and time that the utterance was heard. It can take up to 24 hours for missed utterances to appear in the console.

### To view and add missed utterances to a bot:

1. Follow the first step of Getting Started Exercise 1 to create and test an `OrderFlowers` bot. For instructions, see [Exercise 1: Create an Amazon Lex Bot Using a Blueprint \(Console\) \(p. 24\)](#).

2. Test the bot by typing the following utterances in the **Test Bot** window. Type each utterance several times. The example bot doesn't recognize the following utterances:
  - Order flowers
  - Get me flowers
  - Please order flowers
  - Get me some flowers
3. Wait for Amazon Lex to gather usage data about the missed utterances. Utterance data is generated once per day, generally overnight.
4. Sign in to the AWS Management Console and open the Amazon Lex console at <https://console.aws.amazon.com/lex/>.
5. Choose the `OrderFlowers` bot.
6. Choose the **Monitoring** tab, and then choose **Utterances** from the left menu and then choose the **Missed** button. The pane shows a maximum of 100 missed utterances.



The screenshot shows the 'Utterances' pane in the Amazon Lex console. At the top, there is a button 'Add utterance to Intent' with a dropdown arrow. Below this is a filter section with a search bar labeled 'Filter: Q Filter by keyword' and two tabs: 'Detected' and 'Missed'. The 'Missed' tab is selected. Below the tabs is a table with the following columns: 'Utterances', 'Count', 'Status', and 'Last said date'. Each row in the table has a checkbox on the left. The table contains six rows of missed utterances.

<input type="checkbox"/>	Utterances	Count	Status	Last said date
<input type="checkbox"/>	I want flowers	5	Missed	April 21, 2017 at 10:28:13 A
<input type="checkbox"/>	Order flowers	4	Missed	April 21, 2017 at 10:28:05 A
<input type="checkbox"/>	Get me some flowers	2	Missed	April 21, 2017 at 10:27:49 A
<input type="checkbox"/>	Get me flowers	2	Missed	April 21, 2017 at 10:27:25 A
<input type="checkbox"/>	Please order flowers	1	Missed	April 21, 2017 at 10:26:55 A
<input type="checkbox"/>	get me some flowers	1	Missed	April 21, 2017 at 10:27:18 A

7. To choose the missed utterances that you want to add to the bot, select the check box next to them. To add the utterance to the `$LATEST` version of the intent, choose the down arrow next to the **Add utterance to intent** dropdown, and then choose the intent.
8. To rebuild your bot, choose **Build** and then **Build** again to re-build your bot.
9. To verify that your bot recognizes the new utterances, use the **Test Bot** pane.

## Example: Integrating with a Web site

In this example you integrate a bot with a Web site using text and voice. You use JavaScript and AWS services to build an interactive experience for visitors to your Web site. The complete example is documented in two posts on the [AWS AI Blog](#):

- ["Greetings, visitor!"—Engage Your Web Users with Amazon Lex](#)—Demonstrates using Amazon Lex, the AWS SDK for JavaScript in the Browser, and Amazon Cognito to create a conversational experience on your Web site.

- [Capturing Voice Input in a Browser and Sending it to Amazon Lex](#)—Demonstrates embedding a voice-based chatbot in a Web site using the SDK for JavaScript in the Browser. The application records audio, sends the audio to Amazon Lex, and then plays the response.



# Monitoring Amazon Lex

Monitoring is important for maintaining the reliability, availability, and performance of your Amazon Lex solutions. This topic describes the available monitoring tools and provides reference content for Amazon Lex metrics.

## Topics

- [Monitoring Amazon Lex with Amazon CloudWatch](#) (p. 148)
- [CloudWatch Metrics for Amazon Lex](#) (p. 149)

## Monitoring Amazon Lex with Amazon CloudWatch

With Amazon CloudWatch, you can get metrics for individual Amazon Lex operations or global Amazon Lex operations for your account. Use metrics to track the health of your Amazon Lex solution and to set up alarms to notify you when one or more metrics fall outside of exceeds thresholds that you define. For example, you can monitor the number of requests made to a bot over a particular time period, see the latency of successful requests, or raise an alarm when errors exceed a threshold.

## Using CloudWatch Metrics for Amazon Lex

To use metrics, you must specify the following information:

- The metric dimension. A *dimension* is a set of name-value pairs that you use to identify a metric. Amazon Lex has three dimensions:
  - BotAlias, BotName, Operation
  - BotAlias, BotName, InputMode, Operation
  - BotName, BotVersion, InputMode, Operation
- The metric name, such as `MissedUtteranceCount` or `RuntimeRequestCount`.

You can get metrics for Amazon Lex with the AWS Management Console, the AWS CLI, or the CloudWatch API. You can use the CloudWatch API through one of the Amazon AWS Software Development Kits (SDKs) or the CloudWatch API tools. The Amazon Lex console displays graphs based on raw data from the CloudWatch API.

You must have the appropriate CloudWatch permissions to monitor Amazon Lex with CloudWatch. For more information, see [Authentication and Access Control for Amazon CloudWatch](#) in the *Amazon CloudWatch User Guide*.

## Viewing Metrics for Amazon Lex

Use the following procedures to view Amazon Lex metrics using the Amazon Lex console, the AWS CLI, and the CloudWatch API.

### To view metrics (console)

1. Sign in to the AWS Management Console and open the Amazon Lex console at <https://console.aws.amazon.com/lex/>.
2. From the list of bots, choose the one whose metrics you want to see.
3. Choose **Monitoring**. Metrics are displayed in graphs.

### To view metrics (CloudWatch console)

1. Sign in to the AWS Management Console and open the CloudWatch console at <https://console.aws.amazon.com/cloudwatch/>.
2. Choose **Metrics**, choose the **All Metrics** tab, and then choose **AWS/Lex**.
3. Choose one of the dimensions, and then choose a metric. Add the metric to the graph.
4. Choose a value for the date range. The metric count for the selected date range is displayed in the graph.

## Creating an Alarm

You can create a CloudWatch alarm that sends you an Amazon Simple Notification Service (Amazon SNS) message when the alarm changes state. An alarm watches a single metric over a time period that you specify, and performs one or more actions based on the value of the metric relative to a given threshold over a number of time periods. The action is sending a notification to an Amazon SNS topic or Auto Scaling policy.

CloudWatch alarms invoke actions only when the state changes and has persisted for the period that you specify.

### To set an alarm (console)

1. Sign in to the AWS Management Console and open the CloudWatch console at <https://console.aws.amazon.com/cloudwatch/>.
2. Choose **Alarms**, and then choose **Create Alarm**. This launches the **Create Alarm** wizard.
3. Choose **AWS/Lex Metrics**, and then choose a metric.
4. For **Time Range**, choose a time range to monitor, and then choose **Next**.
5. Type a **Name** and **Description**. For **Whenever**, choose **>=**, and type a maximum value.
6. If you want CloudWatch to send an email when the alarm state is reached, in the **Actions** section choose **State is ALARM** in the **Whenever this alarm** dropdown. In the **Send notification to** dropdown select an existing mailing list or click **New list** to create a new list.
7. Preview the alarm in the **Alarm Preview** section. If you are satisfied with the alarm, choose **Create Alarm** to create and save the alarm.

## CloudWatch Metrics for Amazon Lex

This topic explains the Amazon CloudWatch metrics and the dimensions that are available for Amazon Lex.

## Topics

- [CloudWatch Metrics for Amazon Lex Runtime \(p. 150\)](#)
- [CloudWatch Metrics for Amazon Lex Channel Associations \(p. 153\)](#)

# CloudWatch Metrics for Amazon Lex Runtime

The following table summarizes the Amazon Lex runtime metrics.

Metric	Description
RuntimeInvalidLambdaResponses	<p>The number of invalid Lambda responses in the specified time period that could not be parsed by Amazon Lex.</p> <p>Valid dimensions when Operation=PostContent and InputMode=Text or Speech</p> <ul style="list-style-type: none"><li>• BotName, BotAlias, Operation, InputMode</li></ul> <p>Valid dimensions when Operation=PostText</p> <ul style="list-style-type: none"><li>• BotName, BotAlias, Operation</li></ul>
RuntimeLambdaErrors	<p>The number of Lambda runtime errors in the specified time period.</p> <p>Valid dimensions when Operation=PostContent and InputMode=Text or Speech</p> <ul style="list-style-type: none"><li>• BotName, BotAlias, Operation, InputMode</li></ul> <p>Valid dimensions when Operation=PostText</p> <ul style="list-style-type: none"><li>• BotName, BotAlias, Operation</li></ul>
MissedUtteranceCount	<p>The number of utterances that were not recognized in the specified time period.</p> <p>Valid dimensions when Operation=PostContent and InputMode=Text or Speech</p> <ul style="list-style-type: none"><li>• BotName, BotVersion, Operation, InputMode</li><li>• BotName, BotAlias, Operation, InputMode</li></ul> <p>Valid dimensions when Operation=PostText</p> <ul style="list-style-type: none"><li>• BotName, BotVersion, Operation</li><li>• BotName, BotAlias, Operation</li></ul>
RuntimePollyErrors	<p>The number of invalid Amazon Polly responses in the specified time period.</p> <p>Valid dimensions when Operation=PostContent and InputMode=Text or Speech</p> <ul style="list-style-type: none"><li>• BotName, BotAlias, Operation, InputMode</li></ul>

Metric	Description
	<p>Valid dimensions when Operation=PostText</p> <ul style="list-style-type: none"><li>• BotName, BotAlias, Operation</li></ul>
RuntimeRequestCount	<p>The number of requests in the specified time period.</p> <p>Valid dimensions when Operation=PostContent and InputMode=Text or Speech</p> <ul style="list-style-type: none"><li>• BotName, BotVersion, Operation, InputMode</li><li>• BotName, BotAlias, Operation, InputMode</li></ul> <p>Valid dimensions when Operation=PostText</p> <ul style="list-style-type: none"><li>• BotName, BotVersion, Operation</li><li>• BotName, BotAlias, Operation</li></ul> <p>Unit: Count</p>
RuntimeSuccessfulRequestLatency	<p>The latency for successful requests between the time that the request was made and the response was passed back.</p> <p>Valid dimensions when Operation=PostContent and InputMode=Text or Speech</p> <ul style="list-style-type: none"><li>• BotName, BotVersion, Operation, InputMode</li><li>• BotName, BotAlias, Operation, InputMode</li></ul> <p>Valid dimensions when Operation=PostText</p> <ul style="list-style-type: none"><li>• BotName, BotVersion, Operation</li><li>• BotName, BotAlias, Operation</li></ul> <p>Unit: Milliseconds</p>
RuntimeSystemErrors	<p>The number of system errors in the specified time period. The response code range for a system error is 500 to 599.</p> <p>Valid dimensions when Operation=PostContent and InputMode=Text or Speech</p> <ul style="list-style-type: none"><li>• BotName, BotAlias, Operation, InputMode</li></ul> <p>Valid dimensions when Operation=PostText</p> <ul style="list-style-type: none"><li>• BotName, BotAlias, Operation</li></ul> <p>Unit: Count</p>

Metric	Description
RuntimeThrottledEvents	<p>The number of throttled requests. Amazon Lex throttles a request when it receives more requests than the limit of transactions per second set for your account. If the limit set for your account is frequently exceeded, you can request a limit increase. To request an increase, see <a href="#">AWS Service Limits</a>.</p> <p>Valid dimensions when Operation=PostContent and InputMode=Text or Speech</p> <ul style="list-style-type: none"><li>• BotName, BotAlias, Operation, InputMode</li></ul> <p>Valid dimensions when Operation=PostText</p> <ul style="list-style-type: none"><li>• BotName, BotAlias, Operation</li></ul> <p>Unit: Count</p>
RuntimeUserErrors	<p>The number of user errors in the specified time period. The response code range for a user error is 400 to 499.</p> <p>Valid dimensions when Operation=PostContent and InputMode=Text or Speech</p> <ul style="list-style-type: none"><li>• BotName, BotAlias, Operation, InputMode</li></ul> <p>Valid dimensions when Operation=PostText</p> <ul style="list-style-type: none"><li>• BotName, BotAlias, Operation</li></ul> <p>Unit: Count</p>

Amazon Lex runtime metrics use the `AWS/Lex` namespace, and provide metrics for the following dimensions:

Dimension	Description
BotName, BotAlias, Operation, InputMode	Metrics are grouped by the bot's alias, the bot's name, the operation ( <code>PostContent</code> ), and by whether the input was text or speech.
BotName, BotVersion, Operation, InputMode	Metrics are grouped by the bot's name, the version of the bot, the operation ( <code>PostContent</code> ), and by whether the input was text or speech.
BotName, BotVersion, Operation	Metrics are grouped by the bot's name, the bot's version, and by the operation, <code>PostText</code> .
BotName, BotAlias, Operation	Metrics are grouped by the bot's name, the bot's alias, and by the operation, <code>PostText</code> .

## CloudWatch Metrics for Amazon Lex Channel Associations

A channel association is the association between Amazon Lex and a messaging channel, such as Facebook. The following table describes the Amazon Lex channel association metrics.

Metric	Description
BotChannelAuthErrors	The number of authentication errors returned by the messaging channel in the specified time period. An authentication error indicates that the secret token provided during channel creation is invalid or has expired. Create a new channel with the correct secret token.
BotChannelConfigurationErrors	The number of configuration errors in the specified time period. A configuration error indicates that one or more configuration entries for the channel are invalid.
BotChannelInboundThrottledEvents	The number of times that messages that were sent to Amazon Lex by the messaging channel were throttled in the specified time period.
BotChannelOutboundThrottledEvents	The number of times that outbound events from Amazon Lex to the messaging channel were throttled in the specified time period.
BotChannelRequestCount	The number of requests made on a channel in the specified time period.
BotChannelResponseCardErrors	The number of times that Amazon Lex could not post response cards in the specified time period. Check that response cards are formatted correctly.
BotChannelSystemErrors	The number of internal errors that occurred in Amazon Lex for a channel in the specified time period.

Amazon Lex runtime metrics use the `AWS/Lex` namespace, and provide metrics for the following dimension:

Dimension	Description
BotAlias, BotChannelName, BotName, Source	Groups metrics by the bot's alias, the channel name, the bot's name, and by the source of traffic.

# Guidelines and Limits in Amazon Lex

The following sections provide guidelines and limits when using Amazon Lex.

## Topics

- [General Guidelines \(p. 154\)](#)
- [Limits \(p. 156\)](#)

## General Guidelines

This section describes general guidelines when using Amazon Lex.

- **Signing requests** – All Amazon Lex model-building and runtime API operations in the [API Reference \(p. 173\)](#) use signature V4 for authenticating requests. For more information about authenticating requests, see [Signature Version 4 Signing Process](#) in the *Amazon Web Services General Reference*.

For [PostContent \(p. 286\)](#), uses the unsigned payload option described in [Signature Calculations for the Authorization Header: Transferring Payload in a Single Chunk \(AWS Signature Version 4\)](#) in the *Amazon Simple Storage Service (S3) API Reference*.

When you use the unsigned payload option, don't include the hash of the payload in the canonical request. Instead, you use the literal string "UNSIGNED-PAYLOAD" as the hash of the payload. Also include a header with the name `x-amz-content-sha256` and the value `UNSIGNED-PAYLOAD` in the `PostContent` request.

- Note the following about how Amazon Lex captures slot values from user utterances:

Amazon Lex uses the enumeration values you provide in a slot type definition to train its machine learning models. Suppose you define an intent called `GetPredictionIntent` with the following sample utterance:

```
"Tell me the prediction for {Sign}"
```

Where {Sign} is a slot of custom type `ZodiacSign`. `ZodiacSign` has 12 enumeration values (`Aries` through `Pisces`). From the user utterance "Tell me the prediction for ..." Amazon Lex understands that what follows is a zodiac sign.

If the user says "Tell me the prediction for earth", Amazon Lex infers that "earth" is possibly another `ZodiacSign` and passes it to your fulfillment activity. Therefore, your fulfillment activity must validate the slot values.

When Amazon Lex calls a Lambda function or returns the result of a conversation to your client application, the case of the slot values is not guaranteed. For example, if you are eliciting values for the `AMAZON.Movie` built-in slot type, and a user says or types "Gone with the wind," Amazon Lex may return "Gone with the Wind," "gone with the wind," or "Gone With The Wind."

- Amazon Lex does not support the `AMAZON.LITERAL` built-in slot type that the Alexa Skills Kit supports. However, Amazon Lex supports creating custom slot types that you can use to implement this functionality. As mentioned in the previous bullet, you can capture values outside the custom slot type definition. Add more and diverse enumeration values to boost the automatic speech recognition (ASR) and natural language understanding (NLU) accuracy.
- The `AMAZON.DATE` and `AMAZON.TIME` built-in slot types capture both absolute and relative dates and times. Relative dates and times are resolved in the region where Amazon Lex is processing the request.

For the `AMAZON.TIME` built-in slot type, if the user doesn't specify that a time is before or after noon, the time is ambiguous and Amazon Lex will prompt the user again. We recommend prompts that elicit an absolute time. For example, use a prompt such as "When do you want your pizza delivered? You can say 6 PM or 6 in the evening."

- Providing confusable training data in your bot reduces Amazon Lex's ability to understand user input. Consider these examples:

Suppose you have two intents (`OrderPizza` and `OrderDrink`) in your bot and both are configured with an "I want to order" utterance. This utterance does not map to a specific intent that Amazon Lex can learn from while building the language model for the bot at build time. As a result, when a user inputs this utterance at runtime, Amazon Lex can't pick an intent with a high degree of confidence.

Consider another example where you define a custom intent for getting a confirmation from the user (for example, `MyCustomConfirmationIntent`) and configure the intent with the utterances "Yes" and "No." Note that Amazon Lex also has a language model for understanding user confirmations. This can create conflicting situation. When the user responds with a "Yes," does this mean that this is a confirmation for the ongoing intent or that the user is requesting the custom intent that you created?



In general, the sample utterances you provide should map to a specific intent and, optionally, to specific slot values.

- The runtime API operations [PostContent \(p. 286\)](#) and [PostText \(p. 293\)](#) take a user ID as the required parameter. Developers can set this to any value that meets the constraints described in the API. We recommend you don't use this parameter to send any confidential information such as user logins, emails, or social security numbers. This ID is primarily used to uniquely identify conversation with a bot (there can be multiple users ordering pizza).
- If your client application uses Amazon Cognito for authentication, you might use the Amazon Cognito user ID as Amazon Lex user ID. Note that any Lambda function configured for your bot must have its own authentication mechanism to identify the user on whose behalf Amazon Lex is invoking the Lambda function.
- We encourage you to define an intent that captures a user's intention to discontinue the conversation. For example, you can define an intent (`NothingIntent`) with sample utterances ("I don't want anything", "exit", "bye bye"), no slots, and no Lambda function configured as a code hook. This would let users gracefully close a conversation.

## Limits

This section describes current limits in Amazon Lex. These limits are grouped by categories.

### Topics

- [General Limits \(p. 156\)](#)
- [Runtime Service Limits \(p. 157\)](#)
- [Model Building Limits \(p. 157\)](#)

## General Limits

Currently, Amazon Lex is available in `us-east-1` region.

Service	Region Name	Region	Endpoint	Protocol
Model building service	US East (N. Virginia)	us-east-1	models.lex.us-east-1.amazonaws.com	HTTPS
Runtime service	US East (N. Virginia)	us-east-1	runtime.lex.us-east-1.amazonaws.com	HTTPS

Currently, Amazon Lex supports only US English language. That is, Amazon Lex trains your bots to understand only US English.

## Runtime Service Limits

Currently, Amazon Lex is available in `us-east-1` region.

Region Name	Region	Endpoint	Protocol
US East (N. Virginia)	us-east-1	runtime.lex.us-east-1.amazonaws.com	HTTPS

In addition to the limits described in the API reference, note the following:

- API
  - Input speech in the [PostContent \(p. 286\)](#) can be up to 15 seconds long.
  - In both the runtime API operations [PostContent \(p. 286\)](#) and [PostText \(p. 293\)](#), the input text size can be up to 1024 Unicode characters.
  - The total header size of a `PostContent` header is 16 KB. The total size of the session attributes in a `PostContent` request and response is 12 KB.
  - The maximum input size to a Lambda function is 12 KB. The maximum output size is 25 KB, of which 12 KB can be session attributes.

## Model Building Limits

Model building refers to creating and managing bots. This includes creating and managing bots, intents, slot types, slots, and bot channel associations.

Currently, Amazon Lex is available in `us-east-1` region.

Region Name	Region	Endpoint	Protocol
US East (N. Virginia)	us-east-1	models.lex.us-east-1.amazonaws.com	HTTPS

### Topics

- [Bot Limits \(p. 158\)](#)
- [Intent Limits \(p. 159\)](#)
- [Slot Type Limits \(p. 160\)](#)

## Bot Limits

- You configure prompts and statements throughout the model building API. Each of these prompts or statements can have up to five messages and each message can contain from 1 to 1000 UTF-8 characters.
- You can define sample utterances for intents and slots. You can use a maximum of 200,000 characters for all utterances.
- Bot, alias, and bot channel association names are case insensitive at the time of creation. If you create `PizzaBot` and then try to create `pizzaBot`, you will get an error. However, when accessing a resource, the resource names are case sensitive, you must specify `PizzaBot` and not `pizzaBot`. These names must be between 2 and 50 ASCII characters.
- The maximum number of versions you can publish for all resource types is 100. Note that there is no versioning for aliases.
- Within a bot, intent names and slot names must be unique, you can't have an intent and a slot by the same name.
- You can create a bot that is configured to support multiple intents. If two intents have a slot by the same name, then the corresponding slot type must be the same.

For example, suppose you create a bot to support two intents (`OrderPizza` and `OrderDrink`). If both these intents have the `size` slot, then the slot type must be the same in both places.

In addition, the sample utterances you provide for a slot in one of the intents applies to a slot with the same name in other intents.

- You can associate a maximum of 100 intents with a bot.
- When you create a bot, you specify a session timeout. The session timeout can be between one minute and one day. The default is five minutes.

This timeout determines how long the bot can retain conversation context, such as current user intent and slot data.

In addition, note that after a user starts the conversation with your bot and until the session expires, Amazon Lex uses the same bot version (even if you update the bot alias to point to another version).

- When you update the `$LATEST` version of the bot, Amazon Lex terminates any in-progress conversations for any client application using the `$LATEST` version of the bot). Generally, you should not use the `$LATEST` version of a bot in production because `$LATEST` version can be updated. You should publish a version and use it instead.
- When you update an alias, Amazon Lex takes a few minutes to pick up the change. When you modify the `$LATEST` version of the bot, the change is picked up immediately.
- The `$LATEST` version of your bot should only be used for manual testing while building the bot. Amazon Lex limits the number of runtime requests that you can make to the `$LATEST` version of the bot.
- You can create up to five aliases for a bot.
- You can create up to 100 bots per AWS account.
- You cannot create multiple intents that extend from the same built-in intent.

## Intent Limits

- Intent and slot names are case insensitive at the time of creation. That is, if you create `OrderPizza` intent and then again try to create another `orderPizza` intent, you will get an error. However, when accessing these resources, the resource names are case sensitive, specify `OrderPizza` and not `orderPizza`. These names must be between 1 and 100 ASCII characters.
- An intent can have up to 1,500 sample utterances. A minimum of one sample utterance is required. Each sample utterance can be up to 200 UTF-8 characters long. You can use up to 200,000 characters for all intent and slot utterances in a bot. A sample utterance for an intent:
  - Can refer to zero or more slot names.
  - Can refer to a slot name only once.

For example:

```
I want a pizza
I want a {pizzaSize} pizza
I want a {pizzaSize} {pizzaTopping} pizza
```

- Although each intent supports up to 1,500 utterances, if you use fewer utterances Amazon Lex may have a better ability to recognize inputs outside your provided set.
- Each slot can have up to 10 sample utterances. Each sample utterance must refer to the slot name exactly once. For example:

```
{pizzaSize} please
```

- Each bot can have a maximum of 200,000 characters for intent and slot utterances combined.
- You cannot provide utterances for intents that extend from built-in intents. For all other intents you must provide at least one sample utterance. Intents contain slots, but the slot level sample utterances are optional.
- Built-in intents
  - Currently, Amazon Lex does not support slot elicitation for built-in intents. You cannot create Lambda functions to return the `ElicitSlot` directive in the response with an intent that is derived from built-in intents. For more information, see [Response Format \(p. 88\)](#).
  - The service does not support adding sample utterances to built-in intents. Similarly, you cannot add or remove slots to built-in intents.
- You can create up to 1,000 intents per AWS account. You can create up to 100 slots in an intent.

## Slot Type Limits

- Slot type names are case insensitive at the time of creation. If you create the `PizzaSize` slot type and then again try to create the `pizzaSize` slot type, you will get an error. However, when accessing these resources, the resource names are case sensitive (you must specify `PizzaSize` and not `pizzaSize`). These names must be between 1 and 100 ASCII characters.
- Resource (bot, intent, alias, slot, slot type) names are case insensitive.

A custom slot type you create can have a maximum of 10,000 enumeration values, and each enumeration value can be up to 140 UTF-8 characters long. The enumeration values cannot contain duplicates.

- For a slot type value, where appropriate, specify both upper and lower case. For example, for a slot type called `Procedure`, if value is `MRI`, specify both `MRI` and `mri` as values.
- Built-in slot types – Currently, Amazon Lex doesn't support adding enumeration values for the built-in slot types.

# Authentication and Access Control for Amazon Lex

Access to Amazon Lex requires credentials that AWS can use to authenticate your requests. Those credentials must have permissions to access AWS resources, such as an Amazon Lex chatbot or an Amazon Lex slot type. The following sections provide details on how you can use [AWS Identity and Access Management \(IAM\)](#) and Amazon Lex to help secure your resources by controlling who can access them.

- [Authentication](#) (p. 161)
- [Access Control](#) (p. 162)

## Authentication

You can access AWS as any of the following types of identities:

- **AWS account root user** – When you sign up for AWS, you provide an email address and password that is associated with your AWS account. This is your *AWS account root user*. Its credentials provide complete access to all of your AWS resources.

### Important

For security reasons, we recommend that you use the root user only to create an *administrator*, which is an *IAM user* with full permissions to your AWS account. You can then use this administrator user to create other IAM users and roles with limited permissions. For more information, see [IAM Best Practices](#) and [Creating an Admin User and Group](#) in the *IAM User Guide*.

- **IAM user** – An [IAM user](#) is simply an identity within your AWS account that has specific custom permissions (for example, permissions to create a bot in Amazon Lex). You can use an IAM user name and password to sign in to secure AWS webpages like the [AWS Management Console](#), [AWS Discussion Forums](#), or the [AWS Support Center](#).

In addition to a user name and password, you can also generate [access keys](#) for each user. You can use these keys when you access AWS services programmatically, either through [one of the several SDKs](#) or by using the [AWS Command Line Interface \(CLI\)](#). The SDK and CLI tools use the access keys to cryptographically sign your request. If you don't use the AWS tools, you must sign the request yourself. Amazon Lex supports *Signature Version 4*, a protocol for authenticating inbound API requests. For more information about authenticating requests, see [Signature Version 4 Signing Process](#) in the *AWS General Reference*.

- **IAM role** – An [IAM role](#) is another IAM identity that you can create in your account that has specific permissions. It is similar to an *IAM user*, but it is not associated with a specific person. An IAM role enables you to obtain temporary access keys that can be used to access AWS services and resources. IAM roles with temporary credentials are useful in the following situations:
  - **Federated user access** – Instead of creating an IAM user, you can use preexisting user identities from AWS Directory Service, your enterprise user directory, or a web identity provider. These are known as *federated users*. AWS assigns a role to a federated user when access is requested through an [identity provider](#). For more information about federated users, see [Federated Users and Roles](#) in the *IAM User Guide*.
  - **Cross-account access** – You can use an IAM role in your account to grant another AWS account permissions to access your account's resources. For an example, see [Tutorial: Delegate Access Across AWS Accounts Using IAM Roles](#) in the *IAM User Guide*.
  - **AWS service access** – You can use an IAM role in your account to grant an AWS service permissions to access your account's resources. For example, you can create a role that allows Amazon Redshift to access an Amazon S3 bucket on your behalf and then load data from that bucket into an Amazon Redshift cluster. For more information, see [Creating a Role to Delegate Permissions to an AWS Service](#) in the *IAM User Guide*.
  - **Applications running on Amazon EC2** – You can use an IAM role to manage temporary credentials for applications running on an EC2 instance and making AWS API requests. This is preferable to storing access keys within the EC2 instance. To assign an AWS role to an EC2 instance and make it available to all of its applications, you create an instance profile that is attached to the instance. An instance profile contains the role and enables programs running on the EC2 instance to get temporary credentials. For more information, see [Using Roles for Applications on Amazon EC2](#) in the *IAM User Guide*.

## Access Control

You can have valid credentials to authenticate your requests, but unless you have permissions you cannot create or access Amazon Lex resources. For example, you must have permissions to create an Amazon Lex bot.

The following sections describe how to manage permissions for Amazon Lex. We recommend that you read the overview first.

- [Overview of Managing Access Permissions to Your Amazon Lex Resources](#) (p. 163)
- [Using Identity-Based Policies \(IAM Policies\) for Amazon Lex](#) (p. 166)

# Overview of Managing Access Permissions to Your Amazon Lex Resources

Every AWS resource is owned by an AWS account, and permissions to create or access a resource are governed by permissions policies. An account administrator can attach permissions policies to IAM identities (that is, users, groups, and roles), and some services (such as AWS Lambda) also support attaching permissions policies to resources.

## Note

An *account administrator* (or administrator user) is a user with administrator privileges. For more information, see [IAM Best Practices](#) in the *IAM User Guide*.

When granting permissions, you decide who is getting the permissions, the resources they get permissions for, and the specific actions that you want to allow on those resources.

## Topics

- [Amazon Lex Resources and Operations](#) (p. 163)
- [Understanding Resource Ownership](#) (p. 163)
- [Managing Access to Resources](#) (p. 164)
- [Specifying Policy Elements: Actions, Effects, and Principals](#) (p. 165)
- [Specifying Conditions in a Policy](#) (p. 165)

## Amazon Lex Resources and Operations

In Amazon Lex, the primary resource is a *bot*. Amazon Lex also supports additional resource types, the *intent*, the *slot type*, the *alias*, and the *bot channel association*. Aliases and bot channel associations are referred to as *subresources*. For Amazon Lex, you can create subresources only in the context of an existing bot.

These resources and subresources have unique Amazon Resource Names (ARNs) associated with them as shown in the following table.

Amazon Lex provides a set of operations to work with Amazon Lex resources. For a list of available operations, see Amazon Lex [Actions](#) (p. 173).

## Understanding Resource Ownership

The AWS account owns the resources that are created in the account, regardless of who created the resources. Specifically, the resource owner is the AWS account of the [principal entity](#) (that is, the root account, an IAM user, or an IAM role) that authenticates the resource creation request. The following examples illustrate how this works:

- If you use the root account credentials of your AWS account to create a bot, your AWS account is the owner of the resource (in Amazon Lex, the resource is the bot).
- If you create an IAM user in your AWS account and grant permissions to create a bot to that user, the user can create a bot. However, your AWS account, to which the user belongs, owns the bot resource.
- If you create an IAM role in your AWS account with permissions to create a bot, anyone who can assume the role can create a bot. Your AWS account, to which the role belongs, owns the bot resource.



## Managing Access to Resources

A *permissions policy* describes who has access to what. The following section explains the available options for creating permissions policies.

### Note

This section discusses using IAM in the context of Amazon Lex. It doesn't provide detailed information about the IAM service. For complete IAM documentation, see [What Is IAM?](#) in the *IAM User Guide*. For information about IAM policy syntax and descriptions, see [AWS IAM Policy Reference](#) in the *IAM User Guide*.

Policies attached to an IAM identity are referred to as *identity-based* policies (IAM policies) and policies attached to a resource are referred to as *resource-based* policies. Amazon Lex supports only identity-based policies (IAM policies).

### Topics

- [Identity-Based Policies \(IAM Policies\)](#) (p. 164)
- [Resource-Based Policies](#) (p. 165)

## Identity-Based Policies (IAM Policies)

You can attach policies to IAM identities. For example, you can do the following:

- **Attach a permissions policy to a user or a group in your account** – To grant a user or a group of users permissions to create a Amazon Lex resource, such as a bot, you can attach a permissions policy to a user or group that the user belongs to.
- **Attach a permissions policy to a role (grant cross-account permissions)** – To grant cross-account permissions, you can attach an identity-based permissions policy to an IAM role. For example, the administrator in Account A can create a role to grant cross-account permissions to another AWS account (for example, Account B) or an AWS service as follows:
  1. Account A administrator creates an IAM role and attaches a permissions policy to the role that grants permissions on resources in Account A.
  2. Account A administrator attaches a trust policy to the role identifying Account B as the principal who can assume the role.
  3. Account B administrator can then delegate permissions to assume the role to any users in Account B. Doing this allows users in Account B to create or access resources in Account A. If you want to grant an AWS service permissions to assume the role, the principal in the trust policy can also be an AWS service principal.

For more information about using IAM to delegate permissions, see [Access Management](#) in the *IAM User Guide*.

The following is an example policy that allows the user to perform the `PutBot` action for your AWS account.

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

```
} ]
```

For more information about using identity-based policies with Amazon Lex, see [Using Identity-Based Policies \(IAM Policies\) for Amazon Lex \(p. 166\)](#). For more information about users, groups, roles, and permissions, see [Identities \(Users, Groups, and Roles\)](#) in the *IAM User Guide*.

## Resource-Based Policies

Other services, such as Lambda, support resource-based permissions policies. For example, you can attach a policy to an S3 bucket to manage access permissions to that bucket. Amazon Lex doesn't support resource-based policies. However, it does *use* resource-based policies to access Lambda and Amazon Polly services.

## Specifying Policy Elements: Actions, Effects, and Principals

For each Amazon Lex resource (see [Amazon Lex Resources and Operations \(p. 163\)](#)), the service defines a set of API operations (see [Actions \(p. 173\)](#)). To grant permissions for these API operations, Amazon Lex defines a set of actions that you can specify in a policy. For example, for the Amazon Lex Intent resource, the following actions are defined: `CreateIntent` and `CreateIntentVersion`. Performing an API operation can require permissions for more than one action.

The following are the most basic policy elements:

- **Resource** – In a policy, you use an Amazon Resource Name (ARN) to identify the resource to which the policy applies. For more information, see [Amazon Lex Resources and Operations \(p. 163\)](#).
- **Action** – You use action keywords to identify resource operations that you want to allow or deny. For example, depending on the specified `Effect`, `lex:bot` either allows or denies the user permissions to perform the Amazon Lex `CreateBot` operation.
- **Effect** – You specify the effect of the action that occurs when the user requests the specific action—this can be either allow or deny. If you don't explicitly grant access to (allow) a resource, access is implicitly denied. You can also explicitly deny access to a resource. You might do this to make sure that a user cannot access the resource, even if a different policy grants access.
- **Principal** – In identity-based policies (IAM policies), the user that the policy is attached to is the implicit principal. For resource-based policies, you specify the user, account, service, or other entity that you want to receive permissions. This applies to resource-based policies only. Amazon Lex doesn't support resource-based policies.

To learn more about IAM policy syntax and descriptions, see [AWS IAM Policy Reference](#) in the *IAM User Guide*.

For a table showing all of the Amazon Lex API actions, see [Amazon Lex API Permissions: Actions, Resources, and Conditions Reference \(p. 171\)](#).

## Specifying Conditions in a Policy

When you grant permissions, you use the IAM policy language to specify the conditions under which a policy should take effect. For example, you might want a policy to be applied only after a specific date. For more information about specifying conditions in a policy language, see [Condition](#) in the *IAM User Guide*.

AWS provides a set of predefined condition keys for all AWS services that support IAM for access control. For example, you can use the `aws:userid` condition key to require a specific AWS ID when requesting an

action. For more information and a complete list of AWS-wide keys, see [Available Keys for Conditions](#) in the *IAM User Guide*.

**Note**

Condition keys are case sensitive.

Amazon Lex provides additional condition keys that you can include in `Condition` elements in an IAM permissions policy. The following table shows the Amazon Lex condition keys that apply to Amazon Lex resources.

## Example Policy: Using Condition Keys

The following example shows how to use condition keys in Amazon Lex IAM permissions policies.

### Example 1: Grant Permission to Create Bots Using the OrderPizza Intent

```
{
  "Version": "2012-10-17",
  "Statement": [
    {
      "Effect": "Allow",
      "Action": [
        "lex:PutBot"
      ],
      "Resource": [
        "*"
      ],
      "Condition": {
        "ForAllValues:StringLike": {
          "lex:associatedIntents": [
            "OrderPizza"
          ]
        }
      }
    }
  ]
}
```

## Using Identity-Based Policies (IAM Policies) for Amazon Lex

This topic provides examples of identity-based policies that demonstrate how an account administrator can attach permissions policies to IAM identities (that is, users, groups, and roles) and thereby grant permissions to perform operations on Amazon Lex resources.

**Important**

Before you proceed, we recommend that you review [Overview of Managing Access Permissions to Your Amazon Lex Resources](#) (p. 163).

The sections in this topic cover the following:

- [Permissions Required to Use the Amazon Lex Console](#) (p. 167)
- [AWS Managed \(Predefined\) Policies for Amazon Lex](#) (p. 169)
- [Examples of Customer Managed Policies](#) (p. 170)

The following is an example of a permissions policy:

```
{
  "Version": "2012-10-17",
  "Statement": [
    {
      "Effect": "Allow",
      "Action": [
        "lex:PostText"
      ],
      "Resource": [
        "arn:aws:lex:us-east-1:account-id:bot:OrderPizza:*"
      ]
    }
  ]
}
```

The policy has one statement that grants permission to use the `PostText` action with the `OrderPizza` bot. The resource specifies a wildcard character (\*) to give permission to any alias of the `OrderPizza` bot.

The policy doesn't specify the `Principal` element because you don't specify the principal who gets the permission in an identity-based policy. When you attach a policy to a user, the user is the implicit principal. When you attach a permissions policy to an IAM role, the principal identified in the role's trust policy gets the permissions.

For a table showing all of the Amazon Lex API actions and the resources that they apply to, see [Amazon Lex API Permissions: Actions, Resources, and Conditions Reference \(p. 171\)](#).

## Permissions Required to Use the Amazon Lex Console

The permissions reference table lists the Amazon Lex API operations and shows the required permissions for each operation. For more information about Amazon Lex API operations, see [Amazon Lex API Permissions: Actions, Resources, and Conditions Reference \(p. 171\)](#).

To use the Amazon Lex console, you need to grant permissions for additional actions as shown in the following permissions policy:

```
{
  "Version": "2012-10-17",
  "Statement": [
    {
      "Effect": "Allow",
      "Action": [
        "cloudwatch:GetMetricStatistics",
        "cloudwatch:DescribeAlarms",
        "cloudwatch:DescribeAlarmsForMetric",
        "kms:DescribeKey",
        "kms:ListAliases",
        "lambda:GetPolicy",
        "lambda:ListFunctions",
        "lex:*",
        "polly:DescribeVoices",
        "polly:SynthesizeSpeech"
      ],
      "Resource": [
        "*"
      ]
    },
    {
      "Effect": "Allow",
      "Action": [
        "lambda:AddPermission",

```

```
        "lambda:RemovePermission"
    ],
    "Resource": "*",
    "Condition": {
        "StringLike": {
            "lambda:Principal": "lex.amazonaws.com"
        }
    }
},
{
    "Effect": "Allow",
    "Action": [
        "iam:GetRole",
        "iam:DeleteRole"
    ],
    "Resource": [
        "arn:aws:iam::*:role/aws-service-role/lex.amazonaws.com/
AWSServiceRoleForLexBots",
        "arn:aws:iam::*:role/aws-service-role/channels.lex.amazonaws.com/
AWSServiceRoleForLexChannels"
    ]
},
{
    "Effect": "Allow",
    "Action": [
        "iam:CreateServiceLinkedRole"
    ],
    "Resource": [
        "arn:aws:iam::*:role/aws-service-role/lex.amazonaws.com/
AWSServiceRoleForLexBots"
    ],
    "Condition": {
        "StringLike": {
            "iam:AWSServiceName": "lex.amazonaws.com"
        }
    }
},
{
    "Effect": "Allow",
    "Action": [
        "iam:DetachRolePolicy"
    ],
    "Resource": [
        "arn:aws:iam::*:role/aws-service-role/lex.amazonaws.com/
AWSServiceRoleForLexBots"
    ],
    "Condition": {
        "StringLike": {
            "iam:PolicyArn": "arn:aws:iam::aws:policy/aws-service-role/
AmazonLexBotPolicy"
        }
    }
},
{
    "Effect": "Allow",
    "Action": [
        "iam:CreateServiceLinkedRole"
    ],
    "Resource": [
        "arn:aws:iam::*:role/aws-service-role/channels.lex.amazonaws.com/
AWSServiceRoleForLexChannels"
    ],
    "Condition": {
        "StringLike": {
            "iam:AWSServiceName": "channels.lex.amazonaws.com"
        }
    }
}
```

```
    },
    {
      "Effect": "Allow",
      "Action": [
        "iam:DetachRolePolicy"
      ],
      "Resource": [
        "arn:aws:iam::*:role/aws-service-role/channels.lex.amazonaws.com/AWSServiceRoleForLexChannels"
      ],
      "Condition": {
        "StringLike": {
          "iam:PolicyArn": "arn:aws:iam::aws:policy/aws-service-role/LexChannelPolicy"
        }
      }
    }
  ]
}
```

The Amazon Lex console needs these additional permissions for the following reasons:

- `cloudwatch` permissions to view performance and monitoring information in the console.
- `iam` actions to assume IAM roles for making calls to Lambda functions and processing data for a bot channel association.
- `kms` actions to manage the AWS Key Management Service keys used to encrypt data when creating a bot channel association.
- `lambda` actions to display Lambda functions that your bot can use, and to grant Lex the necessary permissions for your bot to invoke these functions.
- `lex` actions so that the console can display Amazon Lex resources in the account.
- `polly` actions so that the console can display Amazon Polly voices and so that it can translate text to speech.
- `iam` actions so that the console can manage server-linked roles that grant permission to use other AWS resources.

## AWS Managed (Predefined) Policies for Amazon Lex

AWS addresses many common use cases by providing standalone IAM policies that are created and administered by AWS. Managed policies grant necessary permissions for common use cases so you can avoid having to investigate which permissions are needed. For more information, see [AWS Managed Policies](#) in the *IAM User Guide*.

The following AWS managed policies, which you can attach to users in your account, are specific to Amazon Lex:

- **ReadOnly** — Grants read-only access to Amazon Lex resources.
- **RunBotsOnly** — Grants access to run Amazon Lex conversational bots.
- **FullAccess** — Grants full access to create, read, update, delete, and run all Amazon Lex resources. Grants access to associate Lambda functions whose name starts with `AmazonLex` with Amazon Lex intents.

### Note

You can review these permissions policies by signing in to the IAM console and searching for specific policies.

You can also create your own custom IAM policies to allow permissions for Amazon Lex API actions. You can attach these custom policies to the IAM users or groups that require those permissions.

## Examples of Customer Managed Policies

In this section, we provide examples of user policies that grant permissions for various Amazon Lex actions. These policies work with the AWS SDKs or the AWS command line interface (AWS CLI). When you use the console, you need to grant additional permissions specific to the console, which is discussed in [Permissions Required to Use the Amazon Lex Console](#) (p. 167).

### Note

All examples use the us-east-1 Region and contain fictitious account IDs.

### Examples

- [Example 1: Allow a User to Delete Any Bot](#) (p. 170)
- [Example 2: Allow a User to Update a Specific Bot](#) (p. 170)
- [Example 3: Allow a User to Manage a Specific Bot](#) (p. 171)

### Example 1: Allow a User to Delete Any Bot

The following permissions policy grants the user permissions to delete any bot that exists in the us-east-1 Region.

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

### Example 2: Allow a User to Update a Specific Bot

The following policy grants the user permissions to update a specific bot in the us-east-1 Region, in this case, the bot named "PizzaBot."

```
{
  "Version": "2012-10-17",
  "Statement": [
    {
      "Effect": "Allow",
      "Action": [
        "lex:PutBot"
      ],
      "Resource": [
        "arn:aws:lex:us-east-1:account-id:bot:PizzaBot:$LATEST"
      ]
    }
  ]
}
```

### Example 3: Allow a User to Manage a Specific Bot

The following permissions policy grants the user permissions to build and test a pizza ordering, and can only use the `OrderPizza` intent and `Toppings` slot type when editing the bot in the `us-east-1` region. The policy uses the `lex:associatedIntents` and `lex:associatedSlotType` to limit the intent and slot types that the user can use for this bot.

```
{
  "Version": "2012-10-17",
  "Statement": [
    {
      "Effect": "Allow",
      "Action": [
        "lex:Create*Version",
        "lex:Post*",
        "lex:Put*",
        "lex>Delete*"
      ],
      "Resource": [
        "arn:aws:lex:us-east-1::bot:PizzaBot:*",
        "arn:aws:lex:us-east-1::intent:OrderPizza:*",
        "arn:aws:lex:us-east-1::slottype:Toppings:*"
      ],
      "Condition": {
        "ForAllValues:StringEqualsIfExists": {
          "lex:associatedIntents": [
            "OrderPizza"
          ],
          "lex:associatedSlotTypes": [
            "Toppings"
          ]
        }
      }
    },
    {
      "Effect": "Allow",
      "Action": [
        "lex:Get*"
      ],
      "Resource": [
        "arn:aws:lex:us-east-1::bot:*",
        "arn:aws:lex:us-east-1::intent:*",
        "arn:aws:lex:us-east-1::slottype:*"
      ]
    }
  ]
}
```

## Amazon Lex API Permissions: Actions, Resources, and Conditions Reference

Use the following table as a reference when setting up [Access Control \(p. 162\)](#) and writing a permissions policy that you can attach to an IAM identity (an identity-based policy). The list includes each Amazon Lex API operation, the corresponding action for which you can grant permissions to



perform the action, and the AWS resource for which you can grant the permissions. You specify the actions in the policy's `Action` field, and you specify the resource value in the policy's `Resource` field.

To express conditions, you can use AWS-wide condition keys in your Amazon Lex policies. For a complete list of AWS-wide keys, see [Available Keys](#) in the *IAM User Guide*.

**Note**

To specify an action, use the `lex:` prefix followed by the API operation name, for example, `lex:PostText`.

# API Reference

This section provides documentation for the Amazon Lex API operations. Amazon Lex is available in the following AWS region:

Service	Region Name	Region	Endpoint	Protocol
Model building service	US East (N. Virginia)	us-east-1	models.lex.us-east-1.amazonaws.com	HTTPS
Runtime service	US East (N. Virginia)	us-east-1	runtime.lex.us-east-1.amazonaws.com	HTTPS

## Topics

- [Actions \(p. 173\)](#)
- [Data Types \(p. 298\)](#)

## Actions

The following actions are supported by Amazon Lex Model Building Service:

- [CreateBotVersion \(p. 176\)](#)
- [CreateIntentVersion \(p. 181\)](#)
- [CreateSlotTypeVersion \(p. 187\)](#)
- [DeleteBot \(p. 191\)](#)
- [DeleteBotAlias \(p. 193\)](#)
- [DeleteBotChannelAssociation \(p. 195\)](#)
- [DeleteBotVersion \(p. 197\)](#)
- [DeleteIntent \(p. 199\)](#)
- [DeleteIntentVersion \(p. 201\)](#)
- [DeleteSlotType \(p. 203\)](#)
- [DeleteSlotTypeVersion \(p. 205\)](#)

- [DeleteUtterances](#) (p. 207)
- [GetBot](#) (p. 209)
- [GetBotAlias](#) (p. 214)
- [GetBotAliases](#) (p. 217)
- [GetBotChannelAssociation](#) (p. 220)
- [GetBotChannelAssociations](#) (p. 223)
- [GetBots](#) (p. 226)
- [GetBotVersions](#) (p. 229)
- [GetBuiltinIntent](#) (p. 232)
- [GetBuiltinIntents](#) (p. 234)
- [GetBuiltinSlotTypes](#) (p. 236)
- [GetIntent](#) (p. 238)
- [GetIntents](#) (p. 243)
- [GetIntentVersions](#) (p. 246)
- [GetSlotType](#) (p. 249)
- [GetSlotTypes](#) (p. 252)
- [GetSlotTypeVersions](#) (p. 255)
- [GetUtterancesView](#) (p. 258)
- [PutBot](#) (p. 261)
- [PutBotAlias](#) (p. 269)
- [PutIntent](#) (p. 273)
- [PutSlotType](#) (p. 282)

The following actions are supported by Amazon Lex Runtime Service:

- [PostContent](#) (p. 286)
- [PostText](#) (p. 293)

## Amazon Lex Model Building Service

The following actions are supported by Amazon Lex Model Building Service:

- [CreateBotVersion](#) (p. 176)
- [CreateIntentVersion](#) (p. 181)
- [CreateSlotTypeVersion](#) (p. 187)
- [DeleteBot](#) (p. 191)
- [DeleteBotAlias](#) (p. 193)
- [DeleteBotChannelAssociation](#) (p. 195)
- [DeleteBotVersion](#) (p. 197)
- [DeleteIntent](#) (p. 199)
- [DeleteIntentVersion](#) (p. 201)
- [DeleteSlotType](#) (p. 203)
- [DeleteSlotTypeVersion](#) (p. 205)
- [DeleteUtterances](#) (p. 207)
- [GetBot](#) (p. 209)
- [GetBotAlias](#) (p. 214)

- [GetBotAliases](#) (p. 217)
- [GetBotChannelAssociation](#) (p. 220)
- [GetBotChannelAssociations](#) (p. 223)
- [GetBots](#) (p. 226)
- [GetBotVersions](#) (p. 229)
- [GetBuiltinIntent](#) (p. 232)
- [GetBuiltinIntents](#) (p. 234)
- [GetBuiltinSlotTypes](#) (p. 236)
- [GetIntent](#) (p. 238)
- [GetIntents](#) (p. 243)
- [GetIntentVersions](#) (p. 246)
- [GetSlotType](#) (p. 249)
- [GetSlotTypes](#) (p. 252)
- [GetSlotTypeVersions](#) (p. 255)
- [GetUtterancesView](#) (p. 258)
- [PutBot](#) (p. 261)
- [PutBotAlias](#) (p. 269)
- [PutIntent](#) (p. 273)
- [PutSlotType](#) (p. 282)

## CreateBotVersion

Service: Amazon Lex Model Building Service

Creates a new version of the bot based on the `$LATEST` version. If the `$LATEST` version of this resource hasn't changed since you created the last version, Amazon Lex doesn't create a new version. It returns the last created version.

### Note

You can update only the `$LATEST` version of the bot. You can't update the numbered versions that you create with the `CreateBotVersion` operation.

When you create the first version of a bot, Amazon Lex sets the version to 1. Subsequent versions increment by 1. For more information, see [Versioning \(p. 81\)](#).

This operation requires permission for the `lex:CreateBotVersion` action.

## Request Syntax

```
POST /bots/name/versions HTTP/1.1
Content-type: application/json

{
  "checksum": "string"
}
```

## URI Request Parameters

The request requires the following URI parameters.

### `name` (p. 176)

The name of the bot that you want to create a new version of. The name is case sensitive.

Length Constraints: Minimum length of 2. Maximum length of 50.

Pattern: `^[a-zA-Z]+((_[a-zA-Z]+)|([a-zA-Z]+_)*|_)`

## Request Body

The request accepts the following data in JSON format.

### `checksum` (p. 176)

Identifies a specific revision of the `$LATEST` version of the bot. If you specify a checksum and the `$LATEST` version of the bot has a different checksum, a `PreconditionFailedException` exception is returned and Amazon Lex doesn't publish a new version. If you don't specify a checksum, Amazon Lex publishes the `$LATEST` version.

Type: String

Required: No

## Response Syntax

```
HTTP/1.1 201
Content-type: application/json
```

```
{
  "abortStatement": {
    "messages": [
      {
        "content": "string",
        "contentType": "string"
      }
    ],
    "responseCard": "string"
  },
  "checksum": "string",
  "childDirected": boolean,
  "clarificationPrompt": {
    "maxAttempts": number,
    "messages": [
      {
        "content": "string",
        "contentType": "string"
      }
    ],
    "responseCard": "string"
  },
  "createdDate": number,
  "description": "string",
  "failureReason": "string",
  "idleSessionTTLInSeconds": number,
  "intents": [
    {
      "intentName": "string",
      "intentVersion": "string"
    }
  ],
  "lastUpdatedDate": number,
  "locale": "string",
  "name": "string",
  "status": "string",
  "version": "string",
  "voiceId": "string"
}
```

## Response Elements

If the action is successful, the service sends back an HTTP 201 response.

The following data is returned in JSON format by the service.

### **abortStatement (p. 176)**

The message that Amazon Lex uses to abort a conversation. For more information, see [PutBot \(p. 261\)](#).

Type: [Statement \(p. 323\)](#) object

### **checksum (p. 176)**

Checksum identifying the version of the bot that was created.

Type: String

### **childDirected (p. 176)**

For each Amazon Lex bot created with the Amazon Lex Model Building Service, you must specify whether your use of Amazon Lex is related to a website, program, or other application that is directed or targeted, in whole or in part, to children under age 13 and subject to the Children's Online Privacy Protection Act (COPPA) by specifying `true` or `false` in the `childDirected` field. By

specifying `true` in the `childDirected` field, you confirm that your use of Amazon Lex **is** related to a website, program, or other application that is directed or targeted, in whole or in part, to children under age 13 and subject to COPPA. By specifying `false` in the `childDirected` field, you confirm that your use of Amazon Lex **is not** related to a website, program, or other application that is directed or targeted, in whole or in part, to children under age 13 and subject to COPPA. You may not specify a default value for the `childDirected` field that does not accurately reflect whether your use of Amazon Lex is related to a website, program, or other application that is directed or targeted, in whole or in part, to children under age 13 and subject to COPPA.

If your use of Amazon Lex relates to a website, program, or other application that is directed in whole or in part, to children under age 13, you must obtain any required verifiable parental consent under COPPA. For information regarding the use of Amazon Lex in connection with websites, programs, or other applications that are directed or targeted, in whole or in part, to children under age 13, see the [Amazon Lex FAQ](#).

Type: Boolean

#### **clarificationPrompt (p. 176)**

The message that Amazon Lex uses when it doesn't understand the user's request. For more information, see [PutBot \(p. 261\)](#).

Type: [Prompt \(p. 317\)](#) object

#### **createdDate (p. 176)**

The date when the bot version was created.

Type: Timestamp

#### **description (p. 176)**

A description of the bot.

Type: String

Length Constraints: Minimum length of 0. Maximum length of 200.

#### **failureReason (p. 176)**

If `status` is `FAILED`, Amazon Lex provides the reason that it failed to build the bot.

Type: String

#### **idleSessionTTLInSeconds (p. 176)**

The maximum time in seconds that Amazon Lex retains the data gathered in a conversation. For more information, see [PutBot \(p. 261\)](#).

Type: Integer

Valid Range: Minimum value of 60. Maximum value of 86400.

#### **intents (p. 176)**

An array of `Intent` objects. For more information, see [PutBot \(p. 261\)](#).

Type: Array of [Intent \(p. 313\)](#) objects

Array Members: Minimum number of 1 item. Maximum number of 100 items.

#### **lastUpdatedDate (p. 176)**

The date when the `$LATEST` version of this bot was updated.

Type: Timestamp

### **locale (p. 176)**

Specifies the target locale for the bot.

Type: String

Valid Values: `en-US`

### **name (p. 176)**

The name of the bot.

Type: String

Length Constraints: Minimum length of 2. Maximum length of 50.

Pattern: `^[a-zA-Z]+((_[a-zA-Z]+)|([a-zA-Z]+_)*|_)`

### **status (p. 176)**

When you send a request to create or update a bot, Amazon Lex sets the `status` response element to `BUILDING`. After Amazon Lex builds the bot, it sets `status` to `READY`. If Amazon Lex can't build the bot, it sets `status` to `FAILED`. Amazon Lex returns the reason for the failure in the `failureReason` response element.

Type: String

Valid Values: `BUILDING` | `READY` | `FAILED` | `NOT_BUILT`

### **version (p. 176)**

The version of the bot.

Type: String

Length Constraints: Minimum length of 1. Maximum length of 64.

Pattern: `\$LATEST|[0-9]+`

### **voiceId (p. 176)**

The Amazon Polly voice ID that Amazon Lex uses for voice interactions with the user.

Type: String

## **Errors**

### **BadRequestException**

The request is not well formed. For example, a value is invalid or a required field is missing. Check the field values, and try again.

HTTP Status Code: 400

### **ConflictException**

There was a conflict processing the request. Try your request again.

HTTP Status Code: 409

### **InternalFailureException**

An internal Amazon Lex error occurred. Try your request again.

HTTP Status Code: 500



**LimitExceededException**

The request exceeded a limit. Try your request again.

HTTP Status Code: 429

**NotFoundException**

The resource specified in the request was not found. Check the resource and try again.

HTTP Status Code: 404

**PreconditionFailedException**

The checksum of the resource that you are trying to change does not match the checksum in the request. Check the resource's checksum and try again.

HTTP Status Code: 412

## See Also

For more information about using this API in one of the language-specific AWS SDKs, see the following:

- [AWS Command Line Interface](#)
- [AWS SDK for .NET](#)
- [AWS SDK for C++](#)
- [AWS SDK for Go](#)
- [AWS SDK for Java](#)
- [AWS SDK for JavaScript](#)
- [AWS SDK for PHP V3](#)
- [AWS SDK for Python](#)
- [AWS SDK for Ruby V2](#)

## CreateIntentVersion

Service: Amazon Lex Model Building Service

Creates a new version of an intent based on the `$LATEST` version of the intent. If the `$LATEST` version of this intent hasn't changed since you last updated it, Amazon Lex doesn't create a new version. It returns the last version you created.

### Note

You can update only the `$LATEST` version of the intent. You can't update the numbered versions that you create with the `CreateIntentVersion` operation.

When you create a version of an intent, Amazon Lex sets the version to 1. Subsequent versions increment by 1. For more information, see [Versioning \(p. 81\)](#).

This operation requires permissions to perform the `lex:CreateIntentVersion` action.

## Request Syntax

```
POST /intents/name/versions HTTP/1.1
Content-type: application/json

{
  "checksum": "string"
}
```

## URI Request Parameters

The request requires the following URI parameters.

### `name` (p. 181)

The name of the intent that you want to create a new version of. The name is case sensitive.

Length Constraints: Minimum length of 1. Maximum length of 100.

Pattern: `^[a-zA-Z]+((_[a-zA-Z]+)|([a-zA-Z]+_)*|_)`

## Request Body

The request accepts the following data in JSON format.

### `checksum` (p. 181)

Checksum of the `$LATEST` version of the intent that should be used to create the new version. If you specify a checksum and the `$LATEST` version of the intent has a different checksum, Amazon Lex returns a `PreconditionFailedException` exception and doesn't publish a new version. If you don't specify a checksum, Amazon Lex publishes the `$LATEST` version.

Type: String

Required: No

## Response Syntax

```
HTTP/1.1 201
Content-type: application/json
```

```
{
  "checksum": "string",
  "conclusionStatement": {
    "messages": [
      {
        "content": "string",
        "contentType": "string"
      }
    ],
    "responseCard": "string"
  },
  "confirmationPrompt": {
    "maxAttempts": number,
    "messages": [
      {
        "content": "string",
        "contentType": "string"
      }
    ],
    "responseCard": "string"
  },
  "createdAt": number,
  "description": "string",
  "dialogCodeHook": {
    "messageVersion": "string",
    "uri": "string"
  },
  "followUpPrompt": {
    "prompt": {
      "maxAttempts": number,
      "messages": [
        {
          "content": "string",
          "contentType": "string"
        }
      ],
      "responseCard": "string"
    },
    "rejectionStatement": {
      "messages": [
        {
          "content": "string",
          "contentType": "string"
        }
      ],
      "responseCard": "string"
    }
  },
  "fulfillmentActivity": {
    "codeHook": {
      "messageVersion": "string",
      "uri": "string"
    },
    "type": "string"
  },
  "lastUpdatedDate": number,
  "name": "string",
  "parentIntentSignature": "string",
  "rejectionStatement": {
    "messages": [
      {
        "content": "string",
        "contentType": "string"
      }
    ],
    "responseCard": "string"
  }
}
```

```
    },
    "sampleUtterances": [ "string" ],
    "slots": [
      {
        "description": "string",
        "name": "string",
        "priority": number,
        "responseCard": "string",
        "sampleUtterances": [ "string" ],
        "slotConstraint": "string",
        "slotType": "string",
        "slotTypeVersion": "string",
        "valueElicitationPrompt": {
          "maxAttempts": number,
          "messages": [
            {
              "content": "string",
              "contentType": "string"
            }
          ],
          "responseCard": "string"
        }
      }
    ],
    "version": "string"
  }
```

## Response Elements

If the action is successful, the service sends back an HTTP 201 response.

The following data is returned in JSON format by the service.

### checksum (p. 181)

Checksum of the intent version created.

Type: String

### conclusionStatement (p. 181)

After the Lambda function specified in the `fulfillmentActivity` field fulfills the intent, Amazon Lex conveys this statement to the user.

Type: [Statement \(p. 323\)](#) object

### confirmationPrompt (p. 181)

If defined, the prompt that Amazon Lex uses to confirm the user's intent before fulfilling it.

Type: [Prompt \(p. 317\)](#) object

### createdDate (p. 181)

The date that the intent was created.

Type: Timestamp

### description (p. 181)

A description of the intent.

Type: String

Length Constraints: Minimum length of 0. Maximum length of 200.

### **dialogCodeHook (p. 181)**

If defined, Amazon Lex invokes this Lambda function for each user input.

Type: [CodeHook \(p. 309\)](#) object

### **followUpPrompt (p. 181)**

If defined, Amazon Lex uses this prompt to solicit additional user activity after the intent is fulfilled.

Type: [FollowUpPrompt \(p. 311\)](#) object

### **fulfillmentActivity (p. 181)**

Describes how the intent is fulfilled.

Type: [FulfillmentActivity \(p. 312\)](#) object

### **lastUpdatedDate (p. 181)**

The date that the intent was updated.

Type: Timestamp

### **name (p. 181)**

The name of the intent.

Type: String

Length Constraints: Minimum length of 1. Maximum length of 100.

Pattern: `^[a-zA-Z]+((_[a-zA-Z]+)*|([a-zA-Z]+_)*|_)`

### **parentIntentSignature (p. 181)**

A unique identifier for a built-in intent.

Type: String

### **rejectionStatement (p. 181)**

If the user answers "no" to the question defined in `confirmationPrompt`, Amazon Lex responds with this statement to acknowledge that the intent was canceled.

Type: [Statement \(p. 323\)](#) object

### **sampleUtterances (p. 181)**

An array of sample utterances configured for the intent.

Type: Array of strings

Array Members: Minimum number of 0 items. Maximum number of 1500 items.

Length Constraints: Minimum length of 1. Maximum length of 200.

### **slots (p. 181)**

An array of slot types that defines the information required to fulfill the intent.

Type: Array of [Slot \(p. 319\)](#) objects

Array Members: Minimum number of 0 items. Maximum number of 100 items.

### **version (p. 181)**

The version number assigned to the new version of the intent.

Type: String

Length Constraints: Minimum length of 1. Maximum length of 64.

Pattern: `\$LATEST|[0-9]+`

## Errors

### **BadRequestException**

The request is not well formed. For example, a value is invalid or a required field is missing. Check the field values, and try again.

HTTP Status Code: 400

### **ConflictException**

There was a conflict processing the request. Try your request again.

HTTP Status Code: 409

### **InternalFailureException**

An internal Amazon Lex error occurred. Try your request again.

HTTP Status Code: 500

### **LimitExceededException**

The request exceeded a limit. Try your request again.

HTTP Status Code: 429

### **NotFoundException**

The resource specified in the request was not found. Check the resource and try again.

HTTP Status Code: 404

### **PreconditionFailedException**

The checksum of the resource that you are trying to change does not match the checksum in the request. Check the resource's checksum and try again.

HTTP Status Code: 412

## See Also

For more information about using this API in one of the language-specific AWS SDKs, see the following:

- [AWS Command Line Interface](#)
- [AWS SDK for .NET](#)
- [AWS SDK for C++](#)
- [AWS SDK for Go](#)
- [AWS SDK for Java](#)
- [AWS SDK for JavaScript](#)
- [AWS SDK for PHP V3](#)
- [AWS SDK for Python](#)
- [AWS SDK for Ruby V2](#)



## CreateSlotTypeVersion

Service: Amazon Lex Model Building Service

Creates a new version of a slot type based on the `$LATEST` version of the specified slot type. If the `$LATEST` version of this resource has not changed since the last version that you created, Amazon Lex doesn't create a new version. It returns the last version that you created.

### Note

You can update only the `$LATEST` version of a slot type. You can't update the numbered versions that you create with the `CreateSlotTypeVersion` operation.

When you create a version of a slot type, Amazon Lex sets the version to 1. Subsequent versions increment by 1. For more information, see [Versioning \(p. 81\)](#).

This operation requires permissions for the `lex:CreateSlotTypeVersion` action.

## Request Syntax

```
POST /slottypes/name/versions HTTP/1.1
Content-type: application/json

{
  "checksum": "string"
}
```

## URI Request Parameters

The request requires the following URI parameters.

### name (p. 187)

The name of the slot type that you want to create a new version for. The name is case sensitive.

Length Constraints: Minimum length of 1. Maximum length of 100.

Pattern: `^[a-zA-Z]+((_[a-zA-Z]+)|([a-zA-Z]+_)*|_)`

## Request Body

The request accepts the following data in JSON format.

### checksum (p. 187)

Checksum for the `$LATEST` version of the slot type that you want to publish. If you specify a checksum and the `$LATEST` version of the slot type has a different checksum, Amazon Lex returns a `PreconditionFailedException` exception and doesn't publish the new version. If you don't specify a checksum, Amazon Lex publishes the `$LATEST` version.

Type: String

Required: No

## Response Syntax

```
HTTP/1.1 201
Content-type: application/json
```



```
{
  "checksum": "string",
  "createdDate": number,
  "description": "string",
  "enumerationValues": [
    {
      "value": "string"
    }
  ],
  "lastUpdatedDate": number,
  "name": "string",
  "version": "string"
}
```

## Response Elements

If the action is successful, the service sends back an HTTP 201 response.

The following data is returned in JSON format by the service.

### **checksum** (p. 187)

Checksum of the `$LATEST` version of the slot type.

Type: String

### **createdDate** (p. 187)

The date that the slot type was created.

Type: Timestamp

### **description** (p. 187)

A description of the slot type.

Type: String

Length Constraints: Minimum length of 0. Maximum length of 200.

### **enumerationValues** (p. 187)

A list of `EnumerationValue` objects that defines the values that the slot type can take.

Type: Array of [EnumerationValue](#) (p. 310) objects

Array Members: Minimum number of 1 item. Maximum number of 10000 items.

### **lastUpdatedDate** (p. 187)

The date that the slot type was updated. When you create a resource, the creation date and last update date are the same.

Type: Timestamp

### **name** (p. 187)

The name of the slot type.

Type: String

Length Constraints: Minimum length of 1. Maximum length of 100.

Pattern: `^[a-zA-Z]+((_[a-zA-Z]+)|([a-zA-Z]+_))|_`

### version (p. 187)

The version assigned to the new slot type version.

Type: String

Length Constraints: Minimum length of 1. Maximum length of 64.

Pattern: \{\$LATEST|[0-9]\}+

## Errors

### **BadRequestException**

The request is not well formed. For example, a value is invalid or a required field is missing. Check the field values, and try again.

HTTP Status Code: 400

### **ConflictException**

There was a conflict processing the request. Try your request again.

HTTP Status Code: 409

### **InternalFailureException**

An internal Amazon Lex error occurred. Try your request again.

HTTP Status Code: 500

### **LimitExceededException**

The request exceeded a limit. Try your request again.

HTTP Status Code: 429

### **NotFoundException**

The resource specified in the request was not found. Check the resource and try again.

HTTP Status Code: 404

### **PreconditionFailedException**

The checksum of the resource that you are trying to change does not match the checksum in the request. Check the resource's checksum and try again.

HTTP Status Code: 412

## See Also

For more information about using this API in one of the language-specific AWS SDKs, see the following:

- [AWS Command Line Interface](#)
- [AWS SDK for .NET](#)
- [AWS SDK for C++](#)
- [AWS SDK for Go](#)
- [AWS SDK for Java](#)
- [AWS SDK for JavaScript](#)
- [AWS SDK for PHP V3](#)

- [AWS SDK for Python](#)
- [AWS SDK for Ruby V2](#)

## DeleteBot

Service: Amazon Lex Model Building Service

Deletes all versions of the bot, including the `$LATEST` version. To delete a specific version of the bot, use the [DeleteBotVersion \(p. 197\)](#) operation.

If a bot has an alias, you can't delete it. Instead, the `DeleteBot` operation returns a `ResourceInUseException` exception that includes a reference to the alias that refers to the bot. To remove the reference to the bot, delete the alias. If you get the same exception again, delete the referring alias until the `DeleteBot` operation is successful.

This operation requires permissions for the `lex:DeleteBot` action.

## Request Syntax

```
DELETE /bots/name HTTP/1.1
```

## URI Request Parameters

The request requires the following URI parameters.

### **name** (p. 191)

The name of the bot. The name is case sensitive.

Length Constraints: Minimum length of 2. Maximum length of 50.

Pattern: `^[a-zA-Z]+((_[a-zA-Z]+)|([a-zA-Z]+_))*|_`

## Request Body

The request does not have a request body.

## Response Syntax

```
HTTP/1.1 204
```

## Response Elements

If the action is successful, the service sends back an HTTP 204 response with an empty HTTP body.

## Errors

### **BadRequestException**

The request is not well formed. For example, a value is invalid or a required field is missing. Check the field values, and try again.

HTTP Status Code: 400

### **ConflictException**

There was a conflict processing the request. Try your request again.

HTTP Status Code: 409

### **InternalFailureException**

An internal Amazon Lex error occurred. Try your request again.

HTTP Status Code: 500

**LimitExceededException**

The request exceeded a limit. Try your request again.

HTTP Status Code: 429

**NotFoundException**

The resource specified in the request was not found. Check the resource and try again.

HTTP Status Code: 404

**ResourceInUseException**

The resource that you are attempting to delete is referred to by another resource. Use this information to remove references to the resource that you are trying to delete.

The body of the exception contains a JSON object that describes the resource.

```
{ "resourceType": BOT | BOTALIAS | BOTCHANNEL | INTENT,  
  
  "resourceReference": {  
  
    "name": string, "version": string } }
```

HTTP Status Code: 400

## See Also

For more information about using this API in one of the language-specific AWS SDKs, see the following:

- [AWS Command Line Interface](#)
- [AWS SDK for .NET](#)
- [AWS SDK for C++](#)
- [AWS SDK for Go](#)
- [AWS SDK for Java](#)
- [AWS SDK for JavaScript](#)
- [AWS SDK for PHP V3](#)
- [AWS SDK for Python](#)
- [AWS SDK for Ruby V2](#)

## DeleteBotAlias

Service: Amazon Lex Model Building Service

Deletes an alias for the specified bot.

You can't delete an alias that is used in the association between a bot and a messaging channel. If an alias is used in a channel association, the `DeleteBot` operation returns a `ResourceInUseException` exception that includes a reference to the channel association that refers to the bot. You can remove the reference to the alias by deleting the channel association. If you get the same exception again, delete the referring association until the `DeleteBotAlias` operation is successful.

### Request Syntax

```
DELETE /bots/botName/aliases/name HTTP/1.1
```

### URI Request Parameters

The request requires the following URI parameters.

#### **botName** (p. 193)

The name of the bot that the alias points to.

Length Constraints: Minimum length of 2. Maximum length of 50.

Pattern: `^[a-zA-Z]+(("[a-zA-Z"]+)|([a-zA-Z_]+)|_)`

#### **name** (p. 193)

The name of the alias to delete. The name is case sensitive.

Length Constraints: Minimum length of 1. Maximum length of 100.

Pattern: `^[a-zA-Z]+(("[a-zA-Z"]+)|([a-zA-Z_]+)|_)`

### Request Body

The request does not have a request body.

### Response Syntax

```
HTTP/1.1 204
```

### Response Elements

If the action is successful, the service sends back an HTTP 204 response with an empty HTTP body.

### Errors

#### **BadRequestException**

The request is not well formed. For example, a value is invalid or a required field is missing. Check the field values, and try again.

HTTP Status Code: 400

#### **ConflictException**

There was a conflict processing the request. Try your request again.

HTTP Status Code: 409

**InternalFailureException**

An internal Amazon Lex error occurred. Try your request again.

HTTP Status Code: 500

**LimitExceededException**

The request exceeded a limit. Try your request again.

HTTP Status Code: 429

**NotFoundException**

The resource specified in the request was not found. Check the resource and try again.

HTTP Status Code: 404

**ResourceInUseException**

The resource that you are attempting to delete is referred to by another resource. Use this information to remove references to the resource that you are trying to delete.

The body of the exception contains a JSON object that describes the resource.

```
{ "resourceType": BOT | BOTALIAS | BOTCHANNEL | INTENT,  
  "resourceReference": {  
    "name": string, "version": string } }
```

HTTP Status Code: 400

## See Also

For more information about using this API in one of the language-specific AWS SDKs, see the following:

- [AWS Command Line Interface](#)
- [AWS SDK for .NET](#)
- [AWS SDK for C++](#)
- [AWS SDK for Go](#)
- [AWS SDK for Java](#)
- [AWS SDK for JavaScript](#)
- [AWS SDK for PHP V3](#)
- [AWS SDK for Python](#)
- [AWS SDK for Ruby V2](#)

## DeleteBotChannelAssociation

Service: Amazon Lex Model Building Service

Deletes the association between an Amazon Lex bot and a messaging platform.

This operation requires permission for the `lex:DeleteBotChannelAssociation` action.

### Request Syntax

```
DELETE /bots/botName/aliases/aliasName/channels/name HTTP/1.1
```

### URI Request Parameters

The request requires the following URI parameters.

#### **botAlias** (p. 195)

An alias that points to the specific version of the Amazon Lex bot to which this association is being made.

Length Constraints: Minimum length of 1. Maximum length of 100.

Pattern: `^[a-zA-Z]+((_[a-zA-Z]+)*|([a-zA-Z]+_)*|_)`

#### **botName** (p. 195)

The name of the Amazon Lex bot.

Length Constraints: Minimum length of 2. Maximum length of 50.

Pattern: `^[a-zA-Z]+((_[a-zA-Z]+)*|([a-zA-Z]+_)*|_)`

#### **name** (p. 195)

The name of the association. The name is case sensitive.

Length Constraints: Minimum length of 1. Maximum length of 100.

Pattern: `^[a-zA-Z]+((_[a-zA-Z]+)*|([a-zA-Z]+_)*|_)`

### Request Body

The request does not have a request body.

### Response Syntax

```
HTTP/1.1 204
```

### Response Elements

If the action is successful, the service sends back an HTTP 204 response with an empty HTTP body.

### Errors

#### **BadRequestException**

The request is not well formed. For example, a value is invalid or a required field is missing. Check the field values, and try again.



HTTP Status Code: 400

**ConflictException**

There was a conflict processing the request. Try your request again.

HTTP Status Code: 409

**InternalFailureException**

An internal Amazon Lex error occurred. Try your request again.

HTTP Status Code: 500

**LimitExceededException**

The request exceeded a limit. Try your request again.

HTTP Status Code: 429

**NotFoundException**

The resource specified in the request was not found. Check the resource and try again.

HTTP Status Code: 404

## See Also

For more information about using this API in one of the language-specific AWS SDKs, see the following:

- [AWS Command Line Interface](#)
- [AWS SDK for .NET](#)
- [AWS SDK for C++](#)
- [AWS SDK for Go](#)
- [AWS SDK for Java](#)
- [AWS SDK for JavaScript](#)
- [AWS SDK for PHP V3](#)
- [AWS SDK for Python](#)
- [AWS SDK for Ruby V2](#)

## DeleteBotVersion

Service: Amazon Lex Model Building Service

Deletes a specific version of a bot. To delete all versions of a bot, use the [DeleteBot \(p. 191\)](#) operation.

This operation requires permissions for the `lex:DeleteBotVersion` action.

### Request Syntax

```
DELETE /bots/name/versions/version HTTP/1.1
```

### URI Request Parameters

The request requires the following URI parameters.

#### **name** (p. 197)

The name of the bot.

Length Constraints: Minimum length of 2. Maximum length of 50.

Pattern: `^[a-zA-Z]+((_[a-zA-Z]+)*|([a-zA-Z]+_)*|_)`

#### **version** (p. 197)

The version of the bot to delete. You cannot delete the `$LATEST` version of the bot. To delete the `$LATEST` version, use the [DeleteBot \(p. 191\)](#) operation.

Length Constraints: Minimum length of 1. Maximum length of 64.

Pattern: `[0-9]+`

### Request Body

The request does not have a request body.

### Response Syntax

```
HTTP/1.1 204
```

### Response Elements

If the action is successful, the service sends back an HTTP 204 response with an empty HTTP body.

### Errors

#### **BadRequestException**

The request is not well formed. For example, a value is invalid or a required field is missing. Check the field values, and try again.

HTTP Status Code: 400

#### **ConflictException**

There was a conflict processing the request. Try your request again.

HTTP Status Code: 409

### **InternalFailureException**

An internal Amazon Lex error occurred. Try your request again.

HTTP Status Code: 500

### **LimitExceededException**

The request exceeded a limit. Try your request again.

HTTP Status Code: 429

### **NotFoundException**

The resource specified in the request was not found. Check the resource and try again.

HTTP Status Code: 404

### **ResourceInUseException**

The resource that you are attempting to delete is referred to by another resource. Use this information to remove references to the resource that you are trying to delete.

The body of the exception contains a JSON object that describes the resource.

```
{ "resourceType": BOT | BOTALIAS | BOTCHANNEL | INTENT,  
  
  "resourceReference": {  
  
    "name": string, "version": string } }
```

HTTP Status Code: 400

## **See Also**

For more information about using this API in one of the language-specific AWS SDKs, see the following:

- [AWS Command Line Interface](#)
- [AWS SDK for .NET](#)
- [AWS SDK for C++](#)
- [AWS SDK for Go](#)
- [AWS SDK for Java](#)
- [AWS SDK for JavaScript](#)
- [AWS SDK for PHP V3](#)
- [AWS SDK for Python](#)
- [AWS SDK for Ruby V2](#)

## DeleteIntent

Service: Amazon Lex Model Building Service

Deletes all versions of the intent, including the `$LATEST` version. To delete a specific version of the intent, use the [DeleteIntentVersion \(p. 201\)](#) operation.

You can delete a version of an intent only if it is not referenced. To delete an intent that is referred to in one or more bots (see [Amazon Lex: How It Works \(p. 3\)](#)), you must remove those references first.

### Note

If you get the `ResourceInUseException` exception, it provides an example reference that shows where the intent is referenced. To remove the reference to the intent, either update the bot or delete it. If you get the same exception when you attempt to delete the intent again, repeat until the intent has no references and the call to `DeleteIntent` is successful.

This operation requires permission for the `lex:DeleteIntent` action.

## Request Syntax

```
DELETE /intents/name HTTP/1.1
```

## URI Request Parameters

The request requires the following URI parameters.

### name (p. 199)

The name of the intent. The name is case sensitive.

Length Constraints: Minimum length of 1. Maximum length of 100.

Pattern: `^[a-zA-Z]+(_[a-zA-Z]+)*|([a-zA-Z]+_)*|_`

## Request Body

The request does not have a request body.

## Response Syntax

```
HTTP/1.1 204
```

## Response Elements

If the action is successful, the service sends back an HTTP 204 response with an empty HTTP body.

## Errors

### BadRequestException

The request is not well formed. For example, a value is invalid or a required field is missing. Check the field values, and try again.

HTTP Status Code: 400

### ConflictException

There was a conflict processing the request. Try your request again.

HTTP Status Code: 409

**InternalFailureException**

An internal Amazon Lex error occurred. Try your request again.

HTTP Status Code: 500

**LimitExceededException**

The request exceeded a limit. Try your request again.

HTTP Status Code: 429

**NotFoundException**

The resource specified in the request was not found. Check the resource and try again.

HTTP Status Code: 404

**ResourceInUseException**

The resource that you are attempting to delete is referred to by another resource. Use this information to remove references to the resource that you are trying to delete.

The body of the exception contains a JSON object that describes the resource.

```
{ "resourceType": BOT | BOTALIAS | BOTCHANNEL | INTENT,  
  "resourceReference": {  
    "name": string, "version": string } }
```

HTTP Status Code: 400

## See Also

For more information about using this API in one of the language-specific AWS SDKs, see the following:

- [AWS Command Line Interface](#)
- [AWS SDK for .NET](#)
- [AWS SDK for C++](#)
- [AWS SDK for Go](#)
- [AWS SDK for Java](#)
- [AWS SDK for JavaScript](#)
- [AWS SDK for PHP V3](#)
- [AWS SDK for Python](#)
- [AWS SDK for Ruby V2](#)

## DeleteIntentVersion

Service: Amazon Lex Model Building Service

Deletes a specific version of an intent. To delete all versions of a intent, use the [DeleteIntent \(p. 199\)](#) operation.

This operation requires permissions for the `lex:DeleteIntentVersion` action.

### Request Syntax

```
DELETE /intents/name/versions/version HTTP/1.1
```

### URI Request Parameters

The request requires the following URI parameters.

#### **name (p. 201)**

The name of the intent.

Length Constraints: Minimum length of 1. Maximum length of 100.

Pattern: `^[a-zA-Z]+((_[a-zA-Z]+)|([a-zA-Z]+_)*|_)`

#### **version (p. 201)**

The version of the intent to delete. You cannot delete the `$LATEST` version of the intent. To delete the `$LATEST` version, use the [DeleteIntent \(p. 199\)](#) operation.

Length Constraints: Minimum length of 1. Maximum length of 64.

Pattern: `[0-9]+`

### Request Body

The request does not have a request body.

### Response Syntax

```
HTTP/1.1 204
```

### Response Elements

If the action is successful, the service sends back an HTTP 204 response with an empty HTTP body.

### Errors

#### **BadRequestException**

The request is not well formed. For example, a value is invalid or a required field is missing. Check the field values, and try again.

HTTP Status Code: 400

#### **ConflictException**

There was a conflict processing the request. Try your request again.

HTTP Status Code: 409

**InternalFailureException**

An internal Amazon Lex error occurred. Try your request again.

HTTP Status Code: 500

**LimitExceededException**

The request exceeded a limit. Try your request again.

HTTP Status Code: 429

**NotFoundException**

The resource specified in the request was not found. Check the resource and try again.

HTTP Status Code: 404

**ResourceInUseException**

The resource that you are attempting to delete is referred to by another resource. Use this information to remove references to the resource that you are trying to delete.

The body of the exception contains a JSON object that describes the resource.

```
{ "resourceType": BOT | BOTALIAS | BOTCHANNEL | INTENT,  
  "resourceReference": {  
    "name": string, "version": string } }
```

HTTP Status Code: 400

## See Also

For more information about using this API in one of the language-specific AWS SDKs, see the following:

- [AWS Command Line Interface](#)
- [AWS SDK for .NET](#)
- [AWS SDK for C++](#)
- [AWS SDK for Go](#)
- [AWS SDK for Java](#)
- [AWS SDK for JavaScript](#)
- [AWS SDK for PHP V3](#)
- [AWS SDK for Python](#)
- [AWS SDK for Ruby V2](#)

## DeleteSlotType

Service: Amazon Lex Model Building Service

Deletes all versions of the slot type, including the `$LATEST` version. To delete a specific version of the slot type, use the [DeleteSlotTypeVersion \(p. 205\)](#) operation.

You can delete a version of a slot type only if it is not referenced. To delete a slot type that is referred to in one or more intents, you must remove those references first.

### Note

If you get the `ResourceInUseException` exception, the exception provides an example reference that shows the intent where the slot type is referenced. To remove the reference to the slot type, either update the intent or delete it. If you get the same exception when you attempt to delete the slot type again, repeat until the slot type has no references and the `DeleteSlotType` call is successful.

This operation requires permission for the `lex:DeleteSlotType` action.

## Request Syntax

```
DELETE /slottypes/name HTTP/1.1
```

## URI Request Parameters

The request requires the following URI parameters.

### name (p. 203)

The name of the slot type. The name is case sensitive.

Length Constraints: Minimum length of 1. Maximum length of 100.

Pattern: `^[a-zA-Z]+(\_[a-zA-Z]+)*|([a-zA-Z]+\_)|_`

## Request Body

The request does not have a request body.

## Response Syntax

```
HTTP/1.1 204
```

## Response Elements

If the action is successful, the service sends back an HTTP 204 response with an empty HTTP body.

## Errors

### BadRequestException

The request is not well formed. For example, a value is invalid or a required field is missing. Check the field values, and try again.

HTTP Status Code: 400

### ConflictException

There was a conflict processing the request. Try your request again.



HTTP Status Code: 409

**InternalFailureException**

An internal Amazon Lex error occurred. Try your request again.

HTTP Status Code: 500

**LimitExceededException**

The request exceeded a limit. Try your request again.

HTTP Status Code: 429

**NotFoundException**

The resource specified in the request was not found. Check the resource and try again.

HTTP Status Code: 404

**ResourceInUseException**

The resource that you are attempting to delete is referred to by another resource. Use this information to remove references to the resource that you are trying to delete.

The body of the exception contains a JSON object that describes the resource.

```
{ "resourceType": BOT | BOTALIAS | BOTCHANNEL | INTENT,  
  "resourceReference": {  
    "name": string, "version": string } }
```

HTTP Status Code: 400

## See Also

For more information about using this API in one of the language-specific AWS SDKs, see the following:

- [AWS Command Line Interface](#)
- [AWS SDK for .NET](#)
- [AWS SDK for C++](#)
- [AWS SDK for Go](#)
- [AWS SDK for Java](#)
- [AWS SDK for JavaScript](#)
- [AWS SDK for PHP V3](#)
- [AWS SDK for Python](#)
- [AWS SDK for Ruby V2](#)

## DeleteSlotTypeVersion

Service: Amazon Lex Model Building Service

Deletes a specific version of a slot type. To delete all versions of a slot type, use the [DeleteSlotType \(p. 203\)](#) operation.

This operation requires permissions for the `lex:DeleteSlotTypeVersion` action.

### Request Syntax

```
DELETE /slottypes/name/version/version HTTP/1.1
```

### URI Request Parameters

The request requires the following URI parameters.

#### **name (p. 205)**

The name of the slot type.

Length Constraints: Minimum length of 1. Maximum length of 100.

Pattern: `^[a-zA-Z]+((_[a-zA-Z]+)|([a-zA-Z]+_)*|_)`

#### **version (p. 205)**

The version of the slot type to delete. You cannot delete the `$LATEST` version of the slot type. To delete the `$LATEST` version, use the [DeleteSlotType \(p. 203\)](#) operation.

Length Constraints: Minimum length of 1. Maximum length of 64.

Pattern: `[0-9]+`

### Request Body

The request does not have a request body.

### Response Syntax

```
HTTP/1.1 204
```

### Response Elements

If the action is successful, the service sends back an HTTP 204 response with an empty HTTP body.

### Errors

#### **BadRequestException**

The request is not well formed. For example, a value is invalid or a required field is missing. Check the field values, and try again.

HTTP Status Code: 400

#### **ConflictException**

There was a conflict processing the request. Try your request again.

HTTP Status Code: 409

**InternalFailureException**

An internal Amazon Lex error occurred. Try your request again.

HTTP Status Code: 500

**LimitExceededException**

The request exceeded a limit. Try your request again.

HTTP Status Code: 429

**NotFoundException**

The resource specified in the request was not found. Check the resource and try again.

HTTP Status Code: 404

**ResourceInUseException**

The resource that you are attempting to delete is referred to by another resource. Use this information to remove references to the resource that you are trying to delete.

The body of the exception contains a JSON object that describes the resource.

```
{ "resourceType": BOT | BOTALIAS | BOTCHANNEL | INTENT,  
  "resourceReference": {  
    "name": string, "version": string } }
```

HTTP Status Code: 400

## See Also

For more information about using this API in one of the language-specific AWS SDKs, see the following:

- [AWS Command Line Interface](#)
- [AWS SDK for .NET](#)
- [AWS SDK for C++](#)
- [AWS SDK for Go](#)
- [AWS SDK for Java](#)
- [AWS SDK for JavaScript](#)
- [AWS SDK for PHP V3](#)
- [AWS SDK for Python](#)
- [AWS SDK for Ruby V2](#)

## DeleteUtterances

Service: Amazon Lex Model Building Service

Deletes stored utterances.

Amazon Lex stores the utterances that users send to your bot unless the `childDirected` field in the bot is set to `true`. Utterances are stored for 15 days for use with the [GetUtterancesView \(p. 258\)](#) operation, and then stored indefinitely for use in improving the ability of your bot to respond to user input.

Use the `DeleteStoredUtterances` operation to manually delete stored utterances for a specific user.

This operation requires permissions for the `lex:DeleteUtterances` action.

### Request Syntax

```
DELETE /bots/botName/utterances/userId HTTP/1.1
```

### URI Request Parameters

The request requires the following URI parameters.

#### **botName (p. 207)**

The name of the bot that stored the utterances.

Length Constraints: Minimum length of 2. Maximum length of 50.

Pattern: `^[a-zA-Z]+((_[a-zA-Z]+)|([a-zA-Z]+_)*|_)`

#### **userId (p. 207)**

The unique identifier for the user that made the utterances. This is the user ID that was sent in the [PostContent](#) or [PostText](#) operation request that contained the utterance.

Length Constraints: Minimum length of 2. Maximum length of 100.

### Request Body

The request does not have a request body.

### Response Syntax

```
HTTP/1.1 204
```

### Response Elements

If the action is successful, the service sends back an HTTP 204 response with an empty HTTP body.

### Errors

#### **BadRequestException**

The request is not well formed. For example, a value is invalid or a required field is missing. Check the field values, and try again.

HTTP Status Code: 400

**InternalFailureException**

An internal Amazon Lex error occurred. Try your request again.

HTTP Status Code: 500

**LimitExceededException**

The request exceeded a limit. Try your request again.

HTTP Status Code: 429

**NotFoundException**

The resource specified in the request was not found. Check the resource and try again.

HTTP Status Code: 404

## See Also

For more information about using this API in one of the language-specific AWS SDKs, see the following:

- [AWS Command Line Interface](#)
- [AWS SDK for .NET](#)
- [AWS SDK for C++](#)
- [AWS SDK for Go](#)
- [AWS SDK for Java](#)
- [AWS SDK for JavaScript](#)
- [AWS SDK for PHP V3](#)
- [AWS SDK for Python](#)
- [AWS SDK for Ruby V2](#)

## GetBot

Service: Amazon Lex Model Building Service

Returns metadata information for a specific bot. You must provide the bot name and the bot version or alias.

The GetBot operation requires permissions for the `lex:GetBot` action.

### Request Syntax

```
GET /bots/name/versions/versionoralias HTTP/1.1
```

### URI Request Parameters

The request requires the following URI parameters.

#### **name** (p. 209)

The name of the bot. The name is case sensitive.

Length Constraints: Minimum length of 2. Maximum length of 50.

Pattern: `^[a-zA-Z]+((_[a-zA-Z]+)*|([a-zA-Z]+_)*|_)`

#### **versionOrAlias** (p. 209)

The version or alias of the bot.

### Request Body

The request does not have a request body.

### Response Syntax

```
HTTP/1.1 200
Content-type: application/json

{
  "abortStatement": {
    "messages": [
      {
        "content": "string",
        "contentType": "string"
      }
    ],
    "responseCard": "string"
  },
  "checksum": "string",
  "childDirected": boolean,
  "clarificationPrompt": {
    "maxAttempts": number,
    "messages": [
      {
        "content": "string",
        "contentType": "string"
      }
    ],
    "responseCard": "string"
  },
  "createdDate": number,
```

```
"description": "string",
"failureReason": "string",
"idleSessionTTLInSeconds": number,
"intents": [
  {
    "intentName": "string",
    "intentVersion": "string"
  }
],
"lastUpdatedDate": number,
"locale": "string",
"name": "string",
"status": "string",
"version": "string",
"voiceId": "string"
}
```

## Response Elements

If the action is successful, the service sends back an HTTP 200 response.

The following data is returned in JSON format by the service.

### **abortStatement** (p. 209)

The message that Amazon Lex returns when the user elects to end the conversation without completing it. For more information, see [PutBot](#) (p. 261).

Type: [Statement](#) (p. 323) object

### **checksum** (p. 209)

Checksum of the bot used to identify a specific revision of the bot's `$LATEST` version.

Type: String

### **childDirected** (p. 209)

For each Amazon Lex bot created with the Amazon Lex Model Building Service, you must specify whether your use of Amazon Lex is related to a website, program, or other application that is directed or targeted, in whole or in part, to children under age 13 and subject to the Children's Online Privacy Protection Act (COPPA) by specifying `true` or `false` in the `childDirected` field. By specifying `true` in the `childDirected` field, you confirm that your use of Amazon Lex is related to a website, program, or other application that is directed or targeted, in whole or in part, to children under age 13 and subject to COPPA. By specifying `false` in the `childDirected` field, you confirm that your use of Amazon Lex **is not** related to a website, program, or other application that is directed or targeted, in whole or in part, to children under age 13 and subject to COPPA. You may not specify a default value for the `childDirected` field that does not accurately reflect whether your use of Amazon Lex is related to a website, program, or other application that is directed or targeted, in whole or in part, to children under age 13 and subject to COPPA.

If your use of Amazon Lex relates to a website, program, or other application that is directed in whole or in part, to children under age 13, you must obtain any required verifiable parental consent under COPPA. For information regarding the use of Amazon Lex in connection with websites, programs, or other applications that are directed or targeted, in whole or in part, to children under age 13, see the [Amazon Lex FAQ](#).

Type: Boolean

### **clarificationPrompt** (p. 209)

The message Amazon Lex uses when it doesn't understand the user's request. For more information, see [PutBot](#) (p. 261).

Type: [Prompt \(p. 317\)](#) object

**[createdDate \(p. 209\)](#)**

The date that the bot was created.

Type: Timestamp

**[description \(p. 209\)](#)**

A description of the bot.

Type: String

Length Constraints: Minimum length of 0. Maximum length of 200.

**[failureReason \(p. 209\)](#)**

If `status` is `FAILED`, Amazon Lex explains why it failed to build the bot.

Type: String

**[idleSessionTTLInSeconds \(p. 209\)](#)**

The maximum time in seconds that Amazon Lex retains the data gathered in a conversation. For more information, see [PutBot \(p. 261\)](#).

Type: Integer

Valid Range: Minimum value of 60. Maximum value of 86400.

**[intents \(p. 209\)](#)**

An array of `intent` objects. For more information, see [PutBot \(p. 261\)](#).

Type: Array of [Intent \(p. 313\)](#) objects

Array Members: Minimum number of 1 item. Maximum number of 100 items.

**[lastUpdatedDate \(p. 209\)](#)**

The date that the bot was updated. When you create a resource, the creation date and last updated date are the same.

Type: Timestamp

**[locale \(p. 209\)](#)**

The target locale for the bot.

Type: String

Valid Values: `en-US`

**[name \(p. 209\)](#)**

The name of the bot.

Type: String

Length Constraints: Minimum length of 2. Maximum length of 50.

Pattern: `^[a-zA-Z]+((_[a-zA-Z]+)*|([a-zA-Z]+_)*|_)`

**[status \(p. 209\)](#)**

The status of the bot. If the bot is ready to run, the status is `READY`. If there was a problem with building the bot, the status is `FAILED` and the `failureReason` explains why the bot did not build. If the bot was saved but not built, the status is `NOT_BUILT`.



Type: String

Valid Values: BUILDING | READY | FAILED | NOT\_BUILT

**version (p. 209)**

The version of the bot. For a new bot, the version is always \$LATEST.

Type: String

Length Constraints: Minimum length of 1. Maximum length of 64.

Pattern: \ \$LATEST | [0-9] +

**voiceId (p. 209)**

The Amazon Polly voice ID that Amazon Lex uses for voice interaction with the user. For more information, see [PutBot \(p. 261\)](#).

Type: String

## Errors

### **BadRequestException**

The request is not well formed. For example, a value is invalid or a required field is missing. Check the field values, and try again.

HTTP Status Code: 400

### **InternalFailureException**

An internal Amazon Lex error occurred. Try your request again.

HTTP Status Code: 500

### **LimitExceededException**

The request exceeded a limit. Try your request again.

HTTP Status Code: 429

### **NotFoundException**

The resource specified in the request was not found. Check the resource and try again.

HTTP Status Code: 404

## See Also

For more information about using this API in one of the language-specific AWS SDKs, see the following:

- [AWS Command Line Interface](#)
- [AWS SDK for .NET](#)
- [AWS SDK for C++](#)
- [AWS SDK for Go](#)
- [AWS SDK for Java](#)
- [AWS SDK for JavaScript](#)
- [AWS SDK for PHP V3](#)
- [AWS SDK for Python](#)

- [AWS SDK for Ruby V2](#)

## GetBotAlias

Service: Amazon Lex Model Building Service

Returns information about an Amazon Lex bot alias. For more information about aliases, see [Versioning and Aliases \(p. 81\)](#).

This operation requires permissions for the `lex:GetBotAlias` action.

### Request Syntax

```
GET /bots/botName/aliases/name HTTP/1.1
```

### URI Request Parameters

The request requires the following URI parameters.

#### **botName** (p. 214)

The name of the bot.

Length Constraints: Minimum length of 2. Maximum length of 50.

Pattern: `^[a-zA-Z]+(("[a-zA-Z"]+)|([a-zA-Z_]+)|_)`

#### **name** (p. 214)

The name of the bot alias. The name is case sensitive.

Length Constraints: Minimum length of 1. Maximum length of 100.

Pattern: `^[a-zA-Z]+(("[a-zA-Z"]+)|([a-zA-Z_]+)|_)`

### Request Body

The request does not have a request body.

### Response Syntax

```
HTTP/1.1 200
Content-type: application/json

{
  "botName": "string",
  "botVersion": "string",
  "checksum": "string",
  "createdDate": number,
  "description": "string",
  "lastUpdatedDate": number,
  "name": "string"
}
```

### Response Elements

If the action is successful, the service sends back an HTTP 200 response.

The following data is returned in JSON format by the service.

#### **botName** (p. 214)

The name of the bot that the alias points to.

Type: String

Length Constraints: Minimum length of 2. Maximum length of 50.

Pattern: `^[a-zA-Z]+((_[a-zA-Z]+)*|([a-zA-Z]+_)*|_)`

**botVersion (p. 214)**

The version of the bot that the alias points to.

Type: String

Length Constraints: Minimum length of 1. Maximum length of 64.

Pattern: `\$LATEST|[0-9]+`

**checksum (p. 214)**

Checksum of the bot alias.

Type: String

**createdDate (p. 214)**

The date that the bot alias was created.

Type: Timestamp

**description (p. 214)**

A description of the bot alias.

Type: String

Length Constraints: Minimum length of 0. Maximum length of 200.

**lastUpdatedDate (p. 214)**

The date that the bot alias was updated. When you create a resource, the creation date and the last updated date are the same.

Type: Timestamp

**name (p. 214)**

The name of the bot alias.

Type: String

Length Constraints: Minimum length of 1. Maximum length of 100.

Pattern: `^[a-zA-Z]+((_[a-zA-Z]+)*|([a-zA-Z]+_)*|_)`

## Errors

### BadRequestException

The request is not well formed. For example, a value is invalid or a required field is missing. Check the field values, and try again.

HTTP Status Code: 400

### InternalFailureException

An internal Amazon Lex error occurred. Try your request again.

HTTP Status Code: 500

**LimitExceededException**

The request exceeded a limit. Try your request again.

HTTP Status Code: 429

**NotFoundException**

The resource specified in the request was not found. Check the resource and try again.

HTTP Status Code: 404

## See Also

For more information about using this API in one of the language-specific AWS SDKs, see the following:

- [AWS Command Line Interface](#)
- [AWS SDK for .NET](#)
- [AWS SDK for C++](#)
- [AWS SDK for Go](#)
- [AWS SDK for Java](#)
- [AWS SDK for JavaScript](#)
- [AWS SDK for PHP V3](#)
- [AWS SDK for Python](#)
- [AWS SDK for Ruby V2](#)

## GetBotAliases

Service: Amazon Lex Model Building Service

Returns a list of aliases for a specified Amazon Lex bot.

This operation requires permissions for the `lex:GetBotAliases` action.

### Request Syntax

```
GET /bots/botName/aliases/?  
maxResults=maxResults&nameContains=nameContains&nextToken=nextToken HTTP/1.1
```

### URI Request Parameters

The request requires the following URI parameters.

#### **botName** (p. 217)

The name of the bot.

Length Constraints: Minimum length of 2. Maximum length of 50.

Pattern: `^[a-zA-Z]+((_[a-zA-Z]+)*|([a-zA-Z]+_)*|_)`

#### **maxResults** (p. 217)

The maximum number of aliases to return in the response. The default is 50. .

Valid Range: Minimum value of 1. Maximum value of 50.

#### **nameContains** (p. 217)

Substring to match in bot alias names. An alias will be returned if any part of its name matches the substring. For example, "xyz" matches both "xyzabc" and "abcxyz."

Length Constraints: Minimum length of 1. Maximum length of 100.

Pattern: `^[a-zA-Z]+((_[a-zA-Z]+)*|([a-zA-Z]+_)*|_)`

#### **nextToken** (p. 217)

A pagination token for fetching the next page of aliases. If the response to this call is truncated, Amazon Lex returns a pagination token in the response. To fetch the next page of aliases, specify the pagination token in the next request.

### Request Body

The request does not have a request body.

### Response Syntax

```
HTTP/1.1 200  
Content-type: application/json  
  
{  
  "BotAliases": [  
    {  
      "botName": "string",  
      "botVersion": "string",  
      "checksum": "string",
```

```
        "createdDate": number,  
        "description": "string",  
        "lastUpdatedDate": number,  
        "name": "string"  
    }  
  ],  
  "nextToken": "string"  
}
```

## Response Elements

If the action is successful, the service sends back an HTTP 200 response.

The following data is returned in JSON format by the service.

### BotAliases (p. 217)

An array of `BotAliasMetadata` objects, each describing a bot alias.

Type: Array of `BotAliasMetadata` (p. 300) objects

### nextToken (p. 217)

A pagination token for fetching next page of aliases. If the response to this call is truncated, Amazon Lex returns a pagination token in the response. To fetch the next page of aliases, specify the pagination token in the next request.

Type: String

## Errors

### BadRequestException

The request is not well formed. For example, a value is invalid or a required field is missing. Check the field values, and try again.

HTTP Status Code: 400

### InternalFailureException

An internal Amazon Lex error occurred. Try your request again.

HTTP Status Code: 500

### LimitExceededException

The request exceeded a limit. Try your request again.

HTTP Status Code: 429

## See Also

For more information about using this API in one of the language-specific AWS SDKs, see the following:

- [AWS Command Line Interface](#)
- [AWS SDK for .NET](#)
- [AWS SDK for C++](#)
- [AWS SDK for Go](#)
- [AWS SDK for Java](#)

- [AWS SDK for JavaScript](#)
- [AWS SDK for PHP V3](#)
- [AWS SDK for Python](#)
- [AWS SDK for Ruby V2](#)



## GetBotChannelAssociation

Service: Amazon Lex Model Building Service

Returns information about the association between an Amazon Lex bot and a messaging platform.

This operation requires permissions for the `lex:GetBotChannelAssociation` action.

### Request Syntax

```
GET /bots/botName/aliases/aliasName/channels/name HTTP/1.1
```

### URI Request Parameters

The request requires the following URI parameters.

#### **botAlias** (p. 220)

An alias pointing to the specific version of the Amazon Lex bot to which this association is being made.

Length Constraints: Minimum length of 1. Maximum length of 100.

Pattern: `^[a-zA-Z]+((_[a-zA-Z]+)*|([a-zA-Z]+_)*|_)`

#### **botName** (p. 220)

The name of the Amazon Lex bot.

Length Constraints: Minimum length of 2. Maximum length of 50.

Pattern: `^[a-zA-Z]+((_[a-zA-Z]+)*|([a-zA-Z]+_)*|_)`

#### **name** (p. 220)

The name of the association between the bot and the channel. The name is case sensitive.

Length Constraints: Minimum length of 1. Maximum length of 100.

Pattern: `^[a-zA-Z]+((_[a-zA-Z]+)*|([a-zA-Z]+_)*|_)`

### Request Body

The request does not have a request body.

### Response Syntax

```
HTTP/1.1 200  
Content-type: application/json
```

```
{  
  "botAlias": "string",  
  "botConfiguration": {  
    "string": "string"  
  },  
  "botName": "string",  
  "createdDate": number,  
  "description": "string",  
  "name": "string",  
  "type": "string"
```

```
}
```

## Response Elements

If the action is successful, the service sends back an HTTP 200 response.

The following data is returned in JSON format by the service.

### **botAlias** (p. 220)

An alias pointing to the specific version of the Amazon Lex bot to which this association is being made.

Type: String

Length Constraints: Minimum length of 1. Maximum length of 100.

Pattern: `^[a-zA-Z]+((_[a-zA-Z]+)*|([a-zA-Z]+_)*|_)`

### **botConfiguration** (p. 220)

Provides information that the messaging platform needs to communicate with the Amazon Lex bot.

Type: String to string map

### **botName** (p. 220)

The name of the Amazon Lex bot.

Type: String

Length Constraints: Minimum length of 2. Maximum length of 50.

Pattern: `^[a-zA-Z]+((_[a-zA-Z]+)*|([a-zA-Z]+_)*|_)`

### **createdDate** (p. 220)

The date that the association between the bot and the channel was created.

Type: Timestamp

### **description** (p. 220)

A description of the association between the bot and the channel.

Type: String

Length Constraints: Minimum length of 0. Maximum length of 200.

### **name** (p. 220)

The name of the association between the bot and the channel.

Type: String

Length Constraints: Minimum length of 1. Maximum length of 100.

Pattern: `^[a-zA-Z]+((_[a-zA-Z]+)*|([a-zA-Z]+_)*|_)`

### **type** (p. 220)

The type of the messaging platform.

Type: String

Valid Values: `Facebook` | `Slack` | `Twilio-Sms`

## Errors

### **BadRequestException**

The request is not well formed. For example, a value is invalid or a required field is missing. Check the field values, and try again.

HTTP Status Code: 400

### **InternalFailureException**

An internal Amazon Lex error occurred. Try your request again.

HTTP Status Code: 500

### **LimitExceededException**

The request exceeded a limit. Try your request again.

HTTP Status Code: 429

### **NotFoundException**

The resource specified in the request was not found. Check the resource and try again.

HTTP Status Code: 404

## See Also

For more information about using this API in one of the language-specific AWS SDKs, see the following:

- [AWS Command Line Interface](#)
- [AWS SDK for .NET](#)
- [AWS SDK for C++](#)
- [AWS SDK for Go](#)
- [AWS SDK for Java](#)
- [AWS SDK for JavaScript](#)
- [AWS SDK for PHP V3](#)
- [AWS SDK for Python](#)
- [AWS SDK for Ruby V2](#)

## GetBotChannelAssociations

Service: Amazon Lex Model Building Service

Returns a list of all of the channels associated with the specified bot.

The `GetBotChannelAssociations` operation requires permissions for the `lex:GetBotChannelAssociations` action.

### Request Syntax

```
GET /bots/botName/aliases/aliasName/channels/?  
maxResults=maxResults&nameContains=nameContains&nextToken=nextToken HTTP/1.1
```

### URI Request Parameters

The request requires the following URI parameters.

#### **botAlias** (p. 223)

An alias pointing to the specific version of the Amazon Lex bot to which this association is being made.

Length Constraints: Minimum length of 1. Maximum length of 100.

Pattern: `^(-|^([a-zA-Z]+)(\[a-zA-Z\]+)*|([a-zA-Z]+_)*))$`

#### **botName** (p. 223)

The name of the Amazon Lex bot in the association.

Length Constraints: Minimum length of 2. Maximum length of 50.

Pattern: `^[a-zA-Z]+(\[a-zA-Z\]+)*|([a-zA-Z]+_)*|_`

#### **maxResults** (p. 223)

The maximum number of associations to return in the response. The default is 50.

Valid Range: Minimum value of 1. Maximum value of 50.

#### **nameContains** (p. 223)

Substring to match in channel association names. An association will be returned if any part of its name matches the substring. For example, "xyz" matches both "xyzabc" and "abcxyz." To return all bot channel associations, use a hyphen ("-") as the `nameContains` parameter.

Length Constraints: Minimum length of 1. Maximum length of 100.

Pattern: `^[a-zA-Z]+(\[a-zA-Z\]+)*|([a-zA-Z]+_)*|_`

#### **nextToken** (p. 223)

A pagination token for fetching the next page of associations. If the response to this call is truncated, Amazon Lex returns a pagination token in the response. To fetch the next page of associations, specify the pagination token in the next request.

### Request Body

The request does not have a request body.

## Response Syntax

```
HTTP/1.1 200
Content-type: application/json

{
  "botChannelAssociations": [
    {
      "botAlias": "string",
      "botConfiguration": {
        "string": "string"
      },
      "botName": "string",
      "createdDate": number,
      "description": "string",
      "name": "string",
      "type": "string"
    }
  ],
  "nextToken": "string"
}
```

## Response Elements

If the action is successful, the service sends back an HTTP 200 response.

The following data is returned in JSON format by the service.

### **botChannelAssociations** (p. 224)

An array of objects, one for each association, that provides information about the Amazon Lex bot and its association with the channel.

Type: Array of [BotChannelAssociation](#) (p. 302) objects

### **nextToken** (p. 224)

A pagination token that fetches the next page of associations. If the response to this call is truncated, Amazon Lex returns a pagination token in the response. To fetch the next page of associations, specify the pagination token in the next request.

Type: String

## Errors

### **BadRequestException**

The request is not well formed. For example, a value is invalid or a required field is missing. Check the field values, and try again.

HTTP Status Code: 400

### **InternalFailureException**

An internal Amazon Lex error occurred. Try your request again.

HTTP Status Code: 500

### **LimitExceededException**

The request exceeded a limit. Try your request again.

HTTP Status Code: 429

## See Also

For more information about using this API in one of the language-specific AWS SDKs, see the following:

- [AWS Command Line Interface](#)
- [AWS SDK for .NET](#)
- [AWS SDK for C++](#)
- [AWS SDK for Go](#)
- [AWS SDK for Java](#)
- [AWS SDK for JavaScript](#)
- [AWS SDK for PHP V3](#)
- [AWS SDK for Python](#)
- [AWS SDK for Ruby V2](#)

## GetBots

Service: Amazon Lex Model Building Service

Returns bot information as follows:

- If you provide the `nameContains` field, the response includes information for the `$LATEST` version of all bots whose name contains the specified string.
- If you don't specify the `nameContains` field, the operation returns information about the `$LATEST` version of all of your bots.

This operation requires permission for the `lex:GetBots` action.

## Request Syntax

```
GET /bots/?maxResults=maxResults&nameContains=nameContains&nextToken=nextToken HTTP/1.1
```

## URI Request Parameters

The request requires the following URI parameters.

### `maxResults` (p. 226)

The maximum number of bots to return in the response that the request will return. The default is 10.

Valid Range: Minimum value of 1. Maximum value of 50.

### `nameContains` (p. 226)

Substring to match in bot names. A bot will be returned if any part of its name matches the substring. For example, "xyz" matches both "xyzabc" and "abcxyz."

Length Constraints: Minimum length of 2. Maximum length of 50.

Pattern: `^[a-zA-Z]+((_[a-zA-Z]+)*|([a-zA-Z]+_)*|_)`

### `nextToken` (p. 226)

A pagination token that fetches the next page of bots. If the response to this call is truncated, Amazon Lex returns a pagination token in the response. To fetch the next page of bots, specify the pagination token in the next request.

## Request Body

The request does not have a request body.

## Response Syntax

```
HTTP/1.1 200
Content-type: application/json

{
  "bots": [
    {
      "createdDate": number,
      "description": "string",
      "lastUpdatedDate": number,
      "name": "string",
```

```
        "status": "string",  
        "version": "string"  
    },  
    ],  
    "nextToken": "string"  
}
```

## Response Elements

If the action is successful, the service sends back an HTTP 200 response.

The following data is returned in JSON format by the service.

### **bots** (p. 226)

An array of `botMetadata` objects, with one entry for each bot.

Type: Array of [BotMetadata](#) (p. 304) objects

### **nextToken** (p. 226)

If the response is truncated, it includes a pagination token that you can specify in your next request to fetch the next page of bots.

Type: String

## Errors

### **BadRequestException**

The request is not well formed. For example, a value is invalid or a required field is missing. Check the field values, and try again.

HTTP Status Code: 400

### **InternalFailureException**

An internal Amazon Lex error occurred. Try your request again.

HTTP Status Code: 500

### **LimitExceededException**

The request exceeded a limit. Try your request again.

HTTP Status Code: 429

### **NotFoundException**

The resource specified in the request was not found. Check the resource and try again.

HTTP Status Code: 404

## See Also

For more information about using this API in one of the language-specific AWS SDKs, see the following:

- [AWS Command Line Interface](#)
- [AWS SDK for .NET](#)
- [AWS SDK for C++](#)
- [AWS SDK for Go](#)



- [AWS SDK for Java](#)
- [AWS SDK for JavaScript](#)
- [AWS SDK for PHP V3](#)
- [AWS SDK for Python](#)
- [AWS SDK for Ruby V2](#)

## GetBotVersions

Service: Amazon Lex Model Building Service

Gets information about all of the versions of a bot.

The `GetBotVersions` operation returns a `BotMetadata` object for each version of a bot. For example, if a bot has three numbered versions, the `GetBotVersions` operation returns four `BotMetadata` objects in the response, one for each numbered version and one for the `$LATEST` version.

The `GetBotVersions` operation always returns at least one version, the `$LATEST` version.

This operation requires permissions for the `lex:GetBotVersions` action.

## Request Syntax

```
GET /bots/name/versions/?maxResults=maxResults&nextToken=nextToken HTTP/1.1
```

## URI Request Parameters

The request requires the following URI parameters.

### `maxResults` (p. 229)

The maximum number of bot versions to return in the response. The default is 10.

Valid Range: Minimum value of 1. Maximum value of 50.

### `name` (p. 229)

The name of the bot for which versions should be returned.

Length Constraints: Minimum length of 2. Maximum length of 50.

Pattern: `^[a-zA-Z]+((_[a-zA-Z]+)*|([a-zA-Z]+_)*|_)`

### `nextToken` (p. 229)

A pagination token for fetching the next page of bot versions. If the response to this call is truncated, Amazon Lex returns a pagination token in the response. To fetch the next page of versions, specify the pagination token in the next request.

## Request Body

The request does not have a request body.

## Response Syntax

```
HTTP/1.1 200  
Content-type: application/json
```

```
{  
  "bots": [  
    {  
      "createdDate": number,  
      "description": "string",  
      "lastUpdatedDate": number,  
      "name": "string",  
      "status": "string",  
      "version": "string"  
    }  
  ]  
}
```

```
  ],  
  "nextToken": "string"  
}
```

## Response Elements

If the action is successful, the service sends back an HTTP 200 response.

The following data is returned in JSON format by the service.

### **bots** (p. 229)

An array of `BotMetadata` objects, one for each numbered version of the bot plus one for the `$LATEST` version.

Type: Array of [BotMetadata](#) (p. 304) objects

### **nextToken** (p. 229)

A pagination token for fetching the next page of bot versions. If the response to this call is truncated, Amazon Lex returns a pagination token in the response. To fetch the next page of versions, specify the pagination token in the next request.

Type: String

## Errors

### **BadRequestException**

The request is not well formed. For example, a value is invalid or a required field is missing. Check the field values, and try again.

HTTP Status Code: 400

### **InternalFailureException**

An internal Amazon Lex error occurred. Try your request again.

HTTP Status Code: 500

### **LimitExceededException**

The request exceeded a limit. Try your request again.

HTTP Status Code: 429

### **NotFoundException**

The resource specified in the request was not found. Check the resource and try again.

HTTP Status Code: 404

## See Also

For more information about using this API in one of the language-specific AWS SDKs, see the following:

- [AWS Command Line Interface](#)
- [AWS SDK for .NET](#)
- [AWS SDK for C++](#)
- [AWS SDK for Go](#)

- [AWS SDK for Java](#)
- [AWS SDK for JavaScript](#)
- [AWS SDK for PHP V3](#)
- [AWS SDK for Python](#)
- [AWS SDK for Ruby V2](#)

## GetBuiltinIntent

Service: Amazon Lex Model Building Service

Returns information about a built-in intent.

This operation requires permission for the `lex:GetBuiltinIntent` action.

### Request Syntax

```
GET /builtins/intents/signature HTTP/1.1
```

### URI Request Parameters

The request requires the following URI parameters.

#### **signature** (p. 232)

The unique identifier for a built-in intent. To find the signature for an intent, see [Standard Built-in Intents](#) in the *Alexa Skills Kit*.

### Request Body

The request does not have a request body.

### Response Syntax

```
HTTP/1.1 200
Content-type: application/json

{
  "signature": "string",
  "slots": [
    {
      "name": "string"
    }
  ],
  "supportedLocales": [ "string" ]
}
```

### Response Elements

If the action is successful, the service sends back an HTTP 200 response.

The following data is returned in JSON format by the service.

#### **signature** (p. 232)

The unique identifier for a built-in intent.

Type: String

#### **slots** (p. 232)

An array of `BuiltinIntentSlot` objects, one entry for each slot type in the intent.

Type: Array of [BuiltinIntentSlot](#) (p. 307) objects

#### **supportedLocales** (p. 232)

A list of locales that the intent supports.

Type: Array of strings

Valid Values: `en-US`

## Errors

### **BadRequestException**

The request is not well formed. For example, a value is invalid or a required field is missing. Check the field values, and try again.

HTTP Status Code: 400

### **InternalFailureException**

An internal Amazon Lex error occurred. Try your request again.

HTTP Status Code: 500

### **LimitExceededException**

The request exceeded a limit. Try your request again.

HTTP Status Code: 429

### **NotFoundException**

The resource specified in the request was not found. Check the resource and try again.

HTTP Status Code: 404

## See Also

For more information about using this API in one of the language-specific AWS SDKs, see the following:

- [AWS Command Line Interface](#)
- [AWS SDK for .NET](#)
- [AWS SDK for C++](#)
- [AWS SDK for Go](#)
- [AWS SDK for Java](#)
- [AWS SDK for JavaScript](#)
- [AWS SDK for PHP V3](#)
- [AWS SDK for Python](#)
- [AWS SDK for Ruby V2](#)

## GetBuiltinIntents

Service: Amazon Lex Model Building Service

Gets a list of built-in intents that meet the specified criteria.

This operation requires permission for the `lex:GetBuiltinIntents` action.

### Request Syntax

```
GET /builtins/intents/?
locale=locale&maxResults=maxResults&nextToken=nextToken&signatureContains=signatureContains
HTTP/1.1
```

### URI Request Parameters

The request requires the following URI parameters.

#### **locale** (p. 234)

A list of locales that the intent supports.

Valid Values: `en-US`

#### **maxResults** (p. 234)

The maximum number of intents to return in the response. The default is 10.

Valid Range: Minimum value of 1. Maximum value of 50.

#### **nextToken** (p. 234)

A pagination token that fetches the next page of intents. If this API call is truncated, Amazon Lex returns a pagination token in the response. To fetch the next page of intents, use the pagination token in the next request.

#### **signatureContains** (p. 234)

Substring to match in built-in intent signatures. An intent will be returned if any part of its signature matches the substring. For example, "xyz" matches both "xyzabc" and "abcxyz." To find the signature for an intent, see [Standard Built-in Intents](#) in the *Alexa Skills Kit*.

### Request Body

The request does not have a request body.

### Response Syntax

```
HTTP/1.1 200
Content-type: application/json

{
  "intents": [
    {
      "signature": "string",
      "supportedLocales": [ "string" ]
    }
  ],
  "nextToken": "string"
}
```

## Response Elements

If the action is successful, the service sends back an HTTP 200 response.

The following data is returned in JSON format by the service.

### **intents** (p. 234)

An array of `builtinIntentMetadata` objects, one for each intent in the response.

Type: Array of [BuiltinIntentMetadata](#) (p. 306) objects

### **nextToken** (p. 234)

A pagination token that fetches the next page of intents. If the response to this API call is truncated, Amazon Lex returns a pagination token in the response. To fetch the next page of intents, specify the pagination token in the next request.

Type: String

## Errors

### **BadRequestException**

The request is not well formed. For example, a value is invalid or a required field is missing. Check the field values, and try again.

HTTP Status Code: 400

### **InternalFailureException**

An internal Amazon Lex error occurred. Try your request again.

HTTP Status Code: 500

### **LimitExceededException**

The request exceeded a limit. Try your request again.

HTTP Status Code: 429

## See Also

For more information about using this API in one of the language-specific AWS SDKs, see the following:

- [AWS Command Line Interface](#)
- [AWS SDK for .NET](#)
- [AWS SDK for C++](#)
- [AWS SDK for Go](#)
- [AWS SDK for Java](#)
- [AWS SDK for JavaScript](#)
- [AWS SDK for PHP V3](#)
- [AWS SDK for Python](#)
- [AWS SDK for Ruby V2](#)



## GetBuiltinSlotTypes

Service: Amazon Lex Model Building Service

Gets a list of built-in slot types that meet the specified criteria.

For a list of built-in slot types, see [Slot Type Reference](#) in the *Alexa Skills Kit*.

This operation requires permission for the `lex:GetBuiltinSlotTypes` action.

### Request Syntax

```
GET /builtins/slottypes/?  
locale=locale&maxResults=maxResults&nextToken=nextToken&signatureContains=signatureContains  
HTTP/1.1
```

### URI Request Parameters

The request requires the following URI parameters.

#### **locale** (p. 236)

A list of locales that the slot type supports.

Valid Values: `en-US`

#### **maxResults** (p. 236)

The maximum number of slot types to return in the response. The default is 10.

Valid Range: Minimum value of 1. Maximum value of 50.

#### **nextToken** (p. 236)

A pagination token that fetches the next page of slot types. If the response to this API call is truncated, Amazon Lex returns a pagination token in the response. To fetch the next page of slot types, specify the pagination token in the next request.

#### **signatureContains** (p. 236)

Substring to match in built-in slot type signatures. A slot type will be returned if any part of its signature matches the substring. For example, "xyz" matches both "xyzabc" and "abcxyz."

### Request Body

The request does not have a request body.

### Response Syntax

```
HTTP/1.1 200  
Content-type: application/json  
  
{  
  "nextToken": "string",  
  "slotTypes": [  
    {  
      "signature": "string",  
      "supportedLocales": [ "string" ]  
    }  
  ]  
}
```

## Response Elements

If the action is successful, the service sends back an HTTP 200 response.

The following data is returned in JSON format by the service.

### **nextToken** (p. 236)

If the response is truncated, the response includes a pagination token that you can use in your next request to fetch the next page of slot types.

Type: String

### **slotTypes** (p. 236)

An array of `BuiltInSlotTypeMetadata` objects, one entry for each slot type returned.

Type: Array of [BuiltInSlotTypeMetadata](#) (p. 308) objects

## Errors

### **BadRequestException**

The request is not well formed. For example, a value is invalid or a required field is missing. Check the field values, and try again.

HTTP Status Code: 400

### **InternalFailureException**

An internal Amazon Lex error occurred. Try your request again.

HTTP Status Code: 500

### **LimitExceededException**

The request exceeded a limit. Try your request again.

HTTP Status Code: 429

## See Also

For more information about using this API in one of the language-specific AWS SDKs, see the following:

- [AWS Command Line Interface](#)
- [AWS SDK for .NET](#)
- [AWS SDK for C++](#)
- [AWS SDK for Go](#)
- [AWS SDK for Java](#)
- [AWS SDK for JavaScript](#)
- [AWS SDK for PHP V3](#)
- [AWS SDK for Python](#)
- [AWS SDK for Ruby V2](#)

## GetIntent

Service: Amazon Lex Model Building Service

Returns information about an intent. In addition to the intent name, you must specify the intent version.

This operation requires permissions to perform the `lex:GetIntent` action.

### Request Syntax

```
GET /intents/name/versions/version HTTP/1.1
```

### URI Request Parameters

The request requires the following URI parameters.

#### **name** (p. 238)

The name of the intent. The name is case sensitive.

Length Constraints: Minimum length of 1. Maximum length of 100.

Pattern: `^[a-zA-Z]+((_[a-zA-Z]+)*|([a-zA-Z]+_)*|_)`

#### **version** (p. 238)

The version of the intent.

Length Constraints: Minimum length of 1. Maximum length of 64.

Pattern: `\$LATEST|[0-9]+`

### Request Body

The request does not have a request body.

### Response Syntax

```
HTTP/1.1 200
Content-type: application/json

{
  "checksum": "string",
  "conclusionStatement": {
    "messages": [
      {
        "content": "string",
        "contentType": "string"
      }
    ],
    "responseCard": "string"
  },
  "confirmationPrompt": {
    "maxAttempts": number,
    "messages": [
      {
        "content": "string",
        "contentType": "string"
      }
    ],
    "responseCard": "string"
  }
}
```

```
    },
    "createdDate": number,
    "description": "string",
    "dialogCodeHook": {
      "messageVersion": "string",
      "uri": "string"
    },
  },
  "followUpPrompt": {
    "prompt": {
      "maxAttempts": number,
      "messages": [
        {
          "content": "string",
          "contentType": "string"
        }
      ],
      "responseCard": "string"
    },
    "rejectionStatement": {
      "messages": [
        {
          "content": "string",
          "contentType": "string"
        }
      ],
      "responseCard": "string"
    }
  },
  "fulfillmentActivity": {
    "codeHook": {
      "messageVersion": "string",
      "uri": "string"
    },
    "type": "string"
  },
  "lastUpdatedDate": number,
  "name": "string",
  "parentIntentSignature": "string",
  "rejectionStatement": {
    "messages": [
      {
        "content": "string",
        "contentType": "string"
      }
    ],
    "responseCard": "string"
  },
  "sampleUtterances": [ "string" ],
  "slots": [
    {
      "description": "string",
      "name": "string",
      "priority": number,
      "responseCard": "string",
      "sampleUtterances": [ "string" ],
      "slotConstraint": "string",
      "slotType": "string",
      "slotTypeVersion": "string",
      "valueElicitationPrompt": {
        "maxAttempts": number,
        "messages": [
          {
            "content": "string",
            "contentType": "string"
          }
        ]
      }
    }
  ],
```

```
        "responseCard": "string"
      }
    }
  ],
  "version": "string"
}
```

## Response Elements

If the action is successful, the service sends back an HTTP 200 response.

The following data is returned in JSON format by the service.

### **checksum** (p. 238)

Checksum of the intent.

Type: String

### **conclusionStatement** (p. 238)

After the Lambda function specified in the `fulfillmentActivity` element fulfills the intent, Amazon Lex conveys this statement to the user.

Type: [Statement](#) (p. 323) object

### **confirmationPrompt** (p. 238)

If defined in the bot, Amazon Lex uses prompt to confirm the intent before fulfilling the user's request. For more information, see [PutIntent](#) (p. 273).

Type: [Prompt](#) (p. 317) object

### **createdDate** (p. 238)

The date that the intent was created.

Type: Timestamp

### **description** (p. 238)

A description of the intent.

Type: String

Length Constraints: Minimum length of 0. Maximum length of 200.

### **dialogCodeHook** (p. 238)

If defined in the bot, Amazon Amazon Lex invokes this Lambda function for each user input. For more information, see [PutIntent](#) (p. 273).

Type: [CodeHook](#) (p. 309) object

### **followUpPrompt** (p. 238)

If defined in the bot, Amazon Lex uses this prompt to solicit additional user activity after the intent is fulfilled. For more information, see [PutIntent](#) (p. 273).

Type: [FollowUpPrompt](#) (p. 311) object

### **fulfillmentActivity** (p. 238)

Describes how the intent is fulfilled. For more information, see [PutIntent](#) (p. 273).

Type: [FulfillmentActivity](#) (p. 312) object

### **lastUpdatedDate (p. 238)**

The date that the intent was updated. When you create a resource, the creation date and the last updated date are the same.

Type: Timestamp

### **name (p. 238)**

The name of the intent.

Type: String

Length Constraints: Minimum length of 1. Maximum length of 100.

Pattern: `^[a-zA-Z]+((_[a-zA-Z]+)*|([a-zA-Z]+_)*|_)`

### **parentIntentSignature (p. 238)**

A unique identifier for a built-in intent.

Type: String

### **rejectionStatement (p. 238)**

If the user answers "no" to the question defined in `confirmationPrompt`, Amazon Lex responds with this statement to acknowledge that the intent was canceled.

Type: [Statement \(p. 323\)](#) object

### **sampleUtterances (p. 238)**

An array of sample utterances configured for the intent.

Type: Array of strings

Array Members: Minimum number of 0 items. Maximum number of 1500 items.

Length Constraints: Minimum length of 1. Maximum length of 200.

### **slots (p. 238)**

An array of intent slots configured for the intent.

Type: Array of [Slot \(p. 319\)](#) objects

Array Members: Minimum number of 0 items. Maximum number of 100 items.

### **version (p. 238)**

The version of the intent.

Type: String

Length Constraints: Minimum length of 1. Maximum length of 64.

Pattern: `\$LATEST|[0-9]+`

## **Errors**

### **BadRequestException**

The request is not well formed. For example, a value is invalid or a required field is missing. Check the field values, and try again.

HTTP Status Code: 400

**InternalFailureException**

An internal Amazon Lex error occurred. Try your request again.

HTTP Status Code: 500

**LimitExceededException**

The request exceeded a limit. Try your request again.

HTTP Status Code: 429

**NotFoundException**

The resource specified in the request was not found. Check the resource and try again.

HTTP Status Code: 404

## See Also

For more information about using this API in one of the language-specific AWS SDKs, see the following:

- [AWS Command Line Interface](#)
- [AWS SDK for .NET](#)
- [AWS SDK for C++](#)
- [AWS SDK for Go](#)
- [AWS SDK for Java](#)
- [AWS SDK for JavaScript](#)
- [AWS SDK for PHP V3](#)
- [AWS SDK for Python](#)
- [AWS SDK for Ruby V2](#)

## GetIntent

Service: Amazon Lex Model Building Service

Returns intent information as follows:

- If you specify the `nameContains` field, returns the `$LATEST` version of all intents that contain the specified string.
- If you don't specify the `nameContains` field, returns information about the `$LATEST` version of all intents.

The operation requires permission for the `lex:GetIntent` action.

## Request Syntax

```
GET /intents/?maxResults=maxResults&nameContains=nameContains&nextToken=nextToken HTTP/1.1
```

## URI Request Parameters

The request requires the following URI parameters.

### `maxResults` (p. 243)

The maximum number of intents to return in the response. The default is 10.

Valid Range: Minimum value of 1. Maximum value of 50.

### `nameContains` (p. 243)

Substring to match in intent names. An intent will be returned if any part of its name matches the substring. For example, "xyz" matches both "xyzabc" and "abcxyz."

Length Constraints: Minimum length of 1. Maximum length of 100.

Pattern: `^[a-zA-Z]+((_[a-zA-Z]+)*|([a-zA-Z]+_)*|_)`

### `nextToken` (p. 243)

A pagination token that fetches the next page of intents. If the response to this API call is truncated, Amazon Lex returns a pagination token in the response. To fetch the next page of intents, specify the pagination token in the next request.

## Request Body

The request does not have a request body.

## Response Syntax

```
HTTP/1.1 200
Content-type: application/json

{
  "intents": [
    {
      "createdDate": number,
      "description": "string",
      "lastUpdatedDate": number,
      "name": "string",
      "version": "string"
    }
  ]
}
```



```
    }  
  ],  
  "nextToken": "string"  
}
```

## Response Elements

If the action is successful, the service sends back an HTTP 200 response.

The following data is returned in JSON format by the service.

### **intents** (p. 243)

An array of `Intent` objects. For more information, see [PutBot](#) (p. 261).

Type: Array of [IntentMetadata](#) (p. 314) objects

### **nextToken** (p. 243)

If the response is truncated, the response includes a pagination token that you can specify in your next request to fetch the next page of intents.

Type: String

## Errors

### **BadRequestException**

The request is not well formed. For example, a value is invalid or a required field is missing. Check the field values, and try again.

HTTP Status Code: 400

### **InternalFailureException**

An internal Amazon Lex error occurred. Try your request again.

HTTP Status Code: 500

### **LimitExceededException**

The request exceeded a limit. Try your request again.

HTTP Status Code: 429

### **NotFoundException**

The resource specified in the request was not found. Check the resource and try again.

HTTP Status Code: 404

## See Also

For more information about using this API in one of the language-specific AWS SDKs, see the following:

- [AWS Command Line Interface](#)
- [AWS SDK for .NET](#)
- [AWS SDK for C++](#)
- [AWS SDK for Go](#)
- [AWS SDK for Java](#)

- [AWS SDK for JavaScript](#)
- [AWS SDK for PHP V3](#)
- [AWS SDK for Python](#)
- [AWS SDK for Ruby V2](#)

## GetIntentVersions

Service: Amazon Lex Model Building Service

Gets information about all of the versions of an intent.

The `GetIntentVersions` operation returns an `IntentMetadata` object for each version of an intent. For example, if an intent has three numbered versions, the `GetIntentVersions` operation returns four `IntentMetadata` objects in the response, one for each numbered version and one for the `$LATEST` version.

The `GetIntentVersions` operation always returns at least one version, the `$LATEST` version.

This operation requires permissions for the `lex:GetIntentVersions` action.

### Request Syntax

```
GET /intents/name/versions/?maxResults=maxResults&nextToken=nextToken HTTP/1.1
```

### URI Request Parameters

The request requires the following URI parameters.

#### `maxResults` (p. 246)

The maximum number of intent versions to return in the response. The default is 10.

Valid Range: Minimum value of 1. Maximum value of 50.

#### `name` (p. 246)

The name of the intent for which versions should be returned.

Length Constraints: Minimum length of 1. Maximum length of 100.

Pattern: `^[a-zA-Z]+((_[a-zA-Z]+)*|([a-zA-Z]+_)*|_)`

#### `nextToken` (p. 246)

A pagination token for fetching the next page of intent versions. If the response to this call is truncated, Amazon Lex returns a pagination token in the response. To fetch the next page of versions, specify the pagination token in the next request.

### Request Body

The request does not have a request body.

### Response Syntax

```
HTTP/1.1 200
Content-type: application/json
```

```
{
  "intents": [
    {
      "createdDate": number,
      "description": "string",
      "lastUpdatedDate": number,
      "name": "string",
      "version": "string"
    }
  ],
}
```

```
"nextToken": "string"  
}
```

## Response Elements

If the action is successful, the service sends back an HTTP 200 response.

The following data is returned in JSON format by the service.

### **intents** (p. 246)

An array of `IntentMetadata` objects, one for each numbered version of the intent plus one for the `$LATEST` version.

Type: Array of `IntentMetadata` (p. 314) objects

### **nextToken** (p. 246)

A pagination token for fetching the next page of intent versions. If the response to this call is truncated, Amazon Lex returns a pagination token in the response. To fetch the next page of versions, specify the pagination token in the next request.

Type: String

## Errors

### **BadRequestException**

The request is not well formed. For example, a value is invalid or a required field is missing. Check the field values, and try again.

HTTP Status Code: 400

### **InternalFailureException**

An internal Amazon Lex error occurred. Try your request again.

HTTP Status Code: 500

### **LimitExceededException**

The request exceeded a limit. Try your request again.

HTTP Status Code: 429

### **NotFoundException**

The resource specified in the request was not found. Check the resource and try again.

HTTP Status Code: 404

## See Also

For more information about using this API in one of the language-specific AWS SDKs, see the following:

- [AWS Command Line Interface](#)
- [AWS SDK for .NET](#)
- [AWS SDK for C++](#)
- [AWS SDK for Go](#)
- [AWS SDK for Java](#)

- [AWS SDK for JavaScript](#)
- [AWS SDK for PHP V3](#)
- [AWS SDK for Python](#)
- [AWS SDK for Ruby V2](#)

## GetSlotType

Service: Amazon Lex Model Building Service

Returns information about a specific version of a slot type. In addition to specifying the slot type name, you must specify the slot type version.

This operation requires permissions for the `lex:GetSlotType` action.

### Request Syntax

```
GET /slottypes/name/versions/version HTTP/1.1
```

### URI Request Parameters

The request requires the following URI parameters.

#### **name** (p. 249)

The name of the slot type. The name is case sensitive.

Length Constraints: Minimum length of 1. Maximum length of 100.

Pattern: `^[a-zA-Z]+((_[a-zA-Z]+)|([a-zA-Z]+_)*|_)`

#### **version** (p. 249)

The version of the slot type.

Length Constraints: Minimum length of 1. Maximum length of 64.

Pattern: `\$LATEST|[0-9]+`

### Request Body

The request does not have a request body.

### Response Syntax

```
HTTP/1.1 200
Content-type: application/json

{
  "checksum": "string",
  "createdDate": number,
  "description": "string",
  "enumerationValues": [
    {
      "value": "string"
    }
  ],
  "lastUpdatedDate": number,
  "name": "string",
  "version": "string"
}
```

### Response Elements

If the action is successful, the service sends back an HTTP 200 response.

The following data is returned in JSON format by the service.

**checksum (p. 249)**

Checksum of the `$LATEST` version of the slot type.

Type: String

**createdDate (p. 249)**

The date that the slot type was created.

Type: Timestamp

**description (p. 249)**

A description of the slot type.

Type: String

Length Constraints: Minimum length of 0. Maximum length of 200.

**enumerationValues (p. 249)**

A list of `EnumerationValue` objects that defines the values that the slot type can take.

Type: Array of [EnumerationValue \(p. 310\)](#) objects

Array Members: Minimum number of 1 item. Maximum number of 10000 items.

**lastUpdatedDate (p. 249)**

The date that the slot type was updated. When you create a resource, the creation date and last update date are the same.

Type: Timestamp

**name (p. 249)**

The name of the slot type.

Type: String

Length Constraints: Minimum length of 1. Maximum length of 100.

Pattern: `^[a-zA-Z]+((_[a-zA-Z]+)*|([a-zA-Z]+_)*|_)`

**version (p. 249)**

The version of the slot type.

Type: String

Length Constraints: Minimum length of 1. Maximum length of 64.

Pattern: `\$LATEST|[0-9]+`

## Errors

### BadRequestException

The request is not well formed. For example, a value is invalid or a required field is missing. Check the field values, and try again.

HTTP Status Code: 400

**InternalFailureException**

An internal Amazon Lex error occurred. Try your request again.

HTTP Status Code: 500

**LimitExceededException**

The request exceeded a limit. Try your request again.

HTTP Status Code: 429

**NotFoundException**

The resource specified in the request was not found. Check the resource and try again.

HTTP Status Code: 404

## See Also

For more information about using this API in one of the language-specific AWS SDKs, see the following:

- [AWS Command Line Interface](#)
- [AWS SDK for .NET](#)
- [AWS SDK for C++](#)
- [AWS SDK for Go](#)
- [AWS SDK for Java](#)
- [AWS SDK for JavaScript](#)
- [AWS SDK for PHP V3](#)
- [AWS SDK for Python](#)
- [AWS SDK for Ruby V2](#)



## GetSlotTypes

Service: Amazon Lex Model Building Service

Returns slot type information as follows:

- If you specify the `nameContains` field, returns the `$LATEST` version of all slot types that contain the specified string.
- If you don't specify the `nameContains` field, returns information about the `$LATEST` version of all slot types.

The operation requires permission for the `lex:GetSlotTypes` action.

### Request Syntax

```
GET /slottypes/?maxResults=maxResults&nameContains=nameContains&nextToken=nextToken
HTTP/1.1
```

### URI Request Parameters

The request requires the following URI parameters.

#### `maxResults` (p. 252)

The maximum number of slot types to return in the response. The default is 10.

Valid Range: Minimum value of 1. Maximum value of 50.

#### `nameContains` (p. 252)

Substring to match in slot type names. A slot type will be returned if any part of its name matches the substring. For example, "xyz" matches both "xyzabc" and "abcxyz."

Length Constraints: Minimum length of 1. Maximum length of 100.

Pattern: `^[a-zA-Z]+((_[a-zA-Z]+)|([a-zA-Z]+_))*|_`

#### `nextToken` (p. 252)

A pagination token that fetches the next page of slot types. If the response to this API call is truncated, Amazon Lex returns a pagination token in the response. To fetch next page of slot types, specify the pagination token in the next request.

### Request Body

The request does not have a request body.

### Response Syntax

```
HTTP/1.1 200
Content-type: application/json

{
  "nextToken": "string",
  "slotTypes": [
    {
      "createdDate": number,
      "description": "string",
      "lastUpdatedDate": number,
```

```
        "name": "string",  
        "version": "string"  
    }  
]  
}
```

## Response Elements

If the action is successful, the service sends back an HTTP 200 response.

The following data is returned in JSON format by the service.

### **nextToken** (p. 252)

If the response is truncated, it includes a pagination token that you can specify in your next request to fetch the next page of slot types.

Type: String

### **slotTypes** (p. 252)

An array of objects, one for each slot type, that provides information such as the name of the slot type, the version, and a description.

Type: Array of [SlotTypeMetadata](#) (p. 321) objects

## Errors

### **BadRequestException**

The request is not well formed. For example, a value is invalid or a required field is missing. Check the field values, and try again.

HTTP Status Code: 400

### **InternalFailureException**

An internal Amazon Lex error occurred. Try your request again.

HTTP Status Code: 500

### **LimitExceededException**

The request exceeded a limit. Try your request again.

HTTP Status Code: 429

### **NotFoundException**

The resource specified in the request was not found. Check the resource and try again.

HTTP Status Code: 404

## See Also

For more information about using this API in one of the language-specific AWS SDKs, see the following:

- [AWS Command Line Interface](#)
- [AWS SDK for .NET](#)
- [AWS SDK for C++](#)
- [AWS SDK for Go](#)

- [AWS SDK for Java](#)
- [AWS SDK for JavaScript](#)
- [AWS SDK for PHP V3](#)
- [AWS SDK for Python](#)
- [AWS SDK for Ruby V2](#)

## GetSlotTypeVersions

Service: Amazon Lex Model Building Service

Gets information about all versions of a slot type.

The `GetSlotTypeVersions` operation returns a `SlotTypeMetadata` object for each version of a slot type. For example, if a slot type has three numbered versions, the `GetSlotTypeVersions` operation returns four `SlotTypeMetadata` objects in the response, one for each numbered version and one for the `$LATEST` version.

The `GetSlotTypeVersions` operation always returns at least one version, the `$LATEST` version.

This operation requires permissions for the `lex:GetSlotTypeVersions` action.

### Request Syntax

```
GET /slottypes/name/versions/?maxResults=maxResults&nextToken=nextToken HTTP/1.1
```

### URI Request Parameters

The request requires the following URI parameters.

#### `maxResults` (p. 255)

The maximum number of slot type versions to return in the response. The default is 10.

Valid Range: Minimum value of 1. Maximum value of 50.

#### `name` (p. 255)

The name of the slot type for which versions should be returned.

Length Constraints: Minimum length of 1. Maximum length of 100.

Pattern: `^[a-zA-Z]+((_[a-zA-Z]+)|([a-zA-Z]+_)*|_)`

#### `nextToken` (p. 255)

A pagination token for fetching the next page of slot type versions. If the response to this call is truncated, Amazon Lex returns a pagination token in the response. To fetch the next page of versions, specify the pagination token in the next request.

### Request Body

The request does not have a request body.

### Response Syntax

```
HTTP/1.1 200
Content-type: application/json

{
  "nextToken": "string",
  "slotTypes": [
    {
      "createdDate": number,
      "description": "string",
      "lastUpdatedDate": number,
      "name": "string",
      "version": "string"
    }
  ]
}
```

```
}  
  ]  
}
```

## Response Elements

If the action is successful, the service sends back an HTTP 200 response.

The following data is returned in JSON format by the service.

### **nextToken** (p. 255)

A pagination token for fetching the next page of slot type versions. If the response to this call is truncated, Amazon Lex returns a pagination token in the response. To fetch the next page of versions, specify the pagination token in the next request.

Type: String

### **slotTypes** (p. 255)

An array of `SlotTypeMetadata` objects, one for each numbered version of the slot type plus one for the `$LATEST` version.

Type: Array of [SlotTypeMetadata](#) (p. 321) objects

## Errors

### **BadRequestException**

The request is not well formed. For example, a value is invalid or a required field is missing. Check the field values, and try again.

HTTP Status Code: 400

### **InternalFailureException**

An internal Amazon Lex error occurred. Try your request again.

HTTP Status Code: 500

### **LimitExceededException**

The request exceeded a limit. Try your request again.

HTTP Status Code: 429

### **NotFoundException**

The resource specified in the request was not found. Check the resource and try again.

HTTP Status Code: 404

## See Also

For more information about using this API in one of the language-specific AWS SDKs, see the following:

- [AWS Command Line Interface](#)
- [AWS SDK for .NET](#)
- [AWS SDK for C++](#)
- [AWS SDK for Go](#)

- [AWS SDK for Java](#)
- [AWS SDK for JavaScript](#)
- [AWS SDK for PHP V3](#)
- [AWS SDK for Python](#)
- [AWS SDK for Ruby V2](#)

## GetUtterancesView

Service: Amazon Lex Model Building Service

Use the `GetUtterancesView` operation to get information about the utterances that your users have made to your bot. You can use this list to tune the utterances that your bot responds to.

For example, say that you have created a bot to order flowers. After your users have used your bot for a while, use the `GetUtterancesView` operation to see the requests that they have made and whether they have been successful. You might find that the utterance "I want flowers" is not being recognized. You could add this utterance to the `OrderFlowers` intent so that your bot recognizes that utterance.

After you publish a new version of a bot, you can get information about the old version and the new so that you can compare the performance across the two versions.

Data is available for the last 15 days. You can request information for up to 5 versions in each request. The response contains information about a maximum of 100 utterances for each version.

If the bot's `childDirected` field is set to `true`, utterances for the bot are not stored and cannot be retrieved with the `GetUtterancesView` operation. For more information, see [PutBot \(p. 261\)](#).

This operation requires permissions for the `lex:GetUtterancesView` action.

### Request Syntax

```
GET /bots/botname/utterances?view=aggregation?  
bot_versions=botVersions&status_type=statusType HTTP/1.1
```

### URI Request Parameters

The request requires the following URI parameters.

#### **botName (p. 258)**

The name of the bot for which utterance information should be returned.

Length Constraints: Minimum length of 2. Maximum length of 50.

Pattern: `^[a-zA-Z]+(\_[a-zA-Z]+)*|([a-zA-Z]+\_)*|_`

#### **botVersions (p. 258)**

An array of bot versions for which utterance information should be returned. The limit is 5 versions per request.

Array Members: Minimum number of 1 item. Maximum number of 5 items.

Length Constraints: Minimum length of 1. Maximum length of 64.

Pattern: `\$LATEST|[0-9]+`

#### **statusType (p. 258)**

To return utterances that were recognized and handled, use `Detected`. To return utterances that were not recognized, use `Missed`.

Valid Values: `Detected` | `Missed`

### Request Body

The request does not have a request body.

## Response Syntax

```
HTTP/1.1 200
Content-type: application/json

{
  "botName": "string",
  "utterances": [
    {
      "botVersion": "string",
      "utterances": [
        {
          "count": number,
          "distinctUsers": number,
          "firstUtteredDate": number,
          "lastUtteredDate": number,
          "utteranceString": "string"
        }
      ]
    }
  ]
}
```

## Response Elements

If the action is successful, the service sends back an HTTP 200 response.

The following data is returned in JSON format by the service.

### botName (p. 259)

The name of the bot for which utterance information was returned.

Type: String

Length Constraints: Minimum length of 2. Maximum length of 50.

Pattern: `^[a-zA-Z]+((_[a-zA-Z]+)|([a-zA-Z]+_)*|_)`

### utterances (p. 259)

An array of [UtteranceList](#) (p. 325) objects, each containing a list of [UtteranceData](#) (p. 324) objects describing the utterances that were processed by your bot. The response contains a maximum of 100 [UtteranceData](#) objects for each version.

Type: Array of [UtteranceList](#) (p. 325) objects

## Errors

### BadRequestException

The request is not well formed. For example, a value is invalid or a required field is missing. Check the field values, and try again.

HTTP Status Code: 400

### InternalFailureException

An internal Amazon Lex error occurred. Try your request again.

HTTP Status Code: 500



### **LimitExceededException**

The request exceeded a limit. Try your request again.

HTTP Status Code: 429

### **See Also**

For more information about using this API in one of the language-specific AWS SDKs, see the following:

- [AWS Command Line Interface](#)
- [AWS SDK for .NET](#)
- [AWS SDK for C++](#)
- [AWS SDK for Go](#)
- [AWS SDK for Java](#)
- [AWS SDK for JavaScript](#)
- [AWS SDK for PHP V3](#)
- [AWS SDK for Python](#)
- [AWS SDK for Ruby V2](#)

## PutBot

Service: Amazon Lex Model Building Service

Creates an Amazon Lex conversational bot or replaces an existing bot. When you create or update a bot you are only required to specify a name. You can use this to add intents later, or to remove intents from an existing bot. When you create a bot with a name only, the bot is created or updated but Amazon Lex returns the response `FAILED`. You can build the bot after you add one or more intents. For more information about Amazon Lex bots, see [Amazon Lex: How It Works \(p. 3\)](#).

If you specify the name of an existing bot, the fields in the request replace the existing values in the `$LATEST` version of the bot. Amazon Lex removes any fields that you don't provide values for in the request, except for the `idleTTLInSeconds` and `privacySettings` fields, which are set to their default values. If you don't specify values for required fields, Amazon Lex throws an exception.

This operation requires permissions for the `lex:PutBot` action. For more information, see [Authentication and Access Control for Amazon Lex \(p. 161\)](#).

## Request Syntax

```
PUT /bots/name/versions/$LATEST HTTP/1.1
Content-type: application/json
```

```
{
  "abortStatement": {
    "messages": [
      {
        "content": "string",
        "contentType": "string"
      }
    ],
    "responseCard": "string"
  },
  "checksum": "string",
  "childDirected": boolean,
  "clarificationPrompt": {
    "maxAttempts": number,
    "messages": [
      {
        "content": "string",
        "contentType": "string"
      }
    ],
    "responseCard": "string"
  },
  "description": "string",
  "idleSessionTTLInSeconds": number,
  "intents": [
    {
      "intentName": "string",
      "intentVersion": "string"
    }
  ],
  "locale": "string",
  "processBehavior": "string",
  "voiceId": "string"
}
```

## URI Request Parameters

The request requires the following URI parameters.

### name (p. 261)

The name of the bot. The name is *not* case sensitive.

Length Constraints: Minimum length of 2. Maximum length of 50.

Pattern: `^[a-zA-Z]+((_[a-zA-Z]+)|([a-zA-Z]+_)*|_)`

## Request Body

The request accepts the following data in JSON format.

### abortStatement (p. 261)

When Amazon Lex can't understand the user's input in context, it tries to elicit the information a few times. After that, Amazon Lex sends the message defined in `abortStatement` to the user, and then aborts the conversation. To set the number of retries, use the `valueElicitationPrompt` field for the slot type.

For example, in a pizza ordering bot, Amazon Lex might ask a user "What type of crust would you like?" If the user's response is not one of the expected responses (for example, "thin crust, "deep dish," etc.), Amazon Lex tries to elicit a correct response a few more times.

For example, in a pizza ordering application, `OrderPizza` might be one of the intents. This intent might require the `CrustType` slot. You specify the `valueElicitationPrompt` field when you create the `CrustType` slot.

Type: [Statement \(p. 323\)](#) object

Required: No

### checksum (p. 261)

Identifies a specific revision of the `$LATEST` version.

When you create a new bot, leave the `checksum` field blank. If you specify a checksum you get a `BadRequestException` exception.

When you want to update a bot, set the `checksum` field to the checksum of the most recent revision of the `$LATEST` version. If you don't specify the `checksum` field, or if the checksum does not match the `$LATEST` version, you get a `PreconditionFailedException` exception.

Type: String

Required: No

### childDirected (p. 261)

For each Amazon Lex bot created with the Amazon Lex Model Building Service, you must specify whether your use of Amazon Lex is related to a website, program, or other application that is directed or targeted, in whole or in part, to children under age 13 and subject to the Children's Online Privacy Protection Act (COPPA) by specifying `true` or `false` in the `childDirected` field. By specifying `true` in the `childDirected` field, you confirm that your use of Amazon Lex **is** related to a website, program, or other application that is directed or targeted, in whole or in part, to children under age 13 and subject to COPPA. By specifying `false` in the `childDirected` field, you confirm that your use of Amazon Lex **is not** related to a website, program, or other application that is directed or targeted, in whole or in part, to children under age 13 and subject to COPPA. You may not specify a default value for the `childDirected` field that does not accurately reflect whether your use of Amazon Lex is related to a website, program, or other application that is directed or targeted, in whole or in part, to children under age 13 and subject to COPPA.

If your use of Amazon Lex relates to a website, program, or other application that is directed in whole or in part, to children under age 13, you must obtain any required verifiable parental consent under COPPA. For information regarding the use of Amazon Lex in connection with websites, programs, or other applications that are directed or targeted, in whole or in part, to children under age 13, see the [Amazon Lex FAQ](#).

Type: Boolean

Required: Yes

#### **clarificationPrompt (p. 261)**

When Amazon Lex doesn't understand the user's intent, it uses this message to get clarification. To specify how many times Amazon Lex should repeat the clarification prompt, use the `maxAttempts` field. If Amazon Lex still doesn't understand, it sends the message in the `abortStatement` field.

When you create a clarification prompt, make sure that it suggests the correct response from the user. For example, for a bot that orders pizza and drinks, you might create this clarification prompt: "What would you like to do? You can say 'Order a pizza' or 'Order a drink.'"

Type: [Prompt \(p. 317\)](#) object

Required: No

#### **description (p. 261)**

A description of the bot.

Type: String

Length Constraints: Minimum length of 0. Maximum length of 200.

Required: No

#### **idleSessionTTLInSeconds (p. 261)**

The maximum time in seconds that Amazon Lex retains the data gathered in a conversation.

A user interaction session remains active for the amount of time specified. If no conversation occurs during this time, the session expires and Amazon Lex deletes any data provided before the timeout.

For example, suppose that a user chooses the `OrderPizza` intent, but gets sidetracked halfway through placing an order. If the user doesn't complete the order within the specified time, Amazon Lex discards the slot information that it gathered, and the user must start over.

If you don't include the `idleSessionTTLInSeconds` element in a `PutBot` operation request, Amazon Lex uses the default value. This is also true if the request replaces an existing bot.

The default is 300 seconds (5 minutes).

Type: Integer

Valid Range: Minimum value of 60. Maximum value of 86400.

Required: No

#### **intents (p. 261)**

An array of `Intent` objects. Each intent represents a command that a user can express. For example, a pizza ordering bot might support an `OrderPizza` intent. For more information, see [Amazon Lex: How It Works \(p. 3\)](#).

Type: Array of [Intent \(p. 313\)](#) objects

Array Members: Minimum number of 1 item. Maximum number of 100 items.

Required: No

#### **locale (p. 261)**

Specifies the target locale for the bot. Any intent used in the bot must be compatible with the locale of the bot.

The default is `en-US`.

Type: String

Valid Values: `en-US`

Required: Yes

#### **processBehavior (p. 261)**

If you set the `processBehavior` element to `Build`, Amazon Lex builds the bot so that it can be run. If you set the element to `Save` Amazon Lex saves the bot, but doesn't build it.

If you don't specify this value, the default value is `Save`.

Type: String

Valid Values: `SAVE` | `BUILD`

Required: No

#### **voiceId (p. 261)**

The Amazon Polly voice ID that you want Amazon Lex to use for voice interactions with the user. The locale configured for the voice must match the locale of the bot. For more information, see [Available Voices](#) in the *Amazon Polly Developer Guide*.

Type: String

Required: No

## **Response Syntax**

```
HTTP/1.1 200
Content-type: application/json

{
  "abortStatement": {
    "messages": [
      {
        "content": "string",
        "contentType": "string"
      }
    ],
    "responseCard": "string"
  },
  "checksum": "string",
  "childDirected": boolean,
  "clarificationPrompt": {
    "maxAttempts": number,
    "messages": [
      {
        "content": "string",
        "contentType": "string"
      }
    ]
  }
}
```

```
    ],  
    "responseCard": "string"  
  },  
  "createdDate": number,  
  "description": "string",  
  "failureReason": "string",  
  "idleSessionTTLInSeconds": number,  
  "intents": [  
    {  
      "intentName": "string",  
      "intentVersion": "string"  
    }  
  ],  
  "lastUpdatedDate": number,  
  "locale": "string",  
  "name": "string",  
  "status": "string",  
  "version": "string",  
  "voiceId": "string"  
}
```

## Response Elements

If the action is successful, the service sends back an HTTP 200 response.

The following data is returned in JSON format by the service.

### **abortStatement** (p. 264)

The message that Amazon Lex uses to abort a conversation. For more information, see [PutBot](#) (p. 261).

Type: [Statement](#) (p. 323) object

### **checksum** (p. 264)

Checksum of the bot that you created.

Type: String

### **childDirected** (p. 264)

For each Amazon Lex bot created with the Amazon Lex Model Building Service, you must specify whether your use of Amazon Lex is related to a website, program, or other application that is directed or targeted, in whole or in part, to children under age 13 and subject to the Children's Online Privacy Protection Act (COPPA) by specifying `true` or `false` in the `childDirected` field. By specifying `true` in the `childDirected` field, you confirm that your use of Amazon Lex is related to a website, program, or other application that is directed or targeted, in whole or in part, to children under age 13 and subject to COPPA. By specifying `false` in the `childDirected` field, you confirm that your use of Amazon Lex **is not** related to a website, program, or other application that is directed or targeted, in whole or in part, to children under age 13 and subject to COPPA. You may not specify a default value for the `childDirected` field that does not accurately reflect whether your use of Amazon Lex is related to a website, program, or other application that is directed or targeted, in whole or in part, to children under age 13 and subject to COPPA.

If your use of Amazon Lex relates to a website, program, or other application that is directed in whole or in part, to children under age 13, you must obtain any required verifiable parental consent under COPPA. For information regarding the use of Amazon Lex in connection with websites, programs, or other applications that are directed or targeted, in whole or in part, to children under age 13, see the [Amazon Lex FAQ](#).

Type: Boolean

#### **clarificationPrompt (p. 264)**

The prompts that Amazon Lex uses when it doesn't understand the user's intent. For more information, see [PutBot \(p. 261\)](#).

Type: [Prompt \(p. 317\)](#) object

#### **createdDate (p. 264)**

The date that the bot was created.

Type: Timestamp

#### **description (p. 264)**

A description of the bot.

Type: String

Length Constraints: Minimum length of 0. Maximum length of 200.

#### **failureReason (p. 264)**

If `status` is `FAILED`, Amazon Lex provides the reason that it failed to build the bot.

Type: String

#### **idleSessionTTLInSeconds (p. 264)**

The maximum length of time that Amazon Lex retains the data gathered in a conversation. For more information, see [PutBot \(p. 261\)](#).

Type: Integer

Valid Range: Minimum value of 60. Maximum value of 86400.

#### **intents (p. 264)**

An array of `Intent` objects. For more information, see [PutBot \(p. 261\)](#).

Type: Array of [Intent \(p. 313\)](#) objects

Array Members: Minimum number of 1 item. Maximum number of 100 items.

#### **lastUpdatedDate (p. 264)**

The date that the bot was updated. When you create a resource, the creation date and last updated date are the same.

Type: Timestamp

#### **locale (p. 264)**

The target locale for the bot.

Type: String

Valid Values: `en-US`

#### **name (p. 264)**

The name of the bot.

Type: String

Length Constraints: Minimum length of 2. Maximum length of 50.

Pattern: `^[a-zA-Z]+((_[a-zA-Z]+)*|([a-zA-Z]+_)*|_)`

#### [status \(p. 264\)](#)

When you send a request to create a bot with `processBehavior` set to `BUILD`, Amazon Lex sets the `status` response element to `BUILDING`. After Amazon Lex builds the bot, it sets `status` to `READY`. If Amazon Lex can't build the bot, Amazon Lex sets `status` to `FAILED`. Amazon Lex returns the reason for the failure in the `failureReason` response element.

When you set `processBehavior` to `SAVE`, Amazon Lex sets the status code to `NOT_BUILT`.

Type: String

Valid Values: `BUILDING` | `READY` | `FAILED` | `NOT_BUILT`

#### [version \(p. 264\)](#)

The version of the bot. For a new bot, the version is always `$LATEST`.

Type: String

Length Constraints: Minimum length of 1. Maximum length of 64.

Pattern: `\$LATEST|[0-9]+`

#### [voiceId \(p. 264\)](#)

The Amazon Polly voice ID that Amazon Lex uses for voice interaction with the user. For more information, see [PutBot \(p. 261\)](#).

Type: String

## Errors

### **BadRequestException**

The request is not well formed. For example, a value is invalid or a required field is missing. Check the field values, and try again.

HTTP Status Code: 400

### **ConflictException**

There was a conflict processing the request. Try your request again.

HTTP Status Code: 409

### **InternalFailureException**

An internal Amazon Lex error occurred. Try your request again.

HTTP Status Code: 500

### **LimitExceededException**

The request exceeded a limit. Try your request again.

HTTP Status Code: 429

### **PreconditionFailedException**

The checksum of the resource that you are trying to change does not match the checksum in the request. Check the resource's checksum and try again.

HTTP Status Code: 412



## See Also

For more information about using this API in one of the language-specific AWS SDKs, see the following:

- [AWS Command Line Interface](#)
- [AWS SDK for .NET](#)
- [AWS SDK for C++](#)
- [AWS SDK for Go](#)
- [AWS SDK for Java](#)
- [AWS SDK for JavaScript](#)
- [AWS SDK for PHP V3](#)
- [AWS SDK for Python](#)
- [AWS SDK for Ruby V2](#)

## PutBotAlias

Service: Amazon Lex Model Building Service

Creates an alias for the specified version of the bot or replaces an alias for the specified bot. To change the version of the bot that the alias points to, replace the alias. For more information about aliases, see [Versioning and Aliases \(p. 81\)](#).

This operation requires permissions for the `lex:PutBotAlias` action.

### Request Syntax

```
PUT /bots/botName/aliases/name HTTP/1.1
Content-type: application/json

{
  "botVersion": "string",
  "checksum": "string",
  "description": "string"
}
```

### URI Request Parameters

The request requires the following URI parameters.

#### **botName** (p. 269)

The name of the bot.

Length Constraints: Minimum length of 2. Maximum length of 50.

Pattern: `^[a-zA-Z]+((_[a-zA-Z]+)*|([a-zA-Z]+_)*|_)`

#### **name** (p. 269)

The name of the alias. The name is *not* case sensitive.

Length Constraints: Minimum length of 1. Maximum length of 100.

Pattern: `^[a-zA-Z]+((_[a-zA-Z]+)*|([a-zA-Z]+_)*|_)`

### Request Body

The request accepts the following data in JSON format.

#### **botVersion** (p. 269)

The version of the bot.

Type: String

Length Constraints: Minimum length of 1. Maximum length of 64.

Pattern: `\$LATEST|[0-9]+`

Required: Yes

#### **checksum** (p. 269)

Identifies a specific revision of the `$LATEST` version.

When you create a new bot alias, leave the `checksum` field blank. If you specify a checksum you get a `BadRequestException` exception.

When you want to update a bot alias, set the `checksum` field to the checksum of the most recent revision of the `$LATEST` version. If you don't specify the `checksum` field, or if the checksum does not match the `$LATEST` version, you get a `PreconditionFailedException` exception.

Type: String

Required: No

#### [description \(p. 269\)](#)

A description of the alias.

Type: String

Length Constraints: Minimum length of 0. Maximum length of 200.

Required: No

## Response Syntax

```
HTTP/1.1 200
Content-type: application/json

{
  "botName": "string",
  "botVersion": "string",
  "checksum": "string",
  "createdDate": number,
  "description": "string",
  "lastUpdatedDate": number,
  "name": "string"
}
```

## Response Elements

If the action is successful, the service sends back an HTTP 200 response.

The following data is returned in JSON format by the service.

#### [botName \(p. 270\)](#)

The name of the bot that the alias points to.

Type: String

Length Constraints: Minimum length of 2. Maximum length of 50.

Pattern: `^[a-zA-Z]+((_[a-zA-Z]+)*|([a-zA-Z]+_)*|_)`

#### [botVersion \(p. 270\)](#)

The version of the bot that the alias points to.

Type: String

Length Constraints: Minimum length of 1. Maximum length of 64.

Pattern: `\$LATEST|[0-9]+`

#### **checksum (p. 270)**

The checksum for the current version of the alias.

Type: String

#### **createdDate (p. 270)**

The date that the bot alias was created.

Type: Timestamp

#### **description (p. 270)**

A description of the alias.

Type: String

Length Constraints: Minimum length of 0. Maximum length of 200.

#### **lastUpdatedDate (p. 270)**

The date that the bot alias was updated. When you create a resource, the creation date and the last updated date are the same.

Type: Timestamp

#### **name (p. 270)**

The name of the alias.

Type: String

Length Constraints: Minimum length of 1. Maximum length of 100.

Pattern: `^[a-zA-Z]+((_[a-zA-Z]+)|([a-zA-Z]+_)*|_)`

## **Errors**

### **BadRequestException**

The request is not well formed. For example, a value is invalid or a required field is missing. Check the field values, and try again.

HTTP Status Code: 400

### **ConflictException**

There was a conflict processing the request. Try your request again.

HTTP Status Code: 409

### **InternalFailureException**

An internal Amazon Lex error occurred. Try your request again.

HTTP Status Code: 500

### **LimitExceededException**

The request exceeded a limit. Try your request again.

HTTP Status Code: 429

### **PreconditionFailedException**

The checksum of the resource that you are trying to change does not match the checksum in the request. Check the resource's checksum and try again.

HTTP Status Code: 412

## See Also

For more information about using this API in one of the language-specific AWS SDKs, see the following:

- [AWS Command Line Interface](#)
- [AWS SDK for .NET](#)
- [AWS SDK for C++](#)
- [AWS SDK for Go](#)
- [AWS SDK for Java](#)
- [AWS SDK for JavaScript](#)
- [AWS SDK for PHP V3](#)
- [AWS SDK for Python](#)
- [AWS SDK for Ruby V2](#)

## PutIntent

Service: Amazon Lex Model Building Service

Creates an intent or replaces an existing intent.

To define the interaction between the user and your bot, you use one or more intents. For a pizza ordering bot, for example, you would create an `OrderPizza` intent.

To create an intent or replace an existing intent, you must provide the following:

- Intent name. For example, `OrderPizza`.
- Sample utterances. For example, "Can I order a pizza, please." and "I want to order a pizza."
- Information to be gathered. You specify slot types for the information that your bot will request from the user. You can specify standard slot types, such as a date or a time, or custom slot types such as the size and crust of a pizza.
- How the intent will be fulfilled. You can provide a Lambda function or configure the intent to return the intent information to the client application. If you use a Lambda function, when all of the intent information is available, Amazon Lex invokes your Lambda function. If you configure your intent to return the intent information to the client application.

You can specify other optional information in the request, such as:

- A confirmation prompt to ask the user to confirm an intent. For example, "Shall I order your pizza?"
- A conclusion statement to send to the user after the intent has been fulfilled. For example, "I placed your pizza order."
- A follow-up prompt that asks the user for additional activity. For example, asking "Do you want to order a drink with your pizza?"

If you specify an existing intent name to update the intent, Amazon Lex replaces the values in the `$LATEST` version of the slot type with the values in the request. Amazon Lex removes fields that you don't provide in the request. If you don't specify the required fields, Amazon Lex throws an exception.

For more information, see [Amazon Lex: How It Works \(p. 3\)](#).

This operation requires permissions for the `lex:PutIntent` action.

## Request Syntax

```
PUT /intents/name/versions/$LATEST HTTP/1.1
Content-type: application/json
```

```
{
  "checksum": "string",
  "conclusionStatement": {
    "messages": [
      {
        "content": "string",
        "contentType": "string"
      }
    ],
    "responseCard": "string"
  },
  "confirmationPrompt": {
    "maxAttempts": number,
    "messages": [
      {
        "content": "string",
```

```
        "contentType": "string"
      }
    ],
    "responseCard": "string"
  },
  "description": "string",
  "dialogCodeHook": {
    "messageVersion": "string",
    "uri": "string"
  },
  "followUpPrompt": {
    "prompt": {
      "maxAttempts": number,
      "messages": [
        {
          "content": "string",
          "contentType": "string"
        }
      ]
    },
    "responseCard": "string"
  },
  "rejectionStatement": {
    "messages": [
      {
        "content": "string",
        "contentType": "string"
      }
    ],
    "responseCard": "string"
  }
},
"fulfillmentActivity": {
  "codeHook": {
    "messageVersion": "string",
    "uri": "string"
  },
  "type": "string"
},
"parentIntentSignature": "string",
"rejectionStatement": {
  "messages": [
    {
      "content": "string",
      "contentType": "string"
    }
  ],
  "responseCard": "string"
},
"sampleUtterances": [ "string" ],
"slots": [
  {
    "description": "string",
    "name": "string",
    "priority": number,
    "responseCard": "string",
    "sampleUtterances": [ "string" ],
    "slotConstraint": "string",
    "slotType": "string",
    "slotTypeVersion": "string",
    "valueElicitationPrompt": {
      "maxAttempts": number,
      "messages": [
        {
          "content": "string",
          "contentType": "string"
        }
      ]
    }
  }
]
```

```
    ],  
    "responseCard": "string"  
  }  
}  
]  
}
```

## URI Request Parameters

The request requires the following URI parameters.

### name (p. 273)

The name of the intent. The name is *not* case sensitive.

The name can't match a built-in intent name, or a built-in intent name with "AMAZON." removed. For example, because there is a built-in intent called `AMAZON.HelpIntent`, you can't create a custom intent called `HelpIntent`.

For a list of built-in intents, see [Standard Built-in Intents](#) in the *Alexa Skills Kit*.

Length Constraints: Minimum length of 1. Maximum length of 100.

Pattern: `^[a-zA-Z]+((_[a-zA-Z]+)*|([a-zA-Z]+_)*|_)`

## Request Body

The request accepts the following data in JSON format.

### checksum (p. 273)

Identifies a specific revision of the `$LATEST` version.

When you create a new intent, leave the `checksum` field blank. If you specify a checksum you get a `BadRequestException` exception.

When you want to update a intent, set the `checksum` field to the checksum of the most recent revision of the `$LATEST` version. If you don't specify the `checksum` field, or if the checksum does not match the `$LATEST` version, you get a `PreconditionFailedException` exception.

Type: String

Required: No

### conclusionStatement (p. 273)

The statement that you want Amazon Lex to convey to the user after the intent is successfully fulfilled by the Lambda function.

This element is relevant only if you provide a Lambda function in the `fulfillmentActivity`. If you return the intent to the client application, you can't specify this element.

#### Note

The `followUpPrompt` and `conclusionStatement` are mutually exclusive. You can specify only one.

Type: [Statement \(p. 323\)](#) object

Required: No

### confirmationPrompt (p. 273)

Prompts the user to confirm the intent. This question should have a yes or no answer.



Amazon Lex uses this prompt to ensure that the user acknowledges that the intent is ready for fulfillment. For example, with the `OrderPizza` intent, you might want to confirm that the order is correct before placing it. For other intents, such as intents that simply respond to user questions, you might not need to ask the user for confirmation before providing the information.

**Note**

You must provide both the `rejectionStatement` and the `confirmationPrompt`, or neither.

Type: [Prompt \(p. 317\)](#) object

Required: No

**[description \(p. 273\)](#)**

A description of the intent.

Type: String

Length Constraints: Minimum length of 0. Maximum length of 200.

Required: No

**[dialogCodeHook \(p. 273\)](#)**

Specifies a Lambda function to invoke for each user input. You can invoke this Lambda function to personalize user interaction.

For example, suppose your bot determines that the user is John. Your Lambda function might retrieve John's information from a backend database and prepopulate some of the values. For example, if you find that John is gluten intolerant, you might set the corresponding intent slot, `GlutenIntolerant`, to true. You might find John's phone number and set the corresponding session attribute.

Type: [CodeHook \(p. 309\)](#) object

Required: No

**[followUpPrompt \(p. 273\)](#)**

Amazon Lex uses this prompt to solicit additional activity after fulfilling an intent. For example, after the `OrderPizza` intent is fulfilled, you might prompt the user to order a drink.

The action that Amazon Lex takes depends on the user's response, as follows:

- If the user says "Yes" it responds with the clarification prompt that is configured for the bot.
- If the user says "Yes" and continues with an utterance that triggers an intent it starts a conversation for the intent.
- If the user says "No" it responds with the rejection statement configured for the follow-up prompt.
- If it doesn't recognize the utterance it repeats the follow-up prompt again.

The `followUpPrompt` field and the `conclusionStatement` field are mutually exclusive. You can specify only one.

Type: [FollowUpPrompt \(p. 311\)](#) object

Required: No

**[fulfillmentActivity \(p. 273\)](#)**

Required. Describes how the intent is fulfilled. For example, after a user provides all of the information for a pizza order, `fulfillmentActivity` defines how the bot places an order with a local pizza store.

You might configure Amazon Lex to return all of the intent information to the client application, or direct it to invoke a Lambda function that can process the intent (for example, place an order with a pizzeria).

Type: [FulfillmentActivity \(p. 312\)](#) object

Required: No

#### [parentIntentSignature \(p. 273\)](#)

A unique identifier for the built-in intent to base this intent on. To find the signature for an intent, see [Standard Built-in Intents](#) in the *Alexa Skills Kit*.

Type: String

Required: No

#### [rejectionStatement \(p. 273\)](#)

When the user answers "no" to the question defined in `confirmationPrompt`, Amazon Lex responds with this statement to acknowledge that the intent was canceled.

##### **Note**

You must provide both the `rejectionStatement` and the `confirmationPrompt`, or neither.

Type: [Statement \(p. 323\)](#) object

Required: No

#### [sampleUtterances \(p. 273\)](#)

An array of utterances (strings) that a user might say to signal the intent. For example, "I want {PizzaSize} pizza", "Order {Quantity} {PizzaSize} pizzas".

In each utterance, a slot name is enclosed in curly braces.

Type: Array of strings

Array Members: Minimum number of 0 items. Maximum number of 1500 items.

Length Constraints: Minimum length of 1. Maximum length of 200.

Required: No

#### [slots \(p. 273\)](#)

An array of intent slots. At runtime, Amazon Lex elicits required slot values from the user using prompts defined in the slots. For more information, see [Amazon Lex: How It Works \(p. 3\)](#).

Type: Array of [Slot \(p. 319\)](#) objects

Array Members: Minimum number of 0 items. Maximum number of 100 items.

Required: No

## Response Syntax

```
HTTP/1.1 200
Content-type: application/json

{
  "checksum": "string",
  "conclusionStatement": {
```

```
    "messages": [
      {
        "content": "string",
        "contentType": "string"
      }
    ],
    "responseCard": "string"
  },
  "confirmationPrompt": {
    "maxAttempts": number,
    "messages": [
      {
        "content": "string",
        "contentType": "string"
      }
    ],
    "responseCard": "string"
  },
  "createdAt": number,
  "description": "string",
  "dialogCodeHook": {
    "messageVersion": "string",
    "uri": "string"
  },
  "followUpPrompt": {
    "prompt": {
      "maxAttempts": number,
      "messages": [
        {
          "content": "string",
          "contentType": "string"
        }
      ]
    },
    "responseCard": "string"
  },
  "rejectionStatement": {
    "messages": [
      {
        "content": "string",
        "contentType": "string"
      }
    ],
    "responseCard": "string"
  }
},
"fulfillmentActivity": {
  "codeHook": {
    "messageVersion": "string",
    "uri": "string"
  },
  "type": "string"
},
"lastUpdatedDate": number,
"name": "string",
"parentIntentSignature": "string",
"rejectionStatement": {
  "messages": [
    {
      "content": "string",
      "contentType": "string"
    }
  ],
  "responseCard": "string"
},
"sampleUtterances": [ "string" ],
"slots": [
```

```
{
  "description": "string",
  "name": "string",
  "priority": number,
  "responseCard": "string",
  "sampleUtterances": [ "string" ],
  "slotConstraint": "string",
  "slotType": "string",
  "slotTypeVersion": "string",
  "valueElicitationPrompt": {
    "maxAttempts": number,
    "messages": [
      {
        "content": "string",
        "contentType": "string"
      }
    ],
    "responseCard": "string"
  }
},
"version": "string"
}
```

## Response Elements

If the action is successful, the service sends back an HTTP 200 response.

The following data is returned in JSON format by the service.

### checksum (p. 277)

Checksum of the `$LATEST` version of the intent created or updated.

Type: String

### conclusionStatement (p. 277)

After the Lambda function specified in the `fulfillmentActivity` intent fulfills the intent, Amazon Lex conveys this statement to the user.

Type: [Statement \(p. 323\)](#) object

### confirmationPrompt (p. 277)

If defined in the intent, Amazon Lex prompts the user to confirm the intent before fulfilling it.

Type: [Prompt \(p. 317\)](#) object

### createdDate (p. 277)

The date that the intent was created.

Type: Timestamp

### description (p. 277)

A description of the intent.

Type: String

Length Constraints: Minimum length of 0. Maximum length of 200.

### dialogCodeHook (p. 277)

If defined in the intent, Amazon Lex invokes this Lambda function for each user input.

Type: [CodeHook \(p. 309\)](#) object

**[followUpPrompt \(p. 277\)](#)**

If defined in the intent, Amazon Lex uses this prompt to solicit additional user activity after the intent is fulfilled.

Type: [FollowUpPrompt \(p. 311\)](#) object

**[fulfillmentActivity \(p. 277\)](#)**

If defined in the intent, Amazon Lex invokes this Lambda function to fulfill the intent after the user provides all of the information required by the intent.

Type: [FulfillmentActivity \(p. 312\)](#) object

**[lastUpdatedDate \(p. 277\)](#)**

The date that the intent was updated. When you create a resource, the creation date and last update dates are the same.

Type: Timestamp

**[name \(p. 277\)](#)**

The name of the intent.

Type: String

Length Constraints: Minimum length of 1. Maximum length of 100.

Pattern: `^[a-zA-Z]+((_[a-zA-Z]+)|([a-zA-Z]+_))*|_`

**[parentIntentSignature \(p. 277\)](#)**

A unique identifier for the built-in intent that this intent is based on.

Type: String

**[rejectionStatement \(p. 277\)](#)**

If the user answers "no" to the question defined in `confirmationPrompt` Amazon Lex responds with this statement to acknowledge that the intent was canceled.

Type: [Statement \(p. 323\)](#) object

**[sampleUtterances \(p. 277\)](#)**

An array of sample utterances that are configured for the intent.

Type: Array of strings

Array Members: Minimum number of 0 items. Maximum number of 1500 items.

Length Constraints: Minimum length of 1. Maximum length of 200.

**[slots \(p. 277\)](#)**

An array of intent slots that are configured for the intent.

Type: Array of [Slot \(p. 319\)](#) objects

Array Members: Minimum number of 0 items. Maximum number of 100 items.

**[version \(p. 277\)](#)**

The version of the intent. For a new intent, the version is always `$LATEST`.

Type: String

Length Constraints: Minimum length of 1. Maximum length of 64.

Pattern: `\$LATEST|[0-9]+`

## Errors

### **BadRequestException**

The request is not well formed. For example, a value is invalid or a required field is missing. Check the field values, and try again.

HTTP Status Code: 400

### **ConflictException**

There was a conflict processing the request. Try your request again.

HTTP Status Code: 409

### **InternalFailureException**

An internal Amazon Lex error occurred. Try your request again.

HTTP Status Code: 500

### **LimitExceededException**

The request exceeded a limit. Try your request again.

HTTP Status Code: 429

### **PreconditionFailedException**

The checksum of the resource that you are trying to change does not match the checksum in the request. Check the resource's checksum and try again.

HTTP Status Code: 412

## See Also

For more information about using this API in one of the language-specific AWS SDKs, see the following:

- [AWS Command Line Interface](#)
- [AWS SDK for .NET](#)
- [AWS SDK for C++](#)
- [AWS SDK for Go](#)
- [AWS SDK for Java](#)
- [AWS SDK for JavaScript](#)
- [AWS SDK for PHP V3](#)
- [AWS SDK for Python](#)
- [AWS SDK for Ruby V2](#)

## PutSlotType

Service: Amazon Lex Model Building Service

Creates a custom slot type or replaces an existing custom slot type.

To create a custom slot type, specify a name for the slot type and a set of enumeration values, which are the values that a slot of this type can assume. For more information, see [Amazon Lex: How It Works](#) (p. 3).

If you specify the name of an existing slot type, the fields in the request replace the existing values in the `$LATEST` version of the slot type. Amazon Lex removes the fields that you don't provide in the request. If you don't specify required fields, Amazon Lex throws an exception.

This operation requires permissions for the `lex:PutSlotType` action.

### Request Syntax

```
PUT /slottypes/name/versions/$LATEST HTTP/1.1
Content-type: application/json

{
  "checksum": "string",
  "description": "string",
  "enumerationValues": [
    {
      "value": "string"
    }
  ]
}
```

### URI Request Parameters

The request requires the following URI parameters.

#### **name** (p. 282)

The name of the slot type. The name is *not* case sensitive.

The name can't match a built-in slot type name, or a built-in slot type name with "AMAZON." removed. For example, because there is a built-in slot type called `AMAZON.DATE`, you can't create a custom slot type called `DATE`.

For a list of built-in slot types, see [Slot Type Reference](#) in the *Alexa Skills Kit*.

Length Constraints: Minimum length of 1. Maximum length of 100.

Pattern: `^[a-zA-Z]+(\_[a-zA-Z]+)*|([a-zA-Z]+\_)*|_`

### Request Body

The request accepts the following data in JSON format.

#### **checksum** (p. 282)

Identifies a specific revision of the `$LATEST` version.

When you create a new slot type, leave the `checksum` field blank. If you specify a checksum you get a `BadRequestException` exception.

When you want to update a slot type, set the `checksum` field to the checksum of the most recent revision of the `$LATEST` version. If you don't specify the `checksum` field, or if the checksum does not match the `$LATEST` version, you get a `PreconditionFailedException` exception.

Type: String

Required: No

#### [description \(p. 282\)](#)

A description of the slot type.

Type: String

Length Constraints: Minimum length of 0. Maximum length of 200.

Required: No

#### [enumerationValues \(p. 282\)](#)

A list of `EnumerationValue` objects that defines the values that the slot type can take.

Type: Array of [EnumerationValue \(p. 310\)](#) objects

Array Members: Minimum number of 1 item. Maximum number of 10000 items.

Required: No

## Response Syntax

```
HTTP/1.1 200
Content-type: application/json

{
  "checksum": "string",
  "createdDate": number,
  "description": "string",
  "enumerationValues": [
    {
      "value": "string"
    }
  ],
  "lastUpdatedDate": number,
  "name": "string",
  "version": "string"
}
```

## Response Elements

If the action is successful, the service sends back an HTTP 200 response.

The following data is returned in JSON format by the service.

#### [checksum \(p. 283\)](#)

Checksum of the `$LATEST` version of the slot type.

Type: String

#### [createdDate \(p. 283\)](#)

The date that the slot type was created.



Type: Timestamp

**description (p. 283)**

A description of the slot type.

Type: String

Length Constraints: Minimum length of 0. Maximum length of 200.

**enumerationValues (p. 283)**

A list of `EnumerationValue` objects that defines the values that the slot type can take.

Type: Array of `EnumerationValue` (p. 310) objects

Array Members: Minimum number of 1 item. Maximum number of 10000 items.

**lastUpdatedDate (p. 283)**

The date that the slot type was updated. When you create a slot type, the creation date and last update date are the same.

Type: Timestamp

**name (p. 283)**

The name of the slot type.

Type: String

Length Constraints: Minimum length of 1. Maximum length of 100.

Pattern: `^[a-zA-Z]+((_[a-zA-Z]+)|([a-zA-Z]+_)*|_)`

**version (p. 283)**

The version of the slot type. For a new slot type, the version is always `$LATEST`.

Type: String

Length Constraints: Minimum length of 1. Maximum length of 64.

Pattern: `\$LATEST|[0-9]+`

## Errors

### **BadRequestException**

The request is not well formed. For example, a value is invalid or a required field is missing. Check the field values, and try again.

HTTP Status Code: 400

### **ConflictException**

There was a conflict processing the request. Try your request again.

HTTP Status Code: 409

### **InternalFailureException**

An internal Amazon Lex error occurred. Try your request again.

HTTP Status Code: 500

### **LimitExceededException**

The request exceeded a limit. Try your request again.

HTTP Status Code: 429

### **PreconditionFailedException**

The checksum of the resource that you are trying to change does not match the checksum in the request. Check the resource's checksum and try again.

HTTP Status Code: 412

## **See Also**

For more information about using this API in one of the language-specific AWS SDKs, see the following:

- [AWS Command Line Interface](#)
- [AWS SDK for .NET](#)
- [AWS SDK for C++](#)
- [AWS SDK for Go](#)
- [AWS SDK for Java](#)
- [AWS SDK for JavaScript](#)
- [AWS SDK for PHP V3](#)
- [AWS SDK for Python](#)
- [AWS SDK for Ruby V2](#)

## **Amazon Lex Runtime Service**

The following actions are supported by Amazon Lex Runtime Service:

- [PostContent](#) (p. 286)
- [PostText](#) (p. 293)

## PostContent

Service: Amazon Lex Runtime Service

Sends user input (text or speech) to Amazon Lex. Clients use this API to send text and audio requests to Amazon Lex at runtime. Amazon Lex interprets the user input using the machine learning model that it built for the bot.

The `PostContent` operation supports audio input at 8kHz and 16kHz. You can use 8kHz audio to achieve higher speech recognition accuracy in telephone audio applications.

In response, Amazon Lex returns the next message to convey to the user. Consider the following example messages:

- For a user input "I would like a pizza," Amazon Lex might return a response with a message eliciting slot data (for example, `PizzaSize`): "What size pizza would you like?".
- After the user provides all of the pizza order information, Amazon Lex might return a response with a message to get user confirmation: "Order the pizza?".
- After the user replies "Yes" to the confirmation prompt, Amazon Lex might return a conclusion statement: "Thank you, your cheese pizza has been ordered.".

Not all Amazon Lex messages require a response from the user. For example, conclusion statements do not require a response. Some messages require only a yes or no response. In addition to the `message`, Amazon Lex provides additional context about the message in the response that you can use to enhance client behavior, such as displaying the appropriate client user interface. Consider the following examples:

- If the message is to elicit slot data, Amazon Lex returns the following context information:
  - `x-amz-lex-dialog-state` header set to `ElicitSlot`
  - `x-amz-lex-intent-name` header set to the intent name in the current context
  - `x-amz-lex-slot-to-elicited` header set to the slot name for which the `message` is eliciting information
  - `x-amz-lex-slots` header set to a map of slots configured for the intent with their current values
- If the message is a confirmation prompt, the `x-amz-lex-dialog-state` header is set to `Confirmation` and the `x-amz-lex-slot-to-elicited` header is omitted.
- If the message is a clarification prompt configured for the intent, indicating that the user intent is not understood, the `x-amz-dialog-state` header is set to `ElicitIntent` and the `x-amz-slot-to-elicited` header is omitted.

In addition, Amazon Lex also returns your application-specific `sessionAttributes`. For more information, see [Managing Conversation Context](#).

## Request Syntax

```
POST /bot/botName/alias/botAlias/user/userId/content HTTP/1.1
x-amz-lex-session-attributes: sessionAttributes
Content-Type: contentType
Accept: accept

inputStream
```

## URI Request Parameters

The request requires the following URI parameters.

### **accept (p. 286)**

You pass this value as the `Accept` HTTP header.

The message Amazon Lex returns in the response can be either text or speech based on the `Accept` HTTP header value in the request.

- If the value is `text/plain; charset=utf-8`, Amazon Lex returns text in the response.
- If the value begins with `audio/`, Amazon Lex returns speech in the response. Amazon Lex uses Amazon Polly to generate the speech (using the configuration you specified in the `Accept` header). For example, if you specify `audio/mpeg` as the value, Amazon Lex returns speech in the MPEG format.

The following are the accepted values:

- `audio/mpeg`
- `audio/ogg`
- `audio/pcm`
- `text/plain; charset=utf-8`
- `audio/*` (defaults to `mpeg`)

### **botAlias (p. 286)**

Alias of the Amazon Lex bot.

### **botName (p. 286)**

Name of the Amazon Lex bot.

### **contentType (p. 286)**

You pass this value as the `Content-Type` HTTP header.

Indicates the audio format or text. The header value must start with one of the following prefixes:

- PCM format, audio data must be in little-endian byte order.
  - `audio/l16; rate=16000; channels=1`
  - `audio/x-l16; sample-rate=16000; channel-count=1`
  - `audio/lpcm; sample-rate=8000; sample-size-bits=16; channel-count=1; is-big-endian=false`
- Opus format
  - `audio/x-cbr-opus-with-preamble; preamble-size=0; bit-rate=256000; frame-size-milliseconds=4`
- Text format
  - `text/plain; charset=utf-8`

### **sessionAttributes (p. 286)**

You pass this value as the `x-amz-lex-session-attributes` HTTP header.

Application-specific information passed between Amazon Lex and a client application. The value must be a JSON serialized and base64 encoded map with string keys and values.

For more information, see [Setting Session Attributes](#).

### **userId (p. 286)**

The ID of the client application user. Amazon Lex uses this to identify a user's conversation with your bot. At runtime, each request must contain the `userId` field.

To decide the user ID to use for your application, consider the following factors.

- The `userId` field must not contain any personally identifiable information of the user, for example, name, personal identification numbers, or other end user personal information.
- If you want a user to start a conversation on one device and continue on another device, use a user-specific identifier.

- If you want the same user to be able to have two independent conversations on two different devices, choose a device-specific identifier.
- A user can't have two independent conversations with two different versions of the same bot. For example, a user can't have a conversation with the PROD and BETA versions of the same bot. If you anticipate that a user will need to have conversation with two different versions, for example, while testing, include the bot alias in the user ID to separate the two conversations.

Length Constraints: Minimum length of 2. Maximum length of 100.

Pattern: [0-9a-zA-Z.\_:-]+

## Request Body

The request accepts the following binary data.

<varlistentry> [inputStream \(p. 286\)](#)

User input in PCM or Opus audio format or text format as described in the `Content-Type` HTTP header.

You can stream audio data to Amazon Lex or you can create a local buffer that captures all of the audio data before sending. In general, you get better performance if you stream audio data rather than buffering the data locally.

</varlistentry>

## Response Syntax

```
HTTP/1.1 200
Content-Type: contentType
x-amz-lex-intent-name: intentName
x-amz-lex-slots: slots
x-amz-lex-session-attributes: sessionAttributes
x-amz-lex-message: message
x-amz-lex-dialog-state: dialogState
x-amz-lex-slot-to-elicit: slotToElicit
x-amz-lex-input-transcript: inputTranscript

audioStream
```

## Response Elements

If the action is successful, the service sends back an HTTP 200 response.

The response returns the following HTTP headers.

[contentType \(p. 288\)](#)

Content type as specified in the `Accept` HTTP header in the request.

[dialogState \(p. 288\)](#)

Identifies the current state of the user interaction. Amazon Lex returns one of the following values as `dialogState`. The client can optionally use this information to customize the user interface.

- `ElicitIntent` – Amazon Lex wants to elicit the user's intent. Consider the following examples:

For example, a user might utter an intent ("I want to order a pizza"). If Amazon Lex cannot infer the user intent from this utterance, it will return this dialog state.

- `ConfirmIntent` – Amazon Lex is expecting a "yes" or "no" response.

For example, Amazon Lex wants user confirmation before fulfilling an intent. Instead of a simple "yes" or "no" response, a user might respond with additional information. For example, "yes,

but make it a thick crust pizza" or "no, I want to order a drink." Amazon Lex can process such additional information (in these examples, update the crust type slot or change the intent from OrderPizza to OrderDrink).

- `ElicitSlot` – Amazon Lex is expecting the value of a slot for the current intent.

For example, suppose that in the response Amazon Lex sends this message: "What size pizza would you like?". A user might reply with the slot value (e.g., "medium"). The user might also provide additional information in the response (e.g., "medium thick crust pizza"). Amazon Lex can process such additional information appropriately.

- `Fulfilled` – Conveys that the Lambda function has successfully fulfilled the intent.
- `ReadyForFulfillment` – Conveys that the client has to fulfill the request.
- `Failed` – Conveys that the conversation with the user failed.

This can happen for various reasons, including that the user does not provide an appropriate response to prompts from the service (you can configure how many times Amazon Lex can prompt a user for specific information), or if the Lambda function fails to fulfill the intent.

Valid Values: `ElicitIntent` | `ConfirmIntent` | `ElicitSlot` | `Fulfilled` | `ReadyForFulfillment` | `Failed`

#### [inputTranscript \(p. 288\)](#)

The text used to process the request.

If the input was an audio stream, the `inputTranscript` field contains the text extracted from the audio stream. This is the text that is actually processed to recognize intents and slot values. You can use this information to determine if Amazon Lex is correctly processing the audio that you send.

#### [intentName \(p. 288\)](#)

Current user intent that Amazon Lex is aware of.

#### [message \(p. 288\)](#)

Message to convey to the user. It can come from the bot's configuration or a code hook (Lambda function). If the current intent is not configured with a code hook or if the code hook returned `Delegate` as the `dialogAction.type` in its response, then Amazon Lex decides the next course of action and selects an appropriate message from the bot configuration based on the current user interaction context. For example, if Amazon Lex is not able to understand the user input, it uses a clarification prompt message (For more information, see the Error Handling section in the Amazon Lex console). Another example: if the intent requires confirmation before fulfillment, then Amazon Lex uses the confirmation prompt message in the intent configuration. If the code hook returns a message, Amazon Lex passes it as-is in its response to the client.

Length Constraints: Minimum length of 1. Maximum length of 1024.

#### [sessionAttributes \(p. 288\)](#)

Map of key/value pairs representing the session-specific context information.

#### [slots \(p. 288\)](#)

Map of zero or more intent slots (name/value pairs) Amazon Lex detected from the user input during the conversation.

#### [slotToElicit \(p. 288\)](#)

If the `dialogState` value is `ElicitSlot`, returns the name of the slot for which Amazon Lex is eliciting a value.

The response returns the following as the HTTP body.

<varlistentry> [audioStream \(p. 288\)](#)

The prompt (or statement) to convey to the user. This is based on the bot configuration and context. For example, if Amazon Lex did not understand the user intent, it sends the `clarificationPrompt` configured for the bot. If the intent requires confirmation before taking the fulfillment action, it sends the `confirmationPrompt`. Another example: Suppose that the Lambda function successfully fulfilled the intent, and sent a message to convey to the user. Then Amazon Lex sends that message in the response. `</varlistentry>`

## Errors

### **BadGatewayException**

Either the Amazon Lex bot is still building, or one of the dependent services (Amazon Polly, AWS Lambda) failed with an internal service error.

HTTP Status Code: 502

### **BadRequestException**

Request validation failed, there is no usable message in the context, or the bot build failed, is still in progress, or contains unbuilt changes.

HTTP Status Code: 400

### **ConflictException**

Two clients are using the same AWS account, Amazon Lex bot, and user ID.

HTTP Status Code: 409

### **DependencyFailedException**

One of the dependencies, such as AWS Lambda or Amazon Polly, threw an exception. For example,

- If Amazon Lex does not have sufficient permissions to call a Lambda function.
- If a Lambda function takes longer than 30 seconds to execute.
- If a fulfillment Lambda function returns a `Delegate` dialog action without removing any slot values.

HTTP Status Code: 424

### **InternalFailureException**

Internal service error. Retry the call.

HTTP Status Code: 500

### **LimitExceededException**

Exceeded a limit.

HTTP Status Code: 429

### **LoopDetectedException**

This exception is not used.

HTTP Status Code: 508

### **NotAcceptableException**

The accept header in the request does not have a valid value.

HTTP Status Code: 406

### **NotFoundException**

The resource (such as the Amazon Lex bot or an alias) that is referred to is not found.

HTTP Status Code: 404

#### **RequestTimeoutException**

The input speech is too long.

HTTP Status Code: 408

#### **UnsupportedMediaTypeException**

The Content-Type header (PostContent API) has an invalid value.

HTTP Status Code: 415

## Example

### Example 1

In this request, the URI identifies a bot (Traffic), bot version (\$LATEST), and end user name (someuser). The Content-Type header identifies the format of the audio in the body. Amazon Lex also supports other formats. To convert audio from one format to another, if necessary, you can use SoX open source software. You specify the format in which you want to get the response by adding the Accept HTTP header.

In the response, the x-amz-lex-message header shows the response that Amazon Lex returned. The client can then send this response to the user. The same message is sent in audio/MPEG format through chunked encoding (as requested).

### Sample Request

```
"POST /bot/Traffic/alias/$LATEST/user/someuser/content HTTP/1.1[\r][\n]"
"x-amz-lex-session-attributes: eyJlc2VybmFtZSI6IkVvYiJ9[\r][\n]"
"Content-Type: audio/x-l16; channel-count=1; sample-rate=16000f[\r][\n]"
"Accept: audio/mpeg[\r][\n]"
"Host: runtime.lex.us-east-1.amazonaws.com[\r][\n]"
"Authorization: AWS4-HMAC-SHA256 Credential=BLANKED_OUT/20161230/us-east-1/lex/
aws4_request,
SignedHeaders=accept;content-type;host;x-amz-content-sha256;x-amz-date;x-amz-lex-session-
attributes, Signature=78ca5b54ea3f64a17ff7522de02cd90a9acd2365b45a9ce9b96ea105bb1c7ec2[\r]
[\n]"
"X-Amz-Date: 20161230T181426Z[\r][\n]"
"X-Amz-Content-Sha256: e3b0c44298fc1c149afb4c8996fb92427ae41e4649b934ca495991b7852b855[\r]
[\n]"
"Transfer-Encoding: chunked[\r][\n]"
"Connection: Keep-Alive[\r][\n]"
"User-Agent: Apache-HttpClient/4.5.x (Java/1.8.0_112)[\r][\n]"
"Accept-Encoding: gzip,deflate[\r][\n]"
"[\r][\n]"
"1000[\r][\n]"
"[0x7][0x0][0x7][0x0][\n]"
"[0x0][0x7][0x0][0xfc][0xff][\n]"
"[0x0][\n]"
...
```

### Sample Response

```
"HTTP/1.1 200 OK[\r][\n]"
"x-amzn-RequestId: cc8b34af-cebb-11e6-a35c-55f3a992f28d[\r][\n]"
"x-amz-lex-message: Sorry, can you repeat that?[\r][\n]"
"x-amz-lex-dialog-state: ElicitIntent[\r][\n]"
```



...

For more information about using this API in one of the language-specific AWS SDKs, see the following:

## PostText

Service: Amazon Lex Runtime Service

Sends user input (text-only) to Amazon Lex. Client applications can use this API to send requests to Amazon Lex at runtime. Amazon Lex then interprets the user input using the machine learning model it built for the bot.

In response, Amazon Lex returns the next `message` to convey to the user an optional `responseCard` to display. Consider the following example messages:

- For a user input "I would like a pizza", Amazon Lex might return a response with a message eliciting slot data (for example, `PizzaSize`): "What size pizza would you like?"
- After the user provides all of the pizza order information, Amazon Lex might return a response with a message to obtain user confirmation "Proceed with the pizza order?".
- After the user replies to a confirmation prompt with a "yes", Amazon Lex might return a conclusion statement: "Thank you, your cheese pizza has been ordered.".

Not all Amazon Lex messages require a user response. For example, a conclusion statement does not require a response. Some messages require only a "yes" or "no" user response. In addition to the `message`, Amazon Lex provides additional context about the message in the response that you might use to enhance client behavior, for example, to display the appropriate client user interface. These are the `slotToElicit`, `dialogState`, `intentName`, and `slots` fields in the response. Consider the following examples:

- If the message is to elicit slot data, Amazon Lex returns the following context information:
  - `dialogState` set to `ElicitSlot`
  - `intentName` set to the intent name in the current context
  - `slotToElicit` set to the slot name for which the `message` is eliciting information
  - `slots` set to a map of slots, configured for the intent, with currently known values
- If the message is a confirmation prompt, the `dialogState` is set to `ConfirmIntent` and `slotToElicit` is set to null.
- If the message is a clarification prompt (configured for the intent) that indicates that user intent is not understood, the `dialogState` is set to `ElicitIntent` and `slotToElicit` is set to null.

In addition, Amazon Lex also returns your application-specific `sessionAttributes`. For more information, see [Managing Conversation Context](#).

## Request Syntax

```
POST /bot/botName/alias/botAlias/user/userId/text HTTP/1.1
Content-type: application/json

{
  "inputText": "string",
  "sessionAttributes": {
    "string" : "string"
  }
}
```

## URI Request Parameters

The request requires the following URI parameters.

### **botAlias (p. 293)**

The alias of the Amazon Lex bot.

### **botName (p. 293)**

The name of the Amazon Lex bot.

### **userId (p. 293)**

The ID of the client application user. Amazon Lex uses this to identify a user's conversation with your bot. At runtime, each request must contain the `userId` field.

To decide the user ID to use for your application, consider the following factors.

- The `userId` field must not contain any personally identifiable information of the user, for example, name, personal identification numbers, or other end user personal information.
- If you want a user to start a conversation on one device and continue on another device, use a user-specific identifier.
- If you want the same user to be able to have two independent conversations on two different devices, choose a device-specific identifier.
- A user can't have two independent conversations with two different versions of the same bot. For example, a user can't have a conversation with the PROD and BETA versions of the same bot. If you anticipate that a user will need to have conversation with two different versions, for example, while testing, include the bot alias in the user ID to separate the two conversations.

Length Constraints: Minimum length of 2. Maximum length of 100.

Pattern: `[0-9a-zA-Z._:-]+`

## **Request Body**

The request accepts the following data in JSON format.

### **inputText (p. 293)**

The text that the user entered (Amazon Lex interprets this text).

Type: String

Length Constraints: Minimum length of 1. Maximum length of 1024.

Required: Yes

### **sessionAttributes (p. 293)**

Application-specific information passed between Amazon Lex and a client application. The value must be a JSON serialized and base64 encoded map with string keys and values.

For more information, see [Setting Session Attributes](#).

Type: String to string map

Required: No

## **Response Syntax**

HTTP/1.1 200
--------------

Content-type: application/json

```
{
  "dialogState": "string",
  "intentName": "string",
  "message": "string",
  "responseCard": {
    "contentType": "string",
    "genericAttachments": [
      {
        "attachmentLinkUrl": "string",
        "buttons": [
          {
            "text": "string",
            "value": "string"
          }
        ],
        "imageUrl": "string",
        "subTitle": "string",
        "title": "string"
      }
    ],
    "version": "string"
  },
  "sessionAttributes": {
    "string": "string"
  },
  "slots": {
    "string": "string"
  },
  "slotToElicit": "string"
}
```

## Response Elements

If the action is successful, the service sends back an HTTP 200 response.

The following data is returned in JSON format by the service.

### dialogState (p. 294)

Identifies the current state of the user interaction. Amazon Lex returns one of the following values as `dialogState`. The client can optionally use this information to customize the user interface.

- `ElicitIntent` – Amazon Lex wants to elicit user intent.

For example, a user might utter an intent ("I want to order a pizza"). If Amazon Lex cannot infer the user intent from this utterance, it will return this `dialogState`.

- `ConfirmIntent` – Amazon Lex is expecting a "yes" or "no" response.

For example, Amazon Lex wants user confirmation before fulfilling an intent.

Instead of a simple "yes" or "no," a user might respond with additional information. For example, "yes, but make it thick crust pizza" or "no, I want to order a drink". Amazon Lex can process such additional information (in these examples, update the crust type slot value, or change intent from `OrderPizza` to `OrderDrink`).

- `ElicitSlot` – Amazon Lex is expecting a slot value for the current intent.

For example, suppose that in the response Amazon Lex sends this message: "What size pizza would you like?". A user might reply with the slot value (e.g., "medium"). The user might also provide additional information in the response (e.g., "medium thick crust pizza"). Amazon Lex can process such additional information appropriately.

- **Fulfilled** – Conveys that the Lambda function configured for the intent has successfully fulfilled the intent.
- **ReadyForFulfillment** – Conveys that the client has to fulfill the intent.
- **Failed** – Conveys that the conversation with the user failed.

This can happen for various reasons including that the user did not provide an appropriate response to prompts from the service (you can configure how many times Amazon Lex can prompt a user for specific information), or the Lambda function failed to fulfill the intent.

Type: String

Valid Values: `ElicitIntent` | `ConfirmIntent` | `ElicitSlot` | `Fulfilled` | `ReadyForFulfillment` | `Failed`

#### **intentName (p. 294)**

The current user intent that Amazon Lex is aware of.

Type: String

#### **message (p. 294)**

A message to convey to the user. It can come from the bot's configuration or a code hook (Lambda function). If the current intent is not configured with a code hook or the code hook returned `Delegate` as the `dialogAction.type` in its response, then Amazon Lex decides the next course of action and selects an appropriate message from the bot configuration based on the current user interaction context. For example, if Amazon Lex is not able to understand the user input, it uses a clarification prompt message (for more information, see the Error Handling section in the Amazon Lex console). Another example: if the intent requires confirmation before fulfillment, then Amazon Lex uses the confirmation prompt message in the intent configuration. If the code hook returns a message, Amazon Lex passes it as-is in its response to the client.

Type: String

Length Constraints: Minimum length of 1. Maximum length of 1024.

#### **responseCard (p. 294)**

Represents the options that the user has to respond to the current prompt. Response Card can come from the bot configuration (in the Amazon Lex console, choose the settings button next to a slot) or from a code hook (Lambda function).

Type: [ResponseCard \(p. 329\)](#) object

#### **sessionAttributes (p. 294)**

A map of key-value pairs representing the session-specific context information.

Type: String to string map

#### **slots (p. 294)**

The intent slots (name/value pairs) that Amazon Lex detected so far from the user input in the conversation.

Type: String to string map

#### **slotToElicit (p. 294)**

If the `dialogState` value is `ElicitSlot`, returns the name of the slot for which Amazon Lex is eliciting a value.

Type: String

## Errors

### **BadGatewayException**

Either the Amazon Lex bot is still building, or one of the dependent services (Amazon Polly, AWS Lambda) failed with an internal service error.

HTTP Status Code: 502

### **BadRequestException**

Request validation failed, there is no usable message in the context, or the bot build failed, is still in progress, or contains unbuilt changes.

HTTP Status Code: 400

### **ConflictException**

Two clients are using the same AWS account, Amazon Lex bot, and user ID.

HTTP Status Code: 409

### **DependencyFailedException**

One of the dependencies, such as AWS Lambda or Amazon Polly, threw an exception. For example,

- If Amazon Lex does not have sufficient permissions to call a Lambda function.
- If a Lambda function takes longer than 30 seconds to execute.
- If a fulfillment Lambda function returns a `Delegate` dialog action without removing any slot values.

HTTP Status Code: 424

### **InternalFailureException**

Internal service error. Retry the call.

HTTP Status Code: 500

### **LimitExceededException**

Exceeded a limit.

HTTP Status Code: 429

### **LoopDetectedException**

This exception is not used.

HTTP Status Code: 508

### **NotFoundException**

The resource (such as the Amazon Lex bot or an alias) that is referred to is not found.

HTTP Status Code: 404

## See Also

For more information about using this API in one of the language-specific AWS SDKs, see the following:

- [AWS Command Line Interface](#)
- [AWS SDK for .NET](#)
- [AWS SDK for C++](#)

- [AWS SDK for Go](#)
- [AWS SDK for Java](#)
- [AWS SDK for JavaScript](#)
- [AWS SDK for PHP V3](#)
- [AWS SDK for Python](#)
- [AWS SDK for Ruby V2](#)

## Data Types

The following data types are supported by Amazon Lex Model Building Service:

- [BotAliasMetadata](#) (p. 300)
- [BotChannelAssociation](#) (p. 302)
- [BotMetadata](#) (p. 304)
- [BuiltinIntentMetadata](#) (p. 306)
- [BuiltinIntentSlot](#) (p. 307)
- [BuiltinSlotTypeMetadata](#) (p. 308)
- [CodeHook](#) (p. 309)
- [EnumerationValue](#) (p. 310)
- [FollowUpPrompt](#) (p. 311)
- [FulfillmentActivity](#) (p. 312)
- [Intent](#) (p. 313)
- [IntentMetadata](#) (p. 314)
- [Message](#) (p. 316)
- [Prompt](#) (p. 317)
- [ResourceReference](#) (p. 318)
- [Slot](#) (p. 319)
- [SlotTypeMetadata](#) (p. 321)
- [Statement](#) (p. 323)
- [UtteranceData](#) (p. 324)
- [UtteranceList](#) (p. 325)

The following data types are supported by Amazon Lex Runtime Service:

- [Button](#) (p. 326)
- [GenericAttachment](#) (p. 327)
- [ResponseCard](#) (p. 329)

## Amazon Lex Model Building Service

The following data types are supported by Amazon Lex Model Building Service:

- [BotAliasMetadata](#) (p. 300)
- [BotChannelAssociation](#) (p. 302)
- [BotMetadata](#) (p. 304)
- [BuiltinIntentMetadata](#) (p. 306)

- [BuiltinIntentSlot](#) (p. 307)
- [BuiltinSlotTypeMetadata](#) (p. 308)
- [CodeHook](#) (p. 309)
- [EnumerationValue](#) (p. 310)
- [FollowUpPrompt](#) (p. 311)
- [FulfillmentActivity](#) (p. 312)
- [Intent](#) (p. 313)
- [IntentMetadata](#) (p. 314)
- [Message](#) (p. 316)
- [Prompt](#) (p. 317)
- [ResourceReference](#) (p. 318)
- [Slot](#) (p. 319)
- [SlotTypeMetadata](#) (p. 321)
- [Statement](#) (p. 323)
- [UtteranceData](#) (p. 324)
- [UtteranceList](#) (p. 325)



## BotAliasMetadata

Service: Amazon Lex Model Building Service

Provides information about a bot alias.

### Contents

#### **botName**

The name of the bot to which the alias points.

Type: String

Length Constraints: Minimum length of 2. Maximum length of 50.

Pattern: `^[a-zA-Z]+(\_[a-zA-Z]+)*|([a-zA-Z]+_\_)|_\_`

Required: No

#### **botVersion**

The version of the Amazon Lex bot to which the alias points.

Type: String

Length Constraints: Minimum length of 1. Maximum length of 64.

Pattern: `\$LATEST|[0-9]+`

Required: No

#### **checksum**

Checksum of the bot alias.

Type: String

Required: No

#### **createdDate**

The date that the bot alias was created.

Type: Timestamp

Required: No

#### **description**

A description of the bot alias.

Type: String

Length Constraints: Minimum length of 0. Maximum length of 200.

Required: No

#### **lastUpdatedDate**

The date that the bot alias was updated. When you create a resource, the creation date and last updated date are the same.

Type: Timestamp

Required: No

**name**

The name of the bot alias.

Type: String

Length Constraints: Minimum length of 1. Maximum length of 100.

Pattern: `^[a-zA-Z]+((_[a-zA-Z]+)|([a-zA-Z]+_)*|_)`

Required: No

## See Also

For more information about using this API in one of the language-specific AWS SDKs, see the following:

- [AWS SDK for C++](#)
- [AWS SDK for Go](#)
- [AWS SDK for Java](#)
- [AWS SDK for Ruby V2](#)

## BotChannelAssociation

Service: Amazon Lex Model Building Service

Represents an association between an Amazon Lex bot and an external messaging platform.

### Contents

#### botAlias

An alias pointing to the specific version of the Amazon Lex bot to which this association is being made.

Type: String

Length Constraints: Minimum length of 1. Maximum length of 100.

Pattern: `^[a-zA-Z]+((_[a-zA-Z]+)*|([a-zA-Z]+_)*|_)`

Required: No

#### botConfiguration

Provides information necessary to communicate with the messaging platform.

Type: String to string map

Required: No

#### botName

The name of the Amazon Lex bot to which this association is being made.

##### Note

Currently, Amazon Lex supports associations with Facebook and Slack, and Twilio.

Type: String

Length Constraints: Minimum length of 2. Maximum length of 50.

Pattern: `^[a-zA-Z]+((_[a-zA-Z]+)*|([a-zA-Z]+_)*|_)`

Required: No

#### createdDate

The date that the association between the Amazon Lex bot and the channel was created.

Type: Timestamp

Required: No

#### description

A text description of the association you are creating.

Type: String

Length Constraints: Minimum length of 0. Maximum length of 200.

Required: No

#### name

The name of the association between the bot and the channel.

Type: String

Length Constraints: Minimum length of 1. Maximum length of 100.

Pattern: `^[a-zA-Z]+((_[a-zA-Z]+)|([a-zA-Z]+_))*|_`

Required: No

### **type**

Specifies the type of association by indicating the type of channel being established between the Amazon Lex bot and the external messaging platform.

Type: String

Valid Values: `Facebook` | `Slack` | `Twilio-Sms`

Required: No

## **See Also**

For more information about using this API in one of the language-specific AWS SDKs, see the following:

- [AWS SDK for C++](#)
- [AWS SDK for Go](#)
- [AWS SDK for Java](#)
- [AWS SDK for Ruby V2](#)

## BotMetadata

Service: Amazon Lex Model Building Service

Provides information about a bot. .

### Contents

#### **createdDate**

The date that the bot was created.

Type: Timestamp

Required: No

#### **description**

A description of the bot.

Type: String

Length Constraints: Minimum length of 0. Maximum length of 200.

Required: No

#### **lastUpdatedDate**

The date that the bot was updated. When you create a bot, the creation date and last updated date are the same.

Type: Timestamp

Required: No

#### **name**

The name of the bot.

Type: String

Length Constraints: Minimum length of 2. Maximum length of 50.

Pattern: `^[a-zA-Z]+((_[a-zA-Z]+)*|([a-zA-Z]+_)*|_)`

Required: No

#### **status**

The status of the bot.

Type: String

Valid Values: `BUILDING` | `READY` | `FAILED` | `NOT_BUILT`

Required: No

#### **version**

The version of the bot. For a new bot, the version is always `$LATEST`.

Type: String

Length Constraints: Minimum length of 1. Maximum length of 64.

Pattern: `\$LATEST|[0-9]+`

Required: No

## See Also

For more information about using this API in one of the language-specific AWS SDKs, see the following:

- [AWS SDK for C++](#)
- [AWS SDK for Go](#)
- [AWS SDK for Java](#)
- [AWS SDK for Ruby V2](#)

## BuiltinIntentMetadata

Service: Amazon Lex Model Building Service

Provides metadata for a built-in intent.

### Contents

#### **signature**

A unique identifier for the built-in intent. To find the signature for an intent, see [Standard Built-in Intents](#) in the *Alexa Skills Kit*.

Type: String

Required: No

#### **supportedLocales**

A list of identifiers for the locales that the intent supports.

Type: Array of strings

Valid Values: `en-US`

Required: No

### See Also

For more information about using this API in one of the language-specific AWS SDKs, see the following:

- [AWS SDK for C++](#)
- [AWS SDK for Go](#)
- [AWS SDK for Java](#)
- [AWS SDK for Ruby V2](#)

## BuiltinIntentSlot

Service: Amazon Lex Model Building Service

Provides information about a slot used in a built-in intent.

### Contents

#### **name**

A list of the slots defined for the intent.

Type: String

Required: No

### See Also

For more information about using this API in one of the language-specific AWS SDKs, see the following:

- [AWS SDK for C++](#)
- [AWS SDK for Go](#)
- [AWS SDK for Java](#)
- [AWS SDK for Ruby V2](#)



## BuiltinSlotTypeMetadata

Service: Amazon Lex Model Building Service

Provides information about a built in slot type.

### Contents

#### **signature**

A unique identifier for the built-in slot type. To find the signature for a slot type, see [Slot Type Reference](#) in the *Alexa Skills Kit*.

Type: String

Required: No

#### **supportedLocales**

A list of target locales for the slot.

Type: Array of strings

Valid Values: `en-US`

Required: No

### See Also

For more information about using this API in one of the language-specific AWS SDKs, see the following:

- [AWS SDK for C++](#)
- [AWS SDK for Go](#)
- [AWS SDK for Java](#)
- [AWS SDK for Ruby V2](#)

## CodeHook

Service: Amazon Lex Model Building Service

Specifies a Lambda function that verifies requests to a bot or fulfills the user's request to a bot..

### Contents

#### **messageVersion**

The version of the request-response that you want Amazon Lex to use to invoke your Lambda function. For more information, see [Using Lambda Functions \(p. 85\)](#).

Type: String

Length Constraints: Minimum length of 1. Maximum length of 5.

Required: Yes

#### **uri**

The Amazon Resource Name (ARN) of the Lambda function.

Type: String

Length Constraints: Minimum length of 20. Maximum length of 2048.

Pattern: `arn:aws:lambda:[a-z]+-[a-z]+-[0-9]:[0-9]{12}:function:[a-zA-Z0-9-_/]{0-9a-f}{8}-[0-9a-f]{4}-[0-9a-f]{4}-[0-9a-f]{4}-[0-9a-f]{12}}?(:[a-zA-Z0-9-_/]+)?`

Required: Yes

### See Also

For more information about using this API in one of the language-specific AWS SDKs, see the following:

- [AWS SDK for C++](#)
- [AWS SDK for Go](#)
- [AWS SDK for Java](#)
- [AWS SDK for Ruby V2](#)

## EnumerationValue

Service: Amazon Lex Model Building Service

Each slot type can have a set of values. Each enumeration value represents a value the slot type can take.

For example, a pizza ordering bot could have a slot type that specifies the type of crust that the pizza should have. The slot type could include the values

- thick
- thin
- stuffed

## Contents

### value

The value of the slot type.

Type: String

Length Constraints: Minimum length of 1. Maximum length of 140.

Required: Yes

## See Also

For more information about using this API in one of the language-specific AWS SDKs, see the following:

- [AWS SDK for C++](#)
- [AWS SDK for Go](#)
- [AWS SDK for Java](#)
- [AWS SDK for Ruby V2](#)

## FollowUpPrompt

Service: Amazon Lex Model Building Service

A prompt for additional activity after an intent is fulfilled. For example, after the `OrderPizza` intent is fulfilled, you might prompt the user to find out whether the user wants to order drinks.

### Contents

#### **prompt**

Prompts for information from the user.

Type: [Prompt \(p. 317\)](#) object

Required: Yes

#### **rejectionStatement**

If the user answers "no" to the question defined in the `prompt` field, Amazon Lex responds with this statement to acknowledge that the intent was canceled.

Type: [Statement \(p. 323\)](#) object

Required: Yes

### See Also

For more information about using this API in one of the language-specific AWS SDKs, see the following:

- [AWS SDK for C++](#)
- [AWS SDK for Go](#)
- [AWS SDK for Java](#)
- [AWS SDK for Ruby V2](#)

## FulfillmentActivity

Service: Amazon Lex Model Building Service

Describes how the intent is fulfilled after the user provides all of the information required for the intent. You can provide a Lambda function to process the intent, or you can return the intent information to the client application. We recommend that you use a Lambda function so that the relevant logic lives in the Cloud and limit the client-side code primarily to presentation. If you need to update the logic, you only update the Lambda function; you don't need to upgrade your client application.

Consider the following examples:

- In a pizza ordering application, after the user provides all of the information for placing an order, you use a Lambda function to place an order with a pizzeria.
- In a gaming application, when a user says "pick up a rock," this information must go back to the client application so that it can perform the operation and update the graphics. In this case, you want Amazon Lex to return the intent data to the client.

## Contents

### codeHook

A description of the Lambda function that is run to fulfill the intent.

Type: [CodeHook](#) (p. 309) object

Required: No

### type

How the intent should be fulfilled, either by running a Lambda function or by returning the slot data to the client application.

Type: String

Valid Values: `ReturnIntent` | `CodeHook`

Required: Yes

## See Also

For more information about using this API in one of the language-specific AWS SDKs, see the following:

- [AWS SDK for C++](#)
- [AWS SDK for Go](#)
- [AWS SDK for Java](#)
- [AWS SDK for Ruby V2](#)

## Intent

Service: Amazon Lex Model Building Service

Identifies the specific version of an intent.

## Contents

### **intentName**

The name of the intent.

Type: String

Length Constraints: Minimum length of 1. Maximum length of 100.

Pattern: `^[a-zA-Z]+((_[a-zA-Z]+)*|([a-zA-Z]+_)*|_)`

Required: Yes

### **intentVersion**

The version of the intent.

Type: String

Length Constraints: Minimum length of 1. Maximum length of 64.

Pattern: `\$LATEST|[0-9]+`

Required: Yes

## See Also

For more information about using this API in one of the language-specific AWS SDKs, see the following:

- [AWS SDK for C++](#)
- [AWS SDK for Go](#)
- [AWS SDK for Java](#)
- [AWS SDK for Ruby V2](#)

## IntentMetadata

Service: Amazon Lex Model Building Service

Provides information about an intent.

### Contents

#### **createdDate**

The date that the intent was created.

Type: Timestamp

Required: No

#### **description**

A description of the intent.

Type: String

Length Constraints: Minimum length of 0. Maximum length of 200.

Required: No

#### **lastUpdatedDate**

The date that the intent was updated. When you create an intent, the creation date and last updated date are the same.

Type: Timestamp

Required: No

#### **name**

The name of the intent.

Type: String

Length Constraints: Minimum length of 1. Maximum length of 100.

Pattern: `^[a-zA-Z]+((_[a-zA-Z]+)*|([a-zA-Z]+_)*|_)`

Required: No

#### **version**

The version of the intent.

Type: String

Length Constraints: Minimum length of 1. Maximum length of 64.

Pattern: `\$LATEST|[0-9]+`

Required: No

### See Also

For more information about using this API in one of the language-specific AWS SDKs, see the following:

- [AWS SDK for C++](#)

- [AWS SDK for Go](#)
- [AWS SDK for Java](#)
- [AWS SDK for Ruby V2](#)



## Message

Service: Amazon Lex Model Building Service

The message object that provides the message text and its type.

### Contents

#### **content**

The text of the message.

Type: String

Length Constraints: Minimum length of 1. Maximum length of 1000.

Required: Yes

#### **contentType**

The content type of the message string.

Type: String

Valid Values: `PlainText` | `SSML`

Required: Yes

### See Also

For more information about using this API in one of the language-specific AWS SDKs, see the following:

- [AWS SDK for C++](#)
- [AWS SDK for Go](#)
- [AWS SDK for Java](#)
- [AWS SDK for Ruby V2](#)

## Prompt

Service: Amazon Lex Model Building Service

Obtains information from the user. To define a prompt, provide one or more messages and specify the number of attempts to get information from the user. If you provide more than one message, Amazon Lex chooses one of the messages to use to prompt the user. For more information, see [Amazon Lex: How It Works \(p. 3\)](#).

## Contents

### **maxAttempts**

The number of times to prompt the user for information.

Type: Integer

Valid Range: Minimum value of 1. Maximum value of 5.

Required: Yes

### **messages**

An array of objects, each of which provides a message string and its type. You can specify the message string in plain text or in Speech Synthesis Markup Language (SSML).

Type: Array of [Message \(p. 316\)](#) objects

Array Members: Minimum number of 1 item. Maximum number of 5 items.

Required: Yes

### **responseCard**

A response card. Amazon Lex uses this prompt at runtime, in the `PostText` API response. It substitutes session attributes and slot values for placeholders in the response card. For more information, see [Example: Using a Response Card \(p. 143\)](#).

Type: String

Length Constraints: Minimum length of 1. Maximum length of 50000.

Required: No

## See Also

For more information about using this API in one of the language-specific AWS SDKs, see the following:

- [AWS SDK for C++](#)
- [AWS SDK for Go](#)
- [AWS SDK for Java](#)
- [AWS SDK for Ruby V2](#)

## ResourceReference

Service: Amazon Lex Model Building Service

Describes the resource that refers to the resource that you are attempting to delete. This object is returned as part of the `ResourceInUseException` exception.

### Contents

#### **name**

The name of the resource that is using the resource that you are trying to delete.

Type: String

Length Constraints: Minimum length of 1. Maximum length of 64.

Pattern: `[a-zA-Z]+`

Required: No

#### **version**

The version of the resource that is using the resource that you are trying to delete.

Type: String

Length Constraints: Minimum length of 1. Maximum length of 64.

Pattern: `\$LATEST|[0-9]+`

Required: No

### See Also

For more information about using this API in one of the language-specific AWS SDKs, see the following:

- [AWS SDK for C++](#)
- [AWS SDK for Go](#)
- [AWS SDK for Java](#)
- [AWS SDK for Ruby V2](#)

## Slot

Service: Amazon Lex Model Building Service

Identifies the version of a specific slot.

### Contents

#### description

A description of the slot.

Type: String

Length Constraints: Minimum length of 0. Maximum length of 200.

Required: No

#### name

The name of the slot.

Type: String

Length Constraints: Minimum length of 1. Maximum length of 100.

Pattern: `^[a-zA-Z]+(([_|.])[a-zA-Z]+)*|([a-zA-Z]+(_|.))*|(_|.)*`

Required: Yes

#### priority

Directs Lex the order in which to elicit this slot value from the user. For example, if the intent has two slots with priorities 1 and 2, AWS Lex first elicits a value for the slot with priority 1.

If multiple slots share the same priority, the order in which Lex elicits values is arbitrary.

Type: Integer

Valid Range: Minimum value of 0. Maximum value of 100.

Required: No

#### responseCard

A set of possible responses for the slot type used by text-based clients. A user chooses an option from the response card, instead of using text to reply.

Type: String

Length Constraints: Minimum length of 1. Maximum length of 50000.

Required: No

#### sampleUtterances

If you know a specific pattern with which users might respond to an Amazon Lex request for a slot value, you can provide those utterances to improve accuracy. This is optional. In most cases, Amazon Lex is capable of understanding user utterances.

Type: Array of strings

Array Members: Minimum number of 0 items. Maximum number of 10 items.

Length Constraints: Minimum length of 1. Maximum length of 200.

Required: No

**slotConstraint**

Specifies whether the slot is required or optional.

Type: String

Valid Values: `Required` | `Optional`

Required: Yes

**slotType**

The type of the slot, either a custom slot type that you defined or one of the built-in slot types.

Type: String

Length Constraints: Minimum length of 1. Maximum length of 100.

Pattern: `^([a-zA-Z]|AMAZON.)([_[a-zA-Z]+)*|([a-zA-Z]+_)*|_)`

Required: No

**slotTypeVersion**

The version of the slot type.

Type: String

Length Constraints: Minimum length of 1. Maximum length of 64.

Pattern: `\$LATEST|[0-9]+`

Required: No

**valueElicitationPrompt**

The prompt that Amazon Lex uses to elicit the slot value from the user.

Type: [Prompt \(p. 317\)](#) object

Required: No

## See Also

For more information about using this API in one of the language-specific AWS SDKs, see the following:

- [AWS SDK for C++](#)
- [AWS SDK for Go](#)
- [AWS SDK for Java](#)
- [AWS SDK for Ruby V2](#)

## SlotTypeMetadata

Service: Amazon Lex Model Building Service

Provides information about a slot type..

### Contents

#### **createdDate**

The date that the slot type was created.

Type: Timestamp

Required: No

#### **description**

A description of the slot type.

Type: String

Length Constraints: Minimum length of 0. Maximum length of 200.

Required: No

#### **lastUpdatedDate**

The date that the slot type was updated. When you create a resource, the creation date and last updated date are the same.

Type: Timestamp

Required: No

#### **name**

The name of the slot type.

Type: String

Length Constraints: Minimum length of 1. Maximum length of 100.

Pattern: `^[a-zA-Z]+((_[a-zA-Z]+)*|([a-zA-Z]+_)*|_)`

Required: No

#### **version**

The version of the slot type.

Type: String

Length Constraints: Minimum length of 1. Maximum length of 64.

Pattern: `\$LATEST|[0-9]+`

Required: No

### See Also

For more information about using this API in one of the language-specific AWS SDKs, see the following:

- [AWS SDK for C++](#)

- [AWS SDK for Go](#)
- [AWS SDK for Java](#)
- [AWS SDK for Ruby V2](#)

## Statement

Service: Amazon Lex Model Building Service

A collection of messages that convey information to the user. At runtime, Amazon Lex selects the message to convey.

## Contents

### **messages**

A collection of message objects.

Type: Array of [Message \(p. 316\)](#) objects

Array Members: Minimum number of 1 item. Maximum number of 5 items.

Required: Yes

### **responseCard**

At runtime, if the client is using the [PostText](#) API, Amazon Lex includes the response card in the response. It substitutes all of the session attributes and slot values for placeholders in the response card.

Type: String

Length Constraints: Minimum length of 1. Maximum length of 50000.

Required: No

## See Also

For more information about using this API in one of the language-specific AWS SDKs, see the following:

- [AWS SDK for C++](#)
- [AWS SDK for Go](#)
- [AWS SDK for Java](#)
- [AWS SDK for Ruby V2](#)



## UtteranceData

Service: Amazon Lex Model Building Service

Provides information about a single utterance that was made to your bot.

### Contents

#### **count**

The number of times that the utterance was processed.

Type: Integer

Required: No

#### **distinctUsers**

The total number of individuals that used the utterance.

Type: Integer

Required: No

#### **firstUtteredDate**

The date that the utterance was first recorded.

Type: Timestamp

Required: No

#### **lastUtteredDate**

The date that the utterance was last recorded.

Type: Timestamp

Required: No

#### **utteranceString**

The text that was entered by the user or the text representation of an audio clip.

Type: String

Length Constraints: Minimum length of 1. Maximum length of 2000.

Required: No

### See Also

For more information about using this API in one of the language-specific AWS SDKs, see the following:

- [AWS SDK for C++](#)
- [AWS SDK for Go](#)
- [AWS SDK for Java](#)
- [AWS SDK for Ruby V2](#)

## UtteranceList

Service: Amazon Lex Model Building Service

Provides a list of utterances that have been made to a specific version of your bot. The list contains a maximum of 100 utterances.

### Contents

#### **botVersion**

The version of the bot that processed the list.

Type: String

Length Constraints: Minimum length of 1. Maximum length of 64.

Pattern: `\$LATEST|[0-9]+`

Required: No

#### **utterances**

One or more [UtteranceData \(p. 324\)](#) objects that contain information about the utterances that have been made to a bot. The maximum number of object is 100.

Type: Array of [UtteranceData \(p. 324\)](#) objects

Required: No

### See Also

For more information about using this API in one of the language-specific AWS SDKs, see the following:

- [AWS SDK for C++](#)
- [AWS SDK for Go](#)
- [AWS SDK for Java](#)
- [AWS SDK for Ruby V2](#)

## Amazon Lex Runtime Service

The following data types are supported by Amazon Lex Runtime Service:

- [Button \(p. 326\)](#)
- [GenericAttachment \(p. 327\)](#)
- [ResponseCard \(p. 329\)](#)

## Button

Service: Amazon Lex Runtime Service

Represents an option to be shown on the client platform (Facebook, Slack, etc.)

### Contents

#### **text**

Text that is visible to the user on the button.

Type: String

Length Constraints: Minimum length of 1. Maximum length of 15.

Required: Yes

#### **value**

The value sent to Amazon Lex when a user chooses the button. For example, consider button text "NYC." When the user chooses the button, the value sent can be "New York City."

Type: String

Length Constraints: Minimum length of 1. Maximum length of 1000.

Required: Yes

### See Also

For more information about using this API in one of the language-specific AWS SDKs, see the following:

- [AWS SDK for C++](#)
- [AWS SDK for Go](#)
- [AWS SDK for Java](#)
- [AWS SDK for Ruby V2](#)

## GenericAttachment

Service: Amazon Lex Runtime Service

Represents an option rendered to the user when a prompt is shown. It could be an image, a button, a link, or text.

### Contents

#### **attachmentLinkUrl**

The URL of an attachment to the response card.

Type: String

Length Constraints: Minimum length of 1. Maximum length of 2048.

Required: No

#### **buttons**

The list of options to show to the user.

Type: Array of [Button \(p. 326\)](#) objects

Array Members: Minimum number of 0 items. Maximum number of 5 items.

Required: No

#### **imageUrl**

The URL of an image that is displayed to the user.

Type: String

Length Constraints: Minimum length of 1. Maximum length of 2048.

Required: No

#### **subTitle**

The subtitle shown below the title.

Type: String

Length Constraints: Minimum length of 1. Maximum length of 80.

Required: No

#### **title**

The title of the option.

Type: String

Length Constraints: Minimum length of 1. Maximum length of 80.

Required: No

### See Also

For more information about using this API in one of the language-specific AWS SDKs, see the following:

- [AWS SDK for C++](#)

- [AWS SDK for Go](#)
- [AWS SDK for Java](#)
- [AWS SDK for Ruby V2](#)

## ResponseCard

Service: Amazon Lex Runtime Service

If you configure a response card when creating your bots, Amazon Lex substitutes the session attributes and slot values that are available, and then returns it. The response card can also come from a Lambda function ( `dialogCodeHook` and `fulfillmentActivity` on an intent).

### Contents

#### **contentType**

The content type of the response.

Type: String

Valid Values: `application/vnd.amazonaws.card.generic`

Required: No

#### **genericAttachments**

An array of attachment objects representing options.

Type: Array of [GenericAttachment](#) (p. 327) objects

Array Members: Minimum number of 0 items. Maximum number of 10 items.

Required: No

#### **version**

The version of the response card format.

Type: String

Required: No

### See Also

For more information about using this API in one of the language-specific AWS SDKs, see the following:

- [AWS SDK for C++](#)
- [AWS SDK for Go](#)
- [AWS SDK for Java](#)
- [AWS SDK for Ruby V2](#)

# Document History for Amazon Lex

The following table describes the documentation for this release of Amazon Lex.

- **Latest documentation update:** May 22, 2017

Change	Description	Date
Expanded documentation	Added Getting Started examples for the AWS CLI. For more information, see <a href="#">Step 4: Getting Started (AWS CLI)</a> (p. 58).	May 22, 2017
New guide	This is the first release of the <i>Amazon Lex User Guide</i> .	April 19, 2017

# AWS Glossary

For the latest AWS terminology, see the [AWS Glossary](#) in the *AWS General Reference*.