Amazon Lex

V2 Developer Guide
# Table of Contents

What is Amazon Lex V2? ........................................................................................................ 1
Are You a First-time User of Amazon Lex V2? .................................................................. 2
How it works ......................................................................................................................... 3
  Supported languages ......................................................................................................... 4
  Supported languages and locales ...................................................................................... 4
  Languages and locales supported by Amazon Lex V2 features ........................................ 5
Getting started .................................................................................................................... 7
  Step 1: Set Up an Account ............................................................................................... 7
    Sign Up for AWS ........................................................................................................... 7
    Create an IAM User ...................................................................................................... 8
    Next step ....................................................................................................................... 8
  Step 2: Getting started (Console) .................................................................................... 8
    Exercise 1: Create a bot from an example ................................................................... 9
    Exercise 2: Review the conversation flow ................................................................... 10
Building bots ..................................................................................................................... 17
  Creating a bot ................................................................................................................. 17
  Adding a language .......................................................................................................... 18
  Adding intents ............................................................................................................... 18
    Turning off responses ................................................................................................. 19
  Adding slot types .......................................................................................................... 19
  Testing a bot .................................................................................................................. 20
  Creating versions ......................................................................................................... 23
    The Draft version ....................................................................................................... 23
    Creating a version ....................................................................................................... 24
    Updating an Amazon Lex V2 bot ............................................................................... 24
    Deleting an Amazon Lex V2 bot or version ................................................................. 24
Built-in intents and slot types .......................................................................................... 24
  Built-in intents .............................................................................................................. 24
  Built-in slot types ......................................................................................................... 35
Using a custom grammar slot type .................................................................................... 43
  Adding a grammar slot type ......................................................................................... 44
  Grammar definition ...................................................................................................... 44
  Script format ................................................................................................................ 51
  Industry grammars ..................................................................................................... 58
Creating custom slot types ............................................................................................... 140
Creating a custom vocabulary to improve speech recognition ......................................... 141
  Custom vocabulary basics ........................................................................................... 141
  Best practices for creating a custom vocabulary .......................................................... 141
  Creating a custom vocabulary for eliciting intents ......................................................... 142
  Creating a custom vocabulary file ................................................................................ 143
Using multiple values in a slot ......................................................................................... 143
Using a Lambda function ................................................................................................. 145
  Attaching a Lambda function to a bot alias ................................................................. 145
  Input event format ....................................................................................................... 146
  Response format ......................................................................................................... 150
  Lambda router function ............................................................................................... 152
Using the Automated Chatbot Designer (Preview) ............................................................ 154
  Importing conversation transcripts .............................................................................. 154
    Importing transcripts from Contact Lens for Amazon Connect ................................. 155
    Prepare transcripts .................................................................................................... 155
    Upload your transcripts to an S3 bucket .................................................................... 156
    Analyze your transcripts using Amazon Lex V2 console .......................................... 156
  Creating intents and slot types ..................................................................................... 157
  Input transcript format ................................................................................................. 157
<table>
<thead>
<tr>
<th>Section</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>Output transcript format</td>
<td>158</td>
</tr>
<tr>
<td>Deploying bots</td>
<td>160</td>
</tr>
<tr>
<td>Aliases</td>
<td>160</td>
</tr>
<tr>
<td>Integrating with a Java application</td>
<td>161</td>
</tr>
<tr>
<td>Integrating your bots</td>
<td>165</td>
</tr>
<tr>
<td>Messaging platforms</td>
<td>165</td>
</tr>
<tr>
<td>Integrating with Facebook</td>
<td>165</td>
</tr>
<tr>
<td>Integrating with Slack</td>
<td>168</td>
</tr>
<tr>
<td>Integrating with Twilio SMS</td>
<td>171</td>
</tr>
<tr>
<td>Contact centers</td>
<td>172</td>
</tr>
<tr>
<td>Amazon Chime SDK</td>
<td>173</td>
</tr>
<tr>
<td>Amazon Connect</td>
<td>173</td>
</tr>
<tr>
<td>Genesys Cloud</td>
<td>174</td>
</tr>
<tr>
<td>Using bots</td>
<td>175</td>
</tr>
<tr>
<td>Managing conversations</td>
<td>175</td>
</tr>
<tr>
<td>Managing conversation context</td>
<td>176</td>
</tr>
<tr>
<td>Setting intent context</td>
<td>176</td>
</tr>
<tr>
<td>Using default slot values</td>
<td>178</td>
</tr>
<tr>
<td>Setting session attributes</td>
<td>178</td>
</tr>
<tr>
<td>Setting request attributes</td>
<td>180</td>
</tr>
<tr>
<td>Setting the session timeout</td>
<td>180</td>
</tr>
<tr>
<td>Sharing information between intents</td>
<td>181</td>
</tr>
<tr>
<td>Setting complex attributes</td>
<td>181</td>
</tr>
<tr>
<td>Managing sessions</td>
<td>182</td>
</tr>
<tr>
<td>Starting a new session</td>
<td>183</td>
</tr>
<tr>
<td>Switching intents</td>
<td>183</td>
</tr>
<tr>
<td>Resuming a prior intent</td>
<td>184</td>
</tr>
<tr>
<td>Validating slot values</td>
<td>184</td>
</tr>
<tr>
<td>Analyzing sentiment</td>
<td>184</td>
</tr>
<tr>
<td>Using confidence scores</td>
<td>185</td>
</tr>
<tr>
<td>Using intent confidence scores</td>
<td>186</td>
</tr>
<tr>
<td>Using voice transcription confidence scores</td>
<td>187</td>
</tr>
<tr>
<td>Using runtime hints to improve recognition of slot values</td>
<td>194</td>
</tr>
<tr>
<td>Adding slot values in context</td>
<td>195</td>
</tr>
<tr>
<td>Adding hints to a slot</td>
<td>195</td>
</tr>
<tr>
<td>Using spelling styles to capture slot values</td>
<td>196</td>
</tr>
<tr>
<td>Enabling spelling</td>
<td>197</td>
</tr>
<tr>
<td>Example code</td>
<td>197</td>
</tr>
<tr>
<td>Streaming conversations</td>
<td>201</td>
</tr>
<tr>
<td>Starting a stream to a bot</td>
<td>202</td>
</tr>
<tr>
<td>Time sequence of events for an audio conversation</td>
<td>204</td>
</tr>
<tr>
<td>Starting a streaming conversation</td>
<td>206</td>
</tr>
<tr>
<td>Event stream encoding</td>
<td>217</td>
</tr>
<tr>
<td>Enabling your bot to be interrupted</td>
<td>218</td>
</tr>
<tr>
<td>Waiting for the user to provide additional information</td>
<td>219</td>
</tr>
<tr>
<td>Configuring fulfillment progress updates</td>
<td>220</td>
</tr>
<tr>
<td>Fulfillment updates</td>
<td>221</td>
</tr>
<tr>
<td>Post-fulfillment response</td>
<td>222</td>
</tr>
<tr>
<td>Timeouts for user input</td>
<td>222</td>
</tr>
<tr>
<td>Timeouts for voice input</td>
<td>223</td>
</tr>
<tr>
<td>Timeouts for text input</td>
<td>223</td>
</tr>
<tr>
<td>Configuration for DTMF input</td>
<td>224</td>
</tr>
<tr>
<td>Monitoring</td>
<td>225</td>
</tr>
<tr>
<td>Monitoring with conversation logs</td>
<td>225</td>
</tr>
<tr>
<td>Configuring conversation logs</td>
<td>226</td>
</tr>
<tr>
<td>Viewing text logs in Amazon CloudWatch Logs</td>
<td>227</td>
</tr>
<tr>
<td>Accessing audio logs in Amazon S3</td>
<td>230</td>
</tr>
</tbody>
</table>
What is Amazon Lex V2?

Amazon Lex V2 is an AWS service for building conversational interfaces for applications using voice and text. Amazon Lex V2 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 V2 enables any developer to build conversational bots quickly. With Amazon Lex V2, no deep learning expertise is necessary—to create a bot, you specify the basic conversation flow in the Amazon Lex V2 console. Amazon Lex V2 manages the dialog 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 V2 provides integration with AWS Lambda, and you can integrate with many other services on the AWS platform, including Amazon Connect, Amazon Comprehend, and Amazon Kendra. Integration with Lambda provides bots access to pre-built serverless enterprise connectors to link to data in SaaS applications such as Salesforce.

Amazon Lex V2 provides the following benefits:

- **Simplicity** – Amazon Lex V2 guides you through using the console to create your own bot in minutes. You supply a few example phrases, and Amazon Lex V2 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** – Amazon Lex V2 provides ASR and NLU technologies to create a Speech Language Understanding (SLU) system. Through SLU, Amazon Lex V2 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 V2 puts deep learning technologies within reach of all developers. Amazon Lex V2 bots convert incoming speech to text and understand the user intent to generate an intelligent response so you can focus on building your bots with added value for your customers and define entirely new categories of products made possible through conversational interfaces.

- **Seamless deployment and scaling** – With Amazon Lex V2, you can build, test, and deploy your bots directly from the Amazon Lex V2 console. Amazon Lex V2 enables you to publish your voice or text bots for use on mobile devices, web apps, and chat services (for example, Facebook Messenger). Amazon Lex V2 scales automatically. 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 V2 operates natively with other AWS services, such as AWS Lambda and Amazon CloudWatch. 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 V2, 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 V2 free tier, you can easily try Amazon Lex V2 without any initial investment.

Are You a First-time User of Amazon Lex V2?

If you are a first-time user of Amazon Lex V2, we recommend that you read the following sections in order:

1. **How it works (p. 3)** – This section introduces Amazon Lex V2 and the features that you use to create a chatbot.
2. **Getting started with Amazon Lex V2 (p. 7)** – In this section, you set up your account and test Amazon Lex V2.
3. **API reference (p. 306)** – This section provides additional examples that you can use to explore Amazon Lex V2.
How it works

Amazon Lex V2 enables you to build applications using a text or speech interface for a conversation with a user. Following are the typical steps for working with Amazon Lex V2:

1. Create a bot and add one or more languages. Configure the bot so that it understands the user’s goal, engages in conversation with the user to elicit information, and fulfills the user’s intent.
2. Test the bot. You can use the test window client provided by the Amazon Lex V2 console.
3. Publish a version and create an alias.
4. Deploy the bot. You can deploy the bot on your own applications or messaging platforms such as Facebook Messenger or Slack.

Before you get started, familiarize yourself with the following Amazon Lex V2 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 V2 bot is powered by automatic speech recognition (ASR) and natural language understanding (NLU) capabilities.

  Amazon Lex V2 bots can understand user input provided with text or speech and converse natural language.

- **Language** – An Amazon Lex V2 bot can converse in one or more languages. Each language is independent of the others, you can configure Amazon Lex V2 to converse with a user using native words and phrases. For more information, see Languages and locales supported by Amazon Lex V2 (p. 4).

- **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 an intent that orders pizzas 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" 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. We recommend that you create a Lambda function to fulfill the intent.

  You can optionally configure the intent so Amazon Lex V2 returns the information back to the client application for the necessary fulfillment.

  In addition to custom intents, Amazon Lex V2 provides built-in intents to quickly set up your bot. For more information, see Built-in intents and slot types (p. 24).

  Amazon Lex always includes a fallback intent for each bot. The fallback intent is used whenever Amazon Lex can't deduce the user's intent. For more information, see AMAZON.FallbackIntent (p. 25).

- **Slot** – An intent can require zero or more slots, or parameters. You add slots as part of the intent configuration. At runtime, Amazon Lex V2 prompts the user for specific slot values. The user must provide values for all required slots before Amazon Lex V2 can fulfill the intent.

  For example the OrderPizza intent requires slots such as size, crust type, and number of pizzas. For each slot, you provide the slot type and one or more prompts that Amazon Lex V2 sends to the client to elicit values from the user. A user can reply with a slot value that contains additional words, such as "large pizza please" or "let's stick with small." Amazon Lex V2 still understands the slot value.
• **Slot type** – Each slot has a type. You can create your own slot type, or you can 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 V2 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. 24).

• **Version** – 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. Once you create a version, you can use a bot as it existed when the version was made. After you create a version, it stays the same while you continue to work on your application.

• **Alias** – An alias is a pointer to a specific version of a bot. With an alias, you can update the version the 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.

For a list of the AWS Regions where Amazon Lex V2 is available, see [Amazon Lex V2 endpoints and quotas](p. 24) in the *Amazon Web Services General Reference*.

## Languages and locales supported by Amazon Lex V2

Amazon Lex V2 supports a variety of languages and locales. The languages supported and the features they support are listed in the following tables.

### Supported languages and locales

Amazon Lex V2 supports the following languages and locales.

<table>
<thead>
<tr>
<th>Code</th>
<th>Language and locale</th>
</tr>
</thead>
<tbody>
<tr>
<td>ca_ES</td>
<td>Catalan (Spain)</td>
</tr>
<tr>
<td>de_AT</td>
<td>German (Austria)</td>
</tr>
<tr>
<td>de_DE</td>
<td>German (Germany)</td>
</tr>
<tr>
<td>en_AU</td>
<td>English (Australia)</td>
</tr>
<tr>
<td>en_GB</td>
<td>English (UK)</td>
</tr>
<tr>
<td>en_IN</td>
<td>English (India)</td>
</tr>
<tr>
<td>en_US</td>
<td>English (US)</td>
</tr>
<tr>
<td>en_ZA</td>
<td>English (South Africa)</td>
</tr>
<tr>
<td>es_419</td>
<td>Spanish (Latin America)</td>
</tr>
<tr>
<td>es_ES</td>
<td>Spanish (Spain)</td>
</tr>
</tbody>
</table>
Languages and locales supported by Amazon Lex V2 features

All Amazon Lex V2 features are supported in all languages and locales except as listed in this table.

<table>
<thead>
<tr>
<th>Feature</th>
<th>Supported languages and locales</th>
</tr>
</thead>
<tbody>
<tr>
<td>AMAZON.AlphaNumeric (p. 36)</td>
<td>All languages and locales except Korean (ko_KR)</td>
</tr>
<tr>
<td>AMAZON.KendraSearchIntent (p. 27)</td>
<td>English (US) (en_US)</td>
</tr>
<tr>
<td>Creating a custom vocabulary to improve speech recognition (p. 141)</td>
<td>English (GB) (en_GB)</td>
</tr>
<tr>
<td>Automated Chatbot Designer</td>
<td>English (US) (en_US)</td>
</tr>
<tr>
<td>Polly voice output</td>
<td>All languages and locales except Catalan (Spain) (ca_ES)</td>
</tr>
<tr>
<td>Region availability</td>
<td>All languages and locales are available in all Regions except Catalan (Spain) (ca_ES) Portuguese (Brazil) (pt_BR), Portuguese (Portugal), (pt_PT) and Mandarin (PRC) (zh_CN) are not available in the following Regions:</td>
</tr>
<tr>
<td></td>
<td>• Asia Pacific (Singapore) (ap-southeast-1)</td>
</tr>
<tr>
<td></td>
<td>• Africa (Cape Town) (ap-south-1)</td>
</tr>
<tr>
<td>Setting intent context (p. 176)</td>
<td>English (US) (en_US)</td>
</tr>
<tr>
<td>Using a custom grammar slot type (p. 43)</td>
<td>English (Australia) (en_AU)</td>
</tr>
<tr>
<td></td>
<td>English (UK) (en_GB)</td>
</tr>
<tr>
<td></td>
<td>English (US) (en_US)</td>
</tr>
<tr>
<td>Using multiple values in a slot (p. 143)</td>
<td>English (US) (en_US)</td>
</tr>
</tbody>
</table>
## Languages and locales supported by Amazon Lex V2 features

<table>
<thead>
<tr>
<th>Feature</th>
<th>Supported languages and locales</th>
</tr>
</thead>
<tbody>
<tr>
<td>Using runtime hints to improve recognition of slot values (p. 194)</td>
<td>English (UK) (en_GB)</td>
</tr>
<tr>
<td>Using spelling styles to capture slot values (p. 196)</td>
<td>English (UK) (en_GB)</td>
</tr>
<tr>
<td></td>
<td>English (US) (en_US)</td>
</tr>
<tr>
<td>Using voice transcription confidence scores (p. 187)</td>
<td>English (UK) (en_GB)</td>
</tr>
<tr>
<td></td>
<td>English (US) (en_US)</td>
</tr>
</tbody>
</table>
Getting started with Amazon Lex V2

Amazon Lex V2 provides API operations that you can integrate with your existing applications. For a list of supported operations, see the API reference (p. 306). You can use any of the following options:

- AWS SDK — When using the SDKs your requests to Amazon Lex V2 are automatically signed and authenticated using the credentials that you provide. We recommend that you use an SDK to build your application.
- AWS CLI — You can use the AWS CLI to access any Amazon Lex V2 feature without having to write any code.
- AWS Console — The console is the easiest way to get started testing and using Amazon Lex V2

If you are new to Amazon Lex V2, we recommend that you read How it works (p. 3) first.

Topics
- Step 1: Set Up an AWS Account and Create an Administrator User (p. 7)
- Step 2: Getting started (Console) (p. 8)

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

Before you use Amazon Lex V2 for the first time, complete the following tasks:

1. Sign Up for AWS (p. 7)
2. Create an IAM User (p. 8)

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 V2. You are charged only for the services that you use.

With Amazon Lex V2, you pay only for the resources that you use. If you are a new AWS customer, you can get started with Amazon Lex V2 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

2. Follow the online instructions.
Part of the sign-up procedure involves receiving a phone call and entering a verification code on 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 V2, 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:

- AWS Identity and Access Management (IAM)
- Getting started
- IAM User Guide

**Next step**

Step 2: Getting started (Console) (p. 8)

**Step 2: Getting started (Console)**

The easiest way to learn how to use Amazon Lex V2 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 V2 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.
Exercise 1: Create a bot from an example

In this exercise, you create your first Amazon Lex V2 bot and test it in the Amazon Lex V2 console. For this exercise, you use the OrderFlowers example.

Example overview

You use the OrderFlowers example to create an Amazon Lex V2 bot. For more information about the structure of a bot, see How it works (p. 3).

- **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?"

To create an Amazon Lex V2 bot (Console)

1. Sign in to the AWS Management Console and open the Amazon Lex V2 console at https://console.aws.amazon.com/lexv2/home
2. Choose Create bot.
3. For the Creation method, choose Start with an example.
4. In the Example bots section, choose OrderFlowers from the list.
5. In the Bot configuration section give the bot a name and a description. The name must be unique in your account.
6. In the Permissions section, choose Create a new role with basic Amazon Lex permissions. This will create an AWS Identity and Access Management (IAM) role with the permissions that Amazon Lex V2 needs to run your bot.
7. In the Children's Online Privacy Protection Act (COPPA) section, make the appropriate choice.
Exercise 2: Review the conversation flow

In this exercise you review the JSON structures that are sent between your client application and the Amazon Lex V2 bot that you created in Exercise 1: Create a bot from an example (p. 9). The conversation uses the RecognizeText (p. 628) operation to generate the JSON structures. The RecognizeUtterance (p. 635) returns the same information as HTTP headers in the response.

The JSON structures are divided by each turn of the conversation. A turn is a request from the client application and a response from the bot.

Turn 1

During the first turn of the conversation, the client application initiates the conversation with your bot. Both the URI and the body of the request provide information about the request.

```
POST /bots/botId/botAliases/botAliasId/botLocales/localeId/sessions/sessionId/text HTTP/1.1
Content-type: application/json

{
   "text": "I would like to order flowers"
}
```
• The URI identifies the bot that the client application is communicating with. It also includes a session identifier generated by the client application that identifies a specific conversation between a user and the bot.

• The body of the request contains the text that the user typed to the client application. In this case, only the text is sent, however you application can send additional information, such as request attributes or session state. For more information, see the RecognizeText (p. 628) operation.

From text, Amazon Lex V2 detects the user's intent, to order flowers. Amazon Lex V2 chooses one of the intent's slots (FlowerType) and one of the prompts for the slot, and then sends the following response to the client application. The client displays the response to the user.

```json
{
  "interpretations": [
    {
      "intent": {
        "confirmationState": "None",
        "name": "OrderFlowers",
        "slots": {
          "FlowerType": null,
          "PickupDate": null,
          "PickupTime": null
        },
        "state": "InProgress"
      },
      "nluConfidence": {
        "score": 0.95
      }
    },
    {
      "intent": {
        "name": "FallbackIntent",
        "slots": {}
      }
    }
  ],
  "messages": [
    {
      "content": "What type of flowers would you like to order?",
      "contentType": "PlainText"
    }
  ],
  "sessionId": "bf445a49-7165-4fcd-9a9c-a782493fa5c",
  "sessionState": {
    "dialogAction": {
      "slotToElicit": "FlowerType",
      "type": "ElicitSlot"
    },
    "intent": {
      "confirmationState": "None",
      "name": "OrderFlowers",
      "slots": {
        "FlowerType": null,
        "PickupDate": null,
        "PickupTime": null
      },
      "state": "InProgress"
    },
    "originatingRequestId": "9e8add70-4106-4a10-93f5-2ce2c959e5f"
  }
}
```
**Turns 2 and 3**

In turn 2 and 3, the user responds to prompts from the Amazon Lex V2 bot with values that fill the `FlowerType` and `PickupDate` slots.

The URI for the second and third turns is the same as the first.

```
{
  "text": "1 dozen roses"
}
```

The response for turn 2 shows the `FlowerType` slot filled and provides a prompt to elicit the next slot value.

```
{
  "interpretations": [
    {
      "intent": {
        "confirmationState": "None",
        "name": "OrderFlowers",
        "slots": {
          "FlowerType": {
            "value": {
              "interpretedValue": "dozen roses",
              "originalValue": "dozen roses",
              "resolvedValues": []
            }
          },
          "PickupDate": null,
          "PickupTime": null
        },
        "state": "InProgress"
      },
      "nluConfidence": {
        "score": 0.98
      }
    },
    {
      "intent": {
        "name": "FallbackIntent",
        "slots": {}
      }
    }
  ],
  "messages": [
    {
      "content": "What day do you want the dozen roses to be picked up?",
      "contentType": "PlainText"
    }
  ],
  "sessionId": "bf445a49-7165-4fcd-9a9c-a782493fba5c",
  "sessionState": {
    "dialogAction": {
      "slotToElicit": "PickupDate",
      "type": "ElicitSlot"
    },
    "intent": {
      "confirmationState": "None",
      "name": "OrderFlowers",
      "slots": {
        "FlowerType": {
          "value": {
            "interpretedValue": "dozen roses",
            "originalValue": "dozen roses",
            "resolvedValues": []
          }
        },
        "PickupDate": null,
        "PickupTime": null
      }
    }
  }
}
```
In turn 4, the user provides the final slot value for the intent, the time that the flowers are picked up.

```
{
    "text": "5 in the evening"
}
```

In the response, Amazon Lex V2 sends a confirmation prompt to the user to confirm that the order is correct. The `dialogAction` is set to `ConfirmIntent` and the `confirmationState` is `None`.

```
{
    "interpretations": [
    {
        "intent": {
            "confirmationState": "None",
            "name": "OrderFlowers",
            "slots": {
                "FlowerType": {
                    "value": {
                        "interpretedValue": "dozen roses",
                        "originalValue": "dozen roses",
                        "resolvedValues": []
                    }
                },
                "PickupDate": {
                    "value": {
                        "interpretedValue": "2021-01-04",
                        "originalValue": "next monday",
                        "resolvedValues": ["2021-01-04"
                    ]
                },
                "PickupTime": {
                    "value": {
                        "interpretedValue": "17:00",
                        "originalValue": "5 evening",
                        "resolvedValues": ["17:00"
                    ]
                }
            },
            "state": "InProgress"
        }
    }
}
```
In the final turn, the user responds with to the confirmation prompt.
In the response, Amazon Lex V2 sends indicates that the intent has been fulfilled by setting the confirmationState to Confirmed and the dialogAction to close. All of the slot values are available to the client application.

```
{
   "interpretations": [
      {
         "intent": {
            "confirmationState": "Confirmed",
            "name": "OrderFlowers",
            "slots": {
               "FlowerType": {
                  "value": {
                     "interpretedValue": "dozen roses",
                     "originalValue": "dozen roses",
                     "resolvedValues": []
                  }
               },
               "PickupDate": {
                  "value": {
                     "interpretedValue": "2021-01-04",
                     "originalValue": "next monday",
                     "resolvedValues": ["2021-01-04"]
                  }
               },
               "PickupTime": {
                  "value": {
                     "interpretedValue": "17:00",
                     "originalValue": "5 evening",
                     "resolvedValues": ["17:00"]
                  }
               }
            }
         },
         "state": "Fulfilled"
      },
      {
         "intent": {
            "name": "FallbackIntent",
            "slots": {}
         }
      }
   ],
   "messages": [
      {
         "content": "Thanks. ",
         "contentType": "PlainText"
      }
   ],
   "sessionId": "bf445a49-7165-4fcd-9a9c-a782493fba5c",
   "sessionState": {
      "dialogAction": {
         "type": "Close"
      }
   }
}``
```
"type": "Close",
}"
"confirmationState": "Confirmed",
"name": "OrderFlowers",
"slots": {
"FlowerType": {
"value": {
"interpretedValue": "dozen roses",
"originalValue": "dozen roses",
"resolvedValues": []
}
},
"PickupDate": {
"value": {
"interpretedValue": "2021-01-04",
"originalValue": "next Monday",
"resolvedValues": ["2021-01-04"
]
}
},
"PickupTime": {
"value": {
"interpretedValue": "17:00",
"originalValue": "5 evening",
"resolvedValues": ["17:00"
]
}
}
},
"state": "Fulfilled"
},
"originatingRequestId": "9e8add70-4106-4a10-93f5-2ce2cb959e5f"}
Building bots

You create an Amazon Lex V2 bot to interact with your users to elicit information to accomplish a task. For example, you can create a bot that gathers the information needed to order a bouquet of flowers or book a hotel room.

To build a bot, you need the following information:

1. The language that the bot uses to interact with the customer. You can choose one or more languages, each language contains independent intents, slots, and slot types.
2. The intents, or goals, that the bot will help the user fulfill. A bot can contain one or more intents, such as ordering flowers, or booking a hotel and rental car. You need to decide which statements, or utterances, that the user makes to trigger the intent.
3. The information, or slots, that you need to gather from the user to fulfill an intent. For example, you might need to get the type of flowers from the user or the start date of a hotel reservation. You need to define one or more prompts that Amazon Lex V2 uses to elicit the slot value from the user.
4. The type of the slots that you need from the user. You may need to create a custom slot type, such as a list of flowers that a user can order, or you can use a built-in slot type, such as using the AMAZON.Date slot type for the start date of a reservation.
5. The interactions between user and intents. Amazon Lex V2 manages the interactions; you can create an AWS Lambda function to validate and fulfill the intent.

Topics
- Creating a bot (p. 17)
- Adding a language (p. 18)
- Adding intents (p. 18)
- Adding slot types (p. 19)
- Testing a bot using the console (p. 20)
- Creating versions (p. 23)
- Built-in intents and slot types (p. 24)
- Using a custom grammar slot type (p. 43)
- Creating custom slot types (p. 140)
- Creating a custom vocabulary to improve speech recognition (p. 141)
- Using multiple values in a slot (p. 143)
- Using an AWS Lambda function (p. 145)

Creating a bot

Start creating your bot by defining the name, description and some basic information.

To create a bot

1. Sign in to the AWS Management Console and open the Amazon Lex V2 console at https://console.aws.amazon.com/lexv2/home
2. Choose Create bot.
3. In the Creation method section, choose Create.
4. In the Bot configuration section, give the bot a name and an optional description.
5. In the **IAM permissions** section, choose an AWS Identity and Access Management (IAM) role that provides Amazon Lex V2 permission to access other AWS services, such as Amazon CloudWatch. You can have Amazon Lex V2 create the role, or you can choose an existing role with CloudWatch permissions.

6. In the **Children's Online Privacy Protection Act (COPPA)** section, choose the appropriate response.

7. In the **Idle session timeout** section, choose the duration that Amazon Lex V2 keeps a session with a user open. Amazon Lex V2 maintains session variables for the duration of the session so that your bot can resume a conversation with the same variables.

8. In the **Advanced settings** section add tags that help identify the bot, and can be used to control access and monitor resources.

9. Choose **Next** to create the bot and move to adding a language.

---

**Adding a language**

You add one or more languages and locales to your bot to enable it to communicate with users in their languages. You define the intents, slots, and slot types separately for each language so that the utterances, prompts, and slot values are specific to the language.

Your bot must contain at least one language.

**To add a language to your bot**

1. In the **New language** section, choose the language that you want to use. You can add a description to help identify the language in lists.

2. If your bot supports voice interaction, in the **Voice interaction** section, choose the Amazon Polly voice that Amazon Lex V2 uses to communicate with the user. If your bot doesn't support voice, choose **None**.

3. For the **Intent classification confidence score threshold**, set the value that Amazon Lex V2 uses to determine whether an intent is the correct intent. You can adjust this value after testing your bot.

4. Choose **Add**.

---

**Adding intents**

Intents are the goals that your users want to fulfill, such as ordering flowers or booking a hotel. Your bot needs to have at least one intent.

By default, all bots contain a single built-in intent, the fallback intent. This intent is used when Amazon Lex V2 does not recognize any other intent. For example, if a user says "I want to order flowers" to a hotel booking intent, the fallback intent is triggered.

**To add an intent**

1. Sign in to the AWS Management Console and open the Amazon Lex V2 console at  [https://console.aws.amazon.com/lexv2/](https://console.aws.amazon.com/lexv2/)

2. From the list of bots, choose the bot that you want to add the intent to, then from **Add languages** choose **View languages**.

3. Choose the language to add the intent to, then choose **Intents**.

4. Choose **Add intent**, give your intent a name, and then choose **Add**.

5. In the intent editor, add the details of your intent.

- **Conversation flow** – Use the conversation flow diagram to see how a dialog with your bot might look. You can choose different sections of the conversation to jump to that section of the intent editor.
• **Intent details** – Give the intent a name and description to help identify the purpose of the intent. You can also see the unique identifier that Amazon Lex V2 assigned to the intent.

• **Contexts** – Set the input and output contexts for the intent. A context is a state variable associated with an intent. An output context is set when an intent is fulfilled. An intent with an input context can only be recognized with the context is active. An intent with no input contexts can always be recognized.

• **Sample utterances** – You should provide 10 or more phrases that you expect your users to use to trigger an intent. Amazon Lex V2 generalizes from these phrases to recognize that the user wants to trigger the intent.

• **Slots** – Define the slots, or parameters, required to fulfill the intent. Each slot has a type that defines the values that can be entered in the slot. You can choose from your custom slot types, or you can choose a built-in slot type.

• **Confirmation prompts and declination responses** – These responses are used to end the conversation with a user and to confirm or decline fulfillment of the intent. The confirmation prompt asks the user to review slot values. For example, "I've booked a hotel room for Friday. Is this correct?" The declination response is sent to the user when they decline the confirmation.

• **Fulfillment updates** – Response sent to the user at the start of fulfillment and periodically while the fulfillment is in progress. For example, "I'm changing your password, this may take a few minutes" and "I'm still working on your request." Fulfillment updates are used only for streaming conversations. For more information, see Configuring fulfillment progress updates (p. 220).

• **Post fulfillment status** – Response sent to the user when an intent is fulfilled. You can set a success message, a failure message, and a timeout message. You can send post-fulfillment messages for both streaming and regular conversations. For example, if the fulfillment succeeds, you can send "I've changed your password." If the fulfillment doesn't succeed, you can send a response with more information, such as "I couldn't change your password, contact the help desk for assistance." If the fulfillment takes too long to process and exceeds the configured timeout period, you can send a message informing the user, such as "Our servers are very busy right now. Try your request again later." For more information, see Configuring fulfillment progress updates (p. 220).

• **Closing responses** – Response sent to the user after the intent is fulfilled and all other messages are played. For example, a thank you for booking a hotel room. Or it can prompt the user to start a different intent, such as "Thank you for booking a room, would you like to book a rental car?"

• **Codehooks** – Indicate whether you are using an AWS Lambda function to initialize the intent, validate user input, and to fulfill the intent. You specify the Lambda function in the alias that you use to run the bot.

6. Choose **Save intent** to save the intent.

### Turning off responses

While using your bot, you may find it necessary to turn off start and update responses, post-fulfillment responses, confirmation prompts, and closing responses. Responses can be turned on and off for each intent.

Use the **Active** toggle to determine whether or not to use a response.

### Adding slot types

Slot types define the values that users can supply for your intent variables. You define slot types for each language so that the values are specific to that language. For example, for a slot type that lists paint colors, you could include the value "red" in English, "rouge" in French, and "rojo" in Spanish.
This topic describes how to create custom slot types that provide values for your intent's slots. You can also use built-in slot types for standard values. For example, you can use the built-in slot type AMAZON.Country for a list of countries in the world.

**To create a slot type**

1. Sign in to the AWS Management Console and open the Amazon Lex V2 console at https://console.aws.amazon.com/lexv2/
2. From the list of bots, choose the bot that you want to add the language to, then choose Conversation structure and then All languages.
3. Choose the language to add the slot type to, then choose Slot types.
4. Choose Add slot type, give your slot type a name, and then choose Add.
5. In the slot type editor, add the details of your slot type.
   - **Slot value resolution** – Determines how slot values are resolved. If you choose Expand values, Amazon Lex V2 uses the values as representative values for training. If you use Restrict to slot values the allowed values for the slot are restricted to the ones that you provide.
   - **Slot type values** – The values for the slot. If you chose Restrict to slot values, you can add synonyms for the value. For example, for the value "football" you can add the synonym "soccer." If the user enters "soccer" in a conversation with your bot, the actual value of the slot is "football."
   - **Use slot values as custom vocabulary** – Enable this option to help improve recognition of slot values and synonyms in audio conversations. Don't enable this option when the slot values are common terms, such as "yes," "no," "one," "two," "three," etc.
6. Choose Save slot type.

**Testing a bot using the console**

The Amazon Lex V2 console contains a test window that you can use to test the interaction with your bot. You use the test window to have a test conversation with your bot and to see the responses that your application receives from the bot.

There are two types of testing that you can perform with your bot. The first, express testing, enables you to test your bot with the exact phrases that you used for creating the bot. For example, if you added the utterance "I want to pick up flowers" to your intent, you can test the bot using that exact phrase.

The second type, complete testing, enables you to test your bot using phrases related to the utterances that you configured. For example, you can use the phrase "Can I order flowers" to start a conversation with your bot.

You test a bot using a specific alias and language. If you are testing the development version of the bot, you use the TestBotAlias alias for testing.

**To open the test window**

1. Sign in to the AWS Management Console and open the Amazon Lex V2 console at https://console.aws.amazon.com/lexv2/
2. Choose the bot to test from the list of bots.
3. Choose Aliases.
4. From the list of aliases, choose the alias to test.
5. From Languages, choose the radio button of the language to test, and then choose Test.

After you choose Test, the test window opens in the console. You can use the test window to interact with your bot, as shown in the following graphic.
In addition to the conversation, you can also choose Inspect in the test window to see the responses returned from the bot. The first view shows you a summary of the information returned from your bot to the test window.
You can also use the test inspection window to see the JSON structures that are sent between the bot and the test window. You can see both the request from the test window and the response from Amazon Lex V2.
Creating versions

Amazon Lex V2 supports publishing versions of bots 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.

The Draft version

When you create an Amazon Lex V2 bot there is only one version, the Draft version.

Draft is the working copy of your bot. You can update only the Draft version and until you publish your first version, Draft is the only version of the bot that you have.
The **Draft** version of your bot is associated with the **TestBotAlias**. The **TestBotAlias** should only be used for manual testing. Amazon Lex V2 limits the number of runtime requests that you can make to the **TestBotAlias** alias of the bot.

## Creating a version

When you version an Amazon Lex V2 bot you create a numbered snapshot of the bot so that you can use the bot as it existed when the version was made. Once you've created a numeric version it will stay the same while you continue to work on the draft version of your application.

When you create a version, you can choose the locales to include in the version. You don't need to choose all of the locales in a bot. Also, when you create a version you can choose a locale from a previous version. For example, if you have three versions of a bot, you can choose one locale from the **Draft** version and one from version two when you create version four.

If you delete a locale from the **Draft** version, it is not deleted from a numbered version.

If a bot version is not used for six months, Amazon Lex V2 will mark the version inactive. When a version is inactive, you can't use runtime operations with the bot. To make the bot active, rebuild all the languages associated with the version.

## Updating an Amazon Lex V2 bot

You can update only the **Draft** version of an Amazon Lex V2 bot. 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` operation.

## Deleting an Amazon Lex V2 bot or version

Amazon Lex V2 supports deleting a bot or version using the console or one of the API operations:

- `DeleteBot` (p. 386)
- `DeleteBotVersion` (p. 395)

## Built-in intents and slot types

To make it easier to create bots, Amazon Lex V2 allows you to use standard built-in intents and slot types.

**Topics**

- Built-in intents (p. 24)
- Built-in slot types (p. 35)

## Built-in intents

For common actions, you can use the standard built-in intents library. To create an intent from a built-in intent, choose a built-intent in the console, and give it a new name. The new intent has the configuration of the base intent, such as the sample utterances.

In the current implementation, you can't do the following:

- Add or remove sample utterances from the base intent
- Configure slots for built-in intents
To add a built-in intent to a bot

1. Sign in to the AWS Management Console and open the Amazon Lex V2 console at https://console.aws.amazon.com/lexv2/
2. Choose the bot to add the built-in intent to.
3. In the left menu, choose the language and then choose **Intents**.
4. Choose **Add intent**, and then choose **Use built-in intent**.
5. In **Built-in intent**, choose the intent to use.
6. Give the intent a name, and then choose **Add**.
7. Use the intent editor to configure the intent as required for your bot.

Topics

- AMAZON.CancelIntent (p. 25)
- AMAZON.FallbackIntent (p. 25)
- AMAZON.HelpIntent (p. 26)
- AMAZON.KendraSearchIntent (p. 27)
- AMAZON.PauseIntent (p. 34)
- AMAZON.RepeatIntent (p. 34)
- AMAZON.ResumeIntent (p. 34)
- AMAZON.StartOverIntent (p. 35)
- AMAZON.StopIntent (p. 35)

**AMAZON.CancelIntent**

Responds to words and phrases that indicate the user wants to cancel the current interaction. Your application can use this intent to remove slot type values and other attributes before ending the interaction with the user.

Common utterances:

- cancel
- never mind
- forget it

**AMAZON.FallbackIntent**

When a user's input to an intent isn't what a bot expects, you can configure Amazon Lex V2 to invoke a **fallback intent**. For example, if the user input "I'd like to order candy" doesn't match an intent in your OrderFlowers bot, Amazon Lex V2 invokes the fallback intent to handle the response.

The built-in **AMAZON.FallbackIntent** intent type is added to your bot automatically when you create a bot using the console, when you use the API you can specify the intent using the **CreateBot** (p. 313) operation.

Invoking a fallback intent uses two steps. In the first step the fallback intent is matched based on the input from the user. When the fallback intent is matched, the way the bot behaves depends on the number of retries configured for a prompt.

Amazon Lex V2 matches the fallback intent in these situations:

- The user's input to an intent doesn't match the input that the bot expects
• Audio input is noise, or text input isn't recognized as words.
• The user's input is ambiguous and Amazon Lex V2 can't determine which intent to invoke.

The fallback intent is invoked when:
• The bot doesn't recognize the user input as an intent after the configured number of tries for clarification when the conversation is started.
• An intent doesn't recognize the user input as a slot value after the configured number of tries.
• An intent doesn't recognize the user input as a response to a confirmation prompt after the configured number of tries.

You can't add the following to a fallback intent:
• Utterances
• Slots
• A confirmation prompt

Using a Lambda Function with a Fallback Intent

When a fallback intent is invoked, the response depends on the setting of the fulfillmentCodeHook parameter to the CreateIntent (p. 337) operation. The bot does one of the following:
• Returns the intent information to the client application.
• Calls the alias's validation and fulfillment Lambda function. It calls the function with the session variables that are set for the session.

For more information about setting the response when a fallback intent is invoked, see the fulfillmentCodeHook parameter of the CreateIntent (p. 337) operation.

If you use the Lambda function with your fallback intent, you can use this function to call another intent or to perform some form of communication with the user, such as collecting a callback number or opening a session with a customer service representative.

A fallback intent can be invoked multiple times in the same session. For example, suppose that your Lambda function uses the ElicitIntent dialog action to prompt the user for a different intent. If Amazon Lex V2 can't infer the user's intent after the configured number of tries, it invokes the fallback intent again. It also invokes the fallback intent when the user doesn't respond with a valid slot value after the configured number of tries.

You can configure your Lambda function to keep track of the number of times that the fallback intent is called using a session variable. Your Lambda function can take a different action if it is called more times than the threshold that you set in your Lambda function. For more information about session variables, see Setting session attributes (p. 178).

AMAZON.HelpIntent

Responds to words or phrases that indicate the user needs help while interacting with your bot. When this intent is invoked, you can configure your Lambda function or application to provide information about the your bot's capabilities, ask follow up questions about areas of help, or hand the interaction over to a human agent.

Common utterances:
• help
• help me
• can you help me

**AMAZON.KendraSearchIntent**

To search documents that you have indexed with Amazon Kendra, use the **AMAZON.KendraSearchIntent** intent. When Amazon Lex V2 can't determine the next action in a conversation with the user, it triggers the search intent.

The **AMAZON.KendraSearchIntent** is available only in the English (US) (en-US) locale and in the US East (N. Virginia), US West (Oregon) and Europe (Ireland) Regions.

Amazon Kendra is a machine-learning-based search service that indexes natural language documents such as PDF documents or Microsoft Word files. It can search indexed documents and return the following types of responses to a question:

• An answer
• An entry from a FAQ that might answer the question
• A document that is related to the question

For an example of using the **AMAZON.KendraSearchIntent**, see Example: Creating a FAQ Bot for an Amazon Kendra Index (p. 32).

If you configure an **AMAZON.KendraSearchIntent** intent for your bot, Amazon Lex V2 calls the intent whenever it can't determine the user utterance for a slot or intent. For example, if your bot is eliciting a response for a slot type called "pizza topping" and the user says "What is a pizza?," Amazon Lex V2 calls the **AMAZON.KendraSearchIntent** to handle the question. If there is no response from Amazon Kendra, the conversation continues as configured in the bot.

When you use the **AMAZON.KendraSearchIntent** with the **AMAZON.FallbackIntent** in the same bot, Amazon Lex V2 uses the intents as follows:

1. Amazon Lex V2 calls the **AMAZON.KendraSearchIntent**. The intent calls the Amazon Kendra Query operation.
2. If Amazon Kendra returns a response, Amazon Lex V2 displays the result to the user.
3. If there is no response from Amazon Kendra, Amazon Lex V2 re-prompts the user. The next action depends on response from the user.
   • If the response from the user contains an utterance that Amazon Lex V2 recognizes, such as filling a slot value or confirming an intent, the conversation with the user proceeds as configured for the bot.
   • If the response from the user does not contain an utterance that Amazon Lex V2 recognizes, Amazon Lex V2 makes another call to the Query operation.
4. If there is no response after the configured number of retries, Amazon Lex V2 calls the **AMAZON.FallbackIntent** and ends the conversation with the user.

There are three ways to use the **AMAZON.KendraSearchIntent** to make a request to Amazon Kendra:

• Let the search intent make the request for you. Amazon Lex V2 calls Amazon Kendra with the user's utterance as the search string. When you create the intent, you can define a query filter string that limits the number of responses that Amazon Kendra returns. Amazon Lex V2 uses the filter in the query request.
• Add additional query parameters to the request to narrow the search results using your Lambda function. You add a `kendraQueryFilterString` field that contains Amazon Kendra query parameters to the `delegate` dialog action. When you add query parameters to the request with the
Lambda function, they take precedence over the query filter that you defined when you created the intent.

- Create a new query using the Lambda function. You can create a complete Amazon Kendra query request that Amazon Lex V2 sends. You specify the query in the kendraQueryRequestPayload field in the delegate dialog action. The kendraQueryRequestPayload field takes precedence over the kendraQueryFilterString field.

To specify the queryFilterString parameter when you create a bot, or to specify the kendraQueryFilterString field when you call the delegate action in a dialog Lambda function, you specify a string that is used as the attribute filter for the Amazon Kendra query. If the string isn’t a valid attribute filter, you’ll get an InvalidBotConfigException exception at runtime. For more information about attribute filters, see Using document attributes to filter queries in the Amazon Kendra Developer Guide.

To have control over the query that Amazon Lex V2 sends to Amazon Kendra, you can specify a query in the kendraQueryRequestPayload field in your Lambda function. If the query isn’t valid, Amazon Lex V2 returns an InvalidLambdaResponseException exception. For more information, see the Query operation in the Amazon Kendra Developer Guide.

For an example of how to use the AMAZON.KendraSearchIntent, see Example: Creating a FAQ Bot for an Amazon Kendra Index (p. 32).

**IAM Policy for Amazon Kendra Search**

To use the AMAZON.KendraSearchIntent intent, you must use a role that provides AWS Identity and Access Management (IAM) policies that enable Amazon Lex V2 to assume a runtime role that has permission to call the Amazon Kendra query intent. The IAM settings that you use depend on whether you create the AMAZON.KendraSearchIntent using the Amazon Lex V2 console, or using an AWS SDK or the AWS Command Line Interface (AWS CLI). When you use the console, you can choose between adding permission to call Amazon Kendra to the Amazon Lex V2 service-linked role or using a role specifically for calling the Amazon Kendra query operation. When you use the AWS CLI or an SDK to create the intent, you must use a role specifically for calling the Query operation.

**Attaching Permissions**

You can use the console to attach permissions to access the Amazon Kendra Query operation to the default Amazon Lex V2 service-linked role. When you attach permissions to the service-linked role, you don't have to create and manage a runtime role specifically to connect to the Amazon Kendra index.

The user, role, or group that you use to access the Amazon Lex V2 console must have permissions to manage role policies. Attach the following IAM policy to the console access role. When you grant these permissions, the role has permissions to change the existing service-linked role policy.

```json
{
  "Version": "2012-10-17",
  "Statement": [
    {
      "Effect": "Allow",
      "Action": [
        "iam:AttachRolePolicy",
        "iam:PutRolePolicy",
        "iam:GetRolePolicy"
      ],
      "Resource": "arn:aws:iam::*:role/aws-service-role/lexv2.amazonaws.com/AWSServiceRoleForLexBots**"
    },
    {
      "Effect": "Allow",
      "Action": "iam:ListRoles",
      "Resource": "*"
    }
  ]
}
```
Specifying a Role

You can use the console, the AWS CLI, or the API to specify a runtime role to use when calling the Amazon Kendra Query operation.

The IAM user, role, or group that you use to specify the runtime role must have the `iam:PassRole` permission. The following policy defines the permission. You can use the `iam:AssociatedResourceArn` and `iam:PassedToService` condition context keys to further limit the scope of the permissions. For more information, see IAM and AWS STS Condition Context Keys in the AWS Identity and Access Management User Guide.

```json
{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Effect": "Allow",
            "Action": "iam:PassRole",
            "Resource": "arn:aws:iam::account:role/role"
        }
    ]
}
```

The runtime role that Amazon Lex V2 needs to use to call Amazon Kendra must have the `kendra:Query` permissions. When you use an existing IAM role for permission to call the Amazon Kendra Query operation, the role must have the following policy attached.

You can use the IAM console, the IAM API, or the AWS CLI to create a policy and attach it to a role. These instructions use the AWS CLI to create the role and policies.

**Note**
The following code is formatted for Linux and MacOS. For Windows, replace the Linux line continuation character (`\`) with a caret (`^`).

**To add Query operation permission to a role**

1. Create a document called `KendraQueryPolicy.json` in the current directory, add the following code to it, and save it

```json
{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Effect": "Allow",
            "Action": ["kendra:Query"],
            "Resource": ["arn:aws:kendra:region:account:index/index ID"
        ]
    ]
}
```

2. In the AWS CLI, run the following command to create the IAM policy for running the Amazon Kendra Query operation.

```
aws iam create-policy \
```
3. Attach the policy to the IAM role that you are using to call the Query operation.

```
aws iam attach-role-policy
  --policy-name arn:aws:iam::account-id:policy/query-policy-name
  --role-name role-name
```

You can choose to update the Amazon Lex V2 service-linked role or to use a role that you created when you create the AMAZON.KendraSearchIntent for your bot. The following procedure shows how to choose the IAM role to use.

**To specify the runtime role for AMAZON.KendraSearchIntent**

1. Sign in to the AWS Management Console and open the Amazon Lex V2 console at https://console.aws.amazon.com/lexv2/home
2. Choose the bot that you want to add the AMAZON.KendraSearchIntent to.
3. Choose the plus (+) next to Intents.
4. In Add intent, choose Search existing intents.
5. In Search intents, enter AMAZON.KendraSearchIntent and then choose Add.
6. In Copy built-in intent, enter a name for the intent, such as KendraSearchIntent, and then choose Add.
7. Open the Amazon Kendra query section.
8. For IAM role choose one of the following options:
   - To update the Amazon Lex V2 service-linked role to enable your bot to query Amazon Kendra indexes, choose Add Amazon Kendra permissions.
   - To use a role that has permission to call the Amazon Kendra Query operation, choose Use an existing role.

**Using Request and Session Attributes as Filters**

To filter the response from Amazon Kendra to items related to current conversation, use session and request attributes as filters by adding the queryFilterString parameter when you create your bot. You specify a placeholder for the attribute when you create the intent, and then Amazon Lex V2 substitutes a value before it calls Amazon Kendra. For more information about request attributes, see Setting request attributes (p. 180). For more information about session attributes, see Setting session attributes (p. 178).

The following is an example of a queryFilterString parameter that uses a string to filter the Amazon Kendra query.

```
{"equalsTo": {"key": "City", "value": {"stringValue": "Seattle"}}}
```

The following is an example of a queryFilterString parameter that uses a session attribute called "SourceURI" to filter the Amazon Kendra query.

```
{"equalsTo": {"key": "SourceURI","value": {"stringValue": "[FileURL]"}}}
```

The following is an example of a queryFilterString parameter that uses a request attribute called "DepartmentName" to filter the Amazon Kendra query.
The `AMAZON.KendraSearchIntent` filters use the same format as the Amazon Kendra search filters. For more information, see Using document attributes to filter search results in the *Amazon Kendra developer guide*.

The query filter string used with the `AMAZON.KendraSearchIntent` must use lower-case letters for the first letter of each filter. For example, the following is a valid query filter for the `AMAZON.KendraSearchIntent`.

```
{
    "andAllFilters": [
        {
            "equalsTo": {
                "key": "City",
                "value": {
                    "stringValue": "Seattle"
                }
            }
        },
        {
            "equalsTo": {
                "key": "State",
                "value": {
                    "stringValue": "Washington"
                }
            }
        }
    ]
}
```

Using the Search Response

Amazon Kendra returns the response to a search in a response from the intent's `IntentClosingSetting` statement. The intent must have a `closingResponse` statement unless a Lambda function produces a closing response message.

Amazon Kendra has four types of responses.

- `x-amz-lex:kendra-search-response-question_answer-question-<N>` – The question from a FAQ that matches the search.
- `x-amz-lex:kendra-search-response-question_answer-answer-<N>` – The answer from a FAQ that matches the search.
- `x-amz-lex:kendra-search-response-answer-<N>` – An excerpt from a document in the index that answers the question.

The responses are returned in request attributes. There can be up to five responses for each attribute, numbered 1 through 5. For more information about responses, see Types of response in the *Amazon Kendra Developer Guide*.

The `closingResponse` statement must have one or more message groups. Each message group contains one or more messages. Each message can contain one or more placeholder variables that are replaced by request attributes in the response from Amazon Kendra. There must be at least one message in the message group where all of the variables in the message are replaced by request attribute values.
in the runtime response, or there must be a message in the group with no placeholder variables. The request attributes are set off with double parentheses ("(("))"). The following message group messages match any response from Amazon Kendra:

- “I found a FAQ question for you: ((x-amz-lex:kendra-search-response-question_answer-question-1)), and the answer is ((x-amz-lex:kendra-search-response-question_answer-answer-1))”
- “I found an excerpt from a helpful document: ((x-amz-lex:kendra-search-response-document-1))”
- “I think the answer to your questions is ((x-amz-lex:kendra-search-response-answer-1))”

Using a Lambda Function to Manage the Request and Response

The AMAZON.KendraSearchIntent intent can use your dialog code hook and fulfillment code hook to manage the request to Amazon Kendra and the response. Use the dialog code hook Lambda function when you want to modify the query that you send to Amazon Kendra, and the fulfillment code hook Lambda function when you want to modify the response.

Creating a Query with the Dialog Code Hook

You can use the dialog code hook to create a query to send to Amazon Kendra. Using the dialog code hook is optional. If you don’t specify a dialog code hook, Amazon Lex V2 constructs a query from the user utterance and uses the queryFilterString that you provided when you configured the intent, if you provided one.

You can use two fields in the dialog code hook response to modify the request to Amazon Kendra:

- **kendraQueryFilterString** – Use this string to specify attribute filters for the Amazon Kendra request. You can filter the query using any of the index fields defined in your index. For the structure of the filter string, see Using document attributes to filter queries in the Amazon Kendra Developer Guide. If the specified filter string isn’t valid, you will get an InvalidLambdaResponseException exception. The kendraQueryFilterString string overrides any query string specified in the queryFilterString configured for the intent.
- **kendraQueryRequestPayload** – Use this string to specify an Amazon Kendra query. Your query can use any of the features of Amazon Kendra. If you don’t specify a valid query, you get a InvalidLambdaResponseException exception. For more information, see Query in the Amazon Kendra Developer Guide.

After you have created the filter or query string, you send the response to Amazon Lex V2 with the dialogAction field of the response set to delegate. Amazon Lex V2 sends the query to Amazon Kendra and then returns the query response to the fulfillment code hook.

Using the Fulfillment Code Hook for the Response

After Amazon Lex V2 sends a query to Amazon Kendra, the query response is returned to the AMAZON.KendraSearchIntent fulfillment Lambda function. The input event to the code hook contains the complete response from Amazon Kendra. The query data is in the same structure as the one returned by the Amazon Kendra Query operation. For more information, see Query response syntax in the Amazon Kendra Developer Guide.

The fulfillment code hook is optional. If one does not exist, or if the code hook doesn’t return a message in the response, Amazon Lex V2 uses the closingResponse statement for responses.

Example: Creating a FAQ Bot for an Amazon Kendra Index

This example creates an Amazon Lex V2 bot that uses an Amazon Kendra index to provide answers to users’ questions. The FAQ bot manages the dialog for the user. It uses the AMAZON.KendraSearchIntent intent to query the index and to present the response to the user. To create the bot, you:
1. Create a bot that your customers will interact with to get answers from your bot.
2. Create a custom intent. Your bot requires at least one intent with at least one utterance. This intent enables your bot to build, but is not used otherwise.
3. Add the KendraSearchIntent intent to your bot and configure it to work with your Amazon Kendra index.
4. Test the bot by asking questions that are answered by documents stored in your Amazon Kendra index.

Before you can use this example, you need to create an Amazon Kendra index. For more information, see Getting started with an S3 bucket (console) in the Amazon Kendra Developer Guide.

To create a FAQ bot

1. Sign in to the AWS Management Console and open the Amazon Lex console at https://console.aws.amazon.com/lex/.
2. In the navigation pane, choose Bots.
3. Choose Create.
4. Choose Custom bot. Configure the bot as follows:
   - Bot name – Give the bot a name that indicates its purpose, such as KendraTestBot.
   - Output voice – Choose None.
   - Session timeout – Enter 5.
   - Sentiment analysis – Choose No.
   - COPPA – Choose No.
   - User utterance storage – Choose Do not store.
5. Choose Create.

To successfully build a bot, you must create at least one intent with at least one sample utterance. This intent is required to build your Amazon Lex V2 bot, but isn’t used for the FAQ response. The utterance for the intent must not apply to any of the questions that your customer asks.

To create the required intent

1. On the Getting started with your bot page, choose Create intent.
2. For Add intent, choose Create intent.
3. In the Create intent dialog box, give the intent a name, such as RequiredIntent.
4. For Sample utterances, type an utterance, such as Required utterance.
5. Choose Save intent.

Now, create the intent to search an Amazon Kendra index and the response messages that it should return.

To create an AMAZON.KendraSearchIntent intent and response message

1. In the navigation pane, choose the plus (+) next to Intents.
2. For Add intent, choose Search existing intents.
3. In the Search intents box, enter AMAZON.KendraSearchIntent, then choose it from the list.
4. For Copy built-in intent, give the intent a name, such as KendraSearchIntent, and then choose Add.
5. In the intent editor, choose Amazon Kendra query to open the query options.
6. From the **Amazon Kendra index** menu, choose the index that you want the intent to search.

7. In the **Response** section, add the following three messages:

```
I found a FAQ question for you: ((x-amz-lex:kendra-search-response-question_answer-question-1)) and the answer is ((x-amz-lex:kendra-search-response-question_answer-answer-1)).
I found an excerpt from a helpful document: ((x-amz-lex:kendra-search-response-document-1)).
I think the answer to your questions is ((x-amz-lex:kendra-search-response-answer-1)).
```

8. Choose **Save intent**, and then choose **Build** to build the bot.

Finally, use the console test window to test responses from your bot. Your questions should be in the domain that your index supports.

**To test your FAQ bot**

1. In the console test window, type a question for your index.
2. Verify the answer in the test window's response section.
3. To reset the test window for another question, choose **Clear chat history**.

**AMAZON.PauseIntent**

Responds to words and phrases that enable the user to pause an interaction with a bot so that they can return to it later. Your Lambda function or application needs to save intent data in session variables, or you need to use the GetSession (p. 618) operation to retrieve intent data when you resume the current intent.

Common utterances:

- pause
- pause that

**AMAZON.RepeatIntent**

Responds to words and phrases that enable the user to repeat the previous message. Your application needs to use a Lambda function to save the previous intent information in session variables, or you need to use the GetSession (p. 618) operation to get the previous intent information.

Common utterances:

- repeat
- say that again
- repeat that

**AMAZON.ResumeIntent**

Responds to words and phrases the enable the user to resume a previously paused intent. You Lambda function or application must manage the information required to resume the previous intent.

Common utterances:
• resume
• continue
• keep going

**AMAZON.StartOverIntent**

Responds to words and phrases that enable the user to stop processing the current intent and start over from the beginning. You can use your Lambda function or the `PutSession` operation to elicit the first slot value again.

Common utterances:

• start over
• restart
• start again

**AMAZON.StopIntent**

Responds to words and phrases that indicate that the user wants to stop processing the current intent and end the interaction with a bot. Your Lambda function or application should clear any existing attributes and slot type values and then end the interaction.

Common utterances:

• stop
• off
• shut up

**Built-in slot types**

Amazon Lex supports built-in slot types that define how data in the slot is recognized and handled. You can create slots of these types in your intents. This eliminates the need to create enumeration values for commonly used slot data such as date, time, and location. Built-in slot types do not have versions.

<table>
<thead>
<tr>
<th>Slot Type</th>
<th>Short Description</th>
<th>Supported Locales</th>
</tr>
</thead>
<tbody>
<tr>
<td>AMAZON.AlphaNumeric</td>
<td>Recognizes words made up of letters and numbers.</td>
<td>All locales except Korean (ko-KR)</td>
</tr>
<tr>
<td>AMAZON.City</td>
<td>Recognizes words that represent a city.</td>
<td>All locales</td>
</tr>
<tr>
<td>AMAZON.Country</td>
<td>Recognizes words that represent a country.</td>
<td>All locales</td>
</tr>
<tr>
<td>AMAZON.Date</td>
<td>Recognizes words that represent a date and converts them to a standard format.</td>
<td>All locales</td>
</tr>
<tr>
<td>AMAZON.Duration</td>
<td>Recognizes words that represent duration and...</td>
<td>All locales</td>
</tr>
<tr>
<td>Slot Type</td>
<td>Short Description</td>
<td>Supported Locales</td>
</tr>
<tr>
<td>--------------------</td>
<td>------------------------------------------------------------------------------------</td>
<td>-----------------------</td>
</tr>
<tr>
<td>AMAZON.EmailAddress</td>
<td>Recognizes words that represent an email address and converts them into a standard email address.</td>
<td>All locales</td>
</tr>
<tr>
<td>AMAZON.FirstName</td>
<td>Recognizes words that represent a first name.</td>
<td>All locales</td>
</tr>
<tr>
<td>AMAZON.LastName</td>
<td>Recognizes words that represent a last name.</td>
<td>All locales</td>
</tr>
<tr>
<td>AMAZON.Number</td>
<td>Recognizes numeric words and converts them into digits.</td>
<td>All locales</td>
</tr>
<tr>
<td>AMAZON.Percentage</td>
<td>Recognizes words that represent a percentage and converts them to a number and a percent sign (%).</td>
<td>All locales</td>
</tr>
<tr>
<td>AMAZON.PhoneNumber</td>
<td>Recognizes words that represent a phone number and converts them into a numeric string.</td>
<td>All locales</td>
</tr>
<tr>
<td>AMAZON.State</td>
<td>Recognizes words that represent a state.</td>
<td>All locales</td>
</tr>
<tr>
<td>AMAZON.StreetName</td>
<td>Recognizes words that represent a street name.</td>
<td>All locales</td>
</tr>
<tr>
<td>AMAZON.Time</td>
<td>Recognizes words that indicate times and converts them into a time format.</td>
<td>All locales</td>
</tr>
<tr>
<td>AMAZON.UKPostalCode</td>
<td>Recognizes words that represent a UK post code and converts them to a standard form.</td>
<td>English (British) (en-GB) only</td>
</tr>
</tbody>
</table>

**AMAZON.AlphaNumeric**

Recognizes strings made up of letters and numbers, such as `APQ123`.

This slot type is not available in the Korean (ko-KR) locale.

You can use the `AMAZON.AlphaNumeric` slot type for strings that contain:

- Alphabatical characters, such as `ABC`
• Numeric characters, such as 123
• A combination of alphanumeric characters, such as ABC123

The AMAZON.AlphaNumeric slot type supports styled input. You can use the spell-by-letter and spell-by-word styles to help your customers enter letters. For more information, see Using spelling styles to capture slot values (p. 196).

You can add a regular expression to the AMAZON.AlphaNumeric slot type to validate values entered for the slot. For example, you can use a regular expression to validate:

• Canadian postal codes
• Driver’s license numbers
• Vehicle identification numbers

Use a standard regular expression. Amazon Lex V2 supports the following characters in the regular expression:

• A-Z, a-z
• 0-9

Amazon Lex V2 also supports Unicode characters in regular expressions. The form is \uUnicode. Use four digits to represent Unicode characters. For example, \[\u0041-\u005A\] is equivalent to [A-Z].

The following regular expression operators are not supported:

• Infinite repeaters: *, +, or {x,} with no upper bound.
• Wild card (.)

The maximum length of the regular expression is 300 characters. The maximum length of a string stored in an AMAZON.AlphaNumeric slot type that uses a regular expression is 30 characters.

The following are some example regular expressions.

• Alphanumeric strings, such as APQ123 or APQ1: [A-Z]{3}[0-9]{1,3} or a more constrained [A-DF-T]{3} [1-5]{1,3}
• US Postal Service Priority Mail International format, such as CP123456789US: CP[0-9]{9}US
• Bank routing numbers, such as 123456789: [0-9]{9}

To set the regular expression for a slot type, use the console or the CreateSlotType (p. 378) operation. The regular expression is validated when you save the slot type. If the expression isn’t valid, Amazon Lex V2 returns an error message.

When you use a regular expression in a slot type, Amazon Lex V2 checks input to slots of that type against the regular expression. If the input matches the expression, the value is accepted for the slot. If the input does not match, Amazon Lex V2 prompts the user to repeat the input.

**AMAZON.City**

Provides a list of local and world cities. The slot type recognizes common variations of city names. Amazon Lex V2 doesn’t convert from a variation to an official name.

Examples:
• New York
• Reykjavik
• Tokyo
• Versailles

**AMAZON.Country**

The names of countries around the world. Examples:

• Australia
• Germany
• Japan
• United States
• Uruguay

**AMAZON.Date**

Converts words that represent dates into a date format.

The date is provided to your intent in ISO-8601 date format. The date that your intent receives in the slot can vary depending on the specific phrase uttered by the user.

• Utterances that map to a specific date, such as "today," "now," or "November twenty-fifth," convert to a complete date: 2020-11-25. This defaults to dates on or after the current date.

• Utterances that map to a future week, such as "next week," convert to the date of the last day of the current week. In ISO-8601 format, the week starts on Monday and ends on Sunday. For example, if today is 2020-11-25, "next week" converts to 2020-11-29. Dates that map to the current or previous week convert to the first day of the week. For example, if today is 2020-11-25, "last week" converts to 2020-11-16.

• Utterances that map to a future month, but not a specific day, such as "next month," convert to the last day of the month. For example, if today is 2020-11-25, "next month" converts to 2020-12-31. For dates that map to the current or previous month convert to the first day of the month. For example, if today is 2020-11-25, "this month" maps to 2020-11-01.

• Utterances that map to a future year, but not a specific month or day, such as "next year," convert to the last day of the following year. For example, if today is 2020-11-25, "next year" converts to 2021-12-31. For dates that map to the current or previous year convert to the first day of the year. For example, if today is 2020-11-25, "last year" converts to 2019-01-01.

**AMAZON.Duration**

Converts words that indicate durations into a numeric duration.

The duration is resolved to a format based on the ISO-8601 duration format, PnYnMnWnDTnHnMnS. The P indicates that this is a duration, the n is a numeric value, and the capital letter following the n is the specific date or time element. For example, P3D means 3 days. A T is used to indicate that the remaining values represent time elements rather than date elements.

Examples:

• "ten minutes": PT10M
• "five hours": PT5H
• "three days": P3D
• "forty five seconds": PT45S
• "eight weeks": P8W
• "seven years": P7Y
• "five hours ten minutes": PT5H10M
• "two years three hours ten minutes": P2YT3H10M

AMAZON.EmailAddress

Recognizes words that represent an email address provided as username@domain. Addresses can include the following special characters in a user name: underscore (_), hyphen (-), period (.), and the plus sign (+).

The AMAZON.EmailAddress slot type supports styled input. You can use the spell-by-letter and spell-by-word styles to help your customers enter email addresses. For more information, see Using spelling styles to capture slot values (p. 196).

AMAZON.FirstName

Commonly used first names. This slot type recognizes both formal names and informal nicknames. The name sent to your intent is the value sent by the user. Amazon Lex V2 doesn’t convert from the nickname to the formal name.

For first names that sound alike but are spelled differently, Amazon Lex V2 sends your intent a single common form.

The AMAZON.FirstName slot type supports styled input. You can use the spell-by-letter and spell-by-word styles to help your customers enter names. For more information, see Using spelling styles to capture slot values (p. 196).

Examples:
• Emily
• John
• Sophie

AMAZON.LastName

Commonly used last names. For names that sound alike that are spelled differently, Amazon Lex V2 sends your intent a single common form.

The AMAZON.LastName slot type supports styled input. You can use the spell-by-letter and spell-by-word styles to help your customers enter names. For more information, see Using spelling styles to capture slot values (p. 196).

Examples:
• Brosky
• Dasher
• Evers
• Parres
• Welt

**AMAZON.Number**

Converts words or numbers that express a number into digits, including decimal numbers. The following table shows how the **AMAZON.Number** slot type captures numeric words.

<table>
<thead>
<tr>
<th>Input</th>
<th>Response</th>
</tr>
</thead>
<tbody>
<tr>
<td>one hundred twenty three point four five</td>
<td>123.45</td>
</tr>
<tr>
<td>one hundred twenty three dot four five</td>
<td>123.45</td>
</tr>
<tr>
<td>point four two</td>
<td>0.42</td>
</tr>
<tr>
<td>point forty two</td>
<td>0.42</td>
</tr>
<tr>
<td>232.998</td>
<td>232.998</td>
</tr>
<tr>
<td>50</td>
<td>50</td>
</tr>
<tr>
<td>-15</td>
<td>-15</td>
</tr>
<tr>
<td>minus 15</td>
<td>-15</td>
</tr>
</tbody>
</table>

**AMAZON.Percentage**

Converts words and symbols that represent a percentage into a numeric value with a percent sign (%).

If the user enters a number without a percent sign or the word "percent," the slot value is set to the number. The following table shows how the **AMAZON.Percentage** slot type captures percentages.

<table>
<thead>
<tr>
<th>Input</th>
<th>Response</th>
</tr>
</thead>
<tbody>
<tr>
<td>50 percent</td>
<td>50%</td>
</tr>
<tr>
<td>0.4 percent</td>
<td>0.4%</td>
</tr>
<tr>
<td>23.5%</td>
<td>23.5%</td>
</tr>
<tr>
<td>twenty five percent</td>
<td>25%</td>
</tr>
</tbody>
</table>

**AMAZON.PhoneNumber**

Converts the numbers or words that represent a phone number into a string format without punctuation as follows.

<table>
<thead>
<tr>
<th>Type</th>
<th>Description</th>
<th>Input</th>
<th>Result</th>
</tr>
</thead>
<tbody>
<tr>
<td>International number with leading plus (+) sign</td>
<td>11-digit number with leading plus sign.</td>
<td>+61 7 4445 1061</td>
<td>+61744431061</td>
</tr>
<tr>
<td></td>
<td></td>
<td>+1 (509) 555-1212</td>
<td>+15095551212</td>
</tr>
</tbody>
</table>
### Built-in slot types

<table>
<thead>
<tr>
<th>Type</th>
<th>Description</th>
<th>Input</th>
<th>Result</th>
</tr>
</thead>
<tbody>
<tr>
<td>International number without leading plus (+) sign</td>
<td>11-digit number without leading plus sign</td>
<td>1 (509) 555-1212</td>
<td>15095551212</td>
</tr>
<tr>
<td></td>
<td></td>
<td>61 7 4445 1061</td>
<td>61744451061</td>
</tr>
<tr>
<td>National number</td>
<td>10-digit number without international code</td>
<td>(03) 5115 4444</td>
<td>0351154444</td>
</tr>
<tr>
<td></td>
<td></td>
<td>(509) 555-1212</td>
<td>5095551212</td>
</tr>
<tr>
<td>Local number</td>
<td>7-digit phone number without an international code or an area code</td>
<td>555-1212</td>
<td>5551212</td>
</tr>
</tbody>
</table>

#### AMAZON.State

The names of geographical and political regions within countries.

Examples:

- Bavaria
- Fukushima Prefecture
- Pacific Northwest
- Queensland
- Wales

#### AMAZON.StreetName

The names of streets within a typical street address. This includes just the street name, not the house number.

Examples:

- Canberra Avenue
- Front Street
- Market Road

#### AMAZON.Time

Converts words that represent times into time values. Includes resolutions for ambiguous times. When a user enters an ambiguous time, Amazon Lex V2 uses the slots attribute of a Lambda event to pass resolutions for the ambiguous times to your Lambda function. For example, if your bot prompts the user for a delivery time, the user can respond by saying "10 o'clock." This time is ambiguous. It means either 10:00 AM or 10:00 PM. In this case, the value in the interpretedValue field is null, and the resolvedValues field contains the two possible resolutions of the time. Amazon Lex V2 inputs the following into the Lambda function:

```json
"slots": {
  "deliveryTime": {
    "value": {
      "originalValue": "10 o'clock",
      "interpretedValue": null,
```
When the user responds with an unambiguous time, Amazon Lex V2 sends the time to your Lambda function in the `interpretedValue` field of the `slots` attribute of the Lambda event. For example, if your user responds to the prompt for a delivery time with "10:00 AM," Amazon Lex V2 inputs the following into the Lambda function:

```json
"slots": {
  "deliveryTime": {
    "value": {
      "originalValue": "10 AM",
      "interpretedValue": 10:00,
      "resolvedValues": ["10:00"
    ]
  }
}
```

For more information about the data sent from Amazon Lex V2 to a Lambda function, see Input event format (p. 146).

**AMAZON.UKPostalCode**

Converts words that represent a UK postal code to a standard format. The `AMAZON.UKPostalCode` slot type validates and resolves the post code to a set of standardized formats, but it doesn’t check to make sure that the post code is valid. Your application must validate the post code.

The `AMAZON.UKPostalCode` slot type is available only in the English (UK) (en-GB) locale.

The `AMAZON.AlphaNumeric` slot type supports styled input. You can use the spell-by-letter and spell-by-word styles to help your customers enter letters. For more information, see Using spelling styles to capture slot values (p. 196).

The slot type recognizes all valid post code formats, including British overseas territories. The valid formats are (A represents a letter, 9 represents a digit):

- AA9A 9AA
- A9A 9AA
- A9 9AA
- A99 9AA
- AA9 9AA
- AA99 9AA

For text input, the user can enter any mix of upper and lower case letters. The user can use or omit the space in the post code. The resolved value will always include the space in the proper location for the post code.

For spoken input, the user can speak the individual characters, or they can use double letter pronunciations, such as "double A" or "double 9." They can also use double-digit pronunciations, such as "ninety-nine" for "99."
Using a custom grammar slot type

With the grammar slot type, you can author your own grammar in the XML format per the SRGS specification to collect information in a conversation. Amazon Lex V2 recognizes utterances matched by the rules specified in the grammar. You can also provide semantic interpretation rules using ECMAScript tags within the grammar files. Amazon Lex then returns properties set in the tags as resolved values when a match occurs.

You can only create grammar slot types in the English (Australia), English (UK), and English (US) locales.

There are two parts to a grammar slot type. The first is the grammar itself written using the SRGS specification format. The grammar interprets the utterance from the user. If the utterance is accepted by the grammar it is matched, otherwise it is rejected. If an utterance is matched it is passed on to the script if there is one.

The second is part of a grammar slot type is an optional script written in ECMAScript that transforms the input to the resolved values returned by the slot type. For example, you can use a script to convert spoken numbers to digits. ECMAScript statements are enclosed in the <tag> element.

The following example is in the XML format per the SRGS specification that shows a valid grammar accepted by Amazon Lex V2. It defines a grammar slot type that accepts card numbers and determines if they are for regular or premium accounts. For more information about the acceptable syntax, see the Grammar definition (p. 44) and the Script format (p. 51) topics.

```
<grammar version="1.0" xmlns="http://www.w3.org/2001/06/grammar"
   xml:lang="en-US" *tag-format="semantics/1.0"* root="card_number">
  <rule id="card_number" scope="public">
    <item repeat="0-1">card number</item>
    <item>
      seven
      <tag>out.value = "7";</tag>
    </item>
    <item>
      <one-of>
        <item>
          two four one
          <tag>out.value = out.value + "241"; out.card_type = "premium";</tag>
        </item>
        <item>
          zero zero one
          <tag>out.value = out.value + "001"; out.card_type = "regular";</tag>
        </item>
      </one-of>
    </item>
  </rule>
</grammar>
```

The above grammar only accepts two types of card numbers: 7241 or 7001. Both of these may be optionally prefixed with “card number”. It also contains ECMAScript tags that can be used for semantic interpretation. With semantic interpretation, the utterance "card number seven two four one" would return following object:

```
{
   "value": "7241",
   "card_type": "premium"
}
```
Adding a grammar slot type

To add a grammar slot type

1. Upload the XML definition of your slot type to an S3 bucket. Make a note of the bucket name and the path to the file.
2. Sign in to the AWS Management Console and open the Amazon Lex console at https://console.aws.amazon.com/lexv2/.
3. From the left menu, choose Bots and then choose the bot to add the grammar slot type to.
4. Choose View languages, and then choose the language to add the grammar slot type to.
5. Choose View slot types.
6. Choose Add slot type, and then choose Add grammar slot type.
7. Give the slot type a name, and then choose Add.
8. Choose the S3 bucket that contains your definition file and enter the path to the file. Choose Save slot type.

Grammar definition

This topic shows the parts of the SRGS specification that Amazon Lex V2 supports. All of the rules are defined in the SRGS specification. For more information, see the Speech recognition grammar specification version 1.0 W3C recommendation.

Topics

- Header declarations (p. 45)
- Tokens (p. 47)
- Rule reference (p. 48)
- Sequences and encapsulation (p. 49)
- Repeats (p. 49)
- Language (p. 50)
- Tags (p. 51)
- Unsupported rules (p. 51)

This document includes material copied and derived from the W3C Speech Recognition Grammar Specification Version 1.0 (available at https://www.w3.org/TR/speech-grammar/). Citation information follows:

Copyright © 2004 W3C® (MIT, ERCIM, Keio), All Rights Reserved. W3C liability, trademark, document use and software licensing rules apply.

The SRGS specification document, a W3C Recommendation, is available from the W3C under the following license.

License text

License
By using and/or copying this document, or the W3C document from which this statement is linked, you (the licensee) agree that you have read, understood, and will comply with the following terms and conditions:

Permission to copy, and distribute the contents of this document, or the W3C document from which this statement is linked, in any medium for any purpose and without fee or royalty is hereby granted, provided that you include the following on ALL copies of the document, or portions thereof, that you use:

- A link or URL to the original W3C document.
- The pre-existing copyright notice of the original author, or if it doesn't exist, a notice (hypertext is preferred, but a textual representation is permitted) of the form: "Copyright © [$date-of-document] World Wide Web Consortium, (MIT, ERCIM, Keio, Beihang). http://www.w3.org/Consortium/Legal/2015/doc-license"
- If it exists, the STATUS of the W3C document.

When space permits, inclusion of the full text of this NOTICE should be provided. We request that authorship attribution be provided in any software, documents, or other items or products that you create pursuant to the implementation of the contents of this document, or any portion thereof.

No right to create modifications or derivatives of W3C documents is granted pursuant to this license, except as follows: To facilitate implementation of the technical specifications set forth in this document, anyone may prepare and distribute derivative works and portions of this document in software, in supporting materials accompanying software, and in documentation of software, PROVIDED that all such works include the notice below. HOWEVER, the publication of derivative works of this document for use as a technical specification is expressly prohibited.

In addition, "Code Components"—Web IDL in sections clearly marked as Web IDL; and W3C-defined markup (HTML, CSS, etc.) and computer programming language code clearly marked as code examples—are licensed under the W3C Software License.

The notice is:

"Copyright © 2015 W3C® (MIT, ERCIM, Keio, Beihang). This software or document includes material copied from or derived from [title and URI of the W3C document]."

Disclaimers

THIS DOCUMENT IS PROVIDED "AS IS," AND COPYRIGHT HOLDERS MAKE NO REPRESENTATIONS OR WARRANTIES, EXPRESS OR IMPLIED, INCLUDING, BUT NOT LIMITED TO, WARRANTIES OF MERCHANTABILITY, FITNESS FOR A PARTICULAR PURPOSE, NON-INFRINGEMENT, OR TITLE; THAT THE CONTENTS OF THE DOCUMENT ARE SUITABLE FOR ANY PURPOSE; NOR THAT THE IMPLEMENTATION OF SUCH CONTENTS WILL NOT INFRINGE ANY THIRD PARTY PATENTS, COPYRIGHTS, TRADEMARKS OR OTHER RIGHTS.

COPYRIGHT HOLDERS WILL NOT BE LIABLE FOR ANY DIRECT, INDIRECT, SPECIAL OR CONSEQUENTIAL DAMAGES ARISING OUT OF ANY USE OF THE DOCUMENT OR THE PERFORMANCE OR IMPLEMENTATION OF THE CONTENTS THEREOF.

The name and trademarks of copyright holders may NOT be used in advertising or publicity pertaining to this document or its contents without specific, written prior permission. Title to copyright in this document will at all times remain with copyright holders.

**Header declarations**

The following table shows the header declarations supported by the grammar slot type. For more information, see Grammar header declarations in the Speech recognition grammar specification version 1 W3C recommendation.
<table>
<thead>
<tr>
<th>Declaration</th>
<th>Specification requirement</th>
<th>XML form</th>
<th>Amazon Lex support</th>
<th>Specification</th>
</tr>
</thead>
<tbody>
<tr>
<td>Grammar version</td>
<td>Required</td>
<td>4.3: <code>version</code> attribute on <code>grammar</code> element</td>
<td>Required</td>
<td>SRGS</td>
</tr>
<tr>
<td>XML namespace</td>
<td>Required (XML only)</td>
<td>4.3: <code>xmlns</code> attribute on <code>grammar</code> element</td>
<td>Required</td>
<td>SRGS</td>
</tr>
<tr>
<td>Document type</td>
<td>Required (XML only)</td>
<td>4.3: <code>XML DOCTYPE</code></td>
<td>Recommended</td>
<td>SRGS</td>
</tr>
<tr>
<td>Character encoding</td>
<td>Recommended</td>
<td>4.4: <code>encoding</code> attribute in XML declaration</td>
<td>Recommended</td>
<td>SRGS</td>
</tr>
<tr>
<td>Language</td>
<td>Required in voice mode</td>
<td>4.5: <code>xml:lang</code> attribute on <code>grammar</code> element</td>
<td>Required in voice mode</td>
<td>SRGS</td>
</tr>
<tr>
<td></td>
<td>Ignored in DTMF mode</td>
<td></td>
<td>Ignored in DTMF mode</td>
<td></td>
</tr>
<tr>
<td>Mode</td>
<td>Optional</td>
<td>4.6: <code>mode</code> attribute on <code>grammar</code> element</td>
<td>Optional</td>
<td>SRGS</td>
</tr>
<tr>
<td>Root rule</td>
<td>Optional</td>
<td>4.7: <code>root</code> attribute on <code>grammar</code> element</td>
<td>Required</td>
<td>SRGS</td>
</tr>
<tr>
<td>Tag format</td>
<td>Optional</td>
<td>4.8: <code>tag-format</code> attribute on <code>grammar</code> element</td>
<td>String literal and ECMAScript are supported</td>
<td>SRGS, SISR</td>
</tr>
<tr>
<td>Base URI</td>
<td>Optional</td>
<td>4.9: <code>xml:base</code> attribute on <code>grammar</code> element</td>
<td>Optional</td>
<td>SRGS</td>
</tr>
<tr>
<td>Pronunciation lexicon</td>
<td>Optional, multiple allowed</td>
<td>4.10: <code>lexicon</code> element</td>
<td>Not supported</td>
<td>SRGS, PLS</td>
</tr>
<tr>
<td>Metadata</td>
<td>Optional, multiple allowed</td>
<td>4.11.1: <code>meta</code> element</td>
<td>Required</td>
<td>SRGS</td>
</tr>
<tr>
<td>XML metadata</td>
<td>Optional, XML only</td>
<td>4.11.2: <code>metadata</code> element</td>
<td>Optional</td>
<td>SRGS</td>
</tr>
<tr>
<td>Tag</td>
<td>Optional, multiple allowed</td>
<td>4.12: <code>tag</code> element</td>
<td>Global tags not supported</td>
<td>SRGS</td>
</tr>
</tbody>
</table>

**Example**

```xml
<?xml version="1.0" encoding="ISO-8859-1"?>
<!DOCTYPE grammar PUBLIC "-//W3C//DTD GRAMMAR 1.0//EN"
  "http://www.w3.org/TR/speech-grammar/grammar.dtd">
```
Tokens

The following table shows the token specifications supported by the grammar slot type. For more information, see Tokens in the Speech recognition grammar specification version 1 W3C recommendation.

<table>
<thead>
<tr>
<th>Token type</th>
<th>Example</th>
<th>Supported?</th>
</tr>
</thead>
<tbody>
<tr>
<td>Single unquoted token</td>
<td>hello</td>
<td>Yes</td>
</tr>
<tr>
<td>Single unquoted token: non-alphabetic</td>
<td>2</td>
<td>Yes</td>
</tr>
<tr>
<td>Single quoted token, no white space</td>
<td>&quot;hello&quot;</td>
<td>Yes, drop double quotes when it only contains a single token</td>
</tr>
<tr>
<td>Two tokens delimited by white space</td>
<td>bon voyage</td>
<td>Yes</td>
</tr>
<tr>
<td>Four tokens delimited by white space</td>
<td>this is a test</td>
<td>Yes</td>
</tr>
<tr>
<td>Single quoted token, including white space</td>
<td>&quot;San Francisco&quot;</td>
<td>No</td>
</tr>
<tr>
<td>Single XML token in &lt;token&gt; tag</td>
<td>&lt;token&gt;San Francisco&lt;/token&gt;</td>
<td>No (same as single quoted token with white space)</td>
</tr>
</tbody>
</table>

Notes

- Single quoted token including white space – The specification requires words enclosed in double quotes be treated as a single token. Amazon Lex V2 treats them as white space delimited tokens.
- Single XML token in <token> – The specification requires words delimited by <token> to represent one token. Amazon Lex V2 treats them as white space delimited tokens.
- Amazon Lex V2 throws a validation error when either usage is found in your grammar.

Example

```xml
<grammar xmlns="http://www.w3.org/2001/06/grammar"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xm:base="http://www.example.com/base-file-path"
xsi:schemaLocation="http://www.w3.org/2001/06/grammar
http://www.w3.org/TR/speech-grammar/grammar.xsd"
xm:lang="en-US"
version="1.0"
mode="voice"
root="city"
tag-format="semantics/1.0">
    <rule id="state" scope="public">
        <one-of>
            <item>FL</item>
            <item>MA</item>
            <item>NY</item>
        </one-of>
    </rule>
</grammar>
```
## Rule reference

The following table summarizes the various forms of rule reference that are possible within grammar documents. For more information, see Rule reference in the Speech recognition grammar specification version 1 W3C recommendation.

<table>
<thead>
<tr>
<th>Reference type</th>
<th>XML form</th>
<th>Supported</th>
</tr>
</thead>
<tbody>
<tr>
<td>2.2.1 Explicit local rule reference</td>
<td><code>&lt;ruleref uri=&quot;#rulename&quot;/&gt;</code></td>
<td>Yes</td>
</tr>
<tr>
<td>2.2.2 Explicit reference to a named rule of a grammar identified by a URI</td>
<td><code>&lt;ruleref uri=&quot;grammarURI#rulename&quot;/&gt;</code></td>
<td>No</td>
</tr>
<tr>
<td>2.2.2 Implicit reference to the root rule of a grammar identified by a URI</td>
<td><code>&lt;ruleref uri=&quot;grammarURI&quot;/&gt;</code></td>
<td>No</td>
</tr>
<tr>
<td>2.2.2 Explicit reference to a named rule of a grammar identified by a URI with a media type</td>
<td><code>&lt;ruleref uri=&quot;grammarURI#rulename&quot; type=&quot;media-type&quot;/&gt;</code></td>
<td>No</td>
</tr>
<tr>
<td>2.2.2 Implicit reference to the root rule of a grammar identified by a URI with a media type</td>
<td><code>&lt;ruleref uri=&quot;grammarURI&quot; type=&quot;media-type&quot;/&gt;</code></td>
<td>No</td>
</tr>
</tbody>
</table>
| 2.2.3 Special rule definitions | `<ruleref special="NULL"/>`  
`<ruleref special="VOID"/>`  
`<ruleref special="GARBAGE"/>` | No |

### Notes

1. Grammar URI is an external URI. For example, http://grammar.example.com/world-cities.grxml.
2. Media type can be:
   - application/srgs+xml
   - text/plain

### Example

```xml
<rule id="city" scope="public">
  <one-of>
    <item>Boston</item>
    <item>Philadelphia</item>
    <item>Fargo</item>
  </one-of>
</rule>
```
<rule id="state" scope="public">
  <one-of>
    <item>FL</item>
    <item>MA</item>
    <item>NY</item>
  </one-of>
</rule>

<!-- "Boston MA" -> city = Boston, state = MA -->
<rule id="city_state" scope="public">
  <ruleref uri="#city"/> <ruleref uri="#state"/>
</rule>

### Sequences and encapsulation

The following example shows the supported sequences. For more information, see [Sequences and encapsulation](#) in the *Speech recognition grammar specification version 1* W3C recommendation.

**Example**

```xml
<!-- sequence of tokens -->
this is a test

<!-- sequence of rule references-->
<ruleref uri="#action"/> <ruleref uri="#object"/>

<!-- sequence of tokens and rule references-->
the <ruleref uri="#object"/> is <ruleref uri="#color"/>

<!-- sequence container -->
<iitem>fly to <ruleref uri="#city"/></iitem>
```

### Repeats

The following table shows the supported repeated expansions for rules. For more information, see [Repeats](#) in the *Speech recognition grammar specification version 1* W3C recommendation.

<table>
<thead>
<tr>
<th>XML form</th>
<th>Behavior</th>
<th>Supported?</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Example</strong></td>
<td></td>
<td></td>
</tr>
<tr>
<td>repeat=&quot;n&quot;</td>
<td>The contained expression is repeated exactly &quot;n&quot; times. &quot;n&quot; must be &quot;0&quot; or a positive integer.</td>
<td>Yes</td>
</tr>
<tr>
<td>repeat=&quot;6&quot;</td>
<td></td>
<td></td>
</tr>
<tr>
<td>repeat=&quot;m-n&quot;</td>
<td>The contained expansion is repeated between &quot;m&quot; and &quot;n&quot; times (inclusive). &quot;m&quot; and &quot;n&quot; must both be &quot;0&quot; or a positive integer, and &quot;m&quot; must be less than or equal to &quot;n&quot;.</td>
<td>Yes</td>
</tr>
<tr>
<td>repeat=&quot;4-6&quot;</td>
<td></td>
<td></td>
</tr>
<tr>
<td>repeat=&quot;m-&quot;</td>
<td>The contained expansion is repeated &quot;m&quot; times or more (inclusive). &quot;m&quot; must be &quot;0&quot; or a positive integer. For example, &quot;3-&quot; declares that the contained expansion can occur three, four, five, or more times.</td>
<td>Yes</td>
</tr>
<tr>
<td>repeat=&quot;3-&quot;</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
XML form | Behavior | Supported?
---|---|---
repeat="0-1" | The contained expansion is optional. | Yes

Language

The following discussion applies to language identifiers applied to grammars. For more information, see Language in the Speech recognition grammar specification version 1 W3C recommendation.

By default a grammar is a single language document with a language identifier provided in the language declaration in the grammar header. All tokens within that grammar, unless otherwise declared, will be handled according to the grammar's language. Grammar-level language declarations are not supported.

In the following example:

1. The grammar header declaration for the language "en-US" is supported by Amazon Lex V2.
2. Item-level language attachment (highlighted in red) is not supported. Amazon Lex V2 throws a validation error if a language attachment is different from the header declaration.

```xml
<?xml version="1.0" encoding="ISO-8859-1"?>
<!DOCTYPE grammar PUBLIC "-//W3C//DTD GRAMMAR 1.0//EN" "http://www.w3.org/TR/speech-grammar/grammar.dtd">
<!-- the default grammar language is US English -->
  <!-- single language attachment to tokens
  "yes" inherits US English language
  "oui" is Canadian French language
  -->
  <rule id="yes">
    <one-of>
      <item>yes</item>
      <item xml:lang="fr-CA">oui</item>
    </one-of>
  </rule>

  <!-- Single language attachment to an expansion -->
  <rule id="people1">
    <one-of xml:lang="fr-CA">
      <item>Michel Tremblay</item>
      <item>André Roy</item>
    </one-of>
  </rule>
</grammar>
```
Tags

The following discussion applies to defined for grammars. For more information, see Tags in the Speech recognition grammar specification version 1 W3C recommendation.

Based on the SRGS specification, tags can be defined in the following ways:

1. As part of a header declaration as described in Header declarations (p. 45).
2. As part of a <rule> definition.

The following tag formats are supported:

- semantics/1.0 (SISR, ECMAScript)
- semantics/1.0-literals (SISR string literals)

The following tag formats are not supported:

- swi-semantics/1.0 (Nuance proprietary)

Example

```xml
<grammar xmlns="http://www.w3.org/2001/06/grammar"
         xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
         xml:base="http://www.example.com/base-file-path"
         xsi:schemaLocation="http://www.w3.org/2001/06/grammar
              http://www.w3.org/TR/speech-grammar/grammar.xsd"
         xml:lang="en-US"
         version="1.0"
         mode="voice"
         root="city"
         tag-format="semantics/1.0-literals">
  <rule id="no">
    <one-of>
      <item>no</item>
      <item>nope</item>
      <item>no way</item>
    </one-of>
    <tag>no</tag>
  </rule>
</grammar>
```

Unsupported rules

In addition to the unsupported rules mentioned in the previous sections, Amazon Lex V2 doesn't support the following:

- Weights – You can't add the weight attribute to an item element.

Script format

Amazon Lex V2 supports the following ECMAScript features for defining grammars.

Amazon Lex V2 supports the following ECMAScript features when specifying tags in the grammar. tag-format must be sent to semantics/1.0 when ECMAScript tags are used in the grammar. For more information, see the ECMA-262 ECMAScript 2021 language specification.
Topics

• Variable statement (p. 53)
• Expressions (p. 53)
• If statement (p. 55)
• Switch statement (p. 55)
• Function declarations (p. 56)
• Iteration statement (p. 56)
• Block statement (p. 56)
• Comments (p. 56)
• Unsupported statements (p. 57)

This document contains material from the ECMAScript standard (available at https://www.ecma-international.org/publications-and-standards/standards/ecma-262/). The ECMAScript language specification document is available from Ecma International under the following license.

License text

© 2020 Ecma International

This document may be copied, published and distributed to others, and certain derivative works of it may be prepared, copied, published, and distributed, in whole or in part, provided that the above copyright notice and this Copyright License and Disclaimer are included on all such copies and derivative works. The only derivative works that are permissible under this Copyright License and Disclaimer are:

(i) works which incorporate all or portion of this document for the purpose of providing commentary or explanation (such as an annotated version of the document),

(ii) works which incorporate all or portion of this document for the purpose of incorporating features that provide accessibility,

(iii) translations of this document into languages other than English and into different formats and

(iv) works by making use of this specification in standard conformant products by implementing (e.g. by copy and paste wholly or partly) the functionality therein.

However, the content of this document itself may not be modified in any way, including by removing the copyright notice or references to Ecma International, except as required to translate it into languages other than English or into a different format.

The official version of an Ecma International document is the English language version on the Ecma International website. In the event of discrepancies between a translated version and the official version, the official version shall govern.

The limited permissions granted above are perpetual and will not be revoked by Ecma International or its successors or assigns. This document and the information contained herein is provided on an "AS IS" basis and ECMA INTERNATIONAL DISCLAIMS ALL WARRANTIES, EXPRESS OR IMPLIED, INCLUDING BUT NOT LIMITED TO ANY WARRANTY THAT THE USE OF THE INFORMATION HEREIN WILL NOT INFRINGE ANY OWNERSHIP RIGHTS OR ANY IMPLIED WARRANTIES OF MERCHANTABILITY OR FITNESS FOR A PARTICULAR PURPOSE."
Variable statement

A variable statement defines one or more variables.

```javascript
var x = 10;
var x = 10, var y = <expression>;
```

Expressions

<table>
<thead>
<tr>
<th>Expression type</th>
<th>Syntax</th>
<th>Example</th>
<th>Supported?</th>
</tr>
</thead>
<tbody>
<tr>
<td>Regular expression literal</td>
<td>String literal containing valid regex special characters</td>
<td>&quot;^\d.$&quot;</td>
<td>No</td>
</tr>
</tbody>
</table>
| Function                   | functionName(parameters)
functionName(parameters)
{ functionBody}            | var x = function calc() {
return 10;
}                                                                               | No        |
| Delete                     | delete expression                                                      | delete obj.property;                                                  | No        |
| Void                       | void expression                                                        | void (2 == '2');                                                      | No        |
| Typeof                     | typeof expression                                                      | typeof 42;                                                             | No        |
| Member index               | expression [ expressions ]                                            | var fruits = ["apple"];
fruits[0];                                                            | Yes       |
<p>| Member dot                 | expression . identifier                                                | out.value                                                             | Yes       |
| Arguments                  | expression (arguments)                                                | new Date('1994-10-11')                                               | Yes       |
| Post increment             | expression++                                                           | var x=10; x++;                                                        | Yes       |
| Post decrement             | expression--                                                           | var x=10; x--;                                                        | Yes       |
| Pre increment              | ++expression                                                           | var x=10; ++x;                                                        | Yes       |
| Pre decrement              | --expression                                                           | var x=10; --x;                                                        | Yes       |
| Unary plus / Unary minus   | +expression / - expression                                              | +x / -x;                                                              | Yes       |
| Bit not                    | - expression                                                           | const a = 5;                                                         | Yes       |</p>
<table>
<thead>
<tr>
<th>Expression type</th>
<th>Syntax</th>
<th>Example</th>
<th>Supported?</th>
</tr>
</thead>
<tbody>
<tr>
<td>Logical not</td>
<td>! expression</td>
<td>!(a &gt; 0</td>
<td></td>
</tr>
<tr>
<td>Multiplicative</td>
<td>expression ('*'</td>
<td>'/'</td>
<td>'%' ) expression</td>
</tr>
<tr>
<td>Additive</td>
<td>expression ('+'</td>
<td>'-' ) expression</td>
<td>(a + b) - (a - (a + b))</td>
</tr>
<tr>
<td>Bit shift</td>
<td>expression ('&lt;&lt;'</td>
<td>'&gt;&gt;'</td>
<td>'&gt;&gt;&gt;' ) expression</td>
</tr>
<tr>
<td>Relative</td>
<td>expression ('&lt;'</td>
<td>'&gt;'</td>
<td>'&lt;='</td>
</tr>
<tr>
<td>In</td>
<td>expression in expression</td>
<td>fruits[0] in otherFruits;</td>
<td>Yes</td>
</tr>
<tr>
<td>Equality</td>
<td>expression ('=='</td>
<td>'!='</td>
<td>'==='</td>
</tr>
<tr>
<td>Bit and / xor / or</td>
<td>expression ('&amp;'</td>
<td>'^'</td>
<td>'</td>
</tr>
<tr>
<td>Logical and / or</td>
<td>expression ('&amp;&amp;'</td>
<td>'</td>
<td></td>
</tr>
<tr>
<td>Ternary</td>
<td>expression ? expression : expression</td>
<td>a &gt; b ? obj.prop : 0</td>
<td>Yes</td>
</tr>
<tr>
<td>Assignment</td>
<td>expression = expression</td>
<td>out.value = &quot;string&quot;;</td>
<td>Yes</td>
</tr>
<tr>
<td>Assignment operator</td>
<td>expression ('*'</td>
<td>'/='</td>
<td>'+='</td>
</tr>
<tr>
<td>Assignment bitwise operator</td>
<td>expression ('&lt;&lt;='</td>
<td>'&gt;&gt;='</td>
<td>'&gt;&gt;&gt;'</td>
</tr>
<tr>
<td>Identifier</td>
<td>identifierSequence where identifierSequence is a sequence of valid characters</td>
<td>fruits = [10, 20, 30];</td>
<td>Yes</td>
</tr>
<tr>
<td>Expression type</td>
<td>Syntax</td>
<td>Example</td>
<td>Supported?</td>
</tr>
<tr>
<td>-----------------</td>
<td>--------</td>
<td>---------</td>
<td>------------</td>
</tr>
<tr>
<td>Null literal</td>
<td>null</td>
<td>x = null;</td>
<td>Yes</td>
</tr>
<tr>
<td>Boolean literal</td>
<td>true</td>
<td>false</td>
<td>x = true;</td>
</tr>
<tr>
<td>String literal</td>
<td>'string' / &quot;string&quot;</td>
<td>a = 'hello', b = &quot;world&quot;;</td>
<td>Yes</td>
</tr>
<tr>
<td>Decimal literal</td>
<td>integer [. ] digits [exponent]</td>
<td></td>
<td>Yes</td>
</tr>
<tr>
<td>Hex literal</td>
<td>0 (x</td>
<td>X)[0-9a-fA-F]</td>
<td></td>
</tr>
<tr>
<td>Octal literal</td>
<td>O [0-7]</td>
<td>&quot;051&quot;</td>
<td>Yes</td>
</tr>
<tr>
<td>Array literal</td>
<td>[ expression, ... ]</td>
<td>v = [a, b, c];</td>
<td>Yes</td>
</tr>
<tr>
<td>Object literal</td>
<td>{property: value, ...}</td>
<td>out = {value: 1, flag: false};</td>
<td>Yes</td>
</tr>
<tr>
<td>Parenthesized</td>
<td>( expressions )</td>
<td>x + (x + y)</td>
<td>Yes</td>
</tr>
</tbody>
</table>

### If statement

```
if (expressions) {
    statements;
} else {
    statements;
}
```

**Note:** In the preceding example, **expressions** and **statements** must be one of the supported ones from this document.

### Switch statement

```
switch (expression) {
    case (expression):
        statements
    ...
    default:
        statements
}
```

**Note:** In the preceding example, **expressions** and **statements** must be one of the supported ones from this document.
Function declarations

```javascript
function functionIdentifier([parameterList, ...]) {
  <function body>
}
```

Iteration statement

Iteration statements can be any one of the following:

```javascript
// Do..While statement
do {
  statements
} while (expressions)

// While Loop
while (expressions) {
  statements
}

// For Loop
for ([initialization]; [condition]; [final-expression])
  statement

// For..In
for (variable in object) {
  statement
}
```

Block statement

```javascript
{
  statements
}
```

// Example
{
  x = 10;
  if (x > 10) {
    console.log("greater than 10");
  }
}

Note: In the preceding example, statements provided in the block must be one of the supported ones from this document.

Comments

```javascript
// Single Line Comments
"// <comment>"

// Multiline comments
/**
 * <comment>
 ***/
```
Unsupported statements

Amazon Lex V2 doesn't support the following ECMAScript features.

Topics
- Empty statement (p. 57)
- Continue statement (p. 57)
- Break statement (p. 57)
- Return statement (p. 57)
- Throw statement (p. 57)
- Try statement (p. 58)
- Debugger statement (p. 58)
- Labeled statement (p. 58)
- Class declaration (p. 58)

Empty statement

The empty statement is used to provide no statement. The following is the syntax for an empty statement:

```
;
```

Continue statement

The continue statement without a label is supported with the Iteration statement (p. 56). The continue statement with a label isn't supported.

```
// continue with label
// this allows the program to jump to a
// labelled statement (see labelled statement below)
continue <label>;
```

Break statement

The break statement without a label is supported with the Iteration statement (p. 56). The break statement with a label isn't supported.

```
// break with label
// this allows the program to break out of a
// labelled statement (see labelled statement below)
break <label>;
```

Return statement

```
return expression;
```

Throw statement

The throw statement is used to throw a user-defined exception.

```
throw expression;
```
Try statement

```javascript
try {
    statements
} catch (expression) {
    statements
} finally {
    statements
}
```

Debugger statement

The debugger statement is used to invoke debugging functionality provided by the environment.

```javascript
debugger;
```

Labeled statement

The labeled statement can be used with `break` or `continue` statements.

```javascript
label:
    statements

// Example
let str = '';

loop1:
    for (let i = 0; i < 5; i++) {
        if (i === 1) {
            continue loop1;
        }
        str = str + i;
    }

console.log(str);
```

Class declaration

```javascript
class Rectangle {
    constructor(height, width) {
        this.height = height;
        this.width = width;
    }
}
```

Industry grammars

Industry grammars are a set of XML files to use with the grammar slot type. You can use these to quickly deliver a consistent end-user experience as you migrate interactive voice response work flows to Amazon Lex V2. You can select from a range of pre-built grammars across three domains: financial services, insurance, and telecom. There is also a generic set of grammars that you can use as a starting point for your own grammars.

The grammars contain the rules to collect the information and the ECMAScript tags for semantic interpretation.
Grammars for financial services (download)

The following grammars are supported for financial services: account and routing numbers, credit card and loan numbers, credit score, account opening and closing dates, and Social Security number.

Account number

```xml
<?xml version="1.0" encoding="UTF-8" ?>
<grammar xmlns="http://www.w3.org/2001/06/grammar"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xsi:schemaLocation="http://www.w3.org/2001/06/grammar
http://www.w3.org/TR/speech-grammar/grammar.xsd"
xml:lang="en-US" version="1.0"
root="main"
mode="voice"
tag-format="semantics/1.0">
  <!-- Test Cases
  Grammar will support the following inputs:
  Scenario 1:
  Input: My account number is A B C 1 2 3 4
  Output: ABC1234
  Scenario 2:
  Input: My account number is 1 2 3 4 A B C
  Output: 1234ABC
  Scenario 3:
  Input: Hmm My account number is 1 2 3 4 A B C 1
  Output: 123ABC1
  -->
  <rule id="main" scope="public">
    <tag>out=""</tag>
    <item><ruleref uri="#alphanumeric"/><tag>out += rules.alphanumeric.alphanum;</tag></item>
    <item repeat="0-1"><ruleref uri="#alphabets"/><tag>out += rules.alphabets.letters;</tag></item>
    <item repeat="0-1"><ruleref uri="#digits"/><tag>out += rules.digits.numbers</tag></item>
  </rule>
  <rule id="text">
    <one-of>
      <item repeat="0-1">account number is</item>
      <item repeat="0-1">Account Number</item>
      <item repeat="0-1">Here is my Account Number</item>
      <item repeat="0-1">Yes, It is</item>
      <item repeat="0-1">Yes It is</item>
      <item repeat="0-1">Yes It's</item>
      <item repeat="0-1">My account Id is</item>
      <item repeat="0-1">This is the account Id</item>
      <item repeat="0-1">account Id</item>
    </one-of>
  </rule>
  <rule id="hesitation">
    <one-of>
      <item>Hmm</item>
      <item>Mmm</item>
    </one-of>
  </rule>
</grammar>
```
<item>My</item>
</one-of>
</rule>

<rule id="alphanumeric" scope="public">
<tag>out.alphanum=""</tag>
<item><ruleref uri="#alphabets"/></item>
<tag>out.alphanum += rules.alphabets.letters;</tag>
<item repeat="0-1"><ruleref uri="#digits"/></item>
<tag>out.alphanum += rules.digits.numbers</tag>
</rule>

<rule id="alphabets">
<item repeat="0-1"><ruleref uri="#text"></item>
<tag>out.letters=""</tag>
<tag>out.firstOccurence=""</tag>
<item repeat="1-"><one-of>
  <item>A</item>
  <item>B</item>
  <item>C</item>
  <item>D</item>
  <item>E</item>
  <item>F</item>
  <item>G</item>
  <item>H</item>
  <item>I</item>
  <item>J</item>
  <item>K</item>
  <item>L</item>
  <item>M</item>
  <item>N</item>
  <item>O</item>
  <item>P</item>
  <item>Q</item>
  <item>R</item>
  <item>S</item>
  <item>T</item>
  <item>U</item>
  <item>V</item>
  <item>W</item>
  <item>X</item>
  <item>Y</item>
  <item>Z</item>
</one-of>
</item>
</rule>

<rule id="digits">
<item repeat="0-1"><ruleref uri="#text"></item>
<tag>out.numbers=""</tag>
<item repeat="1-10"><one-of>
  <item>0</item>
  <item>1</item>
  <item>2</item>
  <item>3</item>
  <item>4</item>
  <item>5</item>
  <item>6</item>
  <item>7</item>
  <item>8</item>
  <item>9</item>
</one-of>
</item>
</rule>
Routing number

<?xml version="1.0" encoding="UTF-8" ?>
<grammar xmlns="http://www.w3.org/2001/06/grammar"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xsi:schemaLocation="http://www.w3.org/2001/06/grammar
http://www.w3.org/TR/speech-grammar/grammar.xsd"
xml:lang="en-US" version="1.0"
root="digits"
tag-format="semantics/1.0">
<!-- Test Cases
Grammar will support the following inputs:

Scenario 1:
Input: My routing number is 1 2 3 4 5 6 7 8 9
Output: 123456789

Scenario 2:
Input: routing number 1 2 3 4 5 6 7 8 9
Output: 123456789

-->}
<rule id="digits">
  <tag>out=""</tag>
  <item><ruleref uri="#singleDigit"/><tag>out += rules.singleDigit.digit;</tag></item>
</rule>

<rule id="text">
  <item repeat="0-1"><ruleref uri="#hesitation"/></item>
  <one-of>
    <item repeat="0-1">My routing number</item>
    <item repeat="0-1">Routing number of</item>
    <item repeat="0-1">The routing number is</item>
  </one-of>
</rule>

<rule id="hesitation">
  <one-of>
    <item>Hmm</item>
    <item>Mmm</item>
    <item>My</item>
  </one-of>
</rule>

<rule id="singleDigit">
  <item repeat="0-1"><ruleref uri="#text"/></item>
  <tag>out=""</tag>
  <item repeat="16">
    <item>0<tag>out.digit+=0;</tag></item>
    <item>zero<tag>out.digit+=0;</tag></item>
    <item>1<tag>out.digit+=1;</tag></item>
    <item>one<tag>out.digit+=1;</tag></item>
    <item>2<tag>out.digit+=2;</tag></item>
    <item>two<tag>out.digit+=2;</tag></item>
    <item>3<tag>out.digit+=3;</tag></item>
    <item>three<tag>out.digit+=3;</tag></item>
    <item>4<tag>out.digit+=4;</tag></item>
    <item>four<tag>out.digit+=4;</tag></item>
    <item>5<tag>out.digit+=5;</tag></item>
    <item>five<tag>out.digit+=5;</tag></item>
    <item>6<tag>out.digit+=6;</tag></item>
    <item>six<tag>out.digit+=6;</tag></item>
    <item>7<tag>out.digit+=7;</tag></item>
    <item>seven<tag>out.digit+=7;</tag></item>
    <item>8<tag>out.digit+=8;</tag></item>
    <item>eight<tag>out.digit+=8;</tag></item>
    <item>9<tag>out.digit+=9;</tag></item>
    <item>nine<tag>out.digit+=9;</tag></item>
  </item>
</rule>
Credit card number

<?xml version="1.0" encoding="UTF-8" ?>
<grammar xmlns="http://www.w3.org/2001/06/grammar"
  xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
  xsi:schemaLocation="http://www.w3.org/2001/06/grammar
  http://www.w3.org/TR/speech-grammar/grammar.xsd"
  xml:lang="en-US" version="1.0"
  root="digits"
  mode="voice"
  tag-format="semantics/1.0">
  <!-- Test Cases
  Grammar will support the following inputs:
  
  Scenario 1:
  Input: My credit card number is 1 2 3 4 5 6 7 8 9 1 2 3 4 5 6 7
  Output: 1234567891234567
  
  Scenario 2:
  Input: card number 1 2 3 4 5 6 7 8 9 1 2 3 4 5 6 7
  Output: 1234567891234567
  
  -->
  <rule id="digits">
    <tag>out=""</tag>
    <item><ruleref uri="#singleDigit"/></item><tag>out += rules.singleDigit.digit;</tag>
  </item>
</rule>

<rule id="text">
  <item repeat="0-1"><ruleref uri="#hesitation"/></item>
  <one-of>
    <item repeat="0-1">My credit card number is</item>
    <item repeat="0-1">card number</item>
  </one-of>
</rule>

<rule id="hesitation">
  <one-of>
    <item>Hmm</item>
    <item>Mmm</item>
    <item>My</item>
  </one-of>
</rule>
<one-of>
</rule>

<rule id="singleDigit">
  <item repeat="0-1"><ruleref uri="#text"/></item>
  <tag>out.digit=""</tag>
  <item repeat="16">
    <one-of>
      <item>0<tag>out.digit+=0;</tag></item>
      <item>zero<tag>out.digit+=0;</tag></item>
      <item>1<tag>out.digit+=1;</tag></item>
      <item>one<tag>out.digit+=1;</tag></item>
      <item>2<tag>out.digit+=2;</tag></item>
      <item>two<tag>out.digit+=2;</tag></item>
      <item>3<tag>out.digit+=3;</tag></item>
      <item>three<tag>out.digit+=3;</tag></item>
      <item>4<tag>out.digit+=4;</tag></item>
      <item>four<tag>out.digit+=4;</tag></item>
      <item>5<tag>out.digit+=5;</tag></item>
      <item>five<tag>out.digit+=5;</tag></item>
      <item>6<tag>out.digit+=6;</tag></item>
      <item>six<tag>out.digit+=5;</tag></item>
      <item>7<tag>out.digit+=7;</tag></item>
      <item>seven<tag>out.digit+=7;</tag></item>
      <item>8<tag>out.digit+=8;</tag></item>
      <item>eight<tag>out.digit+=8;</tag></item>
      <item>9<tag>out.digit+=9;</tag></item>
      <item>nine<tag>out.digit+=9;</tag></item>
    </one-of>
  </item>
</rule>
</grammar>

Loan ID

<?xml version="1.0" encoding="UTF-8" ?>
<grammar xmlns="http://www.w3.org/2001/06/grammar"
  xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
  xsi:schemaLocation="http://www.w3.org/2001/06/grammar
  http://www.w3.org/TR/speech-grammar/grammar.xsd"
  xml:lang="en-US" version="1.0"
  root="main"
  mode="voice"
  tag-format="semantics/1.0">

  <!-- Test Cases
  Grammar will support the following inputs:

  Scenario 1:
  Input: My loan Id is A B C 1 2 3 4
  Output: ABC1234
  -->

  <rule id="main" scope="public">
    <tag>out=""</tag>
    <item><ruleref uri="#alphanumeric"></tag><tag>out += rules.alphanumeric.alphanum;</tag></item>
    <item repeat="0-1"><ruleref uri="#alphabets"></tag><tag>out += rules.alphabets.letters;</tag></item>
    <item repeat="0-1"><ruleref uri="#digits"></tag><tag>out += rules.digits.numbers</tag></item>
  </rule>
</grammar>
<rule id="text">
  <item repeat="0-1"><ruleref uri="#hesitation"/></item>
  <one-of>
    <item repeat="0-1">my loan number is</item>
    <item repeat="0-1">The loan number</item>
    <item repeat="0-1">The loan is</item>
    <item repeat="0-1">The number is</item>
    <item repeat="0-1">loan number</item>
    <item repeat="0-1">loan number of</item>
    <item repeat="0-1">loan Id is</item>
    <item repeat="0-1">My loan Id is</item>
  </one-of>
</rule>

<rule id="hesitation">
  <one-of>
    <item>Hmm</item>
    <item>Mmm</item>
    <item>My</item>
  </one-of>
</rule>

<rule id="alphanumeric" scope="public">
  <tag>out.alphanum=""></tag>
  <item><ruleref uri="#alphabets"></item>
  <item repeat="0-1"><ruleref uri="#digits"></item>
  </rule>

<rule id="alphabets">
  <item repeat="0-1"><ruleref uri="#text"></item>
  <tag>out.letters=""></tag>
  <item repeat="0-1"><ruleref uri="#digits"></item>
  <tag>out.firstOccurence=""></tag>
  <item repeat="1-">
    <one-of>
      <item>A</item>
      <item>B</item>
      <item>C</item>
      <item>D</item>
      <item>E</item>
      <item>F</item>
      <item>G</item>
      <item>H</item>
      <item>I</item>
      <item>J</item>
      <item>K</item>
      <item>L</item>
      <item>M</item>
      <item>N</item>
      <item>O</item>
      <item>P</item>
      <item>Q</item>
      <item>R</item>
      <item>S</item>
      <item>T</item>
      <item>U</item>
      <item>V</item>
      <item>W</item>
      <item>X</item>
      <item>Y</item>
      <item>Z</item>
    </one-of>
  </item>
</rule>

<rule id="digits">
  <item repeat="0-1">0</item>
  <item repeat="0-1">1</item>
  <item repeat="0-1">2</item>
  <item repeat="0-1">3</item>
  <item repeat="0-1">4</item>
  <item repeat="0-1">5</item>
  <item repeat="0-1">6</item>
  <item repeat="0-1">7</item>
  <item repeat="0-1">8</item>
  <item repeat="0-1">9</item>
</rule>
Credit score

<?xml version="1.0" encoding="UTF-8" ?>
<grammar xmlns="http://www.w3.org/2001/06/grammar"
  xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
  xsi:schemaLocation="http://www.w3.org/2001/06/grammar
  http://www.w3.org/TR/speech-grammar/grammar.xsd"
  xml:lang="en-US" version="1.0"
  root="main"
  mode="voice"
  tag-format="semantics/1.0">
  <!-- Test Cases
Grammar will support the following inputs:

Scenario 1:
Input: The number is fifteen
Output: 15

Scenario 2:
Input: My credit score is fifteen
Output: 15
-->

<rule id="main" scope="public">
  <tag>out=""</tag>
  <one-of>
    <item repeat="1"><ruleref uri="#digits"/><tag>out+= rules.digits;</tag></item>
    <item repeat="1"><ruleref uri="#teens"/><tag>out+= rules.teens;</tag></item>
    <item repeat="1"><ruleref uri="#above_twenty"/><tag>out+= rules.above_twenty;</tag></item>
  </one-of>
</rule>

<rule id="text">
  <one-of>
    <item repeat="0-1">Credit score is</item>
    <item repeat="0-1">Last digits are</item>
    <item repeat="0-1">The number is</item>
  </one-of>
</grammar>
<item repeat="0-1">That's</item>  
<item repeat="0-1">It is</item>  
<item repeat="0-1">My credit score is</item>
</one-of>
</rule>

<rule id="digits">
  <item repeat="0-1"><ruleref uri="#text"/></item>
  <one-of>
    <item>0<tag>out=0;</tag></item>
    <item>1<tag>out=1;</tag></item>
    <item>2<tag>out=2;</tag></item>
    <item>3<tag>out=3;</tag></item>
    <item>4<tag>out=4;</tag></item>
    <item>5<tag>out=5;</tag></item>
    <item>6<tag>out=6;</tag></item>
    <item>7<tag>out=7;</tag></item>
    <item>8<tag>out=8;</tag></item>
    <item>9<tag>out=9;</tag></item>
    <item>one<tag>out=1;</tag></item>
    <item>two<tag>out=2;</tag></item>
    <item>three<tag>out=3;</tag></item>
    <item>four<tag>out=4;</tag></item>
    <item>five<tag>out=5;</tag></item>
    <item>six<tag>out=6;</tag></item>
    <item>seven<tag>out=7;</tag></item>
    <item>eight<tag>out=8;</tag></item>
    <item>nine<tag>out=9;</tag></item>
  </one-of>
</rule>

<rule id="teens">
  <item repeat="0-1"><ruleref uri="#text"/></item>
  <one-of>
    <item>ten<tag>out=10;</tag></item>
    <item>eleven<tag>out=11;</tag></item>
    <item>twelve<tag>out=12;</tag></item>
    <item>thirteen<tag>out=13;</tag></item>
    <item>fourteen<tag>out=14;</tag></item>
    <item>fifteen<tag>out=15;</tag></item>
    <item>sixteen<tag>out=16;</tag></item>
    <item>seventeen<tag>out=17;</tag></item>
    <item>eighteen<tag>out=18;</tag></item>
    <item>nineteen<tag>out=19;</tag></item>
    <item>ten<tag>out=10;</tag></item>
    <item>eleven<tag>out=11;</tag></item>
    <item>twelve<tag>out=12;</tag></item>
    <item>thirteen<tag>out=13;</tag></item>
    <item>fourteen<tag>out=14;</tag></item>
    <item>fifteen<tag>out=15;</tag></item>
    <item>sixteen<tag>out=16;</tag></item>
    <item>seventeen<tag>out=17;</tag></item>
    <item>eighteen<tag>out=18;</tag></item>
    <item>nineteen<tag>out=19;</tag></item>
  </one-of>
</rule>

<rule id="above_twenty">
  <item repeat="0-1"><ruleref uri="#text"/></item>
  <one-of>
    <item>twenty<tag>out=20;</tag></item>
    <item>thirty<tag>out=30;</tag></item>
    <item>forty<tag>out=40;</tag></item>
    <item>fifty<tag>out=50;</tag></item>
    <item>六十<tag>out=60;</tag></item>
    <item>seventy<tag>out=70;</tag>
  </one-of>
</rule>
Account opening date

<?xml version="1.0" encoding="UTF-8" ?>
<grammar xmlns="http://www.w3.org/2001/06/grammar"
  xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xsi:schemaLocation="http://www.w3.org/2001/06/grammar
  http://www.w3.org/TR/speech-grammar/grammar.xsd"
xmllang="en-US" version="1.0"
root="main"
mode="voice"
tag-format="semantics/1.0">
  <!-- Test Cases
Grammar will support the following inputs:
  Scenario 1:
  Input: I opened account on July Two Thousand and Eleven
  Output: 07/11
  Scenario 2:
  Input: I need account number opened on July Two Thousand and Eleven
  Output: 07/11
  -->
  <rule id="main" scope="public">
    <tag>out=""</tag>
    <item repeat="1-10">
      <item repeat="1"><ruleref uri="#months"/><tag>out = out + rules.months.mon + "/";</tag></item>
      <one-of>
      <item><ruleref uri="#thousands"/><tag>out += rules.thousands;</tag></item>
      <item repeat="0-1"><ruleref uri="#digits"/><tag>out += rules.digits;</tag></item>
      <item repeat="0-1"><ruleref uri="#teens"/><tag>out += rules.teens;</tag></item>
      <item repeat="0-1"><ruleref uri="#above_twenty"/><tag>out += rules.above_twenty;</tag></item>
      </one-of>
    </item>
  </rule>
  <rule id="text">
    <item repeat="0-1"><ruleref uri="#hesitation"/></item>
    <one-of>
    </one-of>
  </rule>
</grammar>
<item repeat="0-1">I opened account on </item>
<item repeat="0-1">I need account number opened on </item>
</one-of>
</rule>

<rule id="hesitation">
<one-of>
  <item>Hmm</item>
  <item>Mmm</item>
  <item>My</item>
</one-of>
</rule>

<rule id="months">
<item repeat="0-1"><ruleref uri="#text"></item>
<tag>out.mon=""</tag>
<one-of>
  <item>january<tag>out.mon="01";</tag></item>
  <item>february<tag>out.mon="02";</tag></item>
  <item>april<tag>out.mon="04";</tag></item>
  <item>july<tag>out.mon="07";</tag></item>
  <item>aug<tag>out.mon="08";</tag></item>
  <item>sept<tag>out.mon="09";</tag></item>
  <item>dec<tag>out.mon="12";</tag>
</one-of>
</rule>

<rule id="digits">
<one-of>
  <item>zero<tag>out=0;</tag></item>
  <item>one<tag>out=1;</tag></item>
  <item>two<tag>out=2;</tag></item>
  <item>three<tag>out=3;</tag></item>
  <item>four<tag>out=4;</tag></item>
  <item>five<tag>out=5;</tag></item>
  <item>six<tag>out=6;</tag></item>
  <item>seven<tag>out=7;</tag></item>
  <item>eight<tag>out=8;</tag></item>
  <item>nine<tag>out=9;</tag>
</one-of>
</rule>

<rule id="teens">
<one-of>
  <item>ten<tag>out=10;</tag></item>
  <item>eleven<tag>out=11;</tag></item>
  <item>twelve<tag>out=12;</tag></item>
  <item>thirteen<tag>out=13;</tag></item>
  <item>fourteen<tag>out=14;</tag></item>
  <item>fifteen<tag>out=15;</tag></item>
  <item>sixteen<tag>out=16;</tag></item>
  <item>seventeen<tag>out=17;</tag></item>
  <item>eighteen<tag>out=18;</tag></item>
  <item>nineteen<tag>out=19;</tag>
</one-of>
</rule>
Industry grammars

```xml
<grammar xmlns="http://www.w3.org/2001/06/grammar" xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance" xsi:schemaLocation="http://www.w3.org/2001/06/grammar http://www.w3.org/TR/speech-grammar/grammar.xsd" xml:lang="en-US" version="1.0" root="main" mode="voice" tag-format="semantics/1.0">

<!-- Test Cases
Grammar will support the following inputs:

Scenario 1:
Input: I want to schedule auto pay for twenty five Dollar
Output: $25

Scenario 2:
Input: Setup automatic payments for twenty five dollars
Output: $25

-->  

<rule id="main" scope="public">
  <tag>out="$"</tag>
  <one-of>
    <item><ruleref uri="#sub_hundred"/></item>
    <item><ruleref uri="#subThousands"/></item>
  </one-of>
</rule>
</grammar>
```

Automatic pay date
<rule id="text">
  <item repeat="0-1">ruleref uri="#hesitation"/></item>
  <one-of>
    <item repeat="0-1">I want to schedule auto pay for</item>
    <item repeat="0-1">Setup automatic payments for twenty five dollars</item>
    <item repeat="0-1">Auto pay amount of</item>
    <item repeat="0-1">Set it up for</item>
  </one-of>
</rule>

<rule id="hesitation">
  <one-of>
    <item>Hmm</item>
    <item>Mmm</item>
    <item>My</item>
  </one-of>
</rule>

<rule id="digits">
  <item repeat="0-1">ruleref uri="#text"/></item>
  <tag>out.num = 0;</tag>
  <one-of>
    <item>0</item><tag>out.num+=0;</tag></item>
    <item>1</item><tag>out.num+=1;</tag></item>
    <item>2</item><tag>out.num+=2;</tag></item>
    <item>3</item><tag>out.num+=3;</tag></item>
    <item>4</item><tag>out.num+=4;</tag></item>
    <item>5</item><tag>out.num+=5;</tag></item>
    <item>6</item><tag>out.num+=6;</tag></item>
    <item>7</item><tag>out.num+=7;</tag></item>
    <item>8</item><tag>out.num+=8;</tag></item>
    <item>9</item><tag>out.num+=9;</tag></item>
    <item>one</item><tag>out.num+=1;</tag></item>
    <item>two</item><tag>out.num+=2;</tag></item>
    <item>three</item><tag>out.num+=3;</tag></item>
    <item>four</item><tag>out.num+=4;</tag></item>
    <item>five</item><tag>out.num+=5;</tag></item>
    <item>six</item><tag>out.num+=6;</tag></item>
    <item>seven</item><tag>out.num+=7;</tag></item>
    <item>eight</item><tag>out.num+=8;</tag></item>
    <item>nine</item><tag>out.num+=9;</tag></item>
  </one-of>
  <item repeat="0-1">ruleref uri="#currency"/></item>
</rule>

<rule id="teens">
  <item repeat="0-1">ruleref uri="#text"/></item>
  <tag>out.teen = 0;</tag>
  <one-of>
    <item>ten</item><tag>out.teen+=10;</tag></item>
    <item>eleven</item><tag>out.teen+=11;</tag></item>
    <item>twelve</item><tag>out.teen+=12;</tag></item>
    <item>thirteen</item><tag>out.teen+=13;</tag></item>
    <item>fourteen</item><tag>out.teen+=14;</tag></item>
    <item>fifteen</item><tag>out.teen+=15;</tag></item>
    <item>sixteen</item><tag>out.teen+=16;</tag></item>
    <item>seventeen</item><tag>out.teen+=17;</tag></item>
    <item>eighteen</item><tag>out.teen+=18;</tag></item>
    <item>nineteen</item><tag>out.teen+=19;</tag></item>
  </one-of>
  <item repeat="0-1">ruleref uri="#currency"/></item>
</rule>

<rule id="above_twenty">
  <item repeat="0-1">ruleref uri="#text"/></item>
  <tag>out.tens = 0;</tag>
  <one-of>
    <item>70</item>
  </one-of>
</rule>
<grammar>
  <rule id="currency">
    <one-of>
      <item repeat="0-1">dollars</item>
      <item repeat="0-1">Dollars</item>
      <item repeat="0-1">dollar</item>
      <item repeat="0-1">Dollar</item>
    </one-of>
  </rule>

  <rule id="sub_hundred">
    <item repeat="0-1">dollars</item>
    <item repeat="0-1">Dollars</item>
    <item repeat="0-1">dollar</item>
    <item repeat="0-1">Dollar</item>
  </rule>

  <rule id="subThousands">
    <ruleref uri="#currency"/>
    <tag>out = (100 * rules.sub_hundred.sh);</tag>
    <hundred>
      <item repeat="0-1">dollars</item>
      <item repeat="0-1">Dollars</item>
      <item repeat="0-1">dollar</item>
      <item repeat="0-1">Dollar</item>
    </hundred>
    <rules.above_twenty.tens;></tag>
    <item repeat="0-1">dollars</item>
    <item repeat="0-1">Dollars</item>
    <item repeat="0-1">dollar</item>
    <item repeat="0-1">Dollar</item>
  </rule>

  <rule id="textarea">
    <item repeat="0-1">text</item>
    <tag>out.sh = 0;</tag>
    <one-of>
      <item>ruleref uri="#teens"/></item>
      <tag>out.sh += rules.teens.teen;</tag>
      <item>ruleref uri="#above_twenty"/></item>
      <tag>out.sh += rules.above_twenty.tens;</tag>
      <item>ruleref uri="#digits"/></item>
      <tag>out.sh += rules.digits.num;</tag>
    </one-of>
  </rule>
</grammar>
<rule id="dateCardExpiration" scope="public">
    <tag>out=""</tag>
    <item repeat="1"><ruleref uri="#months"/><tag>out = out + rules.months;</tag></item>
    <item repeat="1"><ruleref uri="#year"/><tag>out += " " + rules.year.yr;</tag></item>
</rule>

<!-- Test Cases
Grammar will support the following inputs:

Scenario 1:
Input: My card expiration date is july eleven
Output: 07 2011

Scenario 2:
Input: My card expiration date is may twenty six
Output: 05 2026
-->

<rule id="text">
    <item repeat="0-1"><ruleref uri="#hesitation"/></item>
    <one-of>
        <item repeat="0-1">My card expiration date is </item>
        <item repeat="0-1">Expiration date is </item>
    </one-of>
</rule>

<rule id="hesitation">
    <one-of>
        <item>Hmm</item>
        <item>Mmm</item>
        <item>My</item>
    </one-of>
</rule>

<rule id="months">
    <item repeat="0-1"><ruleref uri="#text"/></item>
    <one-of>
        <item>january<tag>out="01";</tag></item>
        <item>february<tag>out="02";</tag></item>
        <item>march<tag>out="03";</tag></item>
        <item>april<tag>out="04";</tag></item>
        <item>may<tag>out="05";</tag></item>
        <item>june<tag>out="06";</tag></item>
        <item>july<tag>out="07";</tag></item>
        <item>august<tag>out="08";</tag></item>
        <item>sept<tag>out="09";</tag></item>
        <item>oct<tag>out="10";</tag></item>
        <item>nov<tag>out="11";</tag></item>
        <item>dec<tag>out="12";</tag></item>
        <item>1<tag>out="01";</tag></item>
        <item>2<tag>out="02";</tag></item>
        <item>3<tag>out="03";</tag></item>
        <item>4<tag>out="04";</tag></item>
        <item>5<tag>out="05";</tag></item>
        <item>6<tag>out="06";</tag></item>
    </one-of>
</rule>
<item>7<tag>out="07";/item>
<item>8<tag>out="08";/item>
<item>9<tag>out="09";/item>
<item>ten<tag>out="10";/item>
<item>eleven<tag>out="11";/item>
<item>twelve<tag>out="12";/item>
</one-of>
</rule>
</rule id="digits">
<item repeat="0-1"><ruleref uri="#text"/></item>
<one-of>
<item>0<tag>out=0;</item>
<item>1<tag>out=1;</item>
<item>2<tag>out=2;</item>
<item>3<tag>out=3;</item>
<item>4<tag>out=4;</item>
<item>5<tag>out=5;</item>
<item>6<tag>out=6;</item>
<item>7<tag>out=7;</item>
<item>8<tag>out=8;</item>
<item>9<tag>out=9;</item>
<item>one<tag>out=1;</item>
<item>two<tag>out=2;</item>
<item>three<tag>out=3;</item>
<item>four<tag>out=4;</item>
<item>five<tag>out=5;</item>
<item>six<tag>out=6;</item>
<item>seven<tag>out=7;</item>
<item>eight<tag>out=8;</item>
<item>nine<tag>out=9;</item>
</one-of>
</rule>
</rule id="year">
<tag>out.yr="20";/tag>
<one-of>
<item><ruleref uri="#teens"/>out.yr += rules.teens;</item>
<item><ruleref uri="#above_twenty"/>out.yr += rules.above_twenty;</item>
</one-of>
</rule>
</rule id="teens">
<item repeat="0-1"><ruleref uri="#text"/></item>
<one-of>
<item>ten<tag>out=10;</item>
<item>eleven<tag>out=11;</item>
<item>twelve<tag>out=12;</item>
<item>thirteen<tag>out=13;</item>
<item>fourteen<tag>out=14;</item>
<item>fifteen<tag>out=15;</item>
<item>sixteen<tag>out=16;</item>
<item>seventeen<tag>out=17;</item>
<item>eighteen<tag>out=18;</item>
<item>nineteen<tag>out=19;</item>
<item>10<tag>out=10;</item>
<item>11<tag>out=11;</item>
<item>12<tag>out=12;</item>
<item>13<tag>out=13;</item>
<item>14<tag>out=14;</item>
<item>15<tag>out=15;</item>
<item>16<tag>out=16;</item>
<item>17<tag>out=17;</item>
<item>18<tag>out=18;</item>
<item>19<tag>out=19;</item>
Statement date

<?xml version="1.0" encoding="UTF-8" ?>
<grammar xmlns="http://www.w3.org/2001/06/grammar"
         xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xsi:schemaLocation="http://www.w3.org/2001/06/grammar
http://www.w3.org/TR/speech-grammar/grammar.xsd"
xmllang="en-US" version="1.0"
root="main"
mode="voice"
tag-format="semantics/1.0">
<!-- Test Cases
Grammar will support the following inputs:

Scenario 1:
Input: Show me statements from July Five Two Thousand and Eleven
Output: 07/5/11

Scenario 2:
Input: Show me statements from July Sixteen Two Thousand and Eleven
Output: 07/16/11

Scenario 3:
Input: Show me statements from July Thirty Two Thousand and Eleven
Output: 07/30/11
-->

<rule id="main" scope="public">
  <tag>out=""</tag>
  <item repeat="0-1">
    <ruleref uri="#months"/></item>
  <one-of>
    <item repeat="0-1">
      <ruleref uri="#digits"/>"out += rules.digits;</item>
    <item repeat="0-1">
      <ruleref uri="#months"/></item>
    <one-of>
      <item>twenty</item>
      <item>thirty</item>
      <item>forty</item>
      <item>fifty</item>
      <item>sixty</item>
      <item>seventy</item>
      <item>eighty</item>
      <item>ninety</item>
      <item>20</item>
      <item>30</item>
      <item>40</item>
      <item>50</item>
      <item>60</item>
      <item>70</item>
      <item>80</item>
      <item>90</item>
    </one-of>
  </one-of>
</rule>
</grammar>
<item><ruleref uri="#digits"/> <tag>out += rules.digits + "/";</tag></item>

<item><ruleref uri="#teens"/> <tag>out += rules.teens + "/";</tag></item>

<item><ruleref uri="#above_twenty"/> <tag>out += rules.above_twenty + "/";</tag></item>

</one-of>

<one-of>
<item><ruleref uri="#thousands"/> <tag>out += rules.thousands;</tag></item>
<item repeat="0-1"><ruleref uri="#digits"/> <tag>out += rules.digits;</tag></item>
<item repeat="0-1"><ruleref uri="#teens"/> <tag>out += rules.teens;</tag></item>
<item repeat="0-1"><ruleref uri="#above_twenty"/> <tag>out += rules.above_twenty;</tag></item>
</one-of>

</item>
</rule>

<rule id="text">
<item repeat="0-1"><ruleref uri="#hesitation"/></item>
<one-of>
<item>I want to see bank statements from</item>
<item>Show me statements from</item>
</one-of>
</rule>

<rule id="hesitation">
<one-of>
<item>Hmm</item>
<item>Mmm</item>
<item>My</item>
</one-of>
</rule>

<rule id="months">
<tag>out.mon="";</tag>
<item repeat="0-1"><ruleref uri="#text"/></item>
<one-of>
<item>january</item>
<item>february</item>
<item>march</item>
<item>april</item>
<item>may</item>
<item>june</item>
<item>july</item>
<item>august</item>
<item>september</item>
<item>october</item>
<item>november</item>
<item>december</item>
<item>jan</item>
<item>feb</item>
<item>aug</item>
<item>sept</item>
<item>oct</item>
<item>nov</item>
<item>dec</item>
</one-of>
</rule>

<rule id="digits">
<one-of>
<item>zero</item>
<item>one</item>
<item>two</item>
</one-of>
</rule>
Transaction date

<?xml version="1.0" encoding="UTF-8" ?>
<grammar xmlns="http://www.w3.org/2001/06/grammar"
 xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
 xsi:schemaLocation="http://www.w3.org/2001/06/grammar
 http://www.w3.org/TR/speech-grammar/grammar.xsd"
 xml:lang="en-US" version="1.0"
 root="main"
 mode="voice"
 tag-format="semantics/1.0">
 <!-- Test Cases
 Grammar will support the following inputs:
 Scenario 1:
 Input: My last incorrect transaction date is july twenty three
 Output: 07/23
 </scenario1>
Scenario 2:
Input: My last incorrect transaction date is july fifteen
Output: 07/15

```xml
<rule id="main" scope="public">
  <tag>out=""</tag>
  <item repeat="1-10">
    <item><ruleref uri="#months"/><tag>out= rules.months.mon + "/";</tag></item>
  </item>
  <one-of>
    <item><ruleref uri="#digits"/><tag>out+= rules.digits;</tag></item>
    <item><ruleref uri="#teens"/><tag>out+= rules.teens;</tag></item>
    <item><ruleref uri="#above_twenty"/><tag>out+= rules.above_twenty;</tag></item>
  </one-of>
</item>
</rule>

<rule id="text">
  <item repeat="0-1"><ruleref uri="#hesitation"/></item>
  <one-of>
    <item repeat="0-1">My last incorrect transaction date is</item>
    <item repeat="0-1">It is</item>
  </one-of>
</rule>

<rule id="hesitation">
  <one-of>
    <item>Hmm</item>
    <item>Mmm</item>
    <item>My</item>
  </one-of>
</rule>

<rule id="months">
  <item repeat="0-1"><ruleref uri="#text"/></item>
  <tag>out.mon=""</tag>
  <one-of>
    <item>jane<tag>out.mon+="01";</tag></item>
    <item>feb<tag>out.mon+="02";</tag></item>
    <item>mar<tag>out.mon+="03";</tag></item>
    <item>apr<tag>out.mon+="04";</tag></item>
    <item>may<tag>out.mon+="05";</tag></item>
    <item>jun<tag>out.mon+="06";</tag></item>
    <item>jul<tag>out.mon+="07";</tag></item>
    <item>aug<tag>out.mon+="08";</tag></item>
    <item>sept<tag>out.mon+="09";</tag></item>
    <item>oct<tag>out.mon+="10";</tag></item>
    <item>nov<tag>out.mon+="11";</tag></item>
    <item>dec<tag>out.mon+="12";</tag>
  </one-of>
</rule>

<rule id="digits">
  <item repeat="0-1"><ruleref uri="#text"/></item>
  <one-of>
    <item>0<tag>out=0;</tag></item>
    <item>7<tag>out+=7;</tag></item>
  </one-of>
</rule>
```
<item>1</item><tag>out=1;</tag></item>
  <item>2</item><tag>out=2;</tag></item>
  <item>3</item><tag>out=3;</tag></item>
  <item>4</item><tag>out=4;</tag></item>
  <item>5</item><tag>out=5;</tag></item>
  <item>6</item><tag>out=6;</tag></item>
  <item>7</item><tag>out=7;</tag></item>
  <item>8</item><tag>out=8;</tag></item>
  <item>9</item><tag>out=9;</tag></item>
  <item>first</item><tag>out=01;</tag></item>
  <item>second</item><tag>out=02;</tag></item>
  <item>third</item><tag>out=03;</tag></item>
  <item>fourth</item><tag>out=04;</tag></item>
  <item>fifth</item><tag>out=05;</tag></item>
  <item>sixth</item><tag>out=06;</tag></item>
  <item>seventh</item><tag>out=07;</tag></item>
  <item>eighth</item><tag>out=08;</tag></item>
  <item>ninth</item><tag>out=09;</tag></item>
  <item>one</item><tag>out=1;</tag></item>
  <item>two</item><tag>out=2;</tag></item>
  <item>three</item><tag>out=3;</tag></item>
  <item>four</item><tag>out=4;</tag></item>
  <item>five</item><tag>out=5;</tag></item>
  <item>six</item><tag>out=6;</tag></item>
  <item>seven</item><tag>out=7;</tag></item>
  <item>eight</item><tag>out=8;</tag></item>
  <item>nine</item><tag>out=9;</tag></item>
</one-of>
</rule>

<rule id="teens">
  <item repeat="0-1"><ruleref uri="#text"/></item>
  <one-of>
    <item>ten</item><tag>out=10;</tag></item>
    <item>tenth</item><tag>out=10;</tag></item>
    <item>eleven</item><tag>out=11;</tag></item>
    <item>twelve</item><tag>out=12;</tag></item>
    <item>thirteen</item><tag>out=13;</tag></item>
    <item>fourteen</item><tag>out=14;</tag></item>
    <item>fifteen</item><tag>out=15;</tag></item>
    <item>sixteen</item><tag>out=16;</tag></item>
    <item>seventeen</item><tag>out=17;</tag></item>
    <item>eighteen</item><tag>out=18;</tag></item>
    <item>nineteen</item><tag>out=19;</tag></item>
    <item>twenty</item><tag>out=20;</tag></item>
    <item>thirty</item><tag>out=30;</tag></item>
  </one-of>
</rule>

<rule id="above_twenty">
  <item repeat="0-1"><ruleref uri="#text"/></item>
  <one-of>
    <item>twenty</item><tag>out=20;</tag></item>
    <item>thirty</item><tag>out=30;</tag></item>
  </one-of>
</rule>
Transfer amount

```xml
<?xml version="1.0" encoding="UTF-8" ?>
<grammar xmlns="http://www.w3.org/2001/06/grammar"
  xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
  xsi:schemaLocation="http://www.w3.org/2001/06/grammar
                      http://www.w3.org/TR/speech-grammar/grammar.xsd"
  xml:lang="en-US" version="1.0"
  root="main"
  tag-format="semantics/1.0">
  <!-- Test Cases
  Grammar will support the following inputs:
  Scenario 1:
    Input: I want to transfer twenty five Dollar
    Output: $25
  Scenario 2:
    Input: transfer twenty five dollars
    Output: $25
  -->

  <rule id="main" scope="public">
    <tag>out="$"</tag>
    <one-of>
      <item><ruleref uri="#sub_hundred"><tag>out += rules.sub_hundred.sh;</tag></item>
      <item><ruleref uri="#subThousands"><tag>out += rules.subThousands;</tag></item>
    </one-of>
  </rule>
  <rule id="text">
    <item repeat="0-1"><ruleref uri="#hesitation"/></item>
    <one-of>
      <item repeat="0-1">I want to transfer</item>
      <item repeat="0-1">transfer</item>
      <item repeat="0-1">make a transfer for</item>
    </one-of>
  </rule>
  <rule id="hesitation">
    <one-of>
      <item>Hmm</item>
      <item>Mmm</item>
      <item>My</item>
    </one-of>
  </rule>
  <rule id="digits">
    <item repeat="0-1"><ruleref uri="#text"/></item>
    <tag>out.num = 0;</tag>
    <one-of>
      <item>0<tag>out.num+=0;</tag></item>
      <item>1<tag>out.num+=1;</tag></item>
      <item>2<tag>out.num+=2;</tag></item>
      <item>3<tag>out.num+=3;</tag></item>
      <item>4<tag>out.num+=4;</tag></item>
    </one-of>
  </rule>
</grammar>
```
<item>5</item><tag>out.num+=5;</tag></item>
<Item>6</Item><tag>out.num+=6;</tag></item>
<Item>7</Item><tag>out.num+=7;</tag></item>
<Item>8</Item><tag>out.num+=8;</tag></item>
<Item>9</Item><tag>out.num+=9;</tag></item>
<Item>one</Item><tag>out.num+=1;</tag></item>
<Item>two</Item><tag>out.num+=2;</tag></item>
<Item>three</Item><tag>out.num+=3;</tag></item>
<Item>four</Item><tag>out.num+=4;</tag></item>
<Item>five</Item><tag>out.num+=5;</tag></item>
<Item>six</Item><tag>out.num+=6;</tag></item>
<Item>seven</Item><tag>out.num+=7;</tag></item>
<Item>eight</Item><tag>out.num+=8;</tag></item>
<Item>nine</Item><tag>out.num+=9;</tag></item>
</one-of>
<Item repeat="0-1"><ruleref uri="#currency"></item>
</rule>

<rule id="teens">
<Item repeat="0-1"><ruleref uri="#text"></item>
<tag>out.teen = 0;</tag></item>
<one-of>
<Item>ten</Item><tag>out.teen+=10;</tag></item>
<Item>eleven</Item><tag>out.teen+=11;</tag></item>
<Item>twelve</Item><tag>out.teen+=12;</tag></item>
<Item>thirteen</Item><tag>out.teen+=13;</tag></item>
<Item>fourteen</Item><tag>out.teen+=14;</tag></item>
<Item>fifteen</Item><tag>out.teen+=15;</tag></item>
<Item>sixteen</Item><tag>out.teen+=16;</tag></item>
<Item>seventeen</Item><tag>out.teen+=17;</tag></item>
<Item>eighteen</Item><tag>out.teen+=18;</tag></item>
<Item>nineteen</Item><tag>out.teen+=19;</tag></item>
</one-of>
<Item repeat="0-1"><ruleref uri="#currency"></item>
</rule>

<rule id="above_twenty">
<Item repeat="0-1"><ruleref uri="#text"></item>
<tag>out.tens = 0;</tag></item>
<one-of>
<Item>twenty</Item><tag>out.tens+=20;</tag></item>
<Item>thirty</Item><tag>out.tens+=30;</tag></item>
<Item>forty</Item><tag>out.tens+=40;</tag></item>
<Item>fifty</Item><tag>out.tens+=50;</tag></item>
<Item>sixty</Item><tag>out.tens+=60;</tag></item>
<Item>seventy</Item><tag>out.tens+=70;</tag></item>
<Item>eighty</Item><tag>out.tens+=80;</tag></item>
<Item>ninety</Item><tag>out.tens+=90;</tag></item>
<Item>hundred</Item><tag>out.tens+=100;</tag></item>
</one-of>
<Item repeat="0-1"><ruleref uri="#currency"></item>
<Item repeat="0-1"><ruleref uri="#digits"></item><tag>out.tens += rules.digits.num;</tag></item>
</rule>

<rule id="currency">
<one-of>
<Item repeat="0-1">dollars</item>
<Item repeat="0-1">Dollars</item>
<Item repeat="0-1">dollar</item>
<Item repeat="0-1">Dollar</item>
</one-of>
</rule>

<rule id="sub_hundred">
<Item repeat="0-1"><ruleref uri="#currency"></item>
<Item repeat="0-1"><ruleref uri="#digits"></item><tag>out.tens += rules.digits.num;</tag></item>
</rule>
Industry grammars

Social Security number

```xml
<?xml version="1.0" encoding="UTF-8" ?>
<grammar xmlns="http://www.w3.org/2001/06/grammar"
     xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
     xsi:schemaLocation="http://www.w3.org/2001/06/grammar
                      http://www.w3.org/TR/speech-grammar/grammar.xsd"
     xml:lang="en-US" version="1.0"
     root="main"
     mode="voice"
     tag-format="semantics/1.0">

    <rule id="main" scope="public">
        <tag>out=""</tag>
        <ruleref uri="#digits"/><tag>out += rules.digits.numbers;</tag>
    </rule>

    <rule id="digits">
        <tag>out.numbers=""</tag>
        <one-of>
            <item>0</item><tag>out.numbers+=0;</tag>
            <item>1</item><tag>out.numbers+=1;</tag>
            <item>2</item><tag>out.numbers+=2;</tag>
            <item>3</item><tag>out.numbers+=3;</tag>
            <item>4</item><tag>out.numbers+=4;</tag>
            <item>5</item><tag>out.numbers+=5;</tag>
            <item>6</item><tag>out.numbers+=6;</tag>
            <item>7</item><tag>out.numbers+=7;</tag>
            <item>8</item><tag>out.numbers+=8;</tag>
            <item>9</item><tag>out.numbers+=9;</tag>
            <item>zero</item><tag>out.numbers+=0;</tag>
            <item>one</item><tag>out.numbers+=1;</tag>
            <item>two</item><tag>out.numbers+=2;</tag>
            <item>three</item><tag>out.numbers+=3;</tag>
            <item>four</item><tag>out.numbers+=4;</tag>
            <item>five</item><tag>out.numbers+=5;</tag>
            <item>six</item><tag>out.numbers+=6;</tag>
            <item>seven</item><tag>out.numbers+=7;</tag>
        </one-of>
    </rule>
</grammar>
```
Grammars for insurance (download)

The following grammars are supported for insurance domain: claim and policy numbers, driver’s license and license plate numbers, expiration dates, start dates and renewal dates, claim and policy amounts.

Claim ID

<?xml version="1.0" encoding="UTF-8" ?>
<grammar xmlns="http://www.w3.org/2001/06/grammar"
  xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
  xsi:schemaLocation="http://www.w3.org/2001/06/grammar
  http://www.w3.org/TR/speech-grammar/grammar.xsd"
  xml:lang="en-US" version="1.0"
  root="digits"
  mode="voice"
  tag-format="semantics/1.0">
  <!-- Test Cases
  Grammar will support the following inputs:
  Scenario 1:
  Input: My claim number is One Five Four Two
  Output: 1542
  Scenario 2:
  Input: Claim number One Five Four Four
  Output: 1544
  -->
  <rule id="digits">
    <tag>out=""</tag>
    <item><ruleref uri="#singleDigit"/></item>
    <item><ruleref uri="#text"/></item>
  </rule>
  <rule id="text">
    <item repeat="0-1"><ruleref uri="#hesitation"/></item>
    <one-of>
      <item repeat="0-1">My claim number is</item>
      <item repeat="0-1">Claim number</item>
      <item repeat="0-1">This is for claim</item>
    </one-of>
  </rule>
  <rule id="hesitation">
    <one-of>
      <item>Hmm</item>
      <item>Mmm</item>
      <item>My</item>
    </one-of>
  </rule>
  <rule id="singleDigit">
    <item repeat="0-1"><ruleref uri="#text"/></item>
  </rule>
</grammar>
Policy ID

<?xml version="1.0" encoding="UTF-8" ?>
<grammar xmlns="http://www.w3.org/2001/06/grammar"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xsi:schemaLocation="http://www.w3.org/2001/06/grammar
http://www.w3.org/TR/speech-grammar/grammar.xsd"
xml:lang="en-US" version="1.0"
root="main"
mode="voice"
tag-format="semantics/1.0">

<!-- Test Cases
Grammar will support the following inputs:

Scenario 1:
Input: My policy number is A B C 1 2 3 4
Output: ABC1234

Scenario 2:
Input: This is the policy number 1 2 3 4 A B C
Output: 1234ABC

Scenario 3:
Input: Hmm My policy number is 1 2 3 4 A B C 1
Output: 123ABC1
-->
<item repeat="0-1"><ruleref uri="#digits"/> <tag>out += rules.digits.numbers</tag></item>
</rule>

<rule id="text">
    <item repeat="0-1"><ruleref uri="#thanks"/></item>
</rule>

<rule id="hesitation">
    <one-of>
        <item>Hmm</item>
        <item>Mmm</item>
        <item>My</item>
    </one-of>
</rule>

<rule id="thanks">
    <one-of>
        <item>Thanks</item>
        <item>I think</item>
    </one-of>
</rule>

<rule id="alphanumeric" scope="public">
    <tag>out.alphanum=""</tag>
    <item><ruleref uri="#alphabets"/> <tag>out.alphanum += rules.alphabets.letters;</tag></item>
    <item repeat="0-1"><ruleref uri="#digits"/> <tag>out.alphanum += rules.digits.numbers</tag></item>
</rule>

<rule id="alphabets">
    <item repeat="0-1"><ruleref uri="#text"/></item>
    <tag>out.letters=""</tag>
    <tag>out.firstOccurence=""</tag>
    <item><ruleref uri="#digits"/> <tag>out.firstOccurence += rules.digits.numbers; out.letters += out.firstOccurence;</tag></item>
</rule>

<rule id="hesitation">
    <one-of>
        <item>A</item>
        <item>B</item>
        <item>C</item>
        <item>D</item>
        <item>E</item>
        <item>F</item>
        <item>G</item>
        <item>H</item>
        <item>I</item>
        <item>J</item>
        <item>K</item>
        <item>L</item>
        <item>M</item>
        <item>N</item>
        <item>O</item>
    </one-of>
</rule>
License plate number

<?xml version="1.0" encoding="UTF-8" ?>
<grammar xmlns="http://www.w3.org/2001/06/grammar"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xsi:schemaLocation="http://www.w3.org/2001/06/grammar
http://www.w3.org/TR/speech-grammar/grammar.xsd"
xml:lang="en-US" version="1.0"
root="main"
mode="voice"
tag-format="semantics/1.0">
<!-- Test Cases
Grammar will support the following inputs:

Scenario 1:
Input: my license plate is A B C D 1 2
Output: ABCD12

Scenario 2:
Input: license plate number A B C 1 2 3 4
Output: ABC1234

Scenario 3:
Input: my plates say A F G K 9 8 7 6 Thanks
Output: AFGK9876

-->}

<rule id="main" scope="public">
<tag>out.licenseNum=""</tag>
<item><ruleref uri="#alphabets"/><tag>out.licenseNum += rules.alphabets.letters;</tag></item>
<item repeat="0-1"><ruleref uri="#thanks"/></item>
</rule>

<rule id="text">
<item repeat="0-1"><ruleref uri="#hesitation"/></item>
<one-of>
<item repeat="0-1">my license plate is</item>
<item repeat="0-1">license plate number</item>
<item repeat="0-1">my plates say</item>
</one-of>
</rule>

<rule id="hesitation">
<one-of>
<item>Hmm</item>
<item>Mmm</item>
<item>My</item>
</one-of>
</rule>

<rule id="thanks">
<one-of>
<item>Thanks</item>
<item>I think</item>
</one-of>
</rule>

<rule id="alphabets">
<item repeat="0-1"><ruleref uri="#text"/></item>
<tag>out.letters=""</tag>
<tag>out.firstOccurence=""</tag>
<item repeat="3-4">
<one-of>
<item>A<tag>out.letters+='A';</tag></item>
<item>B<tag>out.letters+='B';</tag></item>
<item>C<tag>out.letters+='C';</tag></item>
<item>D<tag>out.letters+='D';</tag></item>
<item>E<tag>out.letters+='E';</tag></item>
<item>F<tag>out.letters+='F';</tag></item>
<item>G<tag>out.letters+='G';</tag></item>
<item>H<tag>out.letters+='H';</tag></item>
<item>I<tag>out.letters+='I';</tag></item>
<item>J<tag>out.letters+='J';</tag></item>
<item>K<tag>out.letters+='K';</tag></item>
<item>L<tag>out.letters+='L';</tag></item>
</one-of>
</rule>
Credit card expiration date

```xml
<?xml version="1.0" encoding="UTF-8" ?>
<grammar xmlns="http://www.w3.org/2001/06/grammar"
  xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
  xsi:schemaLocation="http://www.w3.org/2001/06/grammar" http://www.w3.org/TR/speech-grammar/grammar.xsd"
  xml:lang="en-US" version="1.0"
  root="dateCardExpiration"
  mode="voice"
  tag-format="semantics/1.0">
  <rule id="dateCardExpiration" scope="public">
    <tag>out=""</tag>
    <item repeat="1"><ruleref uri="#months"/>out = out + rules.months;</item>
    <item repeat="1"><ruleref uri="#year"/>out += " " + rules.year.yr;</item>
    <item repeat="0-1"><ruleref uri="#thanks"/></item>
  </rule>
</grammar>
```

Grammar will support the following inputs:
Scenario 1:
Input: My card expiration date is july eleven
Output: 07 2011

Scenario 2:
Input: My card expiration date is may twenty six
Output: 05 2026

<rule id="text">
  <item repeat="0-1"><ruleref uri="#hesitation"/></item>
  <one-of>
    <item repeat="0-1">My card expiration date is </item>
  </one-of>
</rule>

<rule id="hesitation">
  <one-of>
    <item>Hmm</item>
    <item>Mmm</item>
    <item>My</item>
  </one-of>
</rule>

<rule id="thanks">
  <one-of>
    <item>Thanks</item>
    <item>I think</item>
  </one-of>
</rule>

<rule id="months">
  <item repeat="0-1"><ruleref uri="#text"/></item>
  <one-of>
    <item>january<tag>out="01";</tag></item>
    <item>february<tag>out="02";</tag></item>
    <item>march<tag>out="03";</tag></item>
    <item>april<tag>out="04";</tag></item>
    <item>may<tag>out="05";</tag></item>
    <item>june<tag>out="06";</tag></item>
    <item>july<tag>out="07";</tag></item>
    <item>aug<tag>out="08";</tag></item>
    <item>sept<tag>out="09";</tag></item>
    <item>oct<tag>out="10";</tag></item>
    <item>nov<tag>out="11";</tag></item>
    <item>dec<tag>out="12";</tag></item>
    <item>jan<tag>out="01";</tag></item>
    <item>feb<tag>out="02";</tag></item>
    <item>aug<tag>out="08";</tag></item>
    <item>sept<tag>out="09";</tag></item>
    <item>oct<tag>out="10";</tag></item>
    <item>nov<tag>out="11";</tag></item>
    <item>dec<tag>out="12";</tag></item>
    <item>l<tag>out="01";</tag></item>
    <item>2<tag>out="02";</tag></item>
    <item>3<tag>out="03";</tag></item>
    <item>4<tag>out="04";</tag></item>
    <item>5<tag>out="05";</tag></item>
    <item>6<tag>out="06";</tag></item>
    <item>7<tag>out="07";</tag></item>
    <item>8<tag>out="08";</tag></item>
    <item>9<tag>out="09";</tag></item>
    <item>ten<tag>out="10";</tag>
    <item>eleven<tag>out="11";</tag>
  </one-of>
</rule>
<item>twelve</item><tag>out="12";</tag></item>
</one-of>
</rule>

<rule id="digits">
  <item repeat="0-1"><ruleref uri="#text"></item>
  <one-of>
    <item>0</item><tag>out=0;</tag></item>
    <item>1</item><tag>out=1;</tag></item>
    <item>2</item><tag>out=2;</tag></item>
    <item>3</item><tag>out=3;</tag></item>
    <item>4</item><tag>out=4;</tag></item>
    <item>5</item><tag>out=5;</tag></item>
    <item>6</item><tag>out=6;</tag></item>
    <item>7</item><tag>out=7;</tag></item>
    <item>8</item><tag>out=8;</tag></item>
    <item>9</item><tag>out=9;</tag></item>
    <item>one</item><tag>out=1;</tag></item>
    <item>two</item><tag>out=2;</tag></item>
    <item>three</item><tag>out=3;</tag></item>
    <item>four</item><tag>out=4;</tag></item>
    <item>five</item><tag>out=5;</tag></item>
    <item>six</item><tag>out=6;</tag></item>
    <item>seven</item><tag>out=7;</tag></item>
    <item>eight</item><tag>out=8;</tag></item>
    <item>nine</item><tag>out=9;</tag></item>
  </one-of>
</rule>

<rule id="year">
  <tag>out.yr="20"</tag>
  <one-of>
    <item><ruleref uri="#teens"></item><tag>out.yr += rules.teens;</tag></item>
    <item><ruleref uri="#above_twenty"></item><tag>out.yr += rules.above_twenty;</tag></item>
  </one-of>
</rule>

<rule id="teens">
  <item repeat="0-1"><ruleref uri="#text"></item>
  <one-of>
    <item>ten</item><tag>out=10;</tag></item>
    <item>eleven</item><tag>out=11;</tag></item>
    <item>twelve</item><tag>out=12;</tag></item>
    <item>thirteen</item><tag>out=13;</tag></item>
    <item>fourteen</item><tag>out=14;</tag></item>
    <item>fifteen</item><tag>out=15;</tag></item>
    <item>sixteen</item><tag>out=16;</tag></item>
    <item>seventeen</item><tag>out=17;</tag></item>
    <item>eighteen</item><tag>out=18;</tag></item>
    <item>nineteen</item><tag>out=19;</tag></item>
    <item>10</item><tag>out=10;</tag></item>
    <item>11</item><tag>out=11;</tag></item>
    <item>12</item><tag>out=12;</tag></item>
    <item>13</item><tag>out=13;</tag></item>
    <item>14</item><tag>out=14;</tag></item>
    <item>15</item><tag>out=15;</tag></item>
    <item>16</item><tag>out=16;</tag></item>
    <item>17</item><tag>out=17;</tag></item>
    <item>18</item><tag>out=18;</tag></item>
    <item>19</item><tag>out=19;</tag></item>
  </one-of>
</rule>

<rule id="above_twenty">
  <item repeat="0-1"><ruleref uri="#text"></item>
  <item>90</item>
</rule>
Policy expiration date, day/month/year

<?xml version="1.0" encoding="UTF-8" ?>
<grammar xmlns="http://www.w3.org/2001/06/grammar"
    xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
    xsi:schemaLocation="http://www.w3.org/2001/06/grammar
                        http://www.w3.org/TR/speech-grammar/grammar.xsd"
    xml:lang="en-US" version="1.0"
    root="main"
    mode="voice"
    tag-format="semantics/1.0">
    <!-- Test Cases
    Grammar will support the following inputs:
    Scenario 1:
        Input: My policy expired on July Five Two Thousand and Eleven
        Output: 07/5/11
    Scenario 2:
        Input: My policy will expire on July Sixteen Two Thousand and Eleven
        Output: 07/16/11
    Scenario 3:
        Input: My policy expired on July Thirty Two Thousand and Eleven
        Output: 07/30/11
    -->
    <rule id="main" scope="public">
        <tag>out=""</tag>
        <item>
            <item repeat="1"><ruleref uri="#months"/><tag>out = out + rules.months.mon + "/";</tag></item>
            <one-of>
                <item><ruleref uri="#digits"/><tag>out += rules.digits + "/";</tag></item>
                <item><ruleref uri="#teens"/><tag>out += rules.teens + "/";</tag></item>
                <item><ruleref uri="#above_twenty"/><tag>out += rules.above_twenty + "/";</tag></item>
            </one-of>
        </item>
    </rule>
</grammar>
<one-of>
    <item><ruleref uri="#thousands"/><tag>out += rules.thousands;</tag></item>
    <item repeat="0-1"><ruleref uri="#digits"/><tag>out += rules.digits;</tag></item>
    <item repeat="0-1"><ruleref uri="#teens"/><tag>out += rules.teens;</tag></item>
    <item repeat="0-1"><ruleref uri="#above_twenty"/><tag>out += rules.above_twenty;</tag></item>
</one-of>
</item>
</rule>

<rule id="text">
    <item repeat="0-1"><ruleref uri="#hesitation"/></item>
    <one-of>
        <item repeat="0-1">My policy expired on</item>
        <item repeat="0-1">My policy will expire on</item>
    </one-of>
</rule>

<rule id="hesitation">
    <one-of>
        <item>My</item>
        <item>Hmm</item>
        <item>Mmm</item>
    </one-of>
</rule>

<rule id="months">
    <tag>out.mon=""</tag>
    <item repeat="0-1"><ruleref uri="#text"/></item>
    <one-of>
        <item>january<tag>out.mon="01";</tag></item>
        <item>february<tag>out.mon="02";</tag></item>
        <item>march<tag>out.mon="03";</tag></item>
        <item>april<tag>out.mon="04";</tag></item>
        <item>may<tag>out.mon="05";</tag></item>
        <item>june<tag>out.mon="06";</tag></item>
        <item>july<tag>out.mon="07";</tag></item>
        <item>aug<tag>out.mon="08";</tag></item>
        <item>sept<tag>out.mon="09";</tag></item>
        <item>oct<tag>out.mon="10";</tag></item>
        <item>nov<tag>out.mon="11";</tag></item>
        <item>dec<tag>out.mon="12";</tag></item>
    </one-of>
</rule>

<rule id="digits">
    <one-of>
        <item>zero<tag>out=0;</tag></item>
        <item>one<tag>out=1;</tag></item>
        <item>two<tag>out=2;</tag></item>
        <item>three<tag>out=3;</tag></item>
        <item>four<tag>out=4;</tag></item>
        <item>five<tag>out=5;</tag></item>
        <item>six<tag>out=6;</tag></item>
        <item>seven<tag>out=7;</tag></item>
        <item>eight<tag>out=8;</tag></item>
    </one-of>
</rule>
<item>nine</item><tag>out=9;</tag></item>
</one-of>
</rule>

<rule id="teens">
<one-of>
[item>ten</item><tag>out=10;</tag></item>
[item>eleven</item><tag>out=11;</tag></item>
[item>twelve</item><tag>out=12;</tag></item>
[item>thirteen</item><tag>out=13;</tag></item>
[item>fourteen</item><tag>out=14;</tag></item>
[item>fifteen</item><tag>out=15;</tag></item>
[item>sixteen</item><tag>out=16;</tag></item>
[item>seventeen</item><tag>out=17;</tag></item>
[item>eighteen</item><tag>out=18;</tag></item>
[item>nineteen</item><tag>out=19;</tag></item>
</one-of>
</rule>

<rule id="thousands">
<item>two thousand</item>
<item repeat="0-1">and</item>
<item repeat="0-1"><ruleref uri="#digits"></item><tag>out = rules.digits;</tag></item>
<item repeat="0-1"><ruleref uri="#teens"></item><tag>out = rules.teens;</tag></item>
<item repeat="0-1"><ruleref uri="#above_twenty"></item><tag>out = rules.above_twenty;</tag></item>
</rule>

<rule id="above_twenty">
<one-of>
[item>twenty</item><tag>out=20;</tag></item>
[item>thirty</item><tag>out=30;</tag></item>
</one-of>
<item repeat="0-1"><ruleref uri="#digits"></item><tag>out += rules.digits;</tag></item>
</rule>
</grammar>

Policy renewal date, month/year

<?xml version="1.0" encoding="UTF-8" ?>
<grammar xmlns="http://www.w3.org/2001/06/grammar"
xmns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xsi:schemaLocation="http://www.w3.org/2001/06/grammar
http://www.w3.org/TR/speech-grammar/grammar.xsd"
xm:lang="en-US" version="1.0"
root="main"
mode="voice"
tag-format="semantics/1.0">
<!-- Test Cases
Gramn will support the following inputs:

Scenario 1:
Input: I renewed my policy on July Two Thousand and Eleven
Output: 07/11

Scenario 2:
Input: My policy will renew on July Two Thousand and Eleven
Output: 07/11
-->

<rule id="main" scope="public">
  <tag>out=""</tag>
  <item repeat="1-10">
    <item repeat="1"><ruleref uri="#months"/><tag>out = out + rules.months.mon + "/";</tag></item>
    <item><ruleref uri="#thousands"/><tag>out += rules.thousands;</tag></item>
    <item repeat="0-1"><ruleref uri="#digits"/><tag>out += rules.digits;</tag></item>
    <item repeat="0-1"><ruleref uri="#teens"/><tag>out += rules.teens;</tag></item>
    <item repeat="0-1"><ruleref uri="#above_twenty"/><tag>out += rules.above_twenty;</tag></item>
  </item>
</rule>

<rule id="text">
  <item repeat="0-1"><ruleref uri="#hesitation"/></item>
  <one-of>
    <item>My policy will renew on</item>
    <item>My policy was renewed on</item>
    <item>Renew policy on</item>
    <item>I renewed my policy on</item>
  </one-of>
</rule>

<rule id="hesitation">
  <one-of>
    <item>Hmm</item>
    <item>Mmm</item>
    <item>My</item>
  </one-of>
</rule>

<rule id="months">
  <item repeat="0-1"><ruleref uri="#text"/></item>
  <tag>out.mon=""</tag>
  <one-of>
    <item>january<tag>out.mon="01";</tag></item>
    <item>february<tag>out.mon="02";</tag></item>
    <item>march<tag>out.mon="03";</tag></item>
    <item>april<tag>out.mon="04";</tag></item>
    <item>may<tag>out.mon="05";</tag></item>
    <item>june<tag>out.mon="06";</tag></item>
    <item>aug<tag>out.mon="08";</tag></item>
    <item>sept<tag>out.mon="09";</tag></item>
    <item>oct<tag>out.mon="10";</tag></item>
    <item>nov<tag>out.mon="11";</tag></item>
    <item>dec<tag>out.mon="12";</tag>
  </one-of>
</rule>

<rule id="digits">
  <one-of>
    <item>zero<tag>out=0;</tag></item>
  </one-of>
</rule>
<grammar xmlns="http://www.w3.org/2001/06/grammar" xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance" xsi:schemaLocation="http://www.w3.org/2001/06/grammar http://www.w3.org/TR/speech-grammar/grammar.xsd" xml:lang="en-US" version="1.0" root="main" mode="voice" tag-format="semantics/1.0">

  <rule id="main">
    <one-of>
      <item>one</item>
      <item>two</item>
      <item>three</item>
      <item>four</item>
      <item>five</item>
      <item>six</item>
      <item>seven</item>
      <item>eight</item>
      <item>nine</item>
    </one-of>
  </rule>

  <rule id="teens">
    <one-of>
      <item>ten</item>
      <item>eleven</item>
      <item>twelve</item>
      <item>thirteen</item>
      <item>fourteen</item>
      <item>fifteen</item>
      <item>sixteen</item>
      <item>seventeen</item>
      <item>eighteen</item>
      <item>nineteen</item>
    </one-of>
  </rule>

  <rule id="thousands">
    <item>two thousand</item>
    <item repeat="0-1">and</item>
    <item repeat="0-1"><ruleref uri="#digits"/></item>
    <item repeat="0-1"><ruleref uri="#teens"/></item>
    <item repeat="0-1"><ruleref uri="#above_twenty"/></item>
  </rule>

  <rule id="above_twenty">
    <one-of>
      <item>twenty</item>
      <item>thirty</item>
      <item>forty</item>
      <item>fifty</item>
      <item>sixty</item>
      <item>seventy</item>
      <item>eighty</item>
      <item>ninety</item>
    </one-of>
    <item repeat="0-1"><ruleref uri="#digits"/></item>
  </rule>

</grammar>

Policy start date
Grammar will support the following inputs:

Scenario 1:
Input: I bought my policy on july twenty three
Output: 07/23

Scenario 2:
Input: My policy started on july fifteen
Output: 07/15

<rule id="main" scope="public">
<item repeat="1-10">
<item><ruleref uri="#months"/><tag>out= rules.months.mon + ";</tag></item>
<one-of>
<item><ruleref uri="#digits"/><tag>out+= rules.digits;</tag></item>
<item><ruleref uri="#teens"/><tag>out+= rules.teens;</tag></item>
<item><ruleref uri="#above_twenty"/><tag>out+= rules.above_twenty;</tag></item>
</one-of>
</item>
</rule>
<rule id="text">
<item repeat="0-1"><ruleref uri="#hesitation"/></item>
<one-of>
<item>I bought my policy on</item>
<item>I bought policy on</item>
<item>My policy started on</item>
</one-of>
</rule>
<rule id="hesitation">
<one-of>
<Item>Hmm</Item>
<Item>Mmm</Item>
<Item>My</Item>
</one-of>
</rule>
<rule id="months">
<item repeat="0-1"><ruleref uri="#text"/></item>
<tag>out.mon=""</tag>
<one-of>
<item>january<tag>out.mon+="01";</tag></item>
<item>february<tag>out.mon+="02";</tag></item>
<item>march<tag>out.mon+="03";</tag></item>
<item>april<tag>out.mon+="04";</tag></item>
<item>may<tag>out.mon+="05";</tag></item>
<item>june<tag>out.mon+="06";</tag></item>
<item>july<tag>out.mon+="07";</tag></item>
<item>august<tag>out.mon+="08";</tag></item>
<item>sept<tag>out.mon+="09";</tag></item>
<item>october<tag>out.mon+="10";</tag></item>
<item>november<tag>out.mon+="11";</tag></item>
<item>december<tag>out.mon+="12";</tag></item>
<item>jan<tag>out.mon+="01";</tag></item>
<item>feb<tag>out.mon+="02";</tag></item>
<item>aug<tag>out.mon+="08";</tag></item>
<item>sept<tag>out.mon+="09";</tag></item>
<item>oct<tag>out.mon+="10";</tag></item>
<item>nov<tag>out.mon+="11";</tag></item>
<item>dec<tag>out.mon+="12";</tag></item>
<item>jan<tag>out.mon+="01";</tag></item>
<item>feb<tag>out.mon+="02";</tag></item>
<item>aug<tag>out.mon+="08";</tag></item>
<item>sept<tag>out.mon+="09";</tag></item>
<item>oct</item><tag>out.mon+="10";</tag></item>
<item>nov</item><tag>out.mon+="11";</tag></item>
<item>dec</item><tag>out.mon+="12";</tag></item>
</one-of>
</rule>

<rule id="digits">
  <item repeat="0-1"><ruleref uri="#text"/></item>
  <one-of>
    <item>0</item><tag>out=0;</tag></item>
    <item>1</item><tag>out=1;</tag></item>
    <item>2</item><tag>out=2;</tag></item>
    <item>3</item><tag>out=3;</tag></item>
    <item>4</item><tag>out=4;</tag></item>
    <item>5</item><tag>out=5;</tag></item>
    <item>6</item><tag>out=6;</tag></item>
    <item>7</item><tag>out=7;</tag></item>
    <item>8</item><tag>out=8;</tag></item>
    <item>9</item><tag>out=9;</tag></item>
    <item>first</item><tag>out=01;</tag></item>
    <item>second</item><tag>out=02;</tag></item>
    <item>third</item><tag>out=03;</tag></item>
    <item>fourth</item><tag>out=04;</tag></item>
    <item>fifth</item><tag>out=05;</tag></item>
    <item>sixth</item><tag>out=06;</tag></item>
    <item>seventh</item><tag>out=07;</tag></item>
    <item>eighth</item><tag>out=08;</tag></item>
    <item>ninth</item><tag>out=09;</tag></item>
    <item>one</item><tag>out=1;</tag></item>
    <item>two</item><tag>out=2;</tag></item>
    <item>three</item><tag>out=3;</tag></item>
    <item>four</item><tag>out=4;</tag></item>
    <item>five</item><tag>out=5;</tag></item>
    <item>six</item><tag>out=6;</tag></item>
    <item>seven</item><tag>out=7;</tag></item>
    <item>eight</item><tag>out=8;</tag></item>
    <item>nine</item><tag>out=9;</tag></item>
  </one-of>
</rule>

<rule id="teens">
  <item repeat="0-1"><ruleref uri="#text"/></item>
  <one-of>
    <item>ten</item><tag>out=10;</tag></item>
    <item>tenth</item><tag>out=10;</tag></item>
    <item>eleven</item><tag>out=11;</tag></item>
    <item>twelve</item><tag>out=12;</tag></item>
    <item>thirteen</item><tag>out=13;</tag></item>
    <item>fourteen</item><tag>out=14;</tag></item>
    <item>fifteen</item><tag>out=15;</tag></item>
    <item>sixteen</item><tag>out=16;</tag></item>
    <item>seventeen</item><tag>out=17;</tag></item>
    <item>eighteen</item><tag>out=18;</tag></item>
    <item>nineteen</item><tag>out=19;</tag></item>
    <item>tenth</item><tag>out=10;</tag></item>
    <item>eleventh</item><tag>out=11;</tag></item>
    <item>twelveth</item><tag>out=12;</tag></item>
    <item>thirteenth</item><tag>out=13;</tag></item>
    <item>fourteenth</item><tag>out=14;</tag></item>
    <item>fifteenth</item><tag>out=15;</tag></item>
    <item>sixteenth</item><tag>out=16;</tag></item>
    <item>seventeenth</item><tag>out=17;</tag></item>
    <item>eighteenth</item><tag>out=18;</tag></item>
    <item>nineteenth</item><tag>out=19;</tag></item>
  </one-of>
</rule>
Claim amount

<?xml version="1.0" encoding="UTF-8" ?>
<grammar xmlns="http://www.w3.org/2001/06/grammar"
    xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
    xsi:schemaLocation="http://www.w3.org/2001/06/grammar
    http://www.w3.org/TR/speech-grammar/grammar.xsd"
    xml:lang="en-US" version="1.0"
    root="main"
    mode="voice"
    tag-format="semantics/1.0">
    <!-- Test Cases
    Grammar will support the following inputs:
    Scenario 1:
        Input: I want to make a claim of one hundre ten dollars
        Output: $110
    Scenario 2:
        Input: Requesting claim of Two hundred dollars
        Output: $200
    -->

    <rule id="main" scope="public">
        <tag>out="$"</tag>
        <one-of>
            <item><ruleref uri="#sub_hundred"/></item>
            <item><ruleref uri="#subThousands"/></item>
        </one-of>
        <item repeat="0-1"><ruleref uri="#thanks"/></item>
    </rule>

    <rule id="text">
        <item repeat="0-1"><ruleref uri="#hesitation"/></item>
        <one-of>
            <item repeat="0-1">I want to place a claim for</item>
            <item repeat="0-1">I want to make a claim of</item>
            <item repeat="0-1">I assess damage of</item>
            <item repeat="0-1">Requesting claim of</item>
        </one-of>
    </rule>

    <rule id="hesitation">
        <one-of>
            <item>Hmm</item>
            <item>Mmm</item>
            <item>My</item>
        </one-of>
    </rule>
</grammar>
</one-of>
</rule>

<rule id="thanks">
  <one-of>
    <item>Thanks</item>
    <item>I think</item>
  </one-of>
</rule>

<rule id="digits">
  <item repeat="0-1"><ruleref uri="#text"></item>
  <tag>out.num = 0;</tag>
  <one-of>
    <item>0<tag>out.num+=0;</tag></item>
    <item>1<tag>out.num+=1;</tag></item>
    <item>2<tag>out.num+=2;</tag></item>
    <item>3<tag>out.num+=3;</tag></item>
    <item>4<tag>out.num+=4;</tag></item>
    <item>5<tag>out.num+=5;</tag></item>
    <item>6<tag>out.num+=6;</tag></item>
    <item>7<tag>out.num+=7;</tag></item>
    <item>8<tag>out.num+=8;</tag></item>
    <item>9<tag>out.num+=9;</tag></item>
    <item>one<tag>out.num+=1;</tag></item>
    <item>two<tag>out.num+=2;</tag></item>
    <item>three<tag>out.num+=3;</tag></item>
    <item>four<tag>out.num+=4;</tag></item>
    <item>five<tag>out.num+=5;</tag></item>
    <item>six<tag>out.num+=6;</tag></item>
    <item>seven<tag>out.num+=7;</tag></item>
    <item>eight<tag>out.num+=8;</tag></item>
    <item>nine<tag>out.num+=9;</tag></item>
  </one-of>
  <item repeat="0-1"><ruleref uri="#currency"></item>
</rule>

<rule id="teens">
  <item repeat="0-1"><ruleref uri="#text"></item>
  <tag>out.teen = 0;</tag>
  <one-of>
    <item>ten<tag>out.teen+=10;</tag></item>
    <item>eleven<tag>out.teen+=11;</tag></item>
    <item>twelve<tag>out.teen+=12;</tag></item>
    <item>thirteen<tag>out.teen+=13;</tag></item>
    <item>fourteen<tag>out.teen+=14;</tag></item>
    <item>fifteen<tag>out.teen+=15;</tag></item>
    <item>sixteen<tag>out.teen+=16;</tag></item>
    <item>seventeen<tag>out.teen+=17;</tag></item>
    <item>eighteen<tag>out.teen+=18;</tag></item>
    <item>nineteen<tag>out.teen+=19;</tag></item>
  </one-of>
  <item repeat="0-1"><ruleref uri="#currency"></item>
</rule>

<rule id="above_twenty">
  <item repeat="0-1"><ruleref uri="#text"></item>
  <tag>out.tens = 0;</tag>
  <one-of>
    <item>twenty<tag>out.tens+=20;</tag></item>
    <item>thirty<tag>out.tens+=30;</tag></item>
    <item>forty<tag>out.tens+=40;</tag></item>
    <item>fifty<tag>out.tens+=50;</tag></item>
    <item>sixty<tag>out.tens+=60;</tag></item>
    <item>seventy<tag>out.tens+=70;</tag></item>
    <item>eighty<tag>out.tens+=80;</tag></item>
  </one-of>
</rule>
<item>ninety</item><tag>out.tens+=90;</tag></item>

<hundred</hundred><tag>out.tens+=100;</tag></item>
</one-of>

=item repeat="0-1"><ruleref uri="#currency"></item>
</tag></item>
</rule>

<rule id="currency">
<one-of>
=item repeat="0-1">dollars</item>
  <item repeat="0-1">Dollars</item>
  <item repeat="0-1">dollar</item>
  <item repeat="0-1">Dollar</item>
</one-of>
</rule>

<rule id="sub_hundred">
=item repeat="0-1"><ruleref uri="#text"></item>
  <tag>out.sh = 0;</tag>
  <one-of>
    <item><ruleref uri="#teens"></item>
      <tag>out.sh += rules.teens.teen;</tag></item>
    <item>
      <ruleref uri="#above_twenty"></item>
      <tag>out.sh += rules.above_twenty.tens;</tag>
    </item>
    <item><ruleref uri="#digits"></item>
      <tag>out.sh += rules.digits.num;</tag></item>
  </one-of>
</rule>

<rule id="subThousands">
<ruleref uri="#sub_hundred"></rule>
  <tag>out = (100 * rules.sub_hundred.sh);</tag>
  <hundred
    <item repeat="0-1"><ruleref uri="#above_twenty"></item>
      <tag>out += rules.above_twenty.tens;</tag></item>
    <item repeat="0-1"><ruleref uri="#teens"></item>
      <tag>out += rules.teens.teen;</tag></item>
    <item repeat="0-1"><ruleref uri="#digits"></item>
      <tag>out += rules.digits.num;</tag></item>
    <item repeat="0-1"><ruleref uri="#currency"></item>
</rule>
</grammar>

Premium amount

<?xml version="1.0" encoding="UTF-8" ?>
<grammar xmlns="http://www.w3.org/2001/06/grammar"
  xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
  xsi:schemaLocation="http://www.w3.org/2001/06/grammar
  http://www.w3.org/TR/speech-grammar/grammar.xsd"
  xml:lang="en-US" version="1.0"
  root="main"
  mode="voice"
  tag-format="semantics/1.0">
  <!-- Test Cases
  Grammar will support the following inputs:
  Premium amounts
_scenario 1:
  Input: The premium for one hundred ten dollars
  Output: $110
  -->
</grammar>
Scenario 2:
Input: RPremium amount of Two hundred dollars
Output: $200

-->\

<rule id="main" scope="public">
  <tag>out="#]]></tag>
  <one-of>
    <item><ruleref uri="#sub_hundred"/> <tag>out += rules.sub_hundred.sh;</tag></item>
    <item><ruleref uri="#subThousands"/> <tag>out += rules.subThousands;</tag></item>
  </one-of>
  <item repeat="0-1"><ruleref uri="#thanks"></item>
</rule>

<rule id="text">
  <item repeat="0-1"><ruleref uri="#hesitation"></item>
  <one-of>
    <item> A premium of</item>
    <item> Premium amount of</item>
    <item> The premium for</item>
    <item> Insurance premium for</item>
  </one-of>
</rule>

<rule id="hesitation">
  <one-of>
    <item> Hmm</item>
    <item> Mmm</item>
    <item> My</item>
  </one-of>
</rule>

<rule id="thanks">
  <one-of>
    <item> Thanks</item>
    <item> I think</item>
  </one-of>
</rule>

<rule id="digits">
  <item repeat="0-1"><ruleref uri="#text"></item>
  <tag>out.num = 0;</tag>
  <one-of>
    <item>0</item>
    <item>1</item>
    <item>2</item>
    <item>3</item>
    <item>4</item>
    <item>5</item>
    <item>6</item>
    <item>7</item>
    <item>8</item>
    <item>9</item>
    <item>one</item>
    <item>two</item>
    <item>three</item>
    <item>four</item>
    <item>five</item>
    <item>six</item>
    <item>seven</item>
    <item>eight</item>
    <item>nine</item>
  </one-of>
</rule>
<rule id="teens">
  <item repeat="0-1"><ruleref uri="#text"/></item>
  <tag>out.teen = 0;</tag>
  <one-of>
    <item>ten<tag>out.teen+=10;</tag></item>
    <item>eleven<tag>out.teen+=11;</tag></item>
    <item>twelve<tag>out.teen+=12;</tag></item>
    <item>thirteen<tag>out.teen+=13;</tag></item>
    <item>fourteen<tag>out.teen+=14;</tag></item>
    <item>fifteen<tag>out.teen+=15;</tag></item>
    <item>sixteen<tag>out.teen+=16;</tag></item>
    <item>seventeen<tag>out.teen+=17;</tag></item>
    <item>eighteen<tag>out.teen+=18;</tag></item>
    <item>nineteen<tag>out.teen+=19;</tag></item>
  </one-of>
  <item repeat="0-1"><ruleref uri="#currency"/></item>
</rule>

<rule id="above_twenty">
  <item repeat="0-1"><ruleref uri="#text"/></item>
  <tag>out.tens = 0;</tag>
  <one-of>
    <item>twenty<tag>out.tens+=20;</tag></item>
    <item>thirty<tag>out.tens+=30;</tag></item>
    <item>forty<tag>out.tens+=40;</tag></item>
    <item>fifty<tag>out.tens+=50;</tag></item>
    <item>sixty<tag>out.tens+=60;</tag></item>
    <item>seventy<tag>out.tens+=70;</tag></item>
    <item>eighty<tag>out.tens+=80;</tag></item>
    <item>ninety<tag>out.tens+=90;</tag></item>
    <item>hundred<tag>out.tens+=100;</tag></item>
  </one-of>
  <item repeat="0-1"><ruleref uri="#currency"/></item>
  <item repeat="0-1"><ruleref uri="#digits"/></item>
  <tag>out.tens += rules.digits.num;</tag>
</rule>

<rule id="currency">
  <one-of>
    <item repeat="0-1">dollars</item>
    <item repeat="0-1">Dollars</item>
    <item repeat="0-1">dollar</item>
    <item repeat="0-1">Dollar</item>
  </one-of>
</rule>

<rule id="sub_hundred">
  <item repeat="0-1"><ruleref uri="#text"/></item>
  <tag>out.sh = 0;</tag>
  <one-of>
    <item><ruleref uri="#teens" /></item>
    <item><ruleref uri="#above_twenty" /></item>
    <item><ruleref uri="#digits" /></item>
  </one-of>
</rule>

<rule id="subThousands">
  <item repeat="0-1"><ruleref uri="#sub_hundred" /></item>
  <tag>out = (100 * rules.sub_hundred.sh);</tag>
</rule>
hundred
</item
><ruleref uri="#above_twenty"/>
<tag>out += rules.above_twenty.tens;</tag></item>
</item>
</item>
</rule>
</grammar>

Policy quantity

<?xml version="1.0" encoding="UTF-8" ?>
<grammar xmlns="http://www.w3.org/2001/06/grammar"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xsi:schemaLocation="http://www.w3.org/2001/06/grammar
http://www.w3.org/TR/speech-grammar/grammar.xsd"
xml:lang="en-US" version="1.0"
root="main"
mode="voice"
tag-format="semantics/1.0">
<!-- Test Cases
Grammar will support the following inputs:
Scenario 1:
Input: The number is one
Output: 1
Scenario 2:
Input: I want policy for ten
Output: 10
-->  
<rule id="main" scope="public">
<tag>out=""</tag>
<one-of>
=item repeat="1"><ruleref uri="#digits"/>
<tag>out += rules.digits;</tag></item>
=item repeat="1"><ruleref uri="#teens"/>
<tag>out += rules.teens;</tag></item>
=item repeat="0-1"><ruleref uri="#above_twenty"/>
<tag>out += rules.above_twenty;</tag></item>
</one-of>
</rule>

<rule id="text">
<one-of>
=item repeat="0-1">I want policy for</item>
=item repeat="0-1">I want to order policy for</item>
=item repeat="0-1">The number is</item>
</one-of>
</rule>

<rule id="thanks">
<one-of>
=item>Thanks</item>
=item>I think</item>
</one-of>
</rule>
<rule id="digits">
  <item repeat="0-1"><ruleref uri="#text"/></item>
  <one-of>
    <item><tag>0</tag><ruleref uri="#text"/></item>
    <item><tag>1</tag><ruleref uri="#text"/></item>
    <item><tag>2</tag><ruleref uri="#text"/></item>
    <item><tag>3</tag><ruleref uri="#text"/></item>
    <item><tag>4</tag><ruleref uri="#text"/></item>
    <item><tag>5</tag><ruleref uri="#text"/></item>
    <item><tag>6</tag><ruleref uri="#text"/></item>
    <item><tag>7</tag><ruleref uri="#text"/></item>
    <item><tag>8</tag><ruleref uri="#text"/></item>
    <item><tag>9</tag><ruleref uri="#text"/></item>
    <item><tag>one</tag><ruleref uri="#text"/></item>
    <item><tag>two</tag><ruleref uri="#text"/></item>
    <item><tag>three</tag><ruleref uri="#text"/></item>
    <item><tag>four</tag><ruleref uri="#text"/></item>
    <item><tag>five</tag><ruleref uri="#text"/></item>
    <item><tag>six</tag><ruleref uri="#text"/></item>
    <item><tag>seven</tag><ruleref uri="#text"/></item>
    <item><tag>eight</tag><ruleref uri="#text"/></item>
    <item><tag>nine</tag><ruleref uri="#text"/></item>
  </one-of>
</rule>

<rule id="teens">
  <item repeat="0-1"><ruleref uri="#text"/></item>
  <one-of>
    <item><tag>ten</tag><ruleref uri="#text"/></item>
    <item><tag>eleven</tag><ruleref uri="#text"/></item>
    <item><tag>twelve</tag><ruleref uri="#text"/></item>
    <item><tag>thirteen</tag><ruleref uri="#text"/></item>
    <item><tag>fourteen</tag><ruleref uri="#text"/></item>
    <item><tag>fifteen</tag><ruleref uri="#text"/></item>
    <item><tag>sixteen</tag><ruleref uri="#text"/></item>
    <item><tag>seventeen</tag><ruleref uri="#text"/></item>
    <item><tag>eighteen</tag><ruleref uri="#text"/></item>
    <item><tag>nineteen</tag><ruleref uri="#text"/></item>
    <item><tag>10</tag><ruleref uri="#text"/></item>
    <item><tag>11</tag><ruleref uri="#text"/></item>
    <item><tag>12</tag><ruleref uri="#text"/></item>
    <item><tag>13</tag><ruleref uri="#text"/></item>
    <item><tag>14</tag><ruleref uri="#text"/></item>
    <item><tag>15</tag><ruleref uri="#text"/></item>
    <item><tag>16</tag><ruleref uri="#text"/></item>
    <item><tag>17</tag><ruleref uri="#text"/></item>
    <item><tag>18</tag><ruleref uri="#text"/></item>
    <item><tag>19</tag><ruleref uri="#text"/></item>
  </one-of>
</rule>

<rule id="above_twenty">
  <item repeat="0-1"><ruleref uri="#text"/></item>
  <one-of>
    <item><tag>twenty</tag><ruleref uri="#text"/></item>
    <item><tag>thirty</tag><ruleref uri="#text"/></item>
    <item><tag>forty</tag><ruleref uri="#text"/></item>
    <item><tag>fifty</tag><ruleref uri="#text"/></item>
    <item><tag>sixty</tag><ruleref uri="#text"/></item>
    <item><tag>seventy</tag><ruleref uri="#text"/></item>
    <item><tag>eighty</tag><ruleref uri="#text"/></item>
    <item><tag>ninety</tag><ruleref uri="#text"/></item>
    <item><tag>20</tag><ruleref uri="#text"/></item>
    <item><tag>30</tag><ruleref uri="#text"/></item>
    <item><tag>40</tag><ruleref uri="#text"/></item>
    <item><tag>50</tag><ruleref uri="#text"/></item>
    <item><tag>60</tag><ruleref uri="#text"/></item>
    <item><tag>70</tag><ruleref uri="#text"/></item>
    <item><tag>80</tag><ruleref uri="#text"/></item>
    <item><tag>90</tag><ruleref uri="#text"/></item>
    <item><tag>20</tag><ruleref uri="#text"/></item>
    <item><tag>30</tag><ruleref uri="#text"/></item>
    <item><tag>40</tag><ruleref uri="#text"/></item>
    <item><tag>50</tag><ruleref uri="#text"/></item>
  </one-of>
</rule>
Grammars for telecom (download)

The following grammars are supported for telecom: Phone number, serial number, SIM number, US Zip code, credit card expiration date, plan start, renewal and expiration dates, service start date, equipment quantity and bill amount.

Phone number

```xml
<?xml version="1.0" encoding="UTF-8" ?>
<grammar xmlns="http://www.w3.org/2001/06/grammar"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xsi:schemaLocation="http://www.w3.org/2001/06/grammar
http://www.w3.org/TR/speech-grammar/grammar.xsd"
xml:lang="en-US" version="1.0"
root="digits"
mode="voice"
tag-format="semantics/1.0">
<!-- Test Cases
Grammar will support 10-12 digits number and here are couple of examples of valid inputs:

Scenario 1:
Input: Mmm My phone number is two zero one two five two six seven eight five
Output: 2012526785

Scenario 2:
Input: My phone number is two zero one two five two six seven eight five
Output: 2012526785
-->

<rule id="digits">
<tag>out=""</tag>
=item><ruleref uri="#singleDigit"/></item><tag>out += rules.singleDigit.digit;</tag></item>
</rule>

<rule id="text">
=item repeat="0-1"><ruleref uri="#hesitation"/></item>
<one-of>
=item repeat="0-1">My phone number is</item>
=item repeat="0-1">Phone number is</item>
=item repeat="0-1">It is</item>
=item repeat="0-1">Yes, it's</item>
=item repeat="0-1">Yes, it is</item>
=item repeat="0-1">Yes it is</item>
</one-of>
</rule>
</grammar>
```
<rule id="hesitation">
  <one-of>
    <item>Hmm</item>
    <item>Mmm</item>
    <item>My</item>
  </one-of>
</rule>

<rule id="singleDigit">
  <item repeat="0-1"><ruleref uri="#text"/></item>
  <tag>out.digit=""</tag>
  <item repeat="10-12">
    <one-of>
      <item>0</item><tag>out.digit+=0;</tag></item>
      <item>zero</item><tag>out.digit+=0;</tag></item>
      <item>1</item><tag>out.digit+=1;</tag></item>
      <item>one</item><tag>out.digit+=1;</tag></item>
      <item>2</item><tag>out.digit+=2;</tag></item>
      <item>two</item><tag>out.digit+=2;</tag></item>
      <item>3</item><tag>out.digit+=3;</tag></item>
      <item>three</item><tag>out.digit+=3;</tag></item>
      <item>4</item><tag>out.digit+=4;</tag></item>
      <item>four</item><tag>out.digit+=4;</tag></item>
      <item>5</item><tag>out.digit+=5;</tag></item>
      <item>five</item><tag>out.digit+=5;</tag></item>
      <item>6</item><tag>out.digit+=6;</tag></item>
      <item>six</item><tag>out.digit+=6;</tag></item>
      <item>7</item><tag>out.digit+=7;</tag></item>
      <item>seven</item><tag>out.digit+=7;</tag></item>
      <item>8</item><tag>out.digit+=8;</tag></item>
      <item>eight</item><tag>out.digit+=8;</tag></item>
      <item>9</item><tag>out.digit+=9;</tag></item>
      <item>nine</item><tag>out.digit+=9;</tag></item>
    </one-of>
  </item>
</rule>

<grammar xmlns="http://www.w3.org/2001/06/grammar" xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance" xsi:schemaLocation="http://www.w3.org/2001/06/grammar http://www.w3.org/TR/speech-grammar/grammar.xsd" xml:lang="en-US" version="1.0" root="digits" mode="voice" tag-format="semantics/1.0">
  <!-- Test Cases
  Grammar will support the following inputs:
  Scenario 1:
  Input: My serial number is 1 2 3 4 5 6 7 8 9 1 2 3 4 5 6
  Output: 123456789123456
  Scenario 2:
  Input: Device Serial number 1 2 3 4 5 6 7 8 9 1 2 3 4 5 6
  Output: 123456789123456
  -->
</grammar>

  <rule id="digits">
    <tag>out=""</tag>
    <item><ruleref uri="#singleDigit"/><tag>out += rules.singleDigit.digit;</tag></item>
  </rule>

  <rule id="text">
    <item repeat="0-1"><ruleref uri="#hesitation"/></item>
    <one-of>
      <item repeat="0-1">My serial number is</item>
      <item repeat="0-1">Device Serial number</item>
      <item repeat="0-1">The number is</item>
      <item repeat="0-1">The IMEI number is</item>
    </one-of>
  </rule>

  <rule id="hesitation">
    <one-of>
      <item>Hmm</item>
      <item>Mmm</item>
      <item>My</item>
    </one-of>
  </rule>

  <rule id="singleDigit">
    <item repeat="0-1"><ruleref uri="#text"/></item>
    <tag>out.digit=""</tag>
    <item repeat="15">
      <one-of>
        <item>0</item>
        <item>zero</item>
        <item>1</item>
        <item>one</item>
        <item>2</item>
        <item>two</item>
        <item>3</item>
        <item>three</item>
        <item>4</item>
        <item>four</item>
        <item>5</item>
        <item>five</item>
        <item>6</item>
        <item>six</item>
        <item>7</item>
        <item>seven</item>
        <item>8</item>
        <item>eight</item>
        <item>9</item>
        <item>nine</item>
      </one-of>
    </item>
  </rule>

</grammar>

SIM number
Tag-format="semantics/1.0">

<!-- Test Cases
Grammar will support the following inputs:

Scenario 1:
Input: My SIM number is A B C 1 2 3 4
Output: ABC1234

Scenario 2:
Input: My SIM number is 1 2 3 4 A B C
Output: 1234ABC

Scenario 3:
Input: My SIM number is 1 2 3 4 A B C 1
Output: 123ABC1
-->

<rule id="main" scope="public">
  <tag>out=""</tag>
  <item><ruleref uri="#alphanumeric"/><tag>out += rules.alphanumeric.alphanum;</tag></item>
  <item repeat="0-1"><ruleref uri="#alphabets"/><tag>out += rules.alphabets.letters;</tag></item>
  <item repeat="0-1"><ruleref uri="#digits"/><tag>out += rules.digits.numbers</tag></item>
</rule>

<rule id="text">
  <item repeat="0-1"><ruleref uri="#hesitation"/></item>
  <one-of>
    <item repeat="0-1">My SIM number is</item>
    <item repeat="0-1">SIM number is</item>
  </one-of>
</rule>

<rule id="hesitation">
  <one-of>
    <item>Hmm</item>
    <item>Mmm</item>
    <item>My</item>
  </one-of>
</rule>

<rule id="alphanumeric" scope="public">
  <tag>out.alphanum=""</tag>
  <item><ruleref uri="#alphabets"/><tag>out.alphanum += rules.alphabets.letters;</tag></item>
  <item repeat="0-1"><ruleref uri="#digits"/><tag>out.alphanum += rules.digits.numbers</tag></item>
</rule>

<rule id="alphabets">
  <item repeat="0-1"><ruleref uri="#text"/></item>
  <tag>out.letters=""</tag>
  <tag>out.firstOccurence=""</tag>
  <item repeat="0-1"><ruleref uri="#digits"/><tag>out.firstOccurence += rules.digits.numbers; out.letters += out.firstOccurence;</tag></item>
  <item repeat="1-"/>
  <one-of>
    <item>A<tag>out.letters+='A';</tag></item>
    <item>B<tag>out.letters+='B';</tag></item>
    <item>C<tag>out.letters+='C';</tag></item>
    <item>D<tag>out.letters+='D';</tag></item>
  </one-of>
</rule>
US Zip code

<?xml version="1.0" encoding="UTF-8" ?>
  xml:lang="en-US" version="1.0"
  root="digits"
  mode="voice"
  tag-format="semantics/1.0">
  <!-- Test Cases
  Grammar will support 5 digits code and here are couple of examples of valid inputs:

  Scenario 1:
  -->
</grammar>
Input: Mmmm My zipcode is umm One Oh Nine Eight Seven
Output: 10987

Scenario 2:
Input: My zipcode is One Oh Nine Eight Seven
Output: 10987

-->  

<rule id="digits">
  <tag>out=""</tag>
  <item><ruleref uri="#singleDigit"/><tag>out += rules.singleDigit.digit;</tag></item>
</rule>

<rule id="text">
  <item repeat="0-1"><ruleref uri="#hesitation"/></item>
  <one-of>
    <item repeat="0-1">My zipcode is</item>
    <item repeat="0-1">Zipcode is</item>
    <item repeat="0-1">It is</item>
  </one-of>
</rule>

<rule id="hesitation">
  <one-of>
    <item>Hmm</item>
    <item>Mmm</item>
    <item>My</item>
  </one-of>
</rule>

<rule id="singleDigit">
  <item repeat="0-1"><ruleref uri="#text"/></item>
  <tag>out.digit=""</tag>
  <item repeat="5">
    <one-of>
      <item>0<tag>out.digit+=0;</tag></item>
      <item>zero<tag>out.digit+=0;</tag></item>
      <item>Oh<tag>out.digit+=0;</tag></item>
      <item>1<tag>out.digit+=1;</tag></item>
      <item>one<tag>out.digit+=1;</tag></item>
      <item>2<tag>out.digit+=2;</tag></item>
      <item>two<tag>out.digit+=2;</tag></item>
      <item>3<tag>out.digit+=3;</tag></item>
      <item>three<tag>out.digit+=3;</tag></item>
      <item>4<tag>out.digit+=4;</tag></item>
      <item>four<tag>out.digit+=4;</tag></item>
      <item>5<tag>out.digit+=5;</tag></item>
      <item>five<tag>out.digit+=5;</tag></item>
      <item>6<tag>out.digit+=6;</tag></item>
      <item>six<tag>out.digit+=6;</tag></item>
      <item>7<tag>out.digit+=7;</tag></item>
      <item>seven<tag>out.digit+=7;</tag></item>
      <item>8<tag>out.digit+=8;</tag></item>
      <item>eight<tag>out.digit+=8;</tag></item>
      <item>9<tag>out.digit+=9;</tag></item>
      <item>nine<tag>out.digit+=9;</tag>
    </one-of>
  </item>
</rule>
Credit card expiration date

```xml
<?xml version="1.0" encoding="UTF-8" ?>
<grammar xmlns="http://www.w3.org/2001/06/grammar"
xmni:xsi="http://www.w3.org/2001/XMLSchema-instance"
xsi:schemaLocation="http://www.w3.org/2001/06/grammar
http://www.w3.org/TR/speech-grammar/grammar.xsd"
xmni:lang="en-US" version="1.0"
root="dateCardExpiration"
mode="voice"
tag-format="semantics/1.0">
  <rule id="dateCardExpiration" scope="public">
    <tag>out=""</tag>
    <item repeat="1"><ruleref uri="#months"/><tag>out = out + rules.months;</tag></item>
    <item repeat="1"><ruleref uri="#year"/><tag>out += " " + rules.year.yr;</tag></item>
  </rule>

  <!-- Test Cases
  Grammar will support the following inputs:
  
  Scenario 1:
  Input: My card expiration date is july eleven
  Output: 07 2011
  
  Scenario 2:
  Input: My card expiration date is may twenty six
  Output: 05 2026
  -->

  <rule id="text">
    <item repeat="0-1"><ruleref uri="#hesitation"/></item>
    <one-of>
      <item>My card expiration date is </item>
    </one-of>
  </rule>

  <rule id="hesitation">
    <one-of>
      <item>Hmm</item>
      <item>Mmm</item>
      <item>My</item>
    </one-of>
  </rule>

  <rule id="months">
    <item repeat="0-1"><ruleref uri="#text"/></item>
    <one-of>
      <item>january</item>
      <item>february</item>
      <item>march</item>
      <item>april</item>
      <item>may</item>
      <item>june</item>
      <item>august</item>
      <item>september</item>
      <item>october</item>
      <item>november</item>
      <item>december</item>
      <item>jan</item>
      <item>feb</item>
      <item>mar</item>
      <item>apr</item>
      <item>may</item>
      <item>june</item>
      <item>aug</item>
      <item>sept</item>
      <item>oct</item>
      <item>nov</item>
      <item>dec</item>
      <item>jan</item>
      <item>feb</item>
      <item>mar</item>
      <item>apr</item>
      <item>may</item>
      <item>june</item>
      <item>aug</item>
      <item>sept</item>
      <item>oct</item>
      <item>nov</item>
      <item>dec</item>
      <item>jan</item>
      <item>feb</item>
      <item>mar</item>
      <item>apr</item>
      <item>may</item>
      <item>june</item>
      <item>aug</item>
      <item>sept</item>
      <item>oct</item>
      <item>nov</item>
      <item>dec</item>
      <item>jan</item>
      <item>feb</item>
      <item>mar</item>
      <item>apr</item>
      <item>may</item>
      <item>june</item>
      <item>aug</item>
      <item>sept</item>
      <item>oct</item>
      <item>nov</item>
      <item>dec</item>
      <item>jan</item>
      <item>feb</item>
      <item>mar</item>
      <item>apr</item>
      <item>may</item>
      <item>june</item>
      <item>aug</item>
      <item>sept</item>
      <item>oct</item>
      <item>nov</item>
      <item>dec</item>
      <item>jan</item>
      <item>feb</item>
      <item>mar</item>
      <item>apr</item>
      <item>may</item>
      <item>june</item>
      <item>aug</item>
      <item>sept</item>
      <item>oct</item>
      <item>nov</item>
      <item>dec</item>
      <item>jan</item>
      <item>feb</item>
      
    </one-of>
  </rule>
</grammar>
```
<item>feb</item><tag>out="02";</tag></item>
<item>aug</item><tag>out="08";</tag></item>
<item>sept</item><tag>out="09";</tag></item>
<item>oct</item><tag>out="10";</tag></item>
<item>nov</item><tag>out="11";</tag></item>
<item>dec</item><tag>out="12";</tag></item>
<item>1</item><tag>out="01";</tag></item>
<item>2</item><tag>out="02";</tag></item>
<item>3</item><tag>out="03";</tag></item>
<item>4</item><tag>out="04";</tag></item>
<item>5</item><tag>out="05";</tag></item>
<item>6</item><tag>out="06";</tag></item>
<item>7</item><tag>out="07";</tag></item>
<item>8</item><tag>out="08";</tag></item>
<item>9</item><tag>out="09";</tag></item>
<item>ten</item><tag>out="10";</tag></item>
<item>eleven</item><tag>out="11";</tag></item>
<item>twelve</item><tag>out="12";</tag></item>
</one-of>
</rule>

<rule id="digits">
  <item repeat="0-1"><ruleref uri="#text"></item>
  <one-of>
    <item>0</item><tag>out=0;</tag></item>
    <item>1</item><tag>out=1;</tag></item>
    <item>2</item><tag>out=2;</tag></item>
    <item>3</item><tag>out=3;</tag></item>
    <item>4</item><tag>out=4;</tag></item>
    <item>5</item><tag>out=5;</tag></item>
    <item>6</item><tag>out=6;</tag></item>
    <item>7</item><tag>out=7;</tag></item>
    <item>8</item><tag>out=8;</tag></item>
    <item>9</item><tag>out=9;</tag></item>
    <item>one</item><tag>out=1;</tag></item>
    <item>two</item><tag>out=2;</tag></item>
    <item>three</item><tag>out=3;</tag></item>
    <item>four</item><tag>out=4;</tag></item>
    <item>five</item><tag>out=5;</tag></item>
    <item>six</item><tag>out=6;</tag></item>
    <item>seven</item><tag>out=7;</tag></item>
    <item>eight</item><tag>out=8;</tag></item>
    <item>nine</item><tag>out=9;</tag></item>
  </one-of>
</rule>

<rule id="year">
  <tag>out.yr="20"</tag>
  <one-of>
    <item><ruleref uri="#teens"/><tag>out.yr += rules.teens;</tag></item>
    <item><ruleref uri="#above_twenty"/><tag>out.yr += rules.above_twenty;</tag></item>
  </one-of>
</rule>

<rule id="teens">
  <item repeat="0-1"><ruleref uri="#text"></item>
  <one-of>
    <item>ten</item><tag>out=10;</tag></item>
    <item>eleven</item><tag>out=11;</tag></item>
    <item>twelve</item><tag>out=12;</tag></item>
    <item>thirteen</item><tag>out=13;</tag></item>
    <item>fourteen</item><tag>out=14;</tag></item>
    <item>fifteen</item><tag>out=15;</tag></item>
    <item>sixteen</item><tag>out=16;</tag></item>
    <item>seventeen</item><tag>out=17;</tag></item>
  </one-of>
</rule>
<item>eighteen</item><tag>out=18;</tag></item>
<item>nineteen</item><tag>out=19;</tag></item>
<item>10</item><tag>out=10;</tag></item>
<item>11</item><tag>out=11;</tag></item>
<item>12</item><tag>out=12;</tag></item>
<item>13</item><tag>out=13;</tag></item>
<item>14</item><tag>out=14;</tag></item>
<item>15</item><tag>out=15;</tag></item>
<item>16</item><tag>out=16;</tag></item>
<item>17</item><tag>out=17;</tag></item>
<item>18</item><tag>out=18;</tag></item>
<item>19</item><tag>out=19;</tag></item>
</one-of>
</rule>

<rule id="above_twenty">
  <item repeat="0-1"><ruleref uri="#text"/></item>
  <one-of>
    <item>twenty</item><tag>out=20;</tag></item>
    <item>thirty</item><tag>out=30;</tag></item>
    <item>forty</item><tag>out=40;</tag></item>
    <item>fifty</item><tag>out=50;</tag></item>
    <item>sixty</item><tag>out=60;</tag></item>
    <item>seventy</item><tag>out=70;</tag></item>
    <item>eighty</item><tag>out=80;</tag></item>
    <item>ninety</item><tag>out=90;</tag></item>
    <item>20</item><tag>out=20;</tag></item>
    <item>30</item><tag>out=30;</tag></item>
    <item>40</item><tag>out=40;</tag></item>
    <item>50</item><tag>out=50;</tag></item>
    <item>60</item><tag>out=60;</tag></item>
    <item>70</item><tag>out=70;</tag></item>
    <item>80</item><tag>out=80;</tag></item>
    <item>90</item><tag>out=90;</tag></item>
  </one-of>
  <item repeat="0-1"><ruleref uri="#digits"/><tag>out += rules.digits;</tag></item>
</rule>

</grammar>

Plan expiration date, day/month/year

<?xml version="1.0" encoding="UTF-8" ?>
<grammar xmlns="http://www.w3.org/2001/06/grammar"
  xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
  xsi:schemaLocation="http://www.w3.org/2001/06/grammar
  http://www.w3.org/TR/speech-grammar/grammar.xsd"
  xml:lang="en-US" version="1.0"
  root="main"
  mode="voice"
  tag-format="semantics/1.0">
  <!-- Test Cases
  Grammar will support the following inputs:
  -- Scenario 1: 
  Input: My plan expires on July Five Two Thousand and Eleven 
  Output: 07/5/11
  -- Scenario 2: 
  Input: My plan will expire on July Sixteen Two Thousand and Eleven 
  Output: 07/16/11

<item repeat="0-1"><ruleref uri="#digits"/><tag>out += rules.digits;</tag></item>
</grammar>
Scenario 3:
Input: My plan will expire on July Thirty Two Thousand and Eleven
Output: 07/30/11

```
<rule id="main" scope="public">
  <tag>out=""</tag>
  <item>
    <item repeat="1"><ruleref uri="#months"/><tag>out = out + rules.months.mon + "/";</tag></item>
    <one-of>
      <item><ruleref uri="#digits"/></item>
      <item><ruleref uri="#teens"/></item>
      <item><ruleref uri="#above_twenty"/></item>
    </one-of>
  </item>
  <one-of>
    <item><ruleref uri="#thousands"/></item>
    <item repeat="0-1"><ruleref uri="#digits"/></item>
    <item repeat="0-1"><ruleref uri="#teens"/></item>
    <item repeat="0-1"><ruleref uri="#above_twenty"/></item>
  </one-of>
</item>
</rule>

<rule id="text">
  <item repeat="0-1"><ruleref uri="#hesitation"/></item>
  <one-of>
    <item repeat="0-1">My plan expires on</item>
    <item repeat="0-1">My plan expired on</item>
    <item repeat="0-1">My plan will expire on</item>
  </one-of>
</rule>

<rule id="hesitation">
  <one-of>
    <item>Hmm</item>
    <item>Mmm</item>
    <item>My</item>
  </one-of>
</rule>

<rule id="months">
  <tag>out.mon=""</tag>
  <item repeat="0-1"><ruleref uri="#text"/></item>
  <one-of>
    <item>january<tag>out.mon+="01";</tag></item>
    <item>february<tag>out.mon+="02";</tag></item>
    <item>april<tag>out.mon+="04";</tag></item>
    <item>may<tag>out.mon+="05";</tag></item>
    <item>june<tag>out.mon+="06";</tag></item>
    <item>july<tag>out.mon+="07";</tag></item>
    <item>august<tag>out.mon+="08";</tag></item>
    <item>september<tag>out.mon+="09";</tag></item>
    <item>october<tag>out.mon+="10";</tag></item>
    <item>november<tag>out.mon+="11";</tag></item>
    <item>december<tag>out.mon+="12";</tag></item>
    <item>jan<tag>out.mon+="01";</tag></item>
    <item>feb<tag>out.mon+="02";</tag></item>
  </one-of>
```

Plan renewal date, month/year

<?xml version="1.0" encoding="UTF-8" ?>
<grammar xmlns="http://www.w3.org/2001/06/grammar"
  xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
  xsi:schemaLocation="http://www.w3.org/2001/06/grammar">
  <rule id="digits">
    <one-of>
      <item>zero</item><tag>out=0;</tag>
      <item>one</item><tag>out=1;</tag>
      <item>two</item><tag>out=2;</tag>
      <item>three</item><tag>out=3;</tag>
      <item>four</item><tag>out=4;</tag>
      <item>five</item><tag>out=5;</tag>
      <item>six</item><tag>out=6;</tag>
      <item>seven</item><tag>out=7;</tag>
      <item>eight</item><tag>out=8;</tag>
      <item>nine</item><tag>out=9;</tag>
    </one-of>
  </rule>
  <rule id="teens">
    <one-of>
      <item>ten</item><tag>out=10;</tag>
      <item>eleven</item><tag>out=11;</tag>
      <item>twelve</item><tag>out=12;</tag>
      <item>thir"teen"</item><tag>out=13;</tag>
      <item>fourteen</item><tag>out=14;</tag>
      <item>"fifteen"</item><tag>out=15;</tag>
      <item>sixteen</item><tag>out=16;</tag>
      <item>seventeen</item><tag>out=17;</tag>
      <item>eighteen</item><tag>out=18;</tag>
      <item>nineteen</item><tag>out=19;</tag>
    </one-of>
  </rule>
  <rule id="thousands">
    <item>two thousand</item>
    <item repeat="0-1">and</item>
    <item repeat="0-1"><ruleref uri="#digits"><tag>out = rules.digits;</tag></item>
    <item repeat="0-1"><ruleref uri="#teens"><tag>out = rules.teens;</tag></item>
    <item repeat="0-1"><ruleref uri="#above_twenty"><tag>out = rules.above_twenty;</tag></item>
  </rule>
  <rule id="above_twenty">
    <one-of>
      <item>twenty</item><tag>out=20;</tag>
      <item>thirty</item><tag>out=30;</tag>
    </one-of>
    <item repeat="0-1"><ruleref uri="#digits"><tag>out += rules.digits;</tag></item>
  </rule>
</grammar>
<rule id="main" scope="public">
  <tag>out=""</tag>
  <item repeat="1-10">
    <item repeat="1"><ruleref uri="#months"/><tag>out = out + rules.months.mon + ";";/</tag></item>
    <one-of>
      <item>ruleref uri="#thousands"/><tag>out += rules.thousands;</tag></item>
      <item repeat="0-1">ruleref uri="#digits"/><tag>out += rules.digits;</tag></item>
      <item repeat="0-1">ruleref uri="#teens"/><tag>out += rules.teens;</tag></item>
      <item repeat="0-1">ruleref uri="#above_twenty"/><tag>out += rules.above_twenty;</tag></item>
    </one-of>
  </item>
</rule>

<rule id="text">
  <item repeat="0-1">ruleref uri="#hesitation"/</item>
  <one-of>
    <item repeat="0-1">My plan will renew on</item>
    <item repeat="0-1">My plan was renewed on</item>
    <item repeat="0-1">Renew plan on</item>
  </one-of>
</rule>

<rule id="hesitation">
  <one-of>
    <item>Hmm</item>
    <item>Mmm</item>
    <item>My</item>
  </one-of>
</rule>

<rule id="months">
  <item repeat="0-1">ruleref uri="#text"/</item>
  <tag>out=""</tag>
  <one-of>
    <item>janeary<tag>out.mon="01";/</item>
    <item>february<tag>out.mon="02";/</item>
    <item>march<tag>out.mon="03";/</item>
    <item>april<tag>out.mon="04";/</item>
    <item>may<tag>out.mon="05";/</item>
    <item>june<tag>out.mon="06";/</item>
    <item>july<tag>out.mon="07";/</item>
  </one-of>
</rule>
<rule id="digits">
<one-of>
  <item>zero</item><tag>out=0;</tag></item>
  <item>one</item><tag>out=1;</tag></item>
  <item>two</item><tag>out=2;</tag></item>
  <item>three</item><tag>out=3;</tag></item>
  <item>four</item><tag>out=4;</tag></item>
  <item>five</item><tag>out=5;</tag></item>
  <item>six</item><tag>out=6;</tag></item>
  <item>seven</item><tag>out=7;</tag></item>
  <item>eight</item><tag>out=8;</tag></item>
  <item>nine</item><tag>out=9;</tag></item>
</one-of>
</rule>

<rule id="teens">
<one-of>
  <item>ten</item><tag>out=10;</tag></item>
  <item>eleven</item><tag>out=11;</tag></item>
  <item>twelve</item><tag>out=12;</tag></item>
  <item>thirteen</item><tag>out=13;</tag></item>
  <item>fourteen</item><tag>out=14;</tag></item>
  <item>fifteen</item><tag>out=15;</tag></item>
  <item>sixteen</item><tag>out=16;</tag></item>
  <item>seventeen</item><tag>out=17;</tag></item>
  <item>eighteen</item><tag>out=18;</tag></item>
  <item>nineteen</item><tag>out=19;</tag></item>
</one-of>
</rule>

<rule id="thousands">
<item>two thousand</item>
<item repeat="0-1"><ruleref uri="#digits"/></item><tag>out = rules.digits;</tag></item>
<item repeat="0-1"><ruleref uri="#teens"/></item><tag>out = rules.teens;</tag></item>
<item repeat="0-1"><ruleref uri="#above_twenty"/></item><tag>out = rules.above_twenty;</tag></item>
</rule>

<rule id="above_twenty">
<one-of>
  <item>twenty</item><tag>out=20;</tag></item>
  <item>thirty</item><tag>out=30;</tag></item>
  <item>forty</item><tag>out=40;</tag></item>
  <item>fifty</item><tag>out=50;</tag></item>
  <item>sixty</item><tag>out=60;</tag></item>
  <item>seventy</item><tag>out=70;</tag></item>
  <item>eighty</item><tag>out=80;</tag></item>
  <item>ninety</item><tag>out=90;</tag></item>
</one-of>
</rule>
Plan start date, month/day

<?xml version="1.0" encoding="UTF-8" ?>
<grammar xmlns="http://www.w3.org/2001/06/grammar"
  xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
  xsi:schemaLocation="http://www.w3.org/2001/06/grammar
  http://www.w3.org/TR/speech-grammar/grammar.xsd"
  xml:lang="en-US" version="1.0"
  root="main"
  mode="voice"
  tag-format="semantics/1.0">
  <!-- Test Cases
  Grammar will support the following inputs:
  Scenario 1:
  Input: My plan will start on july twenty three
  Output: 07/23
  Scenario 2:
  Input: My plan will start on july fifteen
  Output: 07/15
  -->

  <rule id="main" scope="public">
    <tag>out=""</tag>
    <item repeat="1-10">
      <item><ruleref uri="#months"/><tag>out= rules.months.mon + "/";</tag></item>
      <one-of>
        <item><ruleref uri="#digits"/><tag>out+= rules.digits;</tag></item>
        <item><ruleref uri="#teens"/><tag>out+= rules.teens;</tag></item>
        <item><ruleref uri="#above_twenty"/><tag>out+= rules.above_twenty;</tag></item>
      </one-of>
    </item>
  </rule>

  <rule id="text">
    <item repeat="0-1"><ruleref uri="#hesitation"/></item>
    <one-of>
      <item repeat="0-1">My plan started on</item>
      <item repeat="0-1">My plan will start on</item>
      <item repeat="0-1">I paid it on</item>
      <item repeat="0-1">I paid bill for</item>
    </one-of>
  </rule>

  <rule id="hesitation">
    <one-of>
      <item>Hmm</item>
      <item>Mmm</item>
      <item>My</item>
    </one-of>
  </rule>

  <rule id="months"/>
</grammar>
<item repeat="0-1"><ruleref uri="#text"></item>
<one-of>
  <item><rule id="text">
    <item>January</item>
    <item>Jan</item>
  </rule></item>
  <item><rule id="text">
    <item>February</item>
    <item>Feb</item>
  </rule></item>
  <item><rule id="text">
    <item>March</item>
    <item>Mar</item>
  </rule></item>
  <item><rule id="text">
    <item>April</item>
    <item>Apr</item>
  </rule></item>
  <item><rule id="text">
    <item>May</item>
    <item>May</item>
  </rule></item>
  <item><rule id="text">
    <item>June</item>
    <item>Jun</item>
  </rule></item>
  <item><rule id="text">
    <item>July</item>
    <item>Jul</item>
  </rule></item>
  <item><rule id="text">
    <item>August</item>
    <item>Aug</item>
  </rule></item>
  <item><rule id="text">
    <item>September</item>
    <item>Sep</item>
  </rule></item>
  <item><rule id="text">
    <item>October</item>
    <item>Oct</item>
  </rule></item>
  <item><rule id="text">
    <item>November</item>
    <item>Nov</item>
  </rule></item>
  <item><rule id="text">
    <item>December</item>
    <item>Dec</item>
  </rule></item>
</one-of>
</rule>

<rule id="digits">
  <item repeat="0-1"><ruleref uri="#text"></item>
  <one-of>
    <item>0</item>
    <item>First</item>
  </one-of>
</rule>

<rule id="teens">
  <item repeat="0-1"><ruleref uri="#text"></item>
  <one-of>
    <item>Ten</item>
    <item>Tenth</item>
    <item>Eleven</item>
    <item>Twelve</item>
  </one-of>
</rule>
Industry grammars

<item>thirteen</item><tag>out=13;</tag></item>
<item>fourteen</item><tag>out=14;</tag></item>
<item>fifteen</item><tag>out=15;</tag></item>
<item>sixteen</item><tag>out=16;</tag></item>
<item>seventeen</item><tag>out=17;</tag></item>
<item>eighteen</item><tag>out=18;</tag></item>
<item>nineteen</item><tag>out=19;</tag></item>
<item>tenth</item><tag>out=10;</tag></item>
<item>eleventh</item><tag>out=11;</tag></item>
<item>twelveth</item><tag>out=12;</tag></item>
<item>thirteenth</item><tag>out=13;</tag></item>
<item>fourteenth</item><tag>out=14;</tag></item>
<item>fifteenth</item><tag>out=15;</tag></item>
<item>sixteenth</item><tag>out=16;</tag></item>
<item>seventeenth</item><tag>out=17;</tag></item>
<item>eighteenth</item><tag>out=18;</tag></item>
<item>nineteenth</item><tag>out=19;</tag></item>

</one-of>
</rule>

<rule id="above_twenty">
  <item repeat="0-1"><ruleref uri="#text"></item>
  <one-of>
    <item>twenty</item><tag>out=20;</tag></item>
    <item>thirty</item><tag>out=30;</tag></item>
  </one-of>
  <item repeat="0-1"><ruleref uri="#digits"></item>
  <tag>out += rules.digits;</tag></item>
</rule>
</grammar>

Service start date, month/day

<?xml version="1.0" encoding="UTF-8" ?>
<grammar xmlns="http://www.w3.org/2001/06/grammar"
  xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
  xsi:schemaLocation="http://www.w3.org/2001/06/grammar
  http://www.w3.org/TR/speech-grammar/grammar.xsd"
  xml:lang="en-US" version="1.0"
  root="main"
  mode="voice"
  tag-format="semantics/1.0">
  <!-- Test Cases

Grammar will support the following inputs:

Scenario 1:
  Input: My plan starts on july twenty three
  Output: 07/23

Scenario 2:
  Input: I want to activate on july fifteen
  Output: 07/15

--> 

<rule id="main" scope="public">
  <tag>out=""</tag>
  <item repeat="1-10">
    <item><ruleref uri="#months"></item>
    <tag>out = rules.months.mon + ";";</tag></item>
  </item>
  <one-of>
    <item><ruleref uri="#digits"></item>
    <tag>out += rules.digits;</tag></item>
  </one-of>
</rule>
</grammar>
<rule id="text">
  <item repeat="0-1"><ruleref uri="#hesitation"/></item>
  <one-of>
    <item repeat="0-1">My plan starts on</item>
    <item repeat="0-1">I want to start my plan on</item>
    <item repeat="0-1">Activation date of</item>
    <item repeat="0-1">Start activation on</item>
    <item repeat="0-1">I want to activate on</item>
    <item repeat="0-1">Activate plan starting</item>
    <item repeat="0-1">Starting</item>
    <item repeat="0-1">Start on</item>
  </one-of>
</rule>

<rule id="hesitation">
  <one-of>
    <item>Hmm</item>
    <item>Mmm</item>
    <item>My</item>
  </one-of>
</rule>

<rule id="months">
  <item repeat="0-1"><ruleref uri="#text"/></item>
  <tag>out.mon=""</tag>
  <one-of>
    <item>january<tag>out.mon="01";</tag></item>
    <item>february<tag>out.mon="02";</tag></item>
    <item>march<tag>out.mon="03";</tag></item>
    <item>april<tag>out.mon="04";</tag></item>
    <item>may<tag>out.mon="05";</tag></item>
    <item>june<tag>out.mon="06";</tag></item>
    <item>july<tag>out.mon="07";</tag></item>
    <item>aug<tag>out.mon="08";</tag></item>
    <item>sept<tag>out.mon="09";</tag></item>
    <item>oct<tag>out.mon="10";</tag></item>
    <item>nov<tag>out.mon="11";</tag></item>
    <item>dec<tag>out.mon="12";</tag>
  </one-of>
</rule>

<rule id="digits">
  <item repeat="0-1"><ruleref uri="#text"/></item>
  <one-of>
    <item>0<tag>out=0;</tag></item>
    <item>1<tag>out=1;</tag></item>
    <item>2<tag>out=2;</tag></item>
    <item>3<tag>out=3;</tag></item>
    <item>4<tag>out=4;</tag></item>
    <item>5<tag>out=5;</tag></item>
    <item>6<tag>out=6;</tag></item>
    <item>7<tag>out=7;</tag>
    <item>8<tag>out=8;</tag>
    <item>9<tag>out=9;</tag>
  </one-of>
</rule>
<item>8<tag>out=8;</tag></item>
<item>9<tag>out=9;</tag></item>
<item>first<tag>out=01;</tag></item>
<item>second<tag>out=02;</tag></item>
<item>third<tag>out=03;</tag></item>
<item>fourth<tag>out=04;</tag></item>
<item>fifth<tag>out=05;</tag></item>
<item>sixth<tag>out=06;</tag></item>
<item>seventh<tag>out=07;</tag></item>
<item>eighth<tag>out=08;</tag></item>
<item>ninth<tag>out=09;</tag></item>
<item>one<tag>out=1;</tag></item>
<item>two<tag>out=2;</tag></item>
<item>three<tag>out=3;</tag></item>
<item>four<tag>out=4;</tag></item>
<item>five<tag>out=5;</tag></item>
<item>six<tag>out=6;</tag></item>
<item>seven<tag>out=7;</tag></item>
<item>eight<tag>out=8;</tag></item>
<item>nine<tag>out=9;</tag></item>
</one-of>
</rule>
</grammar>

<rule id="teens">
  <item repeat="0-1"><ruleref uri="#text"/></item>
  <one-of>
    <item>ten<tag>out=10;</tag></item>
    <item>tenth<tag>out=10;</tag></item>
    <item>eleven<tag>out=11;</tag></item>
    <item>twelve<tag>out=12;</tag></item>
    <item>thirteen<tag>out=13;</tag></item>
    <item>fourteen<tag>out=14;</tag></item>
    <item>fifteen<tag>out=15;</tag></item>
    <item>sixteen<tag>out=16;</tag></item>
    <item>seventeen<tag>out=17;</tag></item>
    <item>eighteen<tag>out=18;</tag></item>
    <item>nineteen<tag>out=19;</tag></item>
    <item>ten<tag>out=10;</tag></item>
    <item>eleven<tag>out=11;</tag></item>
    <item>twelve<tag>out=12;</tag></item>
    <item>thirteen<tag>out=13;</tag></item>
    <item>fourteen<tag>out=14;</tag></item>
    <item>fifteen<tag>out=15;</tag></item>
    <item>sixteen<tag>out=16;</tag></item>
    <item>seventeen<tag>out=17;</tag></item>
    <item>eighteen<tag>out=18;</tag></item>
    <item>nineteen<tag>out=19;</tag></item>
  </one-of>
</rule>

<rule id="above_twenty">
  <item repeat="0-1"><ruleref uri="#text"/></item>
  <one-of>
    <item>twenty<tag>out=20;</tag></item>
    <item>thirty<tag>out=30;</tag></item>
  </one-of>
</rule>

<rule id="digits">
  <item repeat="0-1"><ruleref uri="#digits"/></item>
  <tag>out += rules.digits;</tag></item>
</rule>

<grammar>

Equipment quantity

<?xml version="1.0" encoding="UTF-8" ?>
Test Cases

Grammar will support the following inputs:

---

Scenario 1:
Input: The number is one
Output: 1

Scenario 2:
Input: It is ten
Output: 10

---

<rule id="main" scope="public">
  <tag>out=""</tag>
  <one-of>
    <item repeat="1"><ruleref uri="#digits"/><tag>out+= rules.digits;</tag></item>
    <item repeat="1"><ruleref uri="#teens"/><tag>out+= rules.teens;</tag></item>
    <item repeat="1"><ruleref uri="#above_twenty"/><tag>out+= rules.above_twenty;</tag></item>
    <item repeat="0-1"><ruleref uri="#thanks"/></item>
  </one-of>
</rule>

<rule id="text">
  <item repeat="0-1"><ruleref uri="#hesitation"/></item>
  <one-of>
    <item>It is</item>
    <item>The number is</item>
    <item>I want to order</item>
    <item>Total equipment</item>
  </one-of>
</rule>

<rule id="hesitation">
  <one-of>
    <item>Hmm</item>
    <item>Mmm</item>
    <item>My</item>
  </one-of>
</rule>

<rule id="thanks">
  <one-of>
    <item>Thanks</item>
    <item>I think</item>
  </one-of>
</rule>

<rule id="digits">
  <item repeat="0-1"><ruleref uri="#text"/></item>
  <one-of>
    <item>0<tag>out=0;</tag></item>
    <item>1<tag>out=1;</tag></item>
  </one-of>
</rule>
<item>2</item><tag>out=2;</tag></item>
<item>3</item><tag>out=3;</tag></item>
<item>4</item><tag>out=4;</tag></item>
<item>5</item><tag>out=5;</tag></item>
<item>6</item><tag>out=6;</tag></item>
<item>7</item><tag>out=7;</tag></item>
<item>8</item><tag>out=8;</tag></item>
<item>9</item><tag>out=9;</tag></item>
<item>one</item><tag>out=1;</tag></item>
<item>two</item><tag>out=2;</tag></item>
<item>three</item><tag>out=3;</tag></item>
<item>four</item><tag>out=4;</tag></item>
<item>five</item><tag>out=5;</tag></item>
<item>six</item><tag>out=6;</tag></item>
<item>seven</item><tag>out=7;</tag></item>
<item>eight</item><tag>out=8;</tag></item>
<item>nine</item><tag>out=9;</tag></item>
</one-of>
</rule>

<rule id="teens">
  <item repeat="0-1"><ruleref uri="#text"/></item>
  <one-of>
    <item>ten</item><tag>out=10;</tag></item>
    <item>eleven</item><tag>out=11;</tag></item>
    <item>twelve</item><tag>out=12;</tag></item>
    <item>thirteen</item><tag>out=13;</tag></item>
    <item>fourteen</item><tag>out=14;</tag></item>
    <item> fifteen</item><tag>out=15;</tag></item>
    <item>sixteen</item><tag>out=16;</tag></item>
    <item>seventeen</item><tag>out=17;</tag></item>
    <item>eighteen</item><tag>out=18;</tag></item>
    <item>nineteen</item><tag>out=19;</tag></item>
    <item>10</item><tag>out=10;</tag></item>
    <item>11</item><tag>out=11;</tag></item>
    <item>12</item><tag>out=12;</tag></item>
    <item>13</item><tag>out=13;</tag></item>
    <item>14</item><tag>out=14;</tag></item>
    <item>15</item><tag>out=15;</tag></item>
    <item>16</item><tag>out=16;</tag></item>
    <item>17</item><tag>out=17;</tag></item>
    <item>18</item><tag>out=18;</tag></item>
    <item>19</item><tag>out=19;</tag></item>
  </one-of>
</rule>

<rule id="above_twenty">
  <item repeat="0-1"><ruleref uri="#text"/></item>
  <one-of>
    <item>twenty</item><tag>out=20;</tag></item>
    <item>thirty</item><tag>out=30;</tag></item>
    <item>forty</item><tag>out=40;</tag></item>
    <item>fifty</item><tag>out=50;</tag></item>
    <item>sixty</item><tag>out=60;</tag></item>
    <item>seventy</item><tag>out=70;</tag></item>
    <item>eighty</item><tag>out=80;</tag></item>
    <item>ninety</item><tag>out=90;</tag></item>
    <item>20</item><tag>out=20;</tag></item>
    <item>30</item><tag>out=30;</tag></item>
    <item>40</item><tag>out=40;</tag></item>
    <item>50</item><tag>out=50;</tag></item>
    <item>60</item><tag>out=60;</tag></item>
    <item>70</item><tag>out=70;</tag></item>
    <item>80</item><tag>out=80;</tag></item>
    <item>90</item><tag>out=90;</tag></item>
  </one-of>
</rule>
<item repeat="0-1"><ruleref uri="#digits"/> <tag>out += rules.digits;</tag></item>
</rule>
</grammar>

Bill amount

<grammar xmlns="http://www.w3.org/2001/06/grammar"
         xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xsi:schemaLocation="http://www.w3.org/2001/06/grammar
                      http://www.w3.org/TR/speech-grammar/grammar.xsd"
xmllang="en-US" version="1.0"
root="main"
mode="voice"
tag-format="semantics/1.0">
<!-- Test Cases
Grammar will support the following inputs:
Input: I want to make a payment of one hundred ten dollars
Output: $110
--> 
<rule id="main" scope="public">
<tag>out=""</tag>
one-of
  <item><ruleref uri="#sub_hundred"></item>
  <tag>out += rules.sub_hundred.sh;</tag></item>
</one-of>
<item repeat="0-1"><ruleref uri="#subThousands"></item>
<tag>out += rules.subThousands;</tag></item>
</one-of>
</rule>
<rule id="text">
one-of
  <item repeat="0-1">I want to make a payment for</item>
  <item repeat="0-1">I want to make a payment of</item>
  <item repeat="0-1">Pay a total of</item>
  <item repeat="0-1">Paying</item>
  <item repeat="0-1">Pay bill for</item>
</one-of>
</rule>
<rule id="hesitation">
one-of
  <item>Hmm</item>
  <item>Mmm</item>
  <item>My</item>
</one-of>
</rule>
<rule id="thanks">
one-of
  <item>Thanks</item>
  <item>I think</item>
</one-of>
</rule>
<rule id="digits">
  <item repeat="0-1"><ruleref uri="#text"></item>
  <tag>out.num = 0;</tag>
  <one-of>
    <item>0</item>
      <tag>out.num+=0;</tag>
    <item>1</item>
      <tag>out.num+=1;</tag>
    <item>2</item>
      <tag>out.num+=2;</tag>
    <item>3</item>
      <tag>out.num+=3;</tag>
    <item>4</item>
      <tag>out.num+=4;</tag>
    <item>5</item>
      <tag>out.num+=5;</tag>
    <item>6</item>
      <tag>out.num+=6;</tag>
    <item>7</item>
      <tag>out.num+=7;</tag>
    <item>8</item>
      <tag>out.num+=8;</tag>
    <item>9</item>
      <tag>out.num+=9;</tag>
    <item>one</item>
      <tag>out.num+=1;</tag>
    <item>two</item>
      <tag>out.num+=2;</tag>
    <item>three</item>
      <tag>out.num+=3;</tag>
    <item>four</item>
      <tag>out.num+=4;</tag>
    <item>five</item>
      <tag>out.num+=5;</tag>
    <item>six</item>
      <tag>out.num+=6;</tag>
    <item>seven</item>
      <tag>out.num+=7;</tag>
    <item>eight</item>
      <tag>out.num+=8;</tag>
    <item>nine</item>
      <tag>out.num+=9;</tag>
  </one-of>
  <item repeat="0-1"><ruleref uri="#currency"></item>
</rule>

<rule id="teens">
  <item repeat="0-1"><ruleref uri="#text"></item>
  <tag>out.teen = 0;</tag>
  <one-of>
    <item>ten</item>
      <tag>out.teen+=10;</tag>
    <item>eleven</item>
      <tag>out.teen+=11;</tag>
    <item>twelve</item>
      <tag>out.teen+=12;</tag>
    <item>thirteen</item>
      <tag>out.teen+=13;</tag>
    <item>fourteen</item>
      <tag>out.teen+=14;</tag>
    <item>fifteen</item>
      <tag>out.teen+=15;</tag>
    <item>sixteen</item>
      <tag>out.teen+=16;</tag>
    <item>seventeen</item>
      <tag>out.teen+=17;</tag>
    <item>eighteen</item>
      <tag>out.teen+=18;</tag>
    <item>nineteen</item>
      <tag>out.teen+=19;</tag>
  </one-of>
  <item repeat="0-1"><ruleref uri="#currency"></item>
</rule>

<rule id="above_twenty">
  <item repeat="0-1"><ruleref uri="#text"></item>
  <tag>out.tens = 0;</tag>
  <one-of>
    <item>twenty</item>
      <tag>out.tens+=20;</tag>
    <item>thirty</item>
      <tag>out.tens+=30;</tag>
    <item>forty</item>
      <tag>out.tens+=40;</tag>
    <item>fifty</item>
      <tag>out.tens+=50;</tag>
    <item>sixty</item>
      <tag>out.tens+=60;</tag>
    <item>seventy</item>
      <tag>out.tens+=70;</tag>
    <item>eighty</item>
      <tag>out.tens+=80;</tag>
    <item>ninety</item>
      <tag>out.tens+=90;</tag>
    <item>hundred</item>
      <tag>out.tens+=100;</tag>
  </one-of>
  <item repeat="0-1"><ruleref uri="#currency"></item>
  <item repeat="0-1"><ruleref uri="#digits"><tag>out.tens += rules.digits.num;</tag></item>
</rule>

<rule id="currency">
  <one-of>
  <item repeat="0-1"><ruleref uri="#text"></item>
  <item repeat="0-1"><ruleref uri="#currency"></item>
  <item repeat="0-1"><ruleref uri="#digits"></item>
  </one-of>
</rule>
Generic grammars (download)

We provide the following generic grammars: alphanumeric, currency, date (mm/dd/yy), numbers, greeting, hesitation, and agent.

Alphanumeric

<?xml version="1.0" encoding="UTF-8" ?>
<grammar xmlns="http://www.w3.org/2001/06/grammar"
  xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xsi:schemaLocation="http://www.w3.org/2001/06/grammar
  http://www.w3.org/TR/speech-grammar/grammar.xsd"
xmllang="en-US" version="1.0"
root="main"
mode="voice"
tag-format="semantics/1.0">
  <!-- Test Cases
  Scenario 1:
    Input: A B C 1 2 3 4
    Output: ABC1234
  Scenario 2:
    Input: 1 2 3 4 A B C
    Output: 1234ABC
  Scenario 3:
    Input: 1 2 3 4 A B C 1
  -->
Output: 123ABC1

```xml
<rule id="main" scope="public">
  <tag>out=""</tag>
  <item><ruleref uri="#alphanumeric"/> <tag>out += rules.alphanumeric.alphanum;</tag></item>
  <item repeat="0-1"><ruleref uri="#alphabets"/> <tag>out += rules.alphabets.letters;</tag></item>
  <item repeat="0-1"><ruleref uri="#digits"/> <tag>out += rules.digits.numbers;</tag></item>
</rule>

<rule id="alphanumeric" scope="public">
  <tag>out.alphanum=""</tag>
  <item><ruleref uri="#alphabets"/> <tag>out.alphanum += rules.alphabets.letters;</tag></item>
  <item repeat="0-1"><ruleref uri="#digits"/> <tag>out.alphanum += rules.digits.numbers;</tag></item>
</rule>

<rule id="alphabets">
  <tag>out.letters=""</tag>
  <tag>out.firstOccurence=""</tag>
  <item repeat="0-1"><ruleref uri="#digits"/> <tag>out.firstOccurence += rules.digits.numbers; out.letters += out.firstOccurence;</tag></item>
  <one-of>
    <item>A<tag>out.letters+='A';</tag></item>
    <item>B<tag>out.letters+='B';</tag></item>
    <item>C<tag>out.letters+='C';</tag></item>
    <item>D<tag>out.letters+='D';</tag></item>
    <item>E<tag>out.letters+='E';</tag></item>
    <item>F<tag>out.letters+='F';</tag></item>
    <item>G<tag>out.letters+='G';</tag></item>
    <item>H<tag>out.letters+='H';</tag></item>
    <item>I<tag>out.letters+='I';</tag></item>
    <item>J<tag>out.letters+='J';</tag></item>
    <item>K<tag>out.letters+='K';</tag></item>
    <item>L<tag>out.letters+='L';</tag></item>
    <item>M<tag>out.letters+='M';</tag></item>
    <item>N<tag>out.letters+='N';</tag></item>
    <item>O<tag>out.letters+='O';</tag></item>
    <item>P<tag>out.letters+='P';</tag></item>
    <item>Q<tag>out.letters+='Q';</tag></item>
    <item>R<tag>out.letters+='R';</tag></item>
    <item>S<tag>out.letters+='S';</tag></item>
    <item>T<tag>out.letters+='T';</tag></item>
    <item>U<tag>out.letters+='U';</tag></item>
    <item>V<tag>out.letters+='V';</tag></item>
    <item>W<tag>out.letters+='W';</tag></item>
    <item>X<tag>out.letters+='X';</tag></item>
    <item>Y<tag>out.letters+='Y';</tag></item>
    <item>Z<tag>out.letters+='Z';</tag></item>
  </one-of>
</rule>

<rule id="digits">
  <tag>out.numbers=""</tag>
  <item repeat="1-10">
    <one-of>
      <item>0<tag>out.numbers+=0;</tag></item>
      <item>1<tag>out.numbers+=1;</tag></item>
      <item>2<tag>out.numbers+=2;</tag></item>
      <item>3<tag>out.numbers+=3;</tag></item>
    </one-of>
  </item>
</rule>
```
Currency

```xml
<?xml version="1.0" encoding="UTF-8" ?>
<grammar xmlns="http://www.w3.org/2001/06/grammar"
    xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
    xsi:schemaLocation="http://www.w3.org/2001/06/grammar
                      http://www.w3.org/TR/speech-grammar/grammar.xsd"
    xml:lang="en-US" version="1.0"
    root="main"
    mode="voice"
    tag-format="semantics/1.0">
    <rule id="main" scope="public">
        <tag>out="$"</tag>
        <one-of>
            <item><ruleref uri="#sub_hundred"></item>
            <item><ruleref uri="#subThousands"></item>
        </one-of>
    </rule>
    <rule id="digits">
        <tag>out.num = 0;</tag>
        <one-of>
            <item>0</item>
            <item>1</item>
            <item>2</item>
            <item>3</item>
            <item>4</item>
            <item>5</item>
            <item>6</item>
            <item>7</item>
            <item>8</item>
            <item>9</item>
            <item>one</item>
            <item>two</item>
            <item>three</item>
            <item>four</item>
            <item>five</item>
            <item>six</item>
            <item>seven</item>
            <item>eight</item>
            <item>nine</item>
        </one-of>
        <item repeat="0-1"><ruleref uri="#currency"></item>
    </rule>
    <rule id="teens">
        <tag>out.teen = 0;</tag>
        <one-of>
            <item>ten</item>
            <item>eleven</item>
        </one-of>
        <item repeat="0-1"><ruleref uri="#currency"></item>
    </rule>
</grammar>
```
<item>twelve</item><tag>out.teen+=12;</tag></item>
<item>thirteen</item><tag>out.teen+=13;</tag></item>
<item>fourteen</item><tag>out.teen+=14;</tag></item>
<item>fifteen</item><tag>out.teen+=15;</tag></item>
<item>sixteen</item><tag>out.teen+=16;</tag></item>
<item>seventeen</item><tag>out.teen+=17;</tag></item>
<item>eighteen</item><tag>out.teen+=18;</tag></item>
<item>nineteen</item><tag>out.teen+=19;</tag></item>
</one-of>
</rule>

<rule id="above_twenty">
<tag>out.tens = 0;</tag>
<one-of>
<item>twenty</item><tag>out.tens+=20;</tag></item>
<item>thirty</item><tag>out.tens+=30;</tag></item>
<item>forty</item><tag>out.tens+=40;</tag></item>
<item>fifty</item><tag>out.tens+=50;</tag></item>
<item>sixty</item><tag>out.tens+=60;</tag></item>
<item>seventy</item><tag>out.tens+=70;</tag></item>
<item>eighty</item><tag>out.tens+=80;</tag></item>
<item>ninety</item><tag>out.tens+=90;</tag></item>
</one-of>
</rule>

<rule id="currency">
<one-of>
<item repeat="0-1">dollars</item>
<item repeat="0-1">Dollars</item>
<item repeat="0-1">dollar</item>
<item repeat="0-1">Dollar</item>
</one-of>
</rule>

<rule id="sub_hundred">
<tag>out.sh = 0;</tag>
<one-of>
<item><ruleref uri="#teens"/><tag>out.sh += rules.teens.teen;</tag></item>
<item><ruleref uri="#above_twenty"/><tag>out.sh += rules.above_twenty.tens;</tag></item>
<item><ruleref uri="#digits"/><tag>out.sh += rules.digits.num;</tag></item>
</one-of>
</rule>

<rule id="subThousands">
<tag>out = (100 * rules.sub_hundred.sh);</tag>
<hundred
<item repeat="0-1">ruleref uri="#above_twenty"/><tag>out += rules.above_twenty.tens;</tag></item>
<item repeat="0-1">ruleref uri="#teens"/><tag>out += rules.teens.teen;</tag></item>
<item repeat="0-1">ruleref uri="#digits"/><tag>out += rules.digits.num;</tag></item>
</rule>
</grammar>
Date, dd/mm

```xml
<?xml version="1.0" encoding="UTF-8" ?>
<grammar xmlns="http://www.w3.org/2001/06/grammar"
   xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
   xsi:schemaLocation="http://www.w3.org/2001/06/grammar
   http://www.w3.org/TR/speech-grammar/grammar.xsd"
   xml:lang="en-US" version="1.0"
   root="main"
   mode="voice"
   tag-format="semantics/1.0">
  <rule id="main" scope="public">
    <tag>out=""</tag>
    <item repeat="1-10">
      <one-of>
        <item><ruleref uri="#digits"/></item>
        <item><ruleref uri="#teens"/></item>
        <item><ruleref uri="#above_twenty"/></item>
        <item><ruleref uri="#months"/></item>
      </one-of>
    </item>
  </rule>
  <rule id="months">
    <one-of>
      <item>january</item>
      <item>february</item>
      <item>march</item>
      <item>april</item>
      <item>may</item>
      <item>june</item>
      <item>july</item>
      <item>august</item>
      <item>september</item>
      <item>october</item>
      <item>november</item>
      <item>december</item>
      <item>jan</item>
      <item>feb</item>
      <item>aug</item>
      <item>sept</item>
      <item>oct</item>
      <item>nov</item>
      <item>dec</item>
    </one-of>
  </rule>
  <rule id="digits">
    <one-of>
      <item>0</item>
      <item>1</item>
      <item>2</item>
      <item>3</item>
      <item>4</item>
      <item>5</item>
      <item>6</item>
      <item>7</item>
      <item>8</item>
      <item>9</item>
      <item>first</item>
      <item>second</item>
      <item>third</item>
    </one-of>
  </rule>
</grammar>
```
<item>fourth</item><tag>out=4;</tag></item>  
<item>fifth</item><tag>out=5;</tag></item>  
<item>sixth</item><tag>out=6;</tag></item>  
<item>seventh</item><tag>out=7;</tag></item>  
<item>eightth</item><tag>out=8;</tag></item>  
<item>nineth</item><tag>out=9;</tag></item>  
<item>one</item><tag>out=1;</tag></item>  
<item>two</item><tag>out=2;</tag></item>  
<item>three</item><tag>out=3;</tag></item>  
<item>four</item><tag>out=4;</tag></item>  
<item>five</item><tag>out=5;</tag></item>  
<item>six</item><tag>out=6;</tag></item>  
<item>seven</item><tag>out=7;</tag></item>  
<item>eight</item><tag>out=8;</tag></item>  
<item>nine</item><tag>out=9;</tag></item>  
</one-of>
</rule>

<rule id="teens">
  <one-of>
    <item>ten</item><tag>out=10;</tag></item>  
    <item>tenth</item><tag>out=10;</tag></item>  
    <item>eleven</item><tag>out=11;</tag></item>  
    <item>twelve</item><tag>out=12;</tag></item>  
    <item>thirteen</item><tag>out=13;</tag></item>  
    <item>fourteen</item><tag>out=14;</tag></item>  
    <item>fifteen</item><tag>out=15;</tag></item>  
    <item>sixteen</item><tag>out=16;</tag></item>  
    <item>seventeen</item><tag>out=17;</tag></item>  
    <item>eighteen</item><tag>out=18;</tag></item>  
    <item>nineteen</item><tag>out=19;</tag></item>  
    <item>tenth</item><tag>out=10;</tag></item>  
    <item>eleventh</item><tag>out=11;</tag></item>  
    <item>twelveth</item><tag>out=12;</tag></item>  
    <item>thirteenth</item><tag>out=13;</tag></item>  
    <item>fourteenth</item><tag>out=14;</tag></item>  
    <item>fifteenth</item><tag>out=15;</tag></item>  
    <item>sixteenth</item><tag>out=16;</tag></item>  
    <item>seventeenth</item><tag>out=17;</tag></item>  
    <item>eighteenth</item><tag>out=18;</tag></item>  
    <item>nineteenth</item><tag>out=19;</tag></item>  
  </one-of>
</rule>

<rule id="above_twenty">
  <one-of>
    <item>twenty</item><tag>out=20;</tag></item>  
    <item>thirty</item><tag>out=30;</tag></item>  
  </one-of>
  <item repeat="0-1"><ruleref uri="#digits"/>tag>out += rules.digits;</tag></item>
</rule>
</grammar>

Date, mm/yy

<?xml version="1.0" encoding="UTF-8" ?>
<rule id="main" scope="public">
<item repeat="1-10">
    <item repeat="1"><ruleref uri="#months"/><tag>out = out + rules.months.mon + ";";</tag></item>
    <one-of>
        <item><ruleref uri="#thousands"/><tag>out += rules.thousands;</tag></item>
        <item repeat="0-1"><ruleref uri="#digits"/><tag>out += rules.digits;</tag></item>
        <item repeat="0-1"><ruleref uri="#teens"/><tag>out += rules.teens;</tag></item>
        <item repeat="0-1"><ruleref uri="#above_twenty"/><tag>out += rules.above_twenty;</tag></item>
    </one-of>
</item>
</rule>

<rule id="months">
<one-of>
    <item>january<tag>out.mon+="january";</tag></item>
    <item>february<tag>out.mon+="february";</tag></item>
    <item>march<tag>out.mon+="march";</tag></item>
    <item>april<tag>out.mon+="april";</tag></item>
    <item>may<tag>out.mon+="may";</tag></item>
    <item>june<tag>out.mon+="june";</tag></item>
    <item>july<tag>out.mon+="july";</tag></item>
    <item>august<tag>out.mon+="august";</tag></item>
    <item>sept<tag>out.mon+="september";</tag></item>
    <item>oct<tag>out.mon+="october";</tag></item>
    <item>nov<tag>out.mon+="november";</tag></item>
    <item>dec<tag>out.mon+="december";</tag></item>
</one-of>
</rule>

<rule id="digits">
<one-of>
    <item>zero<tag>out=0;</tag></item>
    <item>one<tag>out=1;</tag></item>
    <item>two<tag>out=2;</tag></item>
    <item>three<tag>out=3;</tag></item>
    <item>four<tag>out=4;</tag></item>
    <item>five<tag>out=5;</tag></item>
    <item>six<tag>out=6;</tag></item>
    <item>seven<tag>out=7;</tag></item>
    <item>eight<tag>out=8;</tag></item>
    <item>nine<tag>out=9;</tag>
</one-of>
</rule>

<rule id="teens">
<one-of>
    <item>ten<tag>out=10;</tag></item>
    <item>eleven<tag>out=11;</tag></item>
    <item>twelve<tag>out=12;</tag></item>
    <item>thirteen<tag>out=13;</tag>
</one-of>
</rule>
<item>fourteen</item><tag>out=14;</tag></item>
<item>fifteen</item><tag>out=15;</tag></item>
<item>sixteen</item><tag>out=16;</tag></item>
<item>seventeen</item><tag>out=17;</tag></item>
<item>eighteen</item><tag>out=18;</tag></item>
<item>nineteen</item><tag>out=19;</tag></item>
</one-of>
</rule>

<!--  <rule id="singleDigit">
  <item><ruleref uri="#digits"/></item>
  <tag>out += rules.digits;</tag></item>
</rule>  -->

<rule id="thousands">
<!-- <item>
  <ruleref uri="#digits"/>
  <tag>out = (1000 * rules.digits);</tag>
  thousand
</item> -->
<item>two thousand</item><tag>out=2000;</tag></item>
<item repeat="0-1">and</item>
<item repeat="0-1"><ruleref uri="#digits"/></item>
<tag>out += rules.digits;</tag></item>
<item repeat="0-1"><ruleref uri="#teens"/></item>
<tag>out += rules.teens;</tag></item>
<item repeat="0-1"><ruleref uri="#above_twenty"/></item>
<tag>out += rules.above_twenty;</tag></item>
</rule>

<rule id="above_twenty">
<one-of>
<item>twenty</item><tag>out=20;</tag></item>
<item>thirty</item><tag>out=30;</tag></item>
<item>forty</item><tag>out=40;</tag></item>
<item>fifty</item><tag>out=50;</tag></item>
<item>sixty</item><tag>out=60;</tag></item>
<item>seventy</item><tag>out=70;</tag></item>
<item>eighty</item><tag>out=80;</tag></item>
<item>ninety</item><tag>out=90;</tag></item>
</one-of>
<item repeat="0-1"><ruleref uri="#digits"/></item>
<tag>out += rules.digits;</tag></item>
</rule>

</grammar>

Date, dd/mm/yyyy

<?xml version="1.0" encoding="UTF-8" ?>
<grammar xmlns="http://www.w3.org/2001/06/grammar"
  xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
  xsi:schemaLocation="http://www.w3.org/2001/06/grammar
  http://www.w3.org/TR/speech-grammar/grammar.xsd"
  xml:lang="en-US" version="1.0"
  root="main"
  mode="voice"
  tag-format="semantics/1.0">
  <rule id="main" scope="public">
    <tag>out=""</tag>
    <item repeat="1-10">
      <one-of>
        <item><ruleref uri="#digits"/></item>
        <tag>out += rules.digits + " ";</tag></item>
        <item><ruleref uri="#teens"/></item>
        <tag>out += rules.teens + " ";</tag></item>
        <item><ruleref uri="#above_twenty"/></item>
        <tag>out += rules.above_twenty;</tag></item>
      </one-of>
    </item>
  </rule>
</grammar>
<item><ruleref uri="#above_twenty"><tag>out += rules.above_twenty+ "</tag></item>
</one-of>
</item>
</rule>

<rule id="months">

<one-of>

<item><ruleref uri="#months"><tag>out = out + rules.months.mon+ " ";</tag></item>

<item><ruleref uri="#months"><tag>out += rules.months.mon</tag></item>
</one-of>
</rule>

<rule id="digits">

<one-of>

<item><ruleref uri="#digits"><tag>out = 0;</tag></item>

<item><ruleref uri="#digits"><tag>out = 1;</tag></item>

<item><ruleref uri="#digits"><tag>out = 2;</tag></item>

<item><ruleref uri="#digits"><tag>out = 3;</tag></item>

<item><ruleref uri="#digits"><tag>out = 4;</tag></item>

<item><ruleref uri="#digits"><tag>out = 5;</tag></item>

<item><ruleref uri="#digits"><tag>out = 6;</tag></item>

<item><ruleref uri="#digits"><tag>out = 7;</tag></item>

<item><ruleref uri="#digits"><tag>out = 8;</tag></item>

<item><ruleref uri="#digits"><tag>out = 9;</tag></item>
</one-of>
</rule>

<rule id="teens">

<one-of>

<item><ruleref uri="#teens"><tag>out = 10;</tag></item>

<item><ruleref uri="#teens"><tag>out = 11;</tag></item>

<item><ruleref uri="#teens"><tag>out = 12;</tag></item>

<item><ruleref uri="#teens"><tag>out = 13;</tag></item>

<item><ruleref uri="#teens"><tag>out = 14;</tag></item>

<item><ruleref uri="#teens"><tag>out = 15;</tag></item>
</one-of>
</rule>
<item>sixteen</item><tag>out=16;</tag></item>
<item>seventeen</item><tag>out=17;</tag></item>
<item>eighteen</item><tag>out=18;</tag></item>
<item>nineteen</item><tag>out=19;</tag></item>
</one-of>
</rule>

<rule id="thousands">
<item>two thousand</item><tag>out=2000;</tag></item>
<item repeat="0-1">and</item>
<item repeat="0-1"><ruleref uri="#digits"></tag>out += rules.digits;</item>
<item repeat="0-1"><ruleref uri="#teens"></tag>out += rules.teens;</item>
<item repeat="0-1"><ruleref uri="#above_twenty"></tag>out += rules.above_twenty;</item>
</rule>

<rule id="above_twenty">
<one-of>
<item>twenty</item><tag>out=20;</tag></item>
<item>thirty</item><tag>out=30;</tag></item>
<item>forty</item><tag>out=40;</tag></item>
<item>fifty</item><tag>out=50;</tag></item>
<item>sixty</item><tag>out=60;</tag></item>
<item>seventy</item><tag>out=70;</tag></item>
<item>eighty</item><tag>out=80;</tag></item>
<item<ninety</item><tag>out=90;</tag></item>
</one-of>
<item repeat="0-1"><ruleref uri="#digits"></tag>out += rules.digits;</item>
</rule>
</grammar>

Numbers, digits

<?xml version="1.0" encoding="UTF-8" ?>
xml:lang="en-US" version="1.0" root="digits" mode="voice" tag-format="semantics/1.0">
<rule id="digits">
<tag>out=""</tag>
<item><ruleref uri="#singleDigit"></tag>out += rules.singleDigit.digit;</item>
</rule>

<rule id="singleDigit">
<tag>out.digit=""</tag>
<item repeat="1-10">
<one-of>
<item>0</item><tag>out.digit+=0;</tag></item>
<item>zero</item><tag>out.digit+=0;</tag></item>
<item>1</item><tag>out.digit+=1;</tag></item>
<item>one</item><tag>out.digit+=1;</tag></item>
<item>2</item><tag>out.digit+=2;</tag></item>
<item>two</item><tag>out.digit+=2;</tag></item>
<item>3</item><tag>out.digit+=3;</tag></item>
<item>four</item><tag>out.digit+=4;</tag></item>
<item>five</item><tag>out.digit+=5;</tag></item>
<item>six</item><tag>out.digit+=6;</tag></item>
<item>seven</item><tag>out.digit+=7;</tag></item>
<item>eight</item><tag>out.digit+=8;</tag></item>
<item>nine</item><tag>out.digit+=9;</tag></item>
</one-of>
</item>
</rule>
<item>three</item><tag>out.digit+=3;</tag></item>
<item>four</item><tag>out.digit+=4;</tag></item>
<item>five</item><tag>out.digit+=5;</tag></item>
<item>six</item><tag>out.digit+=6;</tag></item>
<item>seven</item><tag>out.digit+=7;</tag></item>
<item>eight</item><tag>out.digit+=8;</tag></item>
<item>nine</item><tag>out.digit+=9;</tag></item>
</one-of>
</item>
</rule>
</grammar>

Numbers, ordinal

<?xml version="1.0" encoding="UTF-8" ?>
<grammar xmlns="http://www.w3.org/2001/06/grammar"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xsi:schemaLocation="http://www.w3.org/2001/06/grammar
http://www.w3.org/TR/speech-grammar/grammar.xsd"
xmllang="en-US" version="1.0"
root="main"
mode="voice"
tag-format="semantics/1.0">
<rule id="main" scope="public">
<tag>out=""</tag>
<one-of>
  <item repeat="1"><ruleref uri="#digits"/><tag>out+= rules.digits;</tag></item>
  <item repeat="1"><ruleref uri="#teens"/><tag>out+= rules.teens;</tag></item>
  <item repeat="1"><ruleref uri="#above_twenty"/><tag>out+= rules.above_twenty;</tag></item>
</one-of>
</rule>

<rule id="digits">
<one-of>
  <item>0</item><tag>out=0;</tag>
  <item>1</item><tag>out=1;</tag>
  <item>2</item><tag>out=2;</tag>
  <item>3</item><tag>out=3;</tag>
  <item>4</item><tag>out=4;</tag>
  <item>5</item><tag>out=5;</tag>
  <item>6</item><tag>out=6;</tag>
  <item>7</item><tag>out=7;</tag>
  <item>8</item><tag>out=8;</tag>
  <item>9</item><tag>out=9;</tag>
  <item>one</item><tag>out=1;</tag>
  <item>two</item><tag>out=2;</tag>
  <item>three</item><tag>out=3;</tag>
  <item>four</item><tag>out=4;</tag>
  <item>five</item><tag>out=5;</tag>
  <item>six</item><tag>out=6;</tag>
  <item>seven</item><tag>out=7;</tag>
  <item>eight</item><tag>out=8;</tag>
  <item>nine</item><tag>out=9;</tag>
</one-of>
</rule>
industry grammars

<rule id="teens">
    <one-of>
        <item>ten<tag>out=10;</tag></item>
        <item>eleven<tag>out=11;</tag></item>
        <item>twelve<tag>out=12;</tag></item>
        <item>thirteen<tag>out=13;</tag></item>
        <item>fourteen<tag>out=14;</tag></item>
        <item>fifteen<tag>out=15;</tag></item>
        <item>sixteen<tag>out=16;</tag></item>
        <item>seventeen<tag>out=17;</tag></item>
        <item>eighteen<tag>out=18;</tag></item>
        <item>nineteen<tag>out=19;</tag></item>
        <item>ten<tag>out=10;</tag></item>
        <item>eleven<tag>out=11;</tag></item>
        <item>twelve<tag>out=12;</tag></item>
        <item>thirteen<tag>out=13;</tag></item>
        <item>fourteen<tag>out=14;</tag></item>
        <item>fifteen<tag>out=15;</tag></item>
        <item>sixteen<tag>out=16;</tag></item>
        <item>seventeen<tag>out=17;</tag></item>
        <item>eighteen<tag>out=18;</tag></item>
        <item>nineteen<tag>out=19;</tag></item>
    </one-of>
</rule>

<rule id="above_twenty">
    <one-of>
        <item>twenty<tag>out=20;</tag></item>
        <item>thirty<tag>out=30;</tag></item>
        <item>forty<tag>out=40;</tag></item>
        <item>fifty<tag>out=50;</tag></item>
        <item>sixty<tag>out=60;</tag></item>
        <item>seventy<tag>out=70;</tag></item>
        <item>eighty<tag>out=80;</tag></item>
        <item>ninety<tag>out=90;</tag></item>
        <item>twenty<tag>out=20;</tag></item>
        <item>thirty<tag>out=30;</tag></item>
        <item>forty<tag>out=40;</tag></item>
        <item>fifty<tag>out=50;</tag></item>
        <item>sixty<tag>out=60;</tag></item>
        <item>seventy<tag>out=70;</tag></item>
        <item>eighty<tag>out=80;</tag></item>
        <item>ninety<tag>out=90;</tag></item>
    </one-of>
    <item repeat="0-1"><ruleref uri="#digits"/><tag>out += rules.digits;</tag></item>
</rule>

</grammar>
Industry grammars

Greeting

```xml
<?xml version="1.0" encoding="UTF-8" ?>
<grammar xmlns="http://www.w3.org/2001/06/grammar"
    xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
    xsi:schemaLocation="http://www.w3.org/2001/06/grammar
                        http://www.w3.org/TR/speech-grammar/grammar.xsd"
    xml:lang="en-US" version="1.0"
    root="main"
    mode="voice"
    tag-format="semantics/1.0">
    <rule id="main" scope="public">
        <tag>out=""</tag>
        <ruleref uri="#text"/><tag>out = rules.text</tag>
    </rule>
    <rule id="text">
        <one-of>
            <item>Can I talk to the agent<tag>out="You will be transferred to the agent in a while"</tag></item>
            <item>talk to an agent<tag>out="You will be transferred to the agent in a while"</tag></item>
        </one-of>
    </rule>
</grammar>
```

Hesitation

```xml
<?xml version="1.0" encoding="UTF-8" ?>
<grammar xmlns="http://www.w3.org/2001/06/grammar"
    xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
    xsi:schemaLocation="http://www.w3.org/2001/06/grammar
                        http://www.w3.org/TR/speech-grammar/grammar.xsd"
    xml:lang="en-US" version="1.0"
    root="main"
    mode="voice"
    tag-format="semantics/1.0">
    <rule id="main" scope="public">
        <tag>out=""</tag>
        <ruleref uri="#text"/><tag>out = rules.text</tag>
    </rule>
    <rule id="text">
        <one-of>
            <item>Hmm<tag>out="Waiting for your input"</tag></item>
        </one-of>
    </rule>
</grammar>
```
Creating custom slot types

For each intent, you can specify parameters that indicate the information that the intent needs to fulfill the user's request. These parameters, or slots, have a type. A slot type is a list of values that Amazon Lex V2 uses to train the machine learning model to recognize values for a slot. For example, you can define a slot type called "Genres." Each value in the slot type is the name of a genre, "comedy," "adventure," "documentary," etc. You can define a synonym for a slot type value. For example, you can define the synonyms "funny" and "humorous" for the value "comedy."

You can configure the slot type to restrict resolution to the slot values. The slot values will be used as an enumeration and the value entered by the user will be resolved to the slot value only if it is the same as one of the slot values or a synonym. A synonym is resolved to the corresponding slot value. For example, if the user enters "funny" it will resolve to the slot value "comedy."

Alternately, you can configure the slot type to expand the values. Slot values will be used as training data and the slot is resolved to the value provided by the user if it is similar to the slot values and synonyms. This is the default behavior.

Amazon Lex V2 maintains a list of possible resolutions for a slot. Each entry in the list provides a resolved value that Amazon Lex V2 recognized as additional possibilities for the slot. A resolved value is the best effort to match the slot value. The list contains up to five values.

When the value entered by the user is a synonym, the first entry in the list of resolvedValues is the slot type value. For example, if the user enters "funny," the originalValue field contains "funny" and the first entry in the resolvedValues field is "comedy." You configure the valueSelectionStrategy when you create or update a slot type with the CreateSlotType (p. 378) operation so that the slot value is filled with the first value in the resolution list.

If you are using a Lambda function, the input event to the function includes a resolution list called resolvedValues. The following example shows the slot section of the input to a Lambda function:

```
"slots": {
    "MovieGenre": {
      "value": {
        "originalValue": "funny",
        "interpretedValue": "comedy",
        "resolvedValues": [
          "comedy"
        ]
      }
    }
}
```

For each slot type, you can define a maximum of 10,000 values and synonyms. Each bot can have a total number of 50,000 slot type values and synonyms. For example, you can have 5 slot types, each with 5,000 values and 5,000 synonyms, or you can have 10 slot types, each with 2,500 values and 2,500 synonyms.
Creating a custom vocabulary to improve speech recognition

You can give Amazon Lex V2 more information about how to process audio conversations with a bot by creating a custom vocabulary in a specific language. A custom vocabulary is a list of specific phrases that you want Amazon Lex V2 to recognize in the audio input. These are generally proper nouns or domain-specific words that Amazon Lex V2 doesn't recognize.

For example, suppose that you have a tech support bot. You can add "backup" to a custom vocabulary to help the bot transcribe the audio correctly as "backup," even when the audio sounds like "pack up." A custom vocabulary can also help recognize rare words in the audio such as "solvency" for financial services or proper nouns such as "Cognito" or "Monitron."

Custom vocabulary basics

- A custom vocabulary works on the transcription of audio input to a bot. You must provide sample utterances to recognize an intent or slot value.
- A custom vocabulary is unique to a specific language. You must configure custom vocabularies independently for each language. Custom vocabularies are supported only for the English (UK) language.
- Custom vocabularies support only 8 kHz audio input. It is available with contact center integrations supported by Amazon Lex V2. The test window in the Amazon Lex V2 console does not support custom vocabularies as it uses 16 kHz audio input.

Amazon Lex V2 uses custom vocabularies to elicit both intents and slots. You can configure them independently.

**Eliciting an intent** – You can create a custom vocabulary for eliciting an intent. These phrases are used to transcription when your bot is determining the user's intent. For example, if you configured the phrase "backup" in your custom vocabulary, Amazon Lex V2 transcribes the user input to "can you please backup my photos?"—even when the audio sounds like "can you please pack up my photos." You can also specify the degree of boosting for each phrase by configuring a weight of 1, 2, or 3.

The custom vocabulary phrases used for improving transcription during intent elicitation don't affect transcriptions while eliciting slots. For more information about creating a custom vocabulary for eliciting intents, see Creating a custom vocabulary for eliciting intents (p. 142).

**Eliciting custom slots** – You can use a custom vocabulary to improve slot recognition for audio conversations. To improve your Amazon Lex V2 bot's ability to recognize slot values, create a custom slot and add the slot values to the custom slot, then choose Use slot values as custom vocabulary. Examples of slot values include product names, catalogs, or proper nouns. You shouldn't use common words or phrases such as "yes" and "no" in custom vocabularies.

After the slot values are added, these values are used for improving slot recognition when the bot is expecting input for the custom slot. These values aren't used for transcription when eliciting an intent. For more information, see Adding slot types (p. 19).

Best practices for creating a custom vocabulary

**Eliciting an intent**

- Custom vocabularies work best when used to target specific words or phrases. Only add words to a custom vocabulary if they are not readily recognized by Amazon Lex V2.
Creating a custom vocabulary for eliciting intents

- Decide how much weight to give a word based on how often the word isn’t recognized in the transcription and how rare the word is in the input. Difficult to pronounce words require a higher weight.
- Use a representative test set to determine if a weight is appropriate. You can collect an audio test set by turning on audio logging in conversation logs.
- Avoid using short words like “on,” “it,” “to,” “yes,” “no” in a custom vocabulary.

Eliciting a custom slot

- Add the values to the custom slot type that you expect to be recognized. Add all the possible slot values for the custom slot type, no matter how common or rare the slot value is.
- Enable the option only when the custom slot type contains a list of catalog values or entities such as product names or mutual funds.
- Disable the option if the slot type is used to capture generic phrases such as “yes,” “no,” “I don’t know,” “maybe,” or generic words such as “one,” “two,” “three.”
- Limit the number of slot values and synonyms to 500 or less for best performance.

Creating a custom vocabulary for eliciting intents

You can use the Amazon Lex V2 console to create and manage a custom vocabulary, or you can use Amazon Lex V2 API operations.

Console

To use the console to create a custom vocabulary

1. Create a zip file with the phrases that you want to add to the custom vocabulary. For more information, see Creating a custom vocabulary file (p. 143).
2. Open the Amazon Lex V2 console at https://console.aws.amazon.com/lexv2/home
3. From the list of bots, choose the one to which you want to add the custom vocabulary.
4. On the bot detail page, from the Add languages section, choose View languages.
5. From the list of languages, choose the language to which you want to add the custom vocabulary.
6. In the Custom vocabulary section, choose Import.
7. Browse to the file to import, enter a password if necessary, and then choose Import.

API

To use the API to create a custom vocabulary

1. Create a zip file with the phrases that you want to add to the custom vocabulary. For more information, see Creating a custom vocabulary file (p. 143).
2. Use the CreateUploadUrl (p. 384) operation to get an import ID and upload URL. The response from the CreateUploadUrl operation looks like this:

```json
{
  "importId": "string",
  "uploadUrl": "string"
}
```
3. Use the StartImport (p. 541) operation to upload the custom vocabulary file to the upload URL. The import ID specifies a specific import.

4. You can use the ListImports (p. 509) and DescribeImport (p. 446) operations to see the progress of importing your custom vocabulary.

Creating a custom vocabulary file

A custom vocabulary file is a tab-separated list of values that contain the phrase to recognize and a weight to give the boost. Phrases with a higher boost value are more likely to be used when they appear in the audio input.

The custom vocabulary file must be named `CustomVocabulary.tsv`, and must be compressed in a zip file before it can be imported. The zip file must be less than 300 MB in size. The maximum number of phrases in a custom vocabulary is 500.

- **phrase** 1–4 words that should be recognized. Separate words in the phrase with spaces. You can't have duplicate phrases in the file. The phrase field is required.
- **weight** – The degree to which the phrase recognition is boosted. The value is an integer 1, 2, or 3. If you don't specify a weight, the default value is 1. Decide on the weight based on how often the word isn't recognized in the transcription and on how rare the word is in the input.

The custom vocabulary file must contain a header row with the headers "phrase" and "weight". The headers can be in any order, but must use lower case letters.

The following is a custom vocabulary file. The required tab character to separate the phrase and the weight is represented by the text ":[TAB]". If you use this example, replace the text with a tab character.

<table>
<thead>
<tr>
<th>phrase</th>
<th>weight</th>
</tr>
</thead>
<tbody>
<tr>
<td>Newcastle</td>
<td>2</td>
</tr>
<tr>
<td>Hobart</td>
<td>2</td>
</tr>
<tr>
<td>U. Dub</td>
<td>1</td>
</tr>
<tr>
<td>W. S. U.</td>
<td>3</td>
</tr>
<tr>
<td>Issaquah</td>
<td></td>
</tr>
<tr>
<td>Kennewick</td>
<td></td>
</tr>
</tbody>
</table>

Using multiple values in a slot

**Note**

Multiple value slots are only supported in the English (US) language.

For some intents, you might want to capture multiple values for a single slot. For example, a pizza ordering bot might have an intent with the following utterance:

```
I want a pizza with {toppings}
```

The intent expects that the `{toppings}` slot contains a list of the toppings that the customer wants on their pizza, for example “pepperoni and pineapple”.

To configure a slot to capture multiple values, you set the `allowMultipleValues` field on the slot to true. You can set the field using the console or with the CreateSlot (p. 365) or UpdateSlot (p. 596) operation.
You can only mark slots with custom slot types as multi-value slots.

For a multi-value slot, Amazon Lex V2 returns a list of slot values in the response to the RecognizeText (p. 628) or RecognizeUtterance (p. 635) operation. The following is the slot information returned for the utterance "I want a pizza with pepperoni and pineapple" from the OrderPizza bot.

```json
"slots": {
  "toppings": {
    "shape": "List",
    "value": {
      "interpretedValue": "pepperoni and pineapple",
      "originalValue": "pepperoni and pineapple",
      "resolvedValues": [
        "pepperoni and pineapple"
      ]
    },
    "values": [
      {
        "shape": "Scalar",
        "value": {
          "interpretedValue": "pepperoni",
          "originalValue": "pepperoni",
          "resolvedValues": [
            "pepperoni"
          ]
        }
      },
      {
        "shape": "Scalar",
        "value": {
          "interpretedValue": "pineapple",
          "originalValue": "pineapple",
          "resolvedValues": [
            "pineapple"
          ]
        }
      }
    ]
  }
}
```

Multi-valued slots always return a list of values. When the utterance only contains one value, the list of values returned only contains one response.

Amazon Lex V2 recognizes multiple values separated by spaces, commas (,), and the conjunction "and". Multi-value slots work with both text and voice input.

You can use multi-valued slots in prompts. For example, you can set the confirmation prompt for an intent to

```
Would you like me to order your {toppings} pizza?
```

When Amazon Lex V2 sends the prompt to the user, it sends "Would you like me to order your pepperoni and pineapple pizza?"

Multi-valued slots support default values. For more information, see Using default slot values (p. 178).

You can use slot obfuscation to mask the values of a multi-value slot in conversation logs. When you obfuscate slot values, the value of each of the slot values is replaced with the name of the slot. For more information, see Obscuring slot values in logs (p. 231).
Using an AWS Lambda function

This section describes how to attach a Lambda function to a bot alias and the structure of the event data that Amazon Lex V2 provides to a Lambda function. Use this information to parse the input to your Lambda code. It also explains the format of the response that Amazon Lex V2 expects your Lambda function to return.

Amazon Lex V2 uses one Lambda function per bot alias per language instead of one Lambda function for each intent. To use an individual function for each intent, the Lambda function router section provides a function that you can use.

Topics
- Attaching a Lambda function to a bot alias (p. 145)
- Input event format (p. 146)
- Response format (p. 150)
- Lambda router function (p. 152)

Attaching a Lambda function to a bot alias

With Amazon Lex V2 you indicate that an intent should use a Lambda function for each intent. But you assign the Lambda function that your intents use for each language supported by a bot alias. That way you can create a Lambda function tailored to each language that your bot alias supports.

You indicate that an intent should use a Lambda function using the console or the CreateIntent (p. 337) or UpdateIntent (p. 572) operation. In the intent editor, you turn on Lambda functions in the Code hooks section of the editor.

You define the Lambda function to use in each bot alias. The same Lambda function is used for all intents in a language supported by the bot.

To choose a Lambda function to use with a bot alias

1. Open the Amazon Lex console at https://console.aws.amazon.com/lexv2/.
2. From the list of bots, choose the name of the bot that you want to use.
3. From Create versions and aliases for deployment, choose View aliases.
4. From the list of aliases, choose the name of the alias that you want to use.
5. From the list of supported languages, choose the language that the Lambda function is used for.
6. Choose the name of the Lambda function to use, then choose the version or alias of the function.
7. Choose Save to save your changes.
Input event format

The following is the general format of an Amazon Lex V2 event that is passed to a Lambda function. Use this information when you're writing your Lambda function.

**Note**
The input format may change without a corresponding change to the `messageVersion`. Your code shouldn't throw an error if new fields are present.

```json
{
    "messageVersion": "1.0",
    "invocationSource": "DialogCodeHook | FulfillmentCodeHook",
    "inputMode": "DTMF | Speech | Text",
    "responseContentType": "CustomPayload | ImageResponseCard | PlainText | SSML",
    "sessionId": "string",
    "inputTranscript": "string",
    "bot": {
        "id": "string",
        "name": "string",
        "aliasId": "string",
        "localeId": "string",
        "version": "string"
    },
    "interpretations": [
        {
            "intent": {
                "confirmationState": "Confirmed | Denied | None",
                "name": "string",
                "slots": {
                    "string": {
                        "value": {
                            "interpretedValue": "string",
                            "originalValue": "string",
                            "resolvedValues": [
                                "string"
                            ]
                        }
                    }
                }
            }
        }
    ]
}
```
"resolvedValues": [
  "string"
]
}
}
},
"state": "Failed | Fulfilled | FulfillmentInProgress | InProgress | ReadyForFulfillment | Waiting",
"kendraResponse": {
  // Only present when intent is KendraSearchIntent. For details, see
  // https://docs.aws.amazon.com/kendra/latest/dg/API_Query.html#API_Query_ResponseSyntax
}
},
"nluConfidence": {
  "score": number
},
"sentimentResponse": {
  "sentiment": "string",
  "sentimentScore": {
    "mixed": number,
    "negative": number,
    "neutral": number,
    "positive": number
  }
}
},
"proposedNextState": {
  "dialogAction": {
    "slotToElicit": "string",
    "type": "Close | ConfirmIntent | Delegate | ElicitIntent | ElicitSlot"
  },
  "intent": {
    "name": "string",
    "confirmationState": "Confirmed | Denied | None",
    "slots": {},
    "state": "Failed | Fulfilled | InProgress | ReadyForFulfillment | Waiting"
  }
},
"requestAttributes": {
  "string": "string"
},
"sessionState": {
  "activeContexts": [
    {
      "name": "string",
      "contextAttributes": {
        "string": "string"
      },
      "timeToLive": {
        "timeToLiveInSeconds": number,
        "turnsToLive": number
      }
    }
  ],
  "sessionAttributes": {
    "string": "string"
  },
  "runtimeHints": {
    "slotHints": {
      "string": {
        "runtimeHintValues": [

```
{
    "phrase": "string"
},
{
    "phrase": "string"
}
}
```

```
"dialogAction": {
    "slotToElicit": "string",
    "type": "Close | ConfirmIntent | Delegate | ElicitIntent | ElicitSlot"
},
"intent": {
    "confirmationState": "Confirmed | Denied | None",
    "name": "string",
    "slots": {
        "string": {
            "value": {
                "interpretedValue": "string",
                "originalValue": "string",
                "resolvedValues": [
                    "string"
                ]
            }
        }
    },
    "values": [
        {
            "shape": "List",
            "value": {
                "interpretedValue": "string",
                "originalValue": "string",
                "resolvedValues": [
                    "string"
                ]
            }
        },
        {
            "shape": "Scalar",
            "value": {
                "originalValue": "string",
                "interpretedValue": "string",
                "resolvedValues": [
                    "string"
                ]
            }
        }
    ]
},
"state": "Failed | Fulfilled | FulfillmentInProgress | InProgress | ReadyForFulfillment | Waiting",
"kendraResponse": {
    // Only present when intent is KendraSearchIntent. For details, see
```
Note the following additional information about the event fields:

- **invocationSource** – Indicates the action that called the Lambda function. When the source is `DialogCodeHook`, the Lambda function was called after input from the user. When the source is `FulfillmentCodeHook`, the Lambda function was called after all required slots have been filled and the intent is ready for fulfillment.

- **inputTranscript** – The text that was used to process the input from the user. For text or DTMF input, this is the text that the user typed. For speech input, this is the text that was recognized from the speech.

- **interpretations** – One or more intents that Amazon Lex V2 considers possible matches to the user’s utterance. For more information, see `interpretation` (p. 801).
• **proposedNextState** – The next state of the dialog between the user and the bot if the Lambda function doesn't change the flow. For more information, see proposedNextState structure (p. 150).

• **requestAttributes** – Request-specific attributes that the client sends in the request. Use request attributes to pass information that doesn't need to persist for the entire session.

• **sessionState** – The current state of the conversation between the user and your Amazon Lex V2 bot. For more information about the session state, see SessionState (p. 810).

• **transcriptions** – one or more transcriptions that Amazon Lex V2 considers possible matches to the user’s audio utterance. For more information, see Using voice transcription confidence scores (p. 187).

**proposedNextState structure**

The proposedNextState structure contains information about the next action that the bot will take if the Lambda function sets the dialogAction to Delegate. You can use the information to modify your Lambda function's behavior based on what Amazon Lex V2 proposes as the next action.

For example, suppose an intent has slots for first and last names. After filling in the first-name slot, the proposedNextState indicates the slot to elicit in the next turn is the last-name slot. You can query a database for last names associated with the first name and provide the results as runtime hints for the slot. For more information about runtime hints, see Using runtime hints to improve recognition of slot values (p. 194).

The proposedNextState structure contains the dialog action that Amazon Lex V2 will perform next. It also indicates the slot to elicit and the intent that the bot has determined that the user is trying to fulfill.

The proposedNextState structure is only present when the invocationSource field is DialogCodeHook and when the predicted dialog action is ElicitSlot.

The prediction in proposedNextState holds true only if the Lambda function sets the session state dialogAction to Delegate and if you don't override the dialog's behavior by setting the dialog action or intent. If you override the dialog behavior in sessionState, the next state depends on the settings that you return from your Lambda function.

**Response format**

Amazon Lex V2 expects a response from your Lambda function in the following format:

```json
{
  "sessionState": {
    "activeContexts": [
      {
        "name": "string",
        "contextAttributes": {
          "key": "value"
        },
        "timeToLive": {
          "timeToLiveInSeconds": number,
          "turnsToLive": number
        }
      }
    ],
    "sessionAttributes": {
      "string": "string"
    },
    "runtimeHints": {
      "slotHints": {
        "string": {
          "key": "value"
        }
      }
    }
  }
}
```
"string": {
  "runtimeHintValues": [
    {
      "phrase": "string"
    },
    {
      "phrase": "string"
    }
  ]
},
"dialogAction": {
  "slotElicitationStyle": "Default | SpellByLetter | SpellByWord",
  "slotToElicit": "string",
  "type": "Close | ConfirmIntent | Delegate | ElicitIntent | ElicitSlot"
},
"intent": {
  "confirmationState": "Confirmed | Denied | None",
  "name": "string",
  "slots": {
    "string": {
      "value": {
        "interpretedValue": "string",
        "originalValue": "string",
        "resolvedValues": [
          "string"
        ]
      }
    },
    "string": {
      "shape": "List",
      "value": {
        "originalValue": "string",
        "interpretedValue": "string",
        "resolvedValues": [
          "string"
        ]
      }
    }
  },
  "values": [
    {
      "shape": "Scalar",
      "value": {
        "originalValue": "string",
        "interpretedValue": "string",
        "resolvedValues": [
          "string"
        ]
      }
    },
    {
      "shape": "Scalar",
      "value": {
        "originalValue": "string",
        "interpretedValue": "string",
        "resolvedValues": [
          "string"
        ]
      }
    }
  ]
},
"state": "Failed | Fulfilled | FulfillmentInProgress | InProgress | ReadyForFulfillment | Waiting"
Note the following additional information about the response fields:

- **sessionState** – Required. The current state of the conversation with the user. The actual contents of the structure depends on the type of dialog action.
- **dialogAction** – Determines the type of action that Amazon Lex V2 should take in response to the Lambda function. The `type` field is always required. The `slotToElicit` field is required only when `dialogAction.type` is `ElicitSlot`.
- **intent** – The name of the intent that Amazon Lex V2 should use. Not required when `dialogAction.type` is `Delegate` or `ElicitIntent`.
- **state** – Required. The state can only be `ReadyForFulfillment` if `dialogAction.type` is `Delegate`.
- **messages** – Required if `dialogAction.type` is `ElicitIntent`. One or more messages that Amazon Lex V2 shows to the customer to perform the next turn of the conversation. If you don’t supply messages, Amazon Lex V2 uses the appropriate message defined when the bot was created. For more information, see the `Message` (p. 802) data type.
- **contentType** – The type of message to use.
- **content** – If the message type is `PlainText`, `CustomPayload`, or `SSML`, the `content` field contains the message to send to the user.
- **imageResponseCard** – If the message type is `ImageResponseCard`, contains the definition of the response card to show to the user. For more information, see the `ImageResponseCard` (p. 796) data type.

### Lambda router function

When you build a bot with the Amazon Lex V2 APIs, there is only one Lambda function per bot alias per language instead of a Lambda function for each intent. To use separate functions, you can create a router function that activates a separate function for each intent. The following is a router function that you can use or modify for your application.

```python
import os
import json
import boto3
```

```python
Lambda router function
```
# reuse client connection as global
client = boto3.client('lambda')

def router(event):
    intent_name = event['sessionState']['intent']['name']
    fn_name = os.environ.get(intent_name)
    print(f"Intent: {intent_name} -> Lambda: {fn_name}\n")
    if fn_name):
        # invoke lambda and return result
        invoke_response = client.invoke(FunctionName=fn_name, Payload = json.dumps(event))
        print(invoke_response)
        payload = json.load(invoke_response['Payload'])
        return payload
    raise Exception('No environment variable for intent: ' + intent_name)

def lambda_handler(event, context):
    print(event)
    response = router(event)
    return response
Using the Automated Chatbot Designer (Preview)

The Automated Chatbot Designer helps you design bots from existing conversation transcripts. It analyzes the transcripts and suggests an initial design with intents and slot types. You can iterate on the bot design, add prompts, build, test, and deploy the bot.

After you create a new bot or add a language to your bot using the Amazon Lex V2 console or API, you can upload transcripts of conversations between two parties. The automated chatbot designer analyzes the transcripts and determines the intents and slot types for the bot. It also labels the conversations that influenced the creation of a particular intent or slot type for your review.

You use the Amazon Lex V2 console or the API to analyze conversation transcripts and suggest intents and slot types for a bot.

You can review the suggested intents and slot types after the chatbot designer finishes the analysis. After you've added a suggested intent or slot type, you can modify it or delete it from the bot design using the console or the API.

The automated chatbot designer supports conversation transcript files using the Contact Lens for Amazon Connect schema. If you are using a different contact center application, you must transform the conversation transcripts to the format used by the chatbot designer. For information, see Input transcript format (p. 157).

To use the automated chatbot designer, you must allow the IAM role that is running the designer access. For the specific IAM policy, see Allow users to use the Automated Chatbot Designer (p. 268). To enable Amazon Lex V2 to encrypt output data with an optional AWS KMS key, you need to update the key with the policy shown in Allow users to use a AWS KMS key to encrypt and decrypt files (p. 269).

You can only analyze conversations in the English (US) language.

Topics
- Importing conversation transcripts (p. 154)
- Creating intents and slot types (p. 157)
- Input transcript format (p. 157)
- Output transcript format (p. 158)

Importing conversation transcripts

Importing conversation transcripts is a three-step process:

1. Prepare the transcripts for importing by converting them to the correct format. If you are using Contact Lens for Amazon Connect the transcripts are already in the correct format.
2. Upload the transcripts to an Amazon S3 bucket. If you are using Contact Lens, your transcripts are already in an S3 bucket.
3. Analyze the transcripts using the Amazon Lex V2 console or API operations. The time that it takes to complete training depends on the volume of transcripts and the complexity of the conversation. Typically, 15,000 lines of transcripts are analyzed in 30 minutes.
Each of these steps is described in the following sections.

Importing transcripts from Contact Lens for Amazon Connect

The Amazon Lex V2 automated chatbot designer is compatible with Contact Lens transcript files. To use Contact Lens transcript files, you must turn on Contact Lens and note the location of its output files.

To export transcripts from Contact Lens

1. Turn on Contact Lens in your Amazon Connect instance. For instructions, see Enable Contact Lens for Amazon Connect in the Amazon Connect administrator guide.
2. Note the location of the S3 bucket that Amazon Connect is using for your instance. To see the location, open the Data storage page in the Amazon Connect console. For instructions, see Update instance settings in the Amazon Connect administrator guide.

After you have turned on Contact Lens and noted the location of your transcript files, go to Analyze your transcripts using Amazon Lex V2 console (p. 156) for instructions to import and analyze your transcripts.

Prepare transcripts

Prepare your transcripts by creating transcript files.

- Create one transcript file per conversation listing the interaction between the parties. Each interaction in the conversation can span multiple lines. You can provide both redacted and non-redacted versions of the conversation. Each interaction in the conversation can span multiple lines.
- The file must be in the JSON format specified in Input transcript format (p. 157).
- You must provide at least 1,000 conversation turns. There is no limit to the number of transcript files that you use, nor is there a size restriction.

If you plan to filter the transcripts that you import by date, the files must be in the following directory structure:

```
<path or bucket root>
   --> yyyy
      --> mm
         --> dd
            --> transcript files
```

The transcript file must contain the date in the format "yyyy-mm-dd" somewhere in the file name.

To export transcripts from other contact center applications

1. Use your contact center application’s tools to export conversations. The conversation must contain at least the information specified in Input transcript format (p. 157).
2. Transform the transcripts produced by your contact center application to the format described in Input transcript format (p. 157). You are responsible for performing the transformation.

We provide two scripts for preparing transcripts. They are:

- A script to combine Contact Lens transcripts with Amazon Lex V2 conversation logs. Contact Lens transcripts don't include parts of Amazon Connect conversations that interact with Amazon Lex
Upload your transcripts to an S3 bucket

If you are using Contact Lens, your transcript files are already contained in an S3 bucket. For the location and file names of your transcript files, see Example Contact Lens output files in the Amazon Connect administrator guide.

If you are using another contact center application and you have not set up an S3 bucket for your transcript files, follow this procedure. Otherwise, if you have an existing S3 bucket, after logging in to the Amazon S3 console, follow this procedure starting with step 5.

To upload files to an S3 bucket
1. Sign in to the AWS Management Console and open the Amazon S3 console at https://console.aws.amazon.com/s3/.
2. Choose Create bucket.
3. Give the bucket a name and choose a Region. The Region must be the same one that you use for Amazon Lex V2. Set the other options as required for your use case.
4. Choose Create bucket.
5. From the list of buckets, choose an existing bucket or the bucket that you just created
7. Add the transcript files that you want to upload.
8. Choose Upload.

Analyze your transcripts using Amazon Lex V2 console

You can only use automated bot design in an empty language. You can add a new language to an existing bot, or create a new bot.

To create a new language in a new bot
1. Sign in to the AWS Management Console and open the Amazon Lex V2 console at https://console.aws.amazon.com/lexv2/.
2. Choose Create bot
3. Choose Start with transcripts. Fill out the information to create your new bot.
4. Choose Next
5. In Add language to bot fill out the information for the language.
6. In the Transcript file location on S3 section, choose the S3 bucket that contains your transcript files and the local path to the files if necessary.
7. You can optionally choose the following:
   - A AWS KMS key to encrypt the transcript data during processing. If you don't select a key, a service AWS KMS key is used.
To filter the transcripts to a specific date range. If you choose to filter the transcripts, they must be in the correct folder structure. For more information, see Prepare transcripts (p. 155).

8. Choose Done.

Wait for Amazon Lex V2 to process the transcript. You see a completion message when the analysis is complete.

Creating intents and slot types

After the chatbot designer creates intents and slot types, you select the intents and slot types to add to your bot. You can do this after the import finishes.

To review and add intents and slot type

1. Sign in to the AWS Management Console and open the Amazon Lex V2 console at https://console.aws.amazon.com/lexv2/.
2. From the list of bots, choose the bot you want to work with.
3. Choose View languages.
4. From the list of languages, choose the language to work with.
5. In Conversation structure, choose Review.
6. In the list of intents and slot types, choose the ones to add to the bot. You can choose an intent or slot type to see details and the associated transcripts.

Intents are sorted by the confidence that Amazon Lex V2 has that the intent is associated with the processed transcripts.

Input transcript format

The following is the input file format for generating intents and slot types for your bot. The input file must contain these fields. Other fields are ignored.

The input format is compatible with the output format from Contact Lens for Amazon Connect. If you are using Contact Lens, you don't need to modify your transcript files. For more information, see Example Contact Lens output files. If you are using another contact center application, you must transform your transcript file to this format.

```json
{
  "Participants": [
    {
      "ParticipantId": "string",
      "ParticipantRole": "AGENT | CUSTOMER"
    }
  ],
  "Version": "1.1.0",
  "ContentMetadata": {
    "RedactionTypes": ["PII"],
    "Output": "Raw | Redacted"
  },
  "CustomerMetadata": {
    "ContactId": "string"
  }
}
```
### Output transcript format

The output transcript format is nearly the same as the input transcript format. However, it also includes some customer metadata and a field listing segments that influenced the suggestion of intents and slot types. You can download the output transcript from the Review page in the console or using the Amazon Lex V2 API. For more information, see Input transcript format (p. 157).

```json
{
    "Participants": [  
        {  
            "ParticipantId": "string",
            "ParticipantRole": "AGENT | CUSTOMER"
        },
    ],
    "Version": "1.1.0",
    "ContentMetadata": {   
        "RedactionTypes": [  
            "PII"
        ],
        "Output": "Raw | Redacted"
    },
    "CustomerMetadata": {  
        "ContactId": "string",
        "FileName": "string",
        "InputFormat": "Lex"
    },
    "InfluencingSegments": [  
        {  
            "Id": "string",
            "StartTurnIndex": number,
            "EndTurnIndex": number,
            "Intents": [  
                {  
                    "Id": "string"
                }
            ],
            "SlotTypes": [  
                {  
                    "Id": "string"
                }
            ]
        }
    ]
}
```
- **CustomerMetadata** – There are two fields added to the `CustomerMetadata` field, the name of the input file that contains the conversation and the input format, which is always "Lex".

- **InfluencingSegments** – Identifies the segments of the conversation that influenced the suggestion of an intent or slot type. The ID of the intent or slot type identifies the specific one influenced by the conversation.
Deploying bots

This section provides examples of deploying Amazon Lex V2 bots on messaging platforms, mobile applications, and Web sites.

Topics

- Aliases (p. 160)
- Using a Java application to interact with an Amazon Lex V2 bot (p. 161)

Aliases

Amazon Lex V2 bots 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.

An alias is a pointer to a specific version of an Amazon Lex V2 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.

When you create a bot, Amazon Lex V2 creates an alias called TestBotAlias that you can use for testing your bot. The TestBotAlias alias is always associated with the Draft version of your bot. You should only use the TestBotAlias alias for testing, Amazon Lex V2 limits the number of runtime requests that you can make to the alias.

The following example shows two versions of an Amazon Lex V2 bot, version 1 and version 2. Each of these bot versions has an associated alias, BETA 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 `UpdateBotAlias (p. 553)` 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 Draft 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, Amazon Lex V2 will use the current version for all in-progress sessions. New sessions use the new version.

### Using a Java application to interact with an Amazon Lex V2 bot

The AWS SDK for Java provides an interface that you can use from your Java applications to interact with your bots. Use the SDK for Java to build client applications for users.

The following application interacts with the OrderFlowers bot that you created in Exercise 1: Create a bot from an example (p. 9). It uses the `LexRuntimeV2Client` from the SDK for Java to call the `RecognizeText (p. 628)` operation to conduct a conversation with the bot.

The output from the conversation looks like this:

```
User : I would like to order flowers
Bot  : What type of flowers would you like to order?
User : 1 dozen roses
Bot  : What day do you want the dozen roses to be picked up?
User : Next Monday
Bot  : At what time do you want the dozen roses to be picked up?
User : 5 in the evening
Bot  : Okay, your dozen roses will be ready for pickup by 17:00 on 2021-01-04. Does this sound okay?
```
Integrating with a Java application

User : Yes
Bot : Thanks.

For the JSON structures that are sent between the client application and the Amazon Lex V2 bot, see Exercise 2: Review the conversation flow (p. 10).

To run the application, you must provide the following information:

- **botId** – The identifier assigned to the bot when you created it. You can see the bot ID in the Amazon Lex V2 console on the bot **Settings** page.
- **botAliasId** – The identifier assigned to the bot alias when you created it. You can see the bot alias ID in the Amazon Lex V2 console on the **Aliases** page. If you can't see the alias ID in the list, choose the gear icon on the upper right and turn on **Alias ID**.
- **localeId** – The identifier of the locale that you used for your bot. For a list of locales, see Languages and locales supported by Amazon Lex V2 (p. 4).
- **accessKey** and **secretKey** – The authentication keys for your account. If you don't have a set of keys, create them using the AWS Identity and Access Management console.
- **sessionId** – An identifier for the session with the Amazon Lex V2 bot. In this case, the code uses a random UUID.
- **region** – If your bot is not in the US East (N. Virginia) Region, make sure that you change the Region.

The applications uses a function called getRecognizeTextRequest to create individual requests to the bot. The function builds a request with the required parameters to send to Amazon Lex V2.

```java
package com.lex.recognizetext.sample;

import software.amazon.awssdk.auth.credentials.AwsBasicCredentials;
import software.amazon.awssdk.auth.credentials.AwsCredentialsProvider;
import software.amazon.awssdk.auth.credentials.StaticCredentialsProvider;
import software.amazon.awssdk.regions.Region;
import software.amazon.awssdk.services.lexruntimev2.LexRuntimeV2Client;
import software.amazon.awssdk.services.lexruntimev2.model.RecognizeTextRequest;
import software.amazon.awssdk.services.lexruntimev2.model.RecognizeTextResponse;
import java.net.URISyntaxException;
import java.util.UUID;

/**
 * This is a sample application to interact with a bot using RecognizeText API.
 */
public class OrderFlowersSampleApplication {
    public static void main(String[] args) throws URISyntaxException, InterruptedException {
        String botId = "";
        String botAliasId = "";
        String localeId = "en_US";
        String accessKey = "";
        String secretKey = "";
        String sessionId = UUID.randomUUID().toString();
        Region region = Region.US_EAST_1; // pick an appropriate region

        AwsBasicCredentials awsCreds = AwsBasicCredentials.create(accessKey, secretKey);
        AwsCredentialsProvider awsCredentialsProvider = StaticCredentialsProvider.create(awsCreds);

        LexRuntimeV2Client lexV2Client = LexRuntimeV2Client
            .builder()
```

162
Integrating with a Java application

```java
.credentialsProvider(awsCredentialsProvider)
.region(region)
.build();

// utterance 1
String userInput = "I would like to order flowers";
RecognizeTextRequest recognizeTextRequest = getRecognizeTextRequest(botId, botAliasId, localeId, sessionId, userInput);
RecognizeTextResponse recognizeTextResponse = lexV2Client.recognizeText(recognizeTextRequest);
System.out.println("User : " + userInput);
recognizeTextResponse.messages().forEach(message -> {
    System.out.println("Bot : " + message.content());
});

// utterance 2
userInput = "1 dozen roses";
recognizeTextRequest = getRecognizeTextRequest(botId, botAliasId, localeId, sessionId, userInput);
recognizeTextResponse = lexV2Client.recognizeText(recognizeTextRequest);
System.out.println("User : " + userInput);
recognizeTextResponse.messages().forEach(message -> {
    System.out.println("Bot : " + message.content());
});

// utterance 3
userInput = "next monday";
recognizeTextRequest = getRecognizeTextRequest(botId, botAliasId, localeId, sessionId, userInput);
recognizeTextResponse = lexV2Client.recognizeText(recognizeTextRequest);
System.out.println("User : " + userInput);
recognizeTextResponse.messages().forEach(message -> {
    System.out.println("Bot : " + message.content());
});

// utterance 4
userInput = "5 in evening";
recognizeTextRequest = getRecognizeTextRequest(botId, botAliasId, localeId, sessionId, userInput);
recognizeTextResponse = lexV2Client.recognizeText(recognizeTextRequest);
System.out.println("User : " + userInput);
recognizeTextResponse.messages().forEach(message -> {
    System.out.println("Bot : " + message.content());
});

// utterance 5
userInput = "Yes";
recognizeTextRequest = getRecognizeTextRequest(botId, botAliasId, localeId, sessionId, userInput);
recognizeTextResponse = lexV2Client.recognizeText(recognizeTextRequest);
System.out.println("User : " + userInput);
recognizeTextResponse.messages().forEach(message -> {
    System.out.println("Bot : " + message.content());
});
}

private static RecognizeTextRequest getRecognizeTextRequest(String botId, String botAliasId, String localeId, String sessionId, String userInput) {
    RecognizeTextRequest recognizeTextRequest = RecognizeTextRequest.builder()
        .botAliasId(botAliasId)
        .botId(botId)
        .build();
    return recognizeTextRequest;
}
```
localeId(localeId)
/sessionId(sessionId)
/text(userInput)
.build();
    return recognizeTextRequest;
}
Integrating your bots

This section provides information about integrating your Amazon Lex V2 bots with messaging platforms and contact center applications.

Topics
- Integrating an Amazon Lex V2 bot with a messaging platform (p. 165)
- Integrating an Amazon Lex V2 bot with a contact center (p. 172)

Integrating an Amazon Lex V2 bot with a messaging platform

This section explains how to integrate Amazon Lex V2 bots with the Facebook, Slack, and Twilio messaging platforms.

Note
When storing your Facebook, Slack, or Twilio configurations, Amazon Lex V2 uses an AWS KMS key to encrypt information. The first time that you create a channel to one of these messaging platforms, Amazon Lex V2 creates a default customer managed key (aws/lex) in your AWS account or you can select your own customer managed key. Amazon Lex V2 supports only symmetric keys. For more information, see the AWS Key Management Service Developer Guide.

When a messaging platform sends a request to Amazon Lex V2 it includes platform-specific information as a request attribute to your Lambda function. Use this attribute to customize the way that your bot behaves. For more information, see Setting request attributes (p. 180).

Common request attribute

<table>
<thead>
<tr>
<th>Attribute</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>x-amz-lex:channels:platform</td>
<td>One of the following values:</td>
</tr>
<tr>
<td></td>
<td>• Facebook</td>
</tr>
<tr>
<td></td>
<td>• Slack</td>
</tr>
<tr>
<td></td>
<td>• Twilio</td>
</tr>
</tbody>
</table>

Integrating an Amazon Lex V2 bot with Facebook Messenger

You can host your Amazon Lex V2 bot in Facebook Messenger. When you do, Facebook users can interact with your bot to fulfill intents.

Before you start, you need to sign up for a Facebook developer account at https://developers.facebook.com.

You need to perform the following steps:

Topics
- Step 1: Create an Amazon Lex V2 bot (p. 166)
• Step 2: Create a Facebook application (p. 166)  
• Step 3: Integrate Facebook Messenger with the Amazon Lex V2 bot (p. 166)  
• Step 4: Complete Facebook integration (p. 167)  
• Step 5: Test the integration (p. 167)

**Step 1: Create an Amazon Lex V2 bot**

If you don't already have an Amazon Lex V2 bot, create one. In this topic, we assume that you are using the bot that you created in Exercise 1: Create a bot from an example (p. 9). However, you can use any bot.

**Next step**

**Step 2: Create a Facebook application**

On the Facebook developer portal, create a Facebook application and a Facebook page.

**To create a Facebook application**

1. Open https://developers.facebook.com/apps
2. Choose Create App.
3. In the Create an App, choose Manage Business Integrations, then choose Continue.
4. For the App Display Name, App Contact Email, and App Purpose make the appropriate choices for your app. Choose Create App to continue.
5. From Add Products to Your App, choose Set Up from the Messenger tile.
6. In the Access Tokens section, choose Add or Remove pages.
7. Choose a page to use with your app, then choose Next.
8. For What is app allowed to do, leave the defaults then choose Done.
9. On the confirmation page, choose OK.
10. In the Access Tokens section, choose Generate Token, then copy the token. You enter this token in the Amazon Lex V2 console.
11. From the left menu, choose Settings and then choose Basic.
12. For App Secret, choose Show and then copy the secret. You enter this token in the Amazon Lex V2 console.

**Next step**

**Step 3: Integrate Facebook Messenger with the Amazon Lex V2 bot**

In this step you link your Amazon Lex V2 bot with Facebook.

1. Sign in to the AWS Management Console and open the Amazon Lex V2 console at https://console.aws.amazon.com/lexv2/  
2. From the list of bots, choose the Amazon Lex V2 bot that you created in step 1.  
3. In the left menu, choose Channel integrations and then choose Add channel.  
4. In Create channel, do the following:
a. For **Platform**, choose **Facebook**.

b. For **Identity policies**, choose the AWS KMS key to protect channel information. The default key is provided by Amazon Lex V2.

c. For **Integration configuration**, give the channel a name and an optional description. Choose the alias that points to the version of the bot to use, and choose the language that the channel supports.

d. For **Additional configuration**, enter the following:

   - **Alias** – A string that identifies the app that is calling Amazon Lex V2. You can use any string. Record this string, you enter it in the Facebook developer console.
   - **Page access token** – The page access token that you copied from the Facebook developer console.
   - **App secret key** – The secret key that you copied from the Facebook developer console.

e. Choose **Create**

f. Amazon Lex V2 shows the list of channels for your bot. From the list, choose the channel that you just created.

g. From **Callback URL**, record the callback URL. You enter this URL in the Facebook developer console.

**Next step**

Step 4: Complete Facebook integration (p. 167)

**Step 4: Complete Facebook integration**

In this step, use the Facebook developer console to complete integration with Amazon Lex V2.

**To complete Facebook Messenger integration**

1. Open [https://developers.facebook.com/apps](https://developers.facebook.com/apps)
2. From the list of apps, choose the app that you are integrating with Facebook Messenger.
3. In the left menu, choose **Messenger**, then choose **Settings**.
4. In the **Webhooks** section, choose **Add Callback URL**.
5. In **Edit Callback URL**, enter the following:
   - **Callback URL** – Enter the callback URL that you recorded from the Amazon Lex V2 console.
   - **Verify Token** – Enter the alias that you entered in the Amazon Lex V2 console.
6. Choose **Verify and Save**.

**Next step**

Step 5: Test the integration (p. 167)

**Step 5: Test the integration**

You can now start a conversation from Facebook Messenger with your Amazon Lex V2 bot.

**To test the integration between Facebook Messenger and an Amazon Lex V2 bot**

1. Open the Facebook page that you associated with your bot in step 2.
2. In the Messenger window, use the test utterances provided in Exercise 1: Create a bot from an example (p. 9).
Integrating an Amazon Lex V2 bot with Slack

This topic provides instructions for integrating an Amazon Lex V2 bot with the Slack messaging application. You perform the following steps:

Topics
- Step 1: Create an Amazon Lex V2 bot (p. 168)
- Step 2: Sign up for Slack and create a Slack team (p. 168)
- Step 3: Create a Slack application (p. 168)
- Step 4: Integrate the Slack application with the Amazon Lex V2 bot (p. 169)
- Step 5: Complete Slack integration (p. 170)
- Step 6: Test the integration (p. 170)

Step 1: Create an Amazon Lex V2 bot

If you don’t already have an Amazon Lex V2 bot, create one. In this topic, we assume that you are using the bot that you created in Exercise 1: Create a bot from an example (p. 9). However, you can use any bot.

Next step

Step 2: Sign up for Slack and create a Slack team (p. 168)

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. 168)

Step 3: Create a Slack application

In this section, you do the following:

1. Create a Slack application in the Slack API Console.
2. Configure the application to add interactive messaging to your bot.

At the end of this section, you get application credentials (Client ID, Client Secret, and Verification Token). In the next step, you use this information to integrate the bot in the Amazon Lex V2 console.

To create a Slack application

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:
   - In the left menu, choose Interactivity & Shortcuts.
   - Choose the toggle to turn interactive components on.
In the Request URL box, specify any valid URL. For example, you can use https://slack.com.

Note
For now, enter any valid URL to 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 Save Changes.

4. In the left menu, in Settings, 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 V2 bot (p. 169)

Step 4: Integrate the Slack application with the Amazon Lex V2 bot

To integrate the Slack application with your Amazon Lex V2 bot

1. Sign in to the AWS Management Console and open the Amazon Lex V2 console at https://console.aws.amazon.com/lexv2/
2. From the list of bots, choose the Amazon Lex V2 bot that you created in step 1.
3. In the left menu, choose Channel integrations and then choose Add channel.
4. In Create channel, do the following:
   a. For Platform, choose Slack.
   b. For Identity policies, choose the AWS KMS key to protect channel information. The default key is provided by Amazon Lex V2.
   c. For Integration configuration, give the channel a name and an optional description. Choose the alias that points to the version of the bot to use, and choose the language that the channel supports.
   d. For Additional configuration, enter the following:
      - Client ID – enter the client ID from Slack.
      - Client secret – enter the client secret from Slack.
      - Verification token – enter the verification token from Slack.
      - Success page URL – The URL of the page that Slack should open when the user is authenticated. Typically you leave this blank.
5. Choose Create to create the channel.
6. Amazon Lex V2 shows the list of channels for your bot. From the list, choose the channel that you just created.
7. From Callback URL, record the endpoint and the OAuth endpoint.

Next step

Step 5: Complete Slack integration (p. 170)
Step 5: Complete Slack integration

In this section, use the Slack API console to complete integration with the Slack application.

To complete Slack application integration

2. Update the OAuth & Permissions feature as follows:
   a. In the left menu, choose OAuth & Permissions.
   b. In the Redirect URLs section, add the OAuth endpoint that Amazon Lex provided in the preceding step. Choose Add a new Redirect URL, and then choose Save URLs.
   c. In the Bot Token Scopes section, add two permissions with the Add an OAuth Scope button. Filter the list with the following text:
      - chat:write
      - team:read
3. Update the Interactivity & Shortcuts feature by updating the Request URL value to the endpoint that Amazon Lex provided in the preceding step. Enter the endpoint that you saved in step 4, and then choose Save Changes.
4. Subscribe to the Event Subscriptions feature as follows:
   • Enable events by choosing the On option.
   • Set the Request URL value to the endpoint that Amazon Lex provided in the preceding step.
   • In the Subscribe to Bot Events section, subscribe to the message.im bot event to enable direct messaging between the end user and the Slack bot.
   • Save the changes.
5. Enable sending messages from the messages tab as follows:
   • From the left menu, choose App Home.
   • In the Show Tabs section, choose Allow users to send Slash commands and messages from the messages tab.

Next step

Step 6: Test the integration (p. 170)

Step 6: Test the integration

Now use a browser window to test the integration of Slack with your Amazon Lex V2 bot.

To test your Slack application

1. Open the Slack console at http://api.slack.com. From the list of apps, choose the app to test.
2. Choose Manage Distribution under Settings. Choose Add to Slack to install the application. Choose Allow to authorize the bot to respond to messages.
3. You are redirected to your Slack team. From the left menu, in the Direct Messages section, choose your bot. If you don't see your bot, choose the plus icon (+) next to Direct Messages to search for it.
4. Engage in a chat with your Slack application. Your bot responds to messages.

If you created the bot using Getting started exercise 1, you can use the example conversations from that exercise.
Integrating an Amazon Lex V2 bot with Twilio SMS

This topic provides instructions for integrating an Amazon Lex V2 bot with the Twilio simple message service (SMS). You perform the following steps:

**Topics**
- Step 1: Create an Amazon Lex V2 bot (p. 171)
- Step 2: Create a Twilio SMS account (p. 171)
- Step 3: Integrate the Twilio message service endpoint with the Amazon Lex V2 bot (p. 171)
- Step 4: Complete Twilio integration (p. 172)
- Step 5: Test the integration (p. 172)

**Step 1: Create an Amazon Lex V2 bot**

If you don’t already have an Amazon Lex V2 bot, create one. In this topic, we assume that you are using the bot that you created in Exercise 1: Create a bot from an example (p. 9). However, you can use any bot.

Next step

**Step 2: Create a Twilio SMS account**

Sign up for a Twilio account and record the following account information:

- **ACCOUNT SID**
- **AUTH TOKEN**

For sign-up instructions, see [https://www.twilio.com/console](https://www.twilio.com/console).

Next step

**Step 3: Integrate the Twilio message service endpoint with the Amazon Lex V2 bot**

1. Sign in to the AWS Management Console and open the Amazon Lex V2 console at [https://console.aws.amazon.com/lexv2/](https://console.aws.amazon.com/lexv2/)
2. From the list of bots, choose the Amazon Lex V2 bot that you created in step 1.
3. In the left menu, choose Channel integrations and then choose Add channel.
4. In Create channel, do the following:
   a. For Platform, choose Twilio.
   b. For Identity policies, choose the AWS KMS key to protect channel information. The default key is provided by Amazon Lex V2.
   c. For Integration configuration, give the channel a name and an optional description. Choose the alias that points to the version of the bot to use, and choose the language that the channel supports.
   d. For Additional configuration, enter the account SID and authentication token from the Twilio dashboard.
5. Choose **Create**.
6. From the list of channels, choose the channel that you just created.
7. Copy the **Callback URL**.

**Next step**

**Step 4: Complete Twilio integration (p. 172)**

**Step 4: Complete Twilio integration**

Use the Twilio console to complete the integration of your Amazon Lex V2 bot with Twilio SMS.

2. From the left menu, choose **All Products & Services**, then choose **Phone Number**.
3. If you have a phone number, choose it. If you don't have a phone number, choose **Buy a Number** to get one.
4. In the **Messaging** section, in **A MESSAGE COMES IN**, enter the callback URL from the Amazon Lex V2 console.
5. Choose **Save**.

**Next step**

**Step 5: Test the integration (p. 172)**

**Step 5: Test the integration**

Use your mobile phone to test the integration between Twilio SMS and your bot. Using your mobile phone, send messages to the Twilio number.

If you created the bot using Getting started exercise 1, you can use the example conversations from that exercise.

**Integrating an Amazon Lex V2 bot with a contact center**

You can integrate Amazon Lex V2 bots with your contact centers to enable self-service use-cases using the Amazon Lex V2 streaming API. Use these bots as interactive voice response (IVR) agents on telephony or as a text-based chatbot integrated into your contact center. For more information about the streaming APIs, see [Streaming to an Amazon Lex V2 bot (p. 201)](#).

With streaming APIs, you can enable the following features:

- **Interruptions** ("barge-in") – Callers can interrupt the bot and answer a question before the prompt is complete. For more information, see [Enabling your bot to be interrupted by your user (p. 218)](#).
- **Wait and continue** – Callers can instruct the bot to wait if they need time for retrieving additional information during a call, such as a credit card number or a booking ID. For more information, see [Enabling the bot to wait for the user to provide more information (p. 219)](#).
- **DTMF support** – Callers can provide information via speech or DTMF interchangeably.
- **SSML support** – You can configure Amazon Lex V2 bot prompts using SSML tags for greater control over speech generation from text. For more information, see [Generating speech from SSML documents](#) in the [Amazon Polly developer guide](#).
• **Configurable timeouts** – You can configure how long to wait for customers to finish speaking before Amazon Lex V2 collects their speech input, such as answering a yes or no question, or providing a date or credit card number. For more information, see Configuring timeouts for capturing user input (p. 222).

**Amazon Chime SDK**

Use the Amazon Chime SDK to add real-time audio, video, screen sharing, and messaging capabilities to your web or mobile applications. The Amazon Chime SDK provides public switched telephone network (PSTN) audio service so that you can build custom telephony applications with an AWS Lambda function.

Amazon Chime PSTN audio is integrated with Amazon Lex V2. You can use this integration to access Amazon Lex V2 bots as interactive voice response (IVR) systems in contact centers for audio interactions. Use this to integrate Amazon Lex V2 using PSTN audio services in the following scenarios.

**Contact center integrations**—You can use the Amazon Chime Voice Connector and Amazon Chime SDK PSTN audio service to access Amazon Lex V2 bots. Use them in any contact center application that uses the session initiation protocol (SIP) for voice communications. This integration adds natural language voice conversation experiences to your existing on-premises or cloud-based contact center with SIP support. For a list of supported contact center platforms, see Amazon Chime voice connector resources.

The following diagram shows the integration between a contact center using SIP and Amazon Lex V2.

![Diagram of contact center integration with Amazon Chime and Amazon Lex V2](image)

**Direct telephony support**—You can build customized IVR solutions to directly access Amazon Lex V2 bots using a phone number provisioned in the Amazon Chime SDK.

For more information, see the following topics in the Amazon Chime SDK guide.

- SIP integration using an Amazon Chime voice connector
- Using the Amazon Chime SDK PSTN audio service
- Integrating Amazon Chime PSTN audio with Amazon Lex V2

When the Amazon Chime SDK sends a request to Amazon Lex V2, it includes platform-specific information to your Lambda function and conversation logs. Use this information to determine the contact center application that is sending traffic to your bot.

<table>
<thead>
<tr>
<th>Common request attribute</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>x-amz-lex-channels-platform</td>
<td>Amazon Chime SDK PSTN Audio</td>
</tr>
</tbody>
</table>

**Amazon Connect**

Amazon Connect is an omnichannel cloud contact center. You can set up a contact center in a new steps, add agents located anywhere, and start engaging with your customers. For more information, see Get started with Amazon Connect in the Amazon Connect administrator guide.

You can create personalized experiences for your customers use omnichannel communications. For example, you can offer chat and voice contact based on customer preference and estimated wait times.
Meanwhile agents can handle all customers from just one interface. For example, they can chat with customers, and create or respond to tasks as they are routed to them.

You can use Amazon Connect for audio interactions with your customers, or Amazon Connect Chat for text-only interactions.

For more information, see the following topics in the Amazon Connect administrator guide.

- What is Amazon Connect
- Add an Amazon Lex V2 bot
- Amazon Connect get customer input contact block

When a contact center sends a request to Amazon Lex V2, it includes platform-specific information as a request attribute to your Lambda function and conversation logs. Use this information to determine which contact center application is sending traffic to your bot.

**Common request attribute**

<table>
<thead>
<tr>
<th>Attribute</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>x-amz-lex:channels:platform</td>
<td>One of the following values:</td>
</tr>
<tr>
<td></td>
<td>• Connect</td>
</tr>
<tr>
<td></td>
<td>• Connect Chat</td>
</tr>
</tbody>
</table>

**Genesys Cloud**

Genesys Cloud is a suite of cloud services for enterprise communication, collaboration, and contact center management. Genesys Cloud is built on top of AWS and uses a distributed cloud environment that provides secure access to organizations around the work.

For more information, see the following pages on the Genesys Cloud website.

- About Genesys Cloud contact center
- About the Amazon Lex V2 integration

When a contact center sends a request to Amazon Lex V2 it includes platform-specific information as a request attribute to your Lambda function and conversation logs. Use this information to determine which contact center application is sending traffic to your bot.

**Common request attribute**

<table>
<thead>
<tr>
<th>Attribute</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>x-amz-lex:channels:platform</td>
<td>• Genesys Cloud</td>
</tr>
</tbody>
</table>

*Learn more*

- Power your contact center with Amazon Lex and Genesys Cloud
Managing conversations

After you build a bot, you integrate your client application with the Amazon Lex V2 runtime operations to hold conversations with your bot.

When a user starts a conversation with your bot, Amazon Lex V2 creates a session. A session encapsulates the information exchanged between your application and the bot. For more information, see Managing sessions with the Amazon Lex V2 API (p. 182).

A typical conversation involves a back and forth flow between the user and a bot. For example:

<table>
<thead>
<tr>
<th>User</th>
<th>Bot</th>
<th>User</th>
<th>Bot</th>
<th>User</th>
<th>Bot</th>
</tr>
</thead>
<tbody>
<tr>
<td>I'd like to make an appointment</td>
<td>What type of appointment would you like to schedule?</td>
<td>dental</td>
<td>When should I schedule your dental appointment?</td>
<td>Tomorrow</td>
<td>At what time do you want to schedule the dental appointment on 2021-01-01?</td>
</tr>
<tr>
<td>9 am</td>
<td>09:00 is available, should I go ahead and book your appointment?</td>
<td>Yes</td>
<td>Thank you. Your appointment has been set successfully.</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

When you use the RecognizeText (p. 628) or RecognizeUtterance (p. 635) operation, you must manage the conversation in your client application. When you use the StartConversation (p. 642) operation, Amazon Lex V2 manages the conversation for you.

To manage the conversation, you must send user utterances to the bot until the conversation reaches a logical end. The current conversation is captured in session state. The session state is updated after each user utterance. The session state contains the current state of the conversation and is returned by the bot in a response to each user utterance.

A conversation can be in any of the following states:

- **ElicitIntent** – Indicates that the bot has not yet determined the user’s intent.
- **ElicitSlot** – Indicates that the bot has detected the user’s intent and is gathering the required information to fulfill the intent.
• **ConfirmIntent** – Indicates that the bot is waiting for the user to confirm that the information collected is correct.
• **Closed** – Indicates that the user's intent is complete and that the conversation with the bot reached a logical end.

A user can specify a new intent after the first intent is completed. For more information, see *Managing conversation context* (p. 176).

An intent can have the one of the following states:

• **InProgress** – Indicates that the bot is gathering information necessary to complete the intent. This is in conjunction with the **ElicitSlot** conversation state.
• **Waiting** – Indicates that the user requested the bot to wait when the bot asked for information for a specific slot.
• **Fulfilled** – Indicates that the business logic in a Lambda function associated with the intent ran successfully.
• **ReadyForFulfillment** – Indicates that the bot gathered all of the information required to fulfill the intent and that the client application can run fulfillment business logic.
• **Failed** – Indicates that an intent has failed.

### Managing conversation context

*Conversation context* is information that the user, your client application, or a Lambda function provides to a Amazon Lex bot to fulfill and intent. Conversation context includes slot data that the user provides, request attributes set by the client application, and session attributes that the client application and Lambda functions create.

**Topics**

• Setting intent context (p. 176)
• Using default slot values (p. 178)
• Setting session attributes (p. 178)
• Setting request attributes (p. 180)
• Setting the session timeout (p. 180)
• Sharing information between intents (p. 181)
• Setting complex attributes (p. 181)

### Setting intent context

You can have Amazon Lex trigger intents based on *context*. A *context* is a state variable that can be associated with an intent when you define a bot. You configure the contexts for an intent when you create the intent using the console or using the **CreatIntent** (p. 337) operation. You can only use context in the English (US) (en-US) locale.

There are two types of relationships for contexts, output contexts and input contexts. An *output context* becomes active when an associated intent is fulfilled. An output context is returned to your application in the response from the **RecognizeText** (p. 628) or **RecognizeUtterance** (p. 635) operation, and it is set for the current session. After a context is activated, it stays active for the number of turns or time limit configured when the context was defined.
An input context specifies conditions under which an intent can be recognized. An intent can only be recognized during a conversation when all of its input contexts are active. An intent with no input contexts is always eligible for recognition.

Amazon Lex automatically manages the lifecycle of contexts that are activated by fulfilling intents with output contexts. You can also set active contexts in a call to the RecognizeText or RecognizeUtterance operation.

You can also set the context of a conversation using the Lambda function for the intent. The output context from Amazon Lex is sent to the Lambda function input event. The Lambda function can send contexts in its response. For more information, see Using an AWS Lambda function (p. 145).

For example, suppose you have an intent to book a rental car that is configured to return an output context called "book_car_fulfilled". When the intent is fulfilled, Amazon Lex sets the output context variable "book_car_fulfilled". Since "book_car_fulfilled" is an active context, an intent with the "book_car_fulfilled" context set as an input context is now considered for recognition, as long as a user utterance is recognized as an attempt to elicit that intent. You can use this for intents that only make sense after booking a car, such as emailing a receipt or modifying a reservation.

Output context

Amazon Lex makes an intent’s output contexts active when the intent is fulfilled. You can use the output context to control the intents eligible to follow up the current intent.

Each context has a list of parameters that are maintained in the session. The parameters are the slot values for the fulfilled intent. You can use these parameters to pre-populate slot values for other intents. For more information, see Using default slot values (p. 178).

You configure the output context when you create an intent with the console or with the CreateIntent (p. 337) operation. You can configure an intent with more than one output context. When the intent is fulfilled, all of the output contexts are activated and returned in the RecognizeText (p. 628) or RecognizeUtterance (p. 635) response.

When you define an output context you also define its time to live, the length of time or number of turns that the context is included in responses from Amazon Lex. A turn is one request from your application to Amazon Lex. Once the number of turns or the time has expired, the context is no longer active.

Your application can use the output context as needed. For example, your application can use the output context to:

- Change the behavior of the application based on the context. For example, a travel application could have a different action for the context "book_car_fulfilled" than "rental_hotel_fulfilled."
- Return the output context to Amazon Lex as the input context for the next utterance. If Amazon Lex recognizes the utterance as an attempt to elicit an intent, it uses the context to limit the intents that can be returned to ones with the specified context.

Input context

You set an input context to limit the points in the conversation where the intent is recognized. Intents without an input context are always eligible to be recognized.

You set the input contexts that an intent responds to using the console or the CreateIntent operation. An intent can have more than one input context.

For an intent with more than one input context, all contexts must be active to trigger the intent. You can set an input context when you call the RecognizeText (p. 628), RecognizeUtterance (p. 635), or PutSession (p. 623) operation.
You can configure the slots in an intent to take default values from the current active context. Default values are used when Amazon Lex recognizes a new intent but doesn’t receive a slot value. You specify the context name and slot name in the form #context-name.parameter-name when you define the slot. For more information, see Using default slot values (p. 178).

Using default slot values

When you use a default value, you specify a source for a slot value to be filled for new intents when no slot is provided by the user's input. This source can be previous dialog, request or session attributes, or a fixed value that you set at build-time.

You can use the following as the source for your default values.

- Previous dialog (contexts) – #context-name.parameter-name
- Session attributes – [attribute-name]
- Request attributes – <attribute-name>
- Fixed value – Any value that doesn't match the previous

When you use the CreateIntent (p. 337) operation to add slots to an intent, you can add a list of default values. Default values are used in the order that they are listed. For example, suppose you have an intent with a slot with the following definition:

```json
"slots": [
  {
    "botId": "string",
    "defaultValueSpec": {
      "defaultValueList": [
        {
          "defaultValue": "#book-car-fulfilled.startDate"
        },
        {
          "defaultValue": "[reservationStartDate]"
        }
      ]
    }
  }
]
```

When the intent is recognized, the slot named "reservation-start-date" has its value set to one of the following.

1. If the "book-car-fulfilled" context is active, the value of the "startDate" parameter is used as the default value.
2. If the "book-car-fulfilled" context is not active, or if the "startDate" parameter is not set, the value of the "reservationStartDate" session attribute is used as the default value.
3. If neither of the first two default values are used, then the slot doesn't have a default value and Amazon Lex will elicit a value as usual.

If a default value is used for the slot, the slot is not elicited even if it is required.

Setting session attributes

Session attributes contain application-specific information that is passed between a bot and a client application during a session. 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.

Use session attributes in your Lambda functions to initialize a bot and to 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 RecognizeText (p. 628) or RecognizeUtterance (p. 635) 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 and response cards 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": "Vivian"), Amazon Lex substitutes the name for the placeholder. It then sends a personalized prompt to the user, "Hey Vivian, which toppings would you like?"

Session attributes persist for the duration of the session. Amazon Lex stores them in an encrypted datastore until the session ends. The client can create session attributes in a request by calling either the RecognizeText (p. 628) or RecognizeUtterance (p. 635) 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 creates a session attribute, the stored attribute value is used any time that the client application doesn't include sessionAttribute field in a request to Amazon Lex.

For example, suppose you have two session attributes, {"x": "1", "y": "2"}. If the client calls the RecognizeText or RecognizeUtterance 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 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. If you pass a new set of session attributes, such as {"z": 3}, the existing values are removed and only the new value is kept. When an empty map, {}, is passed, stored values are erased.

To send session attributes to Amazon Lex, you create a string-to-string map of the attributes. The following shows how to map session attributes:

```json
{
   "attributeName": "attributeValue",
   "attributeName": "attributeValue"
}
```

For the RecognizeText operation, you insert the map into the body of the request using the sessionAttributes field of the sessionState structure, as follows:

```json
"sessionState": {
   "sessionAttributes": {
      "attributeName": "attributeValue",
      "attributeName": "attributeValue"
   }
}
```

For the PostContent operation, you base64 encode the map, and then send it as part of the x-amz-lex-session-state header.

If you are sending binary or structured data in a session attribute, you must first transform the data to a simple string. For more information, see Setting complex attributes (p. 181).
Setting request attributes

Request attributes contain request-specific information and apply only to the current request. A client application sends this information to Amazon Lex. Use request attributes to pass information that doesn't need to persist for the entire session. You can create your own request attributes or you can use predefined attributes. To send request attributes, use the `x-amz-lex-request-attributes` header in a RecognizeUtterance (p. 635) or the requestAttributes field in a RecognizeText (p. 628) request. Because request attributes don't persist across requests like session attributes do, they are not returned in RecognizeUtterance or RecognizeText responses.

**Note**
To send information that persists across requests, use session attributes.

Setting user-defined request attributes

A user-defined request attribute is data that you send to your bot in each request. You send the information in the `amz-lex-request-attributes` header of a RecognizeUtterance request or in the requestAttributes field of a RecognizeText request.

To send request attributes to Amazon Lex, you create a string-to-string map of the attributes. The following shows how to map request attributes:

```json
{
    "attributeName": "attributeValue",
    "attributeName": "attributeValue"
}
```

For the `PostText` operation, you insert the map into the body of the request using the requestAttributes field, as follows:

```json
"requestAttributes": {
    "attributeName": "attributeValue",
    "attributeName": "attributeValue"
}
```

For the `PostContent` operation, you base64 encode the map, and then send it as the `x-amz-lex-request-attributes` header.

If you are sending binary or structured data in a request attribute, you must first transform the data to a simple string. For more information, see Setting complex attributes (p. 181).

Setting the session timeout

Amazon Lex retains context information—slot data and session attributes—until a conversation session ends. To control how long a session lasts for a bot, set the session timeout. By default, session duration is 5 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 for slot data. For example, it asks 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 entire session. If the user returns to the session before it expires, they can provide the remaining slot data, and complete the purchase.

In the Amazon Lex console, you set the session timeout when you create a bot. With the AWS command line interface (AWS CLI) or API, you set the timeout when you create a bot with the `CreateBot` (p. 313) operation by setting the `idleSessionTTLInSeconds` field.
Sharing information between intents

Amazon Lex supports sharing information between intents. To share between intents, use output contexts or session attributes.

To use output contexts, you define an output context when you create or update an intent. When the intent is fulfilled, responses from Amazon Lex V2 contain the context and slot values from the intent as context parameters. You can use these parameters as default values in subsequent intents or in your application code or Lambda functions.

To use session attributes, you set the attributes in your Lambda or application code. For example, a user of the ShoeOrdering bot starts by ordering shoes. The bot 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. The bot can ask the user for slot data, such as order number and order date. When the bot has the required information, it returns the status of the order.

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 use the orderNumber session attribute to share information across intents and fulfill the GetOrderStatus intent. The bot does this by returning the status of the last order that the user placed.

Setting complex attributes

Session and request attributes are string-to-string maps of attributes and values. In many cases, you can use the string map to transfer attribute values between your client application and a bot. In some cases, however, you might need to transfer binary data or a complex structure that can't be easily converted to a string map. For example, the following JSON object represents an array of the three most populous cities in the United States:

```json
{
  "cities": [
    {
      "city": {
        "name": "New York",
        "state": "New York",
        "pop": "8537673"
      }
    },
    {
      "city": {
        "name": "Los Angeles",
        "state": "California",
        "pop": "3976322"
      }
    },
    {
      "city": {
        "name": "Chicago",
        "state": "Illinois",
        "pop": "2704958"
      }
    }
  ]
}
```

This array of data doesn't translate well to a string-to-string map. In such a case, you can transform an object to a simple string so that you can send it to your bot with the RecognizeText (p. 628) and RecognizeUtterance (p. 635) operations.
Managing sessions

For example, if you are using JavaScript, you can use the JSON.stringify operation to convert an object to JSON, and the JSON.parse operation to convert JSON text to a JavaScript object:

```javascript
// To convert an object to a string.
var jsonString = JSON.stringify(object, null, 2);
// To convert a string to an object.
var obj = JSON.parse(JSON string);
```

To send attributes with the RecognizeUtterance operation, you must base64 encode the attributes before you add them to the request header, as shown in the following JavaScript code:

```javascript
var encodedAttributes = new Buffer(attributeString).toString("base64");
```

You can send binary data to the RecognizeText and RecognizeUtterance operations by first converting the data to a base64-encoded string, and then sending the string as the value in the session attributes:

```javascript
"sessionAttributes" : {
   "binaryData": "base64 encoded data"
}
```

Managing sessions with the Amazon Lex V2 API

When a user starts a conversation with your bot, Amazon Lex V2 creates a session. The information exchanged between your application and Amazon Lex V2 makes up the session state for the conversation. When you make a request, the session is identified by an identifier that you specify. For more information about the session identifier, see the `sessionId` field in the RecognizeText (p. 628) or RecognizeUtterance (p. 635) operation.

You can modify the session state sent between your application and your bot. For example, you can create and modify session attributes that contain custom information about the session, and you can change the flow of the conversation by setting the dialog context to interpret the next utterance.

There are three ways that you can update session state.

- Pass the session information inline as part of a call to the RecognizeText or RecognizeUtterance operation.

- Use a Lambda function with the RecognizeText or RecognizeUtterance operation that is called after each turn of the conversation. For more information, see Using an AWS Lambda function (p. 145). The other is to use the Amazon Lex V2 runtime API in your application to make changes to the session state.

- Use operations that enable you to manage session information for a conversation with your bot. The operations are the PutSession (p. 623) operation, the GetSession (p. 618) operation, and the DeleteSession (p. 615) operation. You use these operations to get information about the state of your user's session with your bot, and to have fine-grained control over the state.

Use the GetSession operation when you want to get the current state of the session. The operation returns the current state of the session, including the state of the dialog with your user, any session attributes that have been set and slot values for the current intent and any other intents that Amazon Lex V2 has identified as possible intents that match the user's utterance.

The PutSession operation enables you to directly manipulate the current session state. You can set the session, including the type of dialog action that the bot will perform next and the messages that
Amazon Lex V2 sends to the user. This gives you control over the flow of the conversation with the bot. Set the dialog action type field to Delegate to have Amazon Lex V2 determine the next action for the bot.

You can use the PutSession operation to create a new session with a bot and set the intent that the bot should start with. You can also use the PutSession operation to change from one intent to another. When you create a session or change the intent you also can set session state, such as slot values and session attributes. When the new intent is finished, you have the option of restarting the prior intent.

The response from the PutSession operation contains the same information as the RecognizeUtterance operation. You can use this information to prompt the user for the next piece of information, just as you would with the response from the RecognizeUtterance operation.

Use the DeleteSession operation to remove an existing session and start over with a new session. For example, when you are testing your bot you can use the DeleteSession operation to remove test sessions from your bot.

The session operations work with your fulfillment Lambda functions. For example, if your Lambda function returns Failed as the fulfillment state you can use the PutSession operation to set the dialog action type to close and fulfillmentState to ReadyForFulfillment to retry the fulfillment step.

Here are some things that you can do with the session operations:

- Have the bot start a conversation instead of waiting for the user.
- Switch intents during a conversation.
- Return to a previous intent.
- Start or restart a conversation in the middle of the interaction.
- Validate slot values and have the bot re-prompt for values that are not valid.

Each of these are described further below.

**Starting a new session**

If you want to have the bot start the conversation with your user, you can use the PutSession operation.

- Create a welcome intent with no slots and a conclusion message that prompts the user to state an intent. For example, "What would you like to order? You can say 'Order a drink' or 'Order a pizza.'"
- Call the PutSession operation. Set the intent name to the name of your welcome intent and set the dialog action to Delegate.
- Amazon Lex will respond with the prompt from your welcome intent to start the conversation with your user.

**Switching intents**

You can use the PutSession operation to switch from one intent to another. You can also use it to switch back to a previous intent. You can use the PutSession operation to set session attributes or slot values for the new intent.

- Call the PutSession operation. Set the intent name to the name of the new intent and set the dialog action to Delegate. You can also set any slot values or session attributes required for the new intent.
- Amazon Lex will start a conversation with the user using the new intent.
Resuming a prior intent

To resume a prior intent you use the GetSession operation to get the state of the intent, perform the needed interaction, and then use the PutSession operation to set the intent to its previous dialog state.

- Call the GetSession operation. Store the state of the intent.
- Perform another interaction, such as fulfilling a different intent.
- Using the information saved information for the previous intent, call the PutSession operation. This will return the user to the previous intent in the same place in the conversation.

In some cases it may be necessary to resume your user's conversation with your bot. For example, say that you have created a customer service bot. Your application determines that the user needs to talk to a customer service representative. After talking to the user, the representative can direct the conversation back to the bot with the information that they collected.

To resume a session, use steps similar to these:

- Your application determines that the user needs to speak to a customer service representative.
- Use the GetSession operation to get the current dialog state of the intent.
- The customer service representative talks to the user and resolves the issue.
- Use the PutSession operation to set the dialog state of the intent. This may include setting slot values, setting session attributes, or changing the intent.
- The bot resumes the conversation with the user.

Validating slot values

You can validate responses to your bot using your client application. If the response isn't valid, you can use the PutSession operation to get a new response from your user. For example, suppose that your flower ordering bot can only sell tulips, roses, and lilies. If the user orders carnations, your application can do the following:

- Examine the slot value returned from the PostText or PostContent response.
- If the slot value is not valid, call the PutSession operation. Your application should clear the slot value, set the slotToElicit field, and set the dialogAction.type value to elicitSlot. Optionally, you can set the message and messageFormat fields if you want to change the message that Amazon Lex uses to elicit the slot value.

Analyzing the sentiment of user utterances

You can use sentiment analysis to determine the sentiments expressed in a user utterance. With the sentiment information you can manage conversation flow or perform post-call analysis. For example, if the user sentiment is negative you can create a flow to hand over a conversation to a human agent.

Amazon Lex integrates with Amazon Comprehend to detect user sentiment. The response from Amazon Comprehend indicates whether the overall sentiment of the text is positive, neutral, negative, or mixed. The response contains the most likely sentiment for the user utterance and the scores for each of the sentiment categories. The score represents the likelihood that the sentiment was correctly detected.

You enable sentiment analysis for a bot using the console or by using the Amazon Lex API. You enable sentiment analysis on an alias for the bot. On the Amazon Lex console:
Using confidence scores

Amazon Lex V2 uses two steps to determine what a user says. The first, automatic speech recognition (ASR), creates a transcript of the user's audio utterance. The second, natural language understanding (NLU), determines the meaning of the user's utterance to recognize the user's intent or the value of slots.

By default, Amazon Lex V2 returns the most likely result from ASR and NLU. At times it may be difficult for Amazon Lex V2 to determine the most likely result. In that case, it returns several possible results along with a confidence score that indicates how likely the result is correct. A confidence score is a rating that Amazon Lex V2 provides that shows the relative confidence that it has in the result. Confidence scores range from 0.0 to 1.0.

You can use your domain knowledge with the confidence score to help determine the correct interpretation of the ASR or NLU result.

The ASR, or transcription, confidence score is a rating on how confident Amazon Lex V2 is that a particular transcription is correct. The NLU, or intent, confidence score is a rating on how confident Amazon Lex V2 is that the intent specified by the top transcription is correct. Use the confidence score that best fits your application.
Using intent confidence scores

When a user makes an utterance, Amazon Lex V2 uses natural language understanding (NLU) to understand the user's request and return the proper intent. By default, Amazon Lex V2 returns the most likely intent defined by your bot.

In some cases, it may be difficult for Amazon Lex V2 to determine the most likely intent. For example, the user might make an ambiguous utterance, or there might be two intents that are similar. To help determine the proper intent, you can combine your domain knowledge with the NLU confidence scores in a list of interpretations. A confidence score is a rating that Amazon Lex V2 provides that shows how confident it is that an intent is the correct intent.

To determine the difference between two intents within an interpretation, you can compare their confidence scores. For example, if one intent has a confidence score of 0.95 and another has a score of 0.65, the first intent is probably correct. However, if one intent has a score of 0.75 and another has a score of 0.72, there is ambiguity between the two intents that you may be able to discriminate using domain knowledge in your application.

You can also use confidence scores to create test applications that determine if changes to an intent's utterances make a difference in the behavior of the bot. For example, you can get the confidence scores for a bot's intents using a set of utterances, then update the intents with new utterances. You can then check the confidence scores to see if there was an improvement.

The confidence scores that Amazon Lex V2 returns are comparative values. You should not rely on them as an absolute score. The values may change based on improvements to Amazon Lex V2.

Amazon Lex V2 returns the most likely intent and up to 4 alternative intents with their associated scores in the interpretations structure in each response. If all of the confidence scores are less than a threshold, Amazon Lex V2 includes the AMAZON.FallbackIntent. If you have the AMAZON.KendraSearchIntent configured, Amazon Lex V2 returns it as well. You can use the default threshold or you can set your own threshold.

The following JSON code shows the interpretations structure in the response from the RecognizeText (p. 628) operation.

```json
"interpretations": [
  { "intent": {
    "confirmationState": "string",
    "name": "string",
    "slots": {
      "string": {
        "value": {
          "interpretedValue": "string",
          "originalValue": "string",
          "resolvedValues": [ "string" ]
        }
      }
    },
    "state": "string"
  },
  "nluConfidence": {
    "score": number
  }
]"
To change the confidence threshold, set it in the console or using the UpdateBotLocale (p. 559) operation. The threshold must be a number between 1.00 and 0.00.

To use the console, set the confidence threshold when you create or update your bot. You set the confidence threshold for each language that you add to your bot.

**To set the confidence threshold when creating a bot (Console)**

- On Add language, enter a value in the Confidence score threshold field.

**To set or update the confidence threshold (SDK)**

- Set the nluIntentConfidenceThreshold parameter of the CreateBotLocale (p. 324) operation. The following JSON code shows the parameter being set.

```
"nluIntentConfidenceThreshold": 0.75,
```

**Session Management**

To change the intent that Amazon Lex V2 uses in a conversation with the user, you can use the response from your dialog code hook Lambda function, or you can use the session management APIs in your custom application.

**Using a Lambda function**

When you use a Lambda function, Amazon Lex V2 calls it with a JSON structure that contains the input to the function. The JSON structure contains a field called currentIntent that contains the intent that Amazon Lex V2 has identified as the most likely intent for the user's utterance. The JSON structure also includes an alternativeIntents field that contains up to four additional intents that may satisfy the user's intent. Each intent includes a field called nluIntentConfidenceScore that contains the confidence score that Amazon Lex V2 assigned to the intent.

To use an alternative intent, you specify it in the ConfirmIntent or the ElicitSlot dialog action in your Lambda function.

For more information, see Using an AWS Lambda function (p. 145).

**Using the Session Management API**

To use a different intent from the current intent, use the PutSession (p. 623) operation. For example, if you decide that the first alternative is preferable to the intent that Amazon Lex V2 chose, you can use the PutSession operation to change intents so that the next intent that the user interacts with is the one that you selected.

For more information, see Managing sessions with the Amazon Lex V2 API (p. 182).

**Using voice transcription confidence scores**

When a user makes a voice utterance, Amazon Lex V2 uses automatic speech recognition (ASR) to transcribe the user's request before it is interpreted. By default, Amazon Lex V2 uses the most likely transcription of the audio for interpretation.
In some cases there might be more than one possible transcription of the audio. For example, a user might make an utterance with an ambiguous sound, such as "My name is John" that might be understood as "My name is Juan." In this case, you can use disambiguation techniques or combine your domain knowledge with the transcription confidence score to help determine which transcription in a list of transcriptions is the correct one.

Amazon Lex V2 includes the top transcription and up to two alternate transcriptions for user input in the request to your Lambda code hook function. Each transcription contains a confidence score that it is the correct transcription. Each transcription also includes any slot values inferred from the user input.

You can compare the confidence scores of two transcriptions to determine if there is ambiguity between them. For example, if one transcription has a confidence score of 0.95 and the other has a confidence score of 0.65, the first transcription is probably correct and the ambiguity between them is low. If the two transcriptions have confidence scores of 0.75 and 0.72, the ambiguity between them is high. You may be able to discriminate between them using your domain knowledge.

For example, if the inferred slot values in two transcripts with a confidence score of 0.75 and 0.72 are "John" and "Juan", you can query the users in your database for the existence of these names and eliminate one of the transcriptions. If "John" isn't a user in your database and "Juan" is, you can use the dialog code hook to change the inferred slot value for the first name to "Juan."

The confidence scores that Amazon Lex V2 returns are comparative values. Don't rely on them as an absolute score. The values may change based on improvements to Amazon Lex V2.

Audio transcription confidence scores are available only in the English (GB) (en_GB) and English (US) (en_US) languages. Confidence scores are supported only for 8 kHz audio input. Transcription confidence scores aren't provided for audio input from the test window on the Amazon Lex V2 console because it uses 16 kHz audio input.

**Note**
Before you can use audio transcription confidence scores with an existing bot, you must first rebuild the bot. Existing versions of a bot don't support transcription confidence scores. You must create a new version of the bot to use them.

You can use confidence scores for multiple conversation design patterns:

- If the highest confidence score falls below a threshold due to a noisy environment or poor signal quality, you can prompt the user with the same question to capture better quality audio.
- If multiple transcriptions have similar confidence scores for slot values, such as "John" and "Juan," you can compare the values with a pre-existing database to eliminate inputs, or you can prompt the user to select one of the two values. For example, "say 1 for John or say 2 for Juan."
- If your business logic requires intent switching based on specific keywords in an alternative transcript with a confidence score close to the top transcript, you can change the intent using your dialog code hook Lambda function or using session management operations. For more information, see Session management (p. 190).

Amazon Lex V2 sends the following JSON structure with up to three transcriptions for the user's input to your Lambda code hook function:

```json
"transcriptions": [
  
  
  "transcription": "string",
  "transcriptionConfidence": {
      "score": "number"
  },
  "resolvedContext": {
      "intent": "string"
```
The JSON structure contains transcription text, the intent that was resolved for the utterance, and values for any slots detected in the utterance. For text user input, the transcriptions contain a single transcript with a confidence score of 1.0.

The contents of the transcripts depend on the turn of the conversation and the recognized intent.

For the first turn, intent elicitation, Amazon Lex V2 determines the top three transcriptions. For the top transcription, it returns the intent and any inferred slot values in the transcription.

On subsequent turns, slot elicitation, the results depend on the inferred intent for each of the transcriptions, as follows.

- If the inferred intent for the top transcript is the same as the previous turn and all other transcripts have the same intent, then
  - All transcripts contain inferred slot values.

- If the inferred intent for the top transcript is different from the previous turn and all other transcripts have the previous intent, then
  - The top transcript contains the inferred slot values for the new intent.
  - Other transcripts have the previous intent and inferred slot values for the previous intent.

- If the inferred intent for the top transcript is different from the previous turn, one transcript is the same as the previous intent, and one transcript is a different intent, then
Using voice transcription confidence scores

- The top transcript contains the new inferred intent and any inferred slot values in the utterance.
- The transcript that has the previous inferred intent contains inferred slot values for that intent.
- The transcript with the different intent has no inferred intent name and no inferred slot values.

- If the inferred intent for the top transcript is the different from the previous turn and all other transcripts have different intents, then
  - The top transcript contains the new inferred intent and any inferred slot values in the utterance.
  - Other transcripts contain no inferred intents and no inferred slot values.

- If the inferred intent for the top two transcripts is the same and different from the previous turn, and the third transcript is a different intent, then
  - The top two transcripts contain the new inferred intent and any inferred slot values in the utterance.
  - The third transcript has no intent name and no resolved slot values.

Session management

To change the intent that Amazon Lex V2 uses in a conversation with the user, use the response from your dialog code hook Lambda function. Or you can use the session management APIs in your custom application.

Using a Lambda function

When you use a Lambda function, Amazon Lex V2 calls it with a JSON structure that contains the input to the function. The JSON structure contains a field called transcriptions that contains the possible transcriptions that Amazon Lex V2 has determined for the utterance. The transcriptions field contains one to three possible transcriptions, each with a confidence score.

To use the intent from an alternative transcription, you specify it in the ConfirmIntent or the ElicitSlot dialog action in your Lambda function. To use a slot value from an alternative transcription, set the value in the intent field in your Lambda function response. For more information, see Using an AWS Lambda function (p. 145).

Example code

The following code example is a Python Lambda function that uses audio transcriptions to improve the conversation experience for the user.

To use the example code, you must have:

- A bot with one language, either English (GB) (en_GB) or English (US) (en_US).
- One intent, OrderBirthStone. Make sure that the Use a Lambda function for initialization and validation is selected in the Code hooks section of the intent definition.
- The intent should have two slots, "BirthMonth" and "Name," both of type AMAZON.AlphaNumeric.
- An alias with the Lambda function defined. For more information, see Attaching a Lambda function to a bot alias (p. 145).

```python
import time
import os
import logging
```

190
logger = logging.getLogger()
logger.setLevel(logging.DEBUG)

# --- Helpers that build all of the responses ---
def elicit_slot(session_attributes, intent_request, slots, slot_to_elicit, message):
    return {
        'sessionState': {
            'dialogAction': {
                'type': 'ElicitSlot',
                'slotToElicit': slot_to_elicit
            },
            'intent': {
                'name': intent_request['sessionState']['intent']['name'],
                'slots': slots,
                'state': 'InProgress'
            },
            'sessionAttributes': session_attributes,
            'originatingRequestId': 'e3ab4d42-fb5f-4cc3-bb78-caaf6fc7cccd'
        },
        'sessionId': intent_request['sessionId'],
        'messages': [message],
        'requestAttributes': intent_request['requestAttributes'] if 'requestAttributes' in intent_request else None
    }

def close(intent_request, session_attributes, fulfillment_state, message):
    intent_request['sessionState']['intent']['state'] = fulfillment_state
    return {
        'sessionState': {
            'dialogAction': {
                'type': 'Close'
            },
            'intent': intent_request['sessionState']['intent'],
            'originatingRequestId': '3ab4d42-fb5f-4cc3-bb78-caaf6fc7cccd'
        },
        'messages': [message],
        'sessionId': intent_request['sessionId'],
        'requestAttributes': intent_request['requestAttributes'] if 'requestAttributes' in intent_request else None
    }

def delegate(intent_request, session_attributes):
    return {
        'sessionState': {
            'dialogAction': {
                'type': 'Delegate'
            },
            'intent': intent_request['sessionState']['intent'],
            'sessionAttributes': session_attributes,
            'originatingRequestId': 'abc'
        },
        'sessionId': intent_request['sessionId'],
        'requestAttributes': intent_request['requestAttributes'] if 'requestAttributes' in intent_request else None
    }

def get_session_attributes(intent_request):
    sessionState = intent_request['sessionState']
    if 'sessionAttributes' in sessionState:
        return sessionState['sessionAttributes']
def get_slots(intent_request):
    return intent_request['sessionState']['intent']['slots']

def order_birth_stone(intent_request):
    
    Perform the dialog management and fulfillment for ordering a birth stone.
    Beyond fulfillment, the implementation for this intent demonstrates the following:
    1) Use of N best transcriptions to re prompt user when confidence for top transcript is
    below threshold
    2) Overrides resolved slot for birth month from a known fixed list if the top
    transcript is not accurate.
    
    transcriptions = intent_request['transcriptions']

    if intent_request['invocationSource'] == 'DialogCodeHook':
        # Disambiguate if there are multiple transcriptions and the top transcription
        # confidence is below a threshold (0.8 here)
        if len(transcriptions) > 1 and transcriptions[0]['transcriptionConfidence'] < 0.8:
            if transcriptions[0]['resolvedSlots'] is not {} and 'Name' in transcriptions[0]['resolvedSlots'] and
                transcriptions[0]['resolvedSlots']['Name'] is not None:
                return prompt_for_name(intent_request)
            elif transcriptions[0]['resolvedSlots'] is not {} and 'BirthMonth' in
                transcriptions[0]['resolvedSlots'] and
                transcriptions[0]['resolvedSlots']['BirthMonth'] is not None:
                return validate_month(intent_request)

    return continue_conversation(intent_request)

def prompt_for_name(intent_request):
    
    If the confidence for the name is not high enough, re prompt the user with the
    recognized names
    so it can be confirmed.
    
    resolved_names = []
    for transcription in intent_request['transcriptions']:
        if transcription['resolvedSlots'] is not {} and 'Name' in
            transcription['resolvedSlots'] and
            transcription['resolvedSlots']['Name'] is not None:
                resolved_names.append(transcription['resolvedSlots']['Name']['value']
                   ['originalValue'])
        if len(resolved_names) > 1:
            session_attributes = get_session_attributes(intent_request)
            slots = get_slots(intent_request)
            return elicit_slot(session_attributes, intent_request, slots, 'Name',
                {'contentType': 'PlainText',
                 'content': 'Sorry, did you say your name is {} ?'.format(' or
                     '.join(resolved_names))})

    else:
        return continue_conversation(intent_request)

def validate_month(intent_request):
    
    # Implement validation logic here
    return continue_conversation(intent_request)
Validate month from an expected list, if not valid looks for other transcriptions and to see if the month recognized there has an expected value. If there is, replace with that and if not continue conversation.

```python
expected_months = ['january', 'february', 'march']
resolved_months = []
for transcription in intent_request['transcriptions']:
    if transcription['resolvedSlots'] is not {} and 'BirthMonth' in transcription['resolvedSlots'] and transcription['resolvedSlots']['BirthMonth'] is not None:
        resolved_months.append(transcription['resolvedSlots']['BirthMonth']['value']['originalValue'])

for resolved_month in resolved_months:
    if resolved_month in expected_months:
        intent_request['sessionState']['intent']['slots']['BirthMonth']['resolvedValues'] = [resolved_month]
        break

return continue_conversation(intent_request)
```

def continue_conversation(event):
    session_attributes = get_session_attributes(event)
    if event['invocationSource'] == "DialogCodeHook":
        return delegate(event, session_attributes)

# --- Intents ---

def dispatch(intent_request):
    ""
    Called when the user specifies an intent for this bot.
    ""
    logger.debug('dispatch sessionId={}, intentName={}'.format(intent_request['sessionId'], intent_request['sessionState']['intent']['name']))
    intent_name = intent_request['sessionState']['intent']['name']

    # Dispatch to your bot's intent handlers
    if intent_name == 'OrderBirthStone':
        return order_birth_stone(intent_request)
    raise Exception('Intent with name ' + intent_name + ' not supported')

# --- Main handler ---

def lambda_handler(event, context):
    ""
    Route the incoming request based on intent.
    The JSON body of the request is provided in the event slot.
    ""
    # By default, treat the user request as coming from the America/New_York time zone.
    os.environ['TZ'] = 'America/New_York'
    time.tzset()
    logger.debug('event={}'.format(event))
Using the session management API

To use a different intent from the current intent, use the PutSession (p. 623) operation. For example, if you decide that the first alternative is preferable to the intent that Amazon Lex V2 chose, you can use the PutSession operation to change intents. That way the next intent that the user interacts with will be the one that you selected.

You can also use the PutSession operation to change the slot value in the intent structure to use a value from an alternative transcription.

For more information, see Managing sessions with the Amazon Lex V2 API (p. 182).

Using runtime hints to improve recognition of slot values

With runtime hints you can give Amazon Lex V2 a set of slot values based on context to get better recognition in audio conversations and improved slot resolutions. You can use runtime hints to provide a list of phrases at runtime that become candidates for the resolution of a slot value.

For example, if a user interacting with a flight reservation bot frequently travels to San Francisco, Jakarta, Seoul, and Moscow you can configure runtime hints with a list of these four cities when eliciting for the destination to improve recognition for frequently travelled cities.

Runtime hints are only available in the English (UK) (en_GB) language. They can be used with the following slot types:

- Custom slot types
- AMAZON.City
- AMAZON.Country
- AMAZON.FirstName
- AMAZON.LastName
- AMAZON.State
- AMAZON.StreetName

Runtime hints basics

- Runtime hints are used only when eliciting a slot value from a user.
- When you use runtime hints, the values of the hints are preferred over similar values. For example, for a food ordering bot, you can set a list of menu items as runtime hints while eliciting for food items in a custom slot to prefer “fillet” over similar sounding “fella”.
- If the user input is different from the values provided in runtime hints, the original user input will be used for the slot.
- For custom slot types, values provided as runtime hints will be used for resolution of the slot even if they are not part of the custom slot during bot creation.
- Runtime hints are supported only for 8 kHz audio input. They are available with contact center integrations supported by Amazon Lex V2. Runtime hints aren't provided for audio input from the test window on the Amazon Lex V2 console because it uses 16 kHz audio input.
Adding slot values in context

Add context for your bot by providing expected slot values as runtime hints when your application has information about the user’s next likely utterance. This enables your bot to use information that your application has to improve the conversation with the user.

For example, in a banking app you can generate a list of account nicknames for a specific user, and then use the list when eliciting the account that the user wants to access.

Send runtime hints at the start of the conversation when you have context to help your bot interpret user input. For example, if you have the user’s phone number, you can use this information to look up the user so that you can use the PutSession or StartConversation operation to pass first and last name hints to the bot if you are eliciting for the user’s name to validate their credentials.

During a conversation, you might gather information from one slot value that can help with another slot value. For example, in a car care app when you have the user’s account number you can do a look up to find the cars that the customer owns and pass them along as hints to another slot.

Enter acronyms, or other words whose letters should be pronounced individually, as single letters separated by a period and a space. Don’t use individual letters unless they are part of a phrase, such as "J. P. Morgan" or "A.W.S". You can use upper- or lower-case letters to define an acronym.

Adding hints to a slot

To add runtime hints to a slot, you use the runtimeHints structure that is part of the sessionState structure. The following is an example of the runtimeHints structure. It provides hints for two slots, "FirstName" and "LastName" for the "MakeAppointment" intent.

```json
{
   "sessionState": {
      "intent": {},
      "activeContexts": [],
      "dialogAction": {},
      "originatingRequestId": {},
      "sessionAttributes": {},
      "runtimeHints": {
         "slotHints": {
```

Using spelling styles to capture slot values

Amazon Lex V2 provides built-in slots to capture user-specific information such as first name, last name, email address or alphanumeric identifiers. For example, you can use the AMAZON.LastName slot to capture surnames such as "Jackson" or "Garcia." However, Amazon Lex V2 may get confused with surnames that are difficult to pronounce or that are not common in a locale, such as "Xiulan." To capture such names, you can ask the user to provide input in spell by letter or spell by word style.

Amazon Lex V2 provides three slot elicitation styles for you to use. When you set a slot elicitation style, it changes the way Amazon Lex V2 interprets the input from the user.

**Spell by letter** – With this style, you can instruct the bot to listen for spellings instead of the whole phrase. For example, to capture a last name such as "Xiulan," you can tell the user to spell out their last name one letter at a time. The bot will capture the spelling and resolve the letters to a word. For example, if the user says "x i u l a n," the bot captures the last name as "xiulan."

**Spell by word** – In voice conversations, especially using the telephone, there are a few letters, such as "t," "b," "p," that sound similar. When capturing alphanumeric values or spelling names results in an incorrect value, you can prompt the user to provide an identifying word along with the letter. For example, if the voice response to a request for a booking ID is "abp123," your bot might instead recognize the phrase "abb123" instead. If this is an incorrect value, you can ask the user to provide the input as "a as in alpha b as in boy p as in peter one two three." The bot will resolve the input to "abp123."

When using spell by word, you can use the following formats:

- "as in" (a as in apple)
• "for" (a for apple)
• "like" (a like apple)

**Default** – This is the natural style of slot capture using word pronunciation. For example, it can capture names such as "John Stiles" naturally. If a slot elicitation style isn't specified, the bot uses the default style. For the AMAZON.AlphaNumeric and AMAZON.UKPostal code slot types, the default style supports spell by letter input.

If the name "Xiulan" is spoken using a mix of letters and words, such as "x as in x-ray i u l as in lion a n" the slot elicitation style must be set to spell-by-word style. The spell-by-letter style won't recognize it.

You should create a voice interface that captures slot values with natural conversational style for a better experience. For inputs that are not correctly captured using the natural style, you can re-prompt the user and set the slot elicitation style to spell-by-letter or spell-by-word.

You can use spell-by-word and spell-by-letter styles for the following slot types in the English (UK) and English (US) languages:

- AMAZON.AlphaNumeric (p. 36)
- AMAZON.EmailAddress (p. 39)
- AMAZON.FirstName (p. 39)
- AMAZON.LastName (p. 39)
- AMAZON.UKPostalCode (p. 42)

### Enabling spelling

You enable spell-by-letter and spell-by-word at runtime when you are eliciting slots from the user. You can set the spelling style with the PutSession (p. 623), RecognizeText (p. 628), RecognizeUtterance (p. 635), or StartConversation (p. 642) operation. You can also enable spell-by-letter and spell-by-word using a Lambda function.

You set the spelling style using the `dialogAction` field of the request's session state. You can only set the style when the dialog action type is ElicitSlot and when the slot to elicit is one of the supported slot types.

The following JSON code shows the `dialogAction` field set to use the spell-by-word style:

```json
"dialogAction": {
    "slotElicitationStyle": "SpellByWord",
    "slotToElicit": "BookingId",
    "type": "ElicitSlot"
}
```

The `slotElicitationStyle` field can be set to SpellByLetter, SpellByWord, or Default. If you don't specify a value, then the value is set to Default.

### Example code

Changing the spelling style is usually performed if the first attempt to resolve a slot value that didn't work. The following code example is a Python Lambda function that uses the spell-by-word style on the second attempt to resolve a slot.

To use the example code, you must have:

- A bot with one language, English (GB) (en_GB).
• One intent, "CheckAccount" with one sample utterance, "I would like to check my account". Make sure that Use a Lambda function for initialization and validation is selected in the Code hooks section of the intent definition.
• The intent should have one slot, "PostalCode", of the AMAZON.UKPostalCode built-in type.
• An alias with the Lambda function defined. For more information, see Attaching a Lambda function to a bot alias (p. 145).

import json
import time
import os
import logging

logger = logging.getLogger()
logger.setLevel(logging.DEBUG)

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

def get_slots(intent_request):
    return intent_request['sessionState']['intent']['slots']

def get_session_attributes(intent_request):
    sessionState = intent_request['sessionState']
    if 'sessionAttributes' in sessionState:
        return sessionState['sessionAttributes']
    return {}

def get_slot(intent_request, slotName):
    slots = get_slots(intent_request)
    if slots is not None and slotName in slots and slots[slotName] is not None:
        logger.debug('resolvedValue={}'.format(slots[slotName]['value']['resolvedValues']))
        return slots[slotName]['value']['resolvedValues']
    else:
        return None

def elicit_slot(session_attributes, intent_request, slots, slot_to_elicit, slot_elicitation_style, message):
    return {'sessionState': {'dialogAction': {'type': 'ElicitSlot',
                                            'slotToElicit': slot_to_elicit,
                                            'slotElicitationStyle': slot_elicitation_style,
                                            'name': intent_request['sessionState']['intent']['name'],
                                            'slots': slots,
                                            'state': 'InProgress',
                                            'sessionAttributes': session_attributes,
                                            'originatingRequestId': 'REQUESTID'},
            'intent': {'name': intent_request['sessionState']['intent']['name'],
                       'slots': slots,
                       'state': 'InProgress',
                       'sessionAttributes': session_attributes,
                       'originatingRequestId': 'REQUESTID'},
            'requestAttributes': intent_request['requestAttributes'] if 'requestAttributes' in intent_request else None},
            'sessionId': intent_request['sessionId'],
            'messages': [message]}

def build_validation_result(isvalid, violated_slot, slot_elicitation_style, message_content):
    return {'isValid': isvalid,
            'violatedSlot': violated_slot,
            'slotElicitationStyle': slot_elicitation_style,
            'message': {'contentType': 'PlainText',
                        'content': message_content}}
def GetItemInDatabase(postal_code):
    
    Perform database check for transcribed postal code. This is a no-op
    check that shows that postal_code can't be found in the database.
    
    return None

def validate_postal_code(intent_request):
    
    postal_code = get_slot(intent_request, 'PostalCode')
    
    if GetItemInDatabase(postal_code) is None:
        return build_validation_result(
            False,
            'PostalCode',
            'SpellByWord',
            "Sorry, I can't find your information. " +
            "To try again, spell out your postal " +
            "code using words, like a a as in apple."
        )
    return {'isValid': True}

def check_account(intent_request):
    
    slots = get_slots(intent_request)
    postal_code = get_slot(intent_request, 'PostalCode')
    session_attributes = get_session_attributes(intent_request)
    
    if intent_request['invocationSource'] == 'DialogCodeHook':
        # Validate the PostalCode slot. If any aren't valid,
        # re-elicit for the value.
        validation_result = validate_postal_code(intent_request)
        if not validation_result['isValid']:
            slots[validation_result['violatedSlot']] = None
            return elicit_slot(
                session_attributes,
                intent_request,
                slots,
                validation_result['violatedSlot'],
                validation_result['slotElicitationStyle'],
                validation_result['message']
            )
    
    return close(
        intent_request,
        session_attributes,
        'Fulfilled',
        {'contentType': 'PlainText',
        'content': 'Thanks'
    )
}
'sessionAttributes': session_attributes,
'dialogAction': {
   'type': 'Close'
},
'intent': intent_request['sessionState']['intent'],
'originatingRequestId': 'xxxxxxxx-xxxx-xxxx-xxxx-xxxxxxxxxxxx'
},
'messages': [ message ],
'sessionId': intent_request['sessionId'],
'requestAttributes': intent_request['requestAttributes'] if 'requestAttributes' in intent_request else None
}

# --- Intents ---
def dispatch(intent_request):
    """
    Called when the user specifies an intent for this bot.
    """
    intent_name = intent_request['sessionState']['intent']['name']
    response = None

    # Dispatch to your bot's intent handlers
    if intent_name == 'CheckAccount':
        response = check_account(intent_request)

    return response

# --- Main handler ---
def lambda_handler(event, context):
    """
    Route the incoming request based on the intent.
    
    The JSON body of the request is provided in the event slot.
    """

    # By default, treat the user request as coming from
    # Eastern Standard Time.
    os.environ['TZ'] = 'America/New_York'
    time.tzset()

    logger.debug('event={}'.format(json.dumps(event)))
    response = dispatch(event)
    logger.debug("response={}".format(json.dumps(response)))

    return response
Streaming to an Amazon Lex V2 bot

You can use the Amazon Lex V2 streaming API to start a bidirectional stream between an Amazon Lex V2 bot and your application. Starting a stream enables the bot to manage the conversation between the bot and the user. The bot responds to user input without you writing code to handle responses from the user. The bot can:

- Handle interruptions from the user while it's playing a prompt. For more information, see Enabling your bot to be interrupted by your user (p. 218).
- Wait for the user to provide input. For example, the bot can wait for the user to gather credit card information. For more information, see Enabling the bot to wait for the user to provide more information (p. 219).
- Take both dual-tone multiple-frequency (DTMF) and audio input in the same stream.
- Handle pauses in user input better than if you were managing the conversation from your application.

Not only does the Amazon Lex V2 bot respond to data sent from your application, but it also sends information about the state of the conversation to your application. You can use this information to change how your application responds to customers.

The Amazon Lex V2 bot also monitors the connection between the bot and your application. It can determine if the connection has timed out.

To use the API to start a stream to an Amazon Lex V2 bot, see Starting a stream to a bot (p. 202).

When you start streaming to an Amazon Lex V2 bot from your application, you can configure the bot to accept audio input or text input from the user. You can also choose whether the user receives audio or text in response to their input.

If you've configured the Amazon Lex V2 bot to accept audio input from the user, it can't take text input. If you've configured the bot to accept text input, the user can only use written text to communicate with it.

When an Amazon Lex V2 bot takes a streaming audio input, the bot determines when a user begins speaking and when they stop speaking. It handles any pauses or any interruptions from the user. It can also take DTMF (dual-tone multi-frequency) input and speech input in the same stream. This helps the user interact with the bot more naturally. You can present users with welcome messages and prompts. You can also enable users to interrupt those messages and prompts.

When you start a bidirectional stream, Amazon Lex V2 uses the HTTP/2 protocol. Your application and the bot exchange data in a single stream as a series of events. An event can be one of the following:

- Text, audio, or DTMF input from the user.
- Signals from the application to the Amazon Lex V2 bot. These include an indication that audio playback of a message has been completed, or that the user has disconnected from the session.

For more information about events, see Starting a stream to a bot (p. 202). For information about how to encode events, see Event stream encoding (p. 217).

Topics
- Starting a stream to a bot (p. 202)
- Event stream encoding (p. 217)
- Enabling your bot to be interrupted by your user (p. 218)
- Enabling the bot to wait for the user to provide more information (p. 219)
- Configuring fulfillment progress updates (p. 220)
Starting a stream to a bot

You use the StartConversation (p. 642) operation to start a stream between the user and the Amazon Lex V2 bot in your application. The POST request from the application establishes a connection between your application and the Amazon Lex V2 bot. This enables your application and the bot to start exchanging information with each other through events.

The StartConversation operation is supported only in the following SDKs:

- AWS SDK for C++
- AWS SDK for Java V2
- AWS SDK for Ruby V3

The first event your application must send to the Amazon Lex V2 bot is a ConfigurationEvent (p. 789). This event includes information such as the response type format. The following are the parameters that you can use in a configuration event:

- **responseContentType** – Determines whether the bot responds to user input with text or speech.
- **sessionState** – Information relating to the streaming session with the bot such as predetermined intent or dialog state.
- **welcomeMessages** – Specifies the welcome messages that play for the user at the beginning of their conversation with a bot. These messages play before the user provides any input. To activate a welcome message, you must also specify values for the sessionState and dialogAction parameters.
- **disablePlayback** – Determines whether the bot should wait for a cue from the client before it starts listening for caller input. By default, playback is activated, so the value of this field is false.
- **requestAttributes** – Provides additional information for the request.

For information about how to specify values for the preceding parameters, see the ConfigurationEvent (p. 789) data type of the StartConversation (p. 642) operation.

Each stream between a bot and your application can only have one configuration event. After your application has sent a configuration event, the bot can take additional communication from your application.

If you’ve specified that your user is using audio to communicate with the Amazon Lex V2 bot, your application can send the following events to the bot during that conversation:

- **AudioInputEvent** (p. 784) – Contains an audio chunk that has maximum size of 320 bytes. Your application must use multiple audio input events to send a message from the server to the bot. Every audio input event in the stream must have the same audio format.
- **DTMFInputEvent** (p. 794) – Sends a DTMF input to the bot. Each DTMF key press corresponds to a single event.
- **PlaybackCompletionEvent** (p. 803) – Informs the server that a response from the user's input has been played back to them. You must use a playback completion event if you're sending an audio response to the user. If disablePlayback of your configuration event is true, you can't use this feature.
- **DisconnectionEvent** (p. 793) – Informs the bot that the user has disconnected from the conversation.

If you’ve specified that the user is using text to communicate with the bot, your application can send the following events to the bot during that conversation:
Amazon Lex V2 Developer Guide
Starting a stream to a bot

- **TextInputEvent (p. 818)** – Text that is sent from your application to the bot. You can have up to 512 characters in a text input event.
- **PlaybackCompletionEvent (p. 803)** – Informs the server that a response from the user’s input has been played back to them. You must use this event if you’re playing audio back to the user. If `disablePlayback` of your configuration event is `true`, you can’t use this feature.
- **DisconnectionEvent (p. 793)** – Informs the bot that the user has disconnected from the conversation.

You must encode every event that you send to an Amazon Lex V2 bot in the correct format. For more information, see Event stream encoding (p. 217).

Every event has an event ID. To help troubleshoot any issues that might occur in the stream, assign a unique event ID to each input event. You can then troubleshoot any processing failures with the bot.

Amazon Lex V2 also uses timestamps for each event. You can use these timestamps in addition to the event ID to help troubleshoot any network transmission issues.

During the conversation between the user and the Amazon Lex V2 bot, the bot can send the following outbound events in response to the user:

- **IntentResultEvent (p. 799)** – Contains the intent that Amazon Lex V2 determined from the user utterance. Each internal result event includes:
  - **InputMode** – The type of user utterance. Valid values are Speech, DTMF, or Text.
  - **Interpretations** – Interpretations that Amazon Lex V2 determines from the user utterance.
  - **RequestAttributes** – If you haven’t modified the request attributes by using a lambda function, these are the same attributes that were passed at the start of the conversation.
  - **SessionId** – Session identifier used for the conversation.
  - **SessionState** – The state of the user’s session with Amazon Lex V2.
- **TranscriptEvent (p. 820)** – If the user provides an input to your application, this event contains the transcript of the user’s utterance to the bot. Your application does not receive a TranscriptEvent if there’s no user input.

  The value of the transcript event sent to your application depends on whether you’ve specified audio (speech and DTMF) or text as a conversation mode:

  - Transcript of speech input – If the user is speaking with the bot, the transcript event is the transcription of the user’s audio. It’s a transcript of all the speech from the time the user begins speaking to the time they end speaking.
  - Transcript of DTMF input – If the user is typing on a keypad, the transcript event contains all the digits the user pressed in their input.
  - Transcript of text input – If the user is providing text input, the transcript event contains all of the text in the user’s input.

- **TextResponseEvent (p. 819)** – Contains the bot response in text format. A text response is returned by default. If you’ve configured Amazon Lex V2 to return an audio response, this text is used to generate an audio response. Each text response event contains an array of message objects that the bot returns to the user.

- **AudioResponseEvent (p. 786)** – Contains the audio response synthesized from the text generated in the TextResponseEvent. To receive audio response events, you must configure Amazon Lex V2 to provide an audio response. All audio response events have the same audio format. Each event contains audio chunks of no more than 100 bytes. Amazon Lex V2 sends an empty audio chunk with the `bytes` field set to `null` to indicate that the end of the audio response event to your application.

- **PlaybackInterruptionEvent (p. 804)** – When a user interrupts a response that the bot has sent to your application, Amazon Lex V2 triggers this event to stop the playback of the response.

- **HeartbeatEvent (p. 795)** – Amazon Lex V2 sends this event back periodically to keep the connection between your application and the bot from timing out.
Time sequence of events for an audio conversation

The following diagrams show a streaming audio conversation between a user and an Amazon Lex V2 bot. The application continuously streams audio to the bot, and the bot looks for user input from the audio. In this example, both the user and the bot are using speech to communicate. Each diagram corresponds to a user utterance and the response of the bot to that utterance.

The following diagram shows the beginning of a conversation between the application and the bot. The stream begins at time zero (t0).

The following list describes the events of the preceding diagram.

- **t0**: The application sends a configuration event to the bot to start the stream.
- **t1**: The application streams audio data. This data is broken into a series of input events from the application.
- **t2**: For *user utterance 1*, the bot detects an audio input event when the user begins speaking.
- **t2**: While the user is speaking, the bot sends a heartbeat event to maintain the connection. It sends these events intermittently to make sure the connection doesn't time out.
- **t3**: The bot detects the end of the user's utterance.
- **t4**: The bot sends back a transcript event that contains a transcript of the user's speech to the application. This is the beginning of *Bot response to user utterance 1*.
- **t5**: The bot sends an intent result event to indicate the action that the user wants to perform.
• t6: The bot begins providing its response as text in a text response event.
• t7: The bot sends a series of audio response events to the application to play for the user.
• t8: The bot sends another heartbeat event to intermittently maintain the connection.

The following diagram is a continuation of the previous diagram. It shows the application sending a playback completion event to the bot to indicate that it has stopped playing the audio response for the user. The application plays back Bot response to user utterance 1 to the user. The user responds to Bot response to user utterance 1 with User utterance 2.

The following list describes the events of the preceding diagram:
• t10: The application sends a playback completion event to indicate that it has finished playing the bot’s message to the user.
• t11: The application sends the user response back to the bot as User utterance 2.
• t12: For Bot response to user utterance 2, the bot waits for the user to stop speaking and then begins to provide an audio response.
• t13: While the bot sends Bot response to user utterance 2 to the application, the bot detects the start of User utterance 3. The bot stops Bot response to user utterance 2 and sends a playback interruption event.
• t14: The bot sends a playback interruption event to the application to signal that the user has interrupted the prompt.

The following diagram shows the Bot response to user utterance 3, and that the conversation continues after the bot responds to the user utterance.

Using the API to start a streaming conversation

When you start a stream to an Amazon Lex V2 bot, you accomplish the following tasks:

1. Create an initial connection to the server.
2. Configure the security credentials and bot details. Bot details include whether the bot takes DTMF and audio input, or text input.
3. Send events to the server. These events are text data or audio data from the user.
4. Process events sent from the server. In this step, you determine whether the bot output is presented to the user as text or speech.

The following code examples initialize a streaming conversation with an Amazon Lex V2 bot and your local machine. You can modify the code to meet your needs.
The following code is an example request using the AWS SDK for Java to start the connection to a bot and configure the bot details and credentials.

```java
package com.lex.streaming.sample;
import software.amazon.awssdk.auth.credentials.AwsBasicCredentials;
import software.amazon.awssdk.auth.credentials.AwsCredentialsProvider;
import software.amazon.awssdk.auth.credentials.StaticCredentialsProvider;
import software.amazon.awssdk.regions.Region;
import software.amazon.awssdk.services.lexruntimev2.LexRuntimeV2AsyncClient;
import software.amazon.awssdk.services.lexruntimev2.model.ConversationMode;
import software.amazon.awssdk.services.lexruntimev2.model.StartConversationRequest;
import java.net.URISyntaxException;
import java.util.UUID;
import java.util.concurrent.CompletableFuture;
/**
 * The following code creates a connection with the Amazon Lex bot and configures the bot
details and credentials.
 * Prerequisite: To use this example, you must be familiar with the Reactive streams
programming model.
 * For more information, see
 * This example uses AWS SDK for Java for Amazon Lex V2.
 * <p>
 * The following sample application interacts with an Amazon Lex bot with the streaming
API. It uses the Audio
 * conversation mode to return audio responses to the user's input.
 * <p>
 * The code in this example accomplishes the following:
 * <p>
 * 1. Configure details about the conversation between the user and the Amazon Lex bot.
 These details include the conversation mode and the specific bot the user is speaking
 with.
 * 2. Create an events publisher that passes the audio events to the Amazon Lex bot after
you establish the connection. The code we provide in this example tells your computer to
pick up the audio from
 * your microphone and send that audio data to Amazon Lex.
 * 3. Create a response handler that handles the audio responses from the Amazon Lex bot
and plays back the audio to you.
 */
public class LexBidirectionalStreamingExample {
    public static void main(String[] args) throws URISyntaxException, InterruptedException {
        String botId = "";
        String botAliasId = "";
        String localeId = "";
        String accessKey = "";
        String secretKey = "";
        String sessionId = UUID.randomUUID().toString();
        Region region = Region.region_name; // Choose an AWS Region where the Amazon Lex
        Streaming API is available.

        AwsCredentialsProvider awsCredentialsProvider = StaticCredentialsProvider
        .create(AwsBasicCredentials.create(accessKey, secretKey));

        // Create a new SDK client. You need to use an asynchronous client.
        System.out.println("step 1: creating a new Lex SDK client");
        LexRuntimeV2AsyncClient lexRuntimeServiceClient = LexRuntimeV2AsyncClient.builder()
        .region(region)
        .credentialsProvider(awsCredentialsProvider)
        .build();
```
// Configure the bot, alias and locale that you'll use to have a conversation.
System.out.println("step 2: configuring bot details");
StartConversationRequest.Builder startConversationRequestBuilder =
StartConversationRequest.builder()
  .botId(botId)
  .botAliasId(botAliasId)
  .localeId(localeId);

// Configure the conversation mode of the bot. By default, the
// conversation mode is audio.
System.out.println("step 3: choosing conversation mode");
startConversationRequestBuilder =
startConversationRequestBuilder.conversationMode(ConversationMode.AUDIO);

// Assign a unique identifier for the conversation.
System.out.println("step 4: choosing a unique conversation identifier");
startConversationRequestBuilder =
startConversationRequestBuilder.sessionId(sessionId);

// Start the initial request.
StartConversationRequest startConversationRequest =
startConversationRequestBuilder.build();

// Create a stream of audio data to the Amazon Lex bot. The stream will start after
the connection is established with the bot.
EventsPublisher eventsPublisher = new EventsPublisher();

// Create a class to handle responses from bot. After the server processes the user
data you've streamed, the server responds
// on another stream.
BotResponseHandler botResponseHandler = new BotResponseHandler(eventsPublisher);

// Start a connection and pass in the publisher that streams the audio and process
the responses from the bot.
System.out.println("step 5: starting the conversation ...");
CompletableFuture<Void> conversation = lexRuntimeServiceClient.startConversation(
  startConversationRequest,
  eventsPublisher,
  botResponseHandler);

// Wait until the conversation finishes. The conversation finishes if the dialog
state reaches the "Closed" state.
// The client stops the connection. If an exception occurs during the conversation,
the
// client sends a disconnection event.
conversation.whenComplete((result, exception) -> {
  if (exception != null) {
    eventsPublisher.disconnect();
  }
});

// The conversation finishes when the dialog state is closed and last prompt has
been played.
while (!botResponseHandler.isConversationComplete()) {
  Thread.sleep(100);
}

// Randomly sleep for 100 milliseconds to prevent JVM from exiting.
// You won't need this in your production code because your JVM is
// likely to always run.
// When the conversation finishes, the following code block stops publishing more
data and informs the Amazon Lex bot that there is no more data to send.
if (botResponseHandler.isConversationComplete()) {
  System.out.println("conversation is complete.");
The following code is an example request using the AWS SDK for Java to send events to the bot. The code in this example uses the microphone on your computer to send audio events.

```java
package com.lex.streaming.sample;
import org.reactivestreams.Publisher;
import org.reactivestreams.Subscriber;
import software.amazon.awssdk.services.lexruntimev2.model.StartConversationRequestEventStream;

/**
 * You use the Events publisher to send events to the Amazon Lex bot. When you establish a connection, the bot uses the
 * subscribe() method and enables the events publisher starts sending events to
 * your computer. The bot uses the "request" method of the subscription to make more
 * requests. For more information on the request method, see https://github.com/reactive-streams/reactive-streams-jvm.
 */
public class EventsPublisher implements Publisher<StartConversationRequestEventStream> {
    private AudioEventsSubscription audioEventsSubscription;

    @Override
    public void subscribe(Subscriber<? super StartConversationRequestEventStream> subscriber) {
        if (audioEventsSubscription == null) {
            audioEventsSubscription = new AudioEventsSubscription(subscriber);
            subscriber.onNext(audioEventsSubscription);
        } else {
            throw new IllegalStateException("received unexpected subscription request");
        }
    }

    public void disconnect() {
        if (audioEventsSubscription != null) {
            audioEventsSubscription.disconnect();
        }
    }

    public void stop() {
        if (audioEventsSubscription != null) {
            audioEventsSubscription.stop();
        }
    }

    public void playbackFinished() {
        if (audioEventsSubscription != null) {
            audioEventsSubscription.playbackFinished();
        }
    }
}
```
The following code is an example request using the AWS SDK for Java to handle responses from the bot. The code in this example configures Amazon Lex V2 to play an audio response back to you.

```java
package com.lex.streaming.sample;

import javazoom.jl.decoder.JavaLayerException;
import javazoom.jl.player.advanced.AdvancedPlayer;
import javazoom.jl.player.advanced.PlaybackEvent;
import javazoom.jl.player.advanced.PlaybackListener;
import software.amazon.awssdk.core.async.SdkPublisher;
import software.amazon.awssdk.services.lexruntimev2.model.AudioResponseEvent;
import software.amazon.awssdk.services.lexruntimev2.model.DialogActionType;
import software.amazon.awssdk.services.lexruntimev2.model.IntentResultEvent;
import software.amazon.awssdk.services.lexruntimev2.model.PlaybackInterruptionEvent;
import software.amazon.awssdk.services.lexruntimev2.model.StartConversationResponse;
import software.amazon.awssdk.services.lexruntimev2.model.StartConversationResponseEventStream;
import software.amazon.awssdk.services.lexruntimev2.model.TranscriptEvent;
import java.io.IOException;
import java.io.UncheckedIOException;
import java.util.concurrent.CompletableFuture;

/**
 * The following class is responsible for processing events sent from the Amazon Lex bot. The bot sends multiple audio events, so the following code concatenates those audio events and uses a publicly available Java audio player to play out the message to the user.
 */
public class BotResponseHandler implements StartConversationResponseHandler {

    private final EventsPublisher eventsPublisher;
    private boolean lastBotResponsePlayedBack;
    private boolean isDialogStateClosed;
    private AudioResponse audioResponse;

    public BotResponseHandler(EventsPublisher eventsPublisher) {
        this.eventsPublisher = eventsPublisher;
        this.lastBotResponsePlayedBack = false;
        this.isDialogStateClosed = false;
    }

    @Override
    public void responseReceived(StartConversationResponse startConversationResponse) {
        System.out.println("successfully established the connection with server. request id:" + startConversationResponse.responseMetadata().requestId()); // would have 2XX, request id.
    }

    @Override
    public void onEventStream(SdkPublisher<StartConversationResponseEventStream> sdkPublisher) {
        sdkPublisher.subscribe(event -> {
            if (event instanceof PlaybackInterruptionEvent) {
                handle((PlaybackInterruptionEvent) event);
            } else if (event instanceof TranscriptEvent) {
                handle((TranscriptEvent) event);
            }
        });
    }
}
```
} else if (event instanceof IntentResultEvent) {
    handle((IntentResultEvent) event);
} else if (event instanceof TextResponseEvent) {
    handle((TextResponseEvent) event);
} else if (event instanceof AudioResponseEvent) {
    handle((AudioResponseEvent) event);
}
});
}

@Override
public void exceptionOccurred(Throwable throwable) {
    System.err.println("got an exception:" + throwable);
}

@Override
public void complete() {
    System.out.println("on complete");
}

private void handle(PlaybackInterruptionEvent event) {
    System.out.println("Got a PlaybackInterruptionEvent: " + event);
}

private void handle(TranscriptEvent event) {
    System.out.println("Got a TranscriptEvent: " + event);
}

private void handle(IntentResultEvent event) {
    System.out.println("Got an IntentResultEvent: " + event);
    isDialogStateClosed =
    DialogActionType.CLOSE.equals(event.sessionState().dialogAction().type());
}

private void handle(TextResponseEvent event) {
    System.out.println("Got an TextResponseEvent: " + event);
    event.messages().forEach(message -> {
        System.out.println("Message content type:" + message.contentType());
        System.out.println("Message content:" + message.content());
    });
}

private void handle(AudioResponseEvent event) {//Synthesize speech
    //System.out.println("Got a AudioResponseEvent: " + event);
    if (audioResponse == null) {
        audioResponse = new AudioResponse();
        //Start an audio player in a different thread.
        CompletableFuture.runAsync(() -> {
            try {
                AdvancedPlayer audioPlayer = new AdvancedPlayer(audioResponse);
                audioPlayer.setPlaybackListener(new PlaybackListener() {
                    @Override
                    public void playbackFinished(PlaybackEvent evt) {
                        super.playbackFinished(evt);
                        // Inform the Amazon Lex bot that the playback has finished.
                        eventsPublisher.playbackFinished();
                        if (isDialogStateClosed) {
                            lastBotResponsePlayedBack = true;
                        }
                    }
                });
                audioPlayer.play();
            } catch (JavaLayerException e) {
                }
        });
        audioPlayer.play();
    } catch (JavaLayerException e) {
}}
throw new RuntimeException("got an exception when using audio player", e);
    }
  });
}
if (event.audioChunk() != null) {
    audioResponse.write(event.audioChunk().asByteArray());
} else {
  // The audio audio prompt has ended when the audio response has no
  // audio bytes.
  try {
    audioResponse.close();
  } catch (IOException e) {
    audioResponse = null; // Prepare for the next audio prompt.
    throw new UncheckedIOException("got an exception when closing the audio
    response", e);
  }
}
// The conversation with the Amazon Lex bot is complete when the bot marks the Dialog
as DialogActionType.CLOSE
// and any prompt playback is finished. For more information, see
public boolean isConversationComplete() {
    return isDialogStateClosed && lastBotResponsePlayedBack;
}
}

To configure a bot to respond to input events with audio, you must first subscribe to audio events from
Amazon Lex V2 and then configure the bot to provide an audio response to the input events from the
user.

The following code is an AWS SDK for Java example for subscribing to audio events from Amazon Lex V2.

```java
package com.lex.streaming.sample;
import org.reactivestreams.Subscriber;
import org.reactivestreams.Subscription;
import software.amazon.awssdk.core.SdkBytes;
import software.amazon.awssdk.services.lexruntimev2.model.AudioInputEvent;
import software.amazon.awssdk.services.lexruntimev2.model.ConfigurationEvent;
import software.amazon.awssdk.services.lexruntimev2.model.DisconnectionEvent;
import software.amazon.awssdk.services.lexruntimev2.model.PlaybackCompletionEvent;
import software.amazon.awssdk.services.lexruntimev2.model.StartConversationRequestEventStream;
import javax.sound.sampled.AudioFormat;
import javax.sound.sampled.AudioInputStream;
import javax.sound.sampled.AudioSystem;
import javax.sound.sampled.DataLine;
import javax.sound.sampled.LineUnavailableException;
import java.io.IOException;
import java.io.UncheckedIOException;
import java.nio.ByteBuffer;
import java.util.Arrays;
import java.util.concurrent.BlockingQueue;
import java.util.concurrent.CompletableFuture;
```
import java.util.concurrent.LinkedBlockingQueue;
import java.util.concurrent.atomic.AtomicLong;

public class AudioEventsSubscription implements Subscription {
    private static final AudioFormat MIC_FORMAT = new AudioFormat(8000, 16, 1, true, false);
    private static final String AUDIO_CONTENT_TYPE = "audio/lpcm; sample-rate=8000; sample-size-bits=16; channel-count=1; is-big-endian=false";
    //private static final String RESPONSE_TYPE = "audio/pcm; sample-rate=8000";
    private static final String RESPONSE_TYPE = "audio/mpeg";
    private static final int BYTES_IN_AUDIO_CHUNK = 320;
    private static final AtomicLong eventIdGenerator = new AtomicLong(0);

    private final AudioInputStream audioInputStream;
    private final Subscriber<? super StartConversationRequestEventStream> subscriber;
    private final EventWriter eventWriter;
    private CompletableFuture eventWriterFuture;

    public AudioEventsSubscription(Subscriber<? super StartConversationRequestEventStream> subscriber) {
        this.audioInputStream = getMicStream();
        this.subscriber = subscriber;
        this.eventWriter = new EventWriter(subscriber, audioInputStream);
        configureConversation();
    }

    private AudioInputStream getMicStream() {
        try {
            DataLine.Info dataLineInfo = new DataLine.Info(TargetDataLine.class, MIC_FORMAT);
            TargetDataLine targetDataLine = (TargetDataLine) AudioSystem.getLine(dataLineInfo);
            targetDataLine.open(MIC_FORMAT);
            targetDataLine.start();
            return new AudioInputStream(targetDataLine);
        } catch (LineUnavailableException e) {
            throw new RuntimeException(e);
        }
    }

    @Override
    public void request(long demand) {
        // If a thread to write events has not been started, start it.
        if (eventWriterFuture == null) {
            eventWriterFuture = CompletableFuture.runAsync(eventWriter);
        }
        eventWriter.addDemand(demand);
    }

    @Override
    public void cancel() {
        subscriber.onError(new RuntimeException("stream was cancelled"));
        try {
            audioInputStream.close();
        } catch (IOException e) {
            throw new UncheckedIOException(e);
        }
    }

    public void configureConversation() {
        String eventId = "ConfigurationEvent-" + String.valueOf(eventIdGenerator.incrementAndGet());
    }
}
ConfigurationEvent configurationEvent = StartConversationRequestEventStream
    .configurationEventBuilder()
    .eventId(eventId)
    .clientTimestampMillis(System.currentTimeMillis())
    .responseContentType(RESPONSE_TYPE)
    .build();

System.out.println("writing config event");
eventWriter.writeConfigurationEvent(configurationEvent);
}

public void disconnect() {

    String eventId = "DisconnectionEvent-" +
    String.valueOf(eventIdGenerator.incrementAndGet());

    DisconnectionEvent disconnectionEvent = StartConversationRequestEventStream
        .disconnectionEventBuilder()
        .eventId(eventId)
        .clientTimestampMillis(System.currentTimeMillis())
        .build();

    eventWriter.writeDisconnectEvent(disconnectionEvent);

    try {
        audioInputStream.close();
    } catch (IOException e) {
        throw new UncheckedIOException(e);
    }
}

//Notify the subscriber that we've finished.
public void stop() {
    subscriber.onComplete();
}

public void playbackFinished() {

    String eventId = "PlaybackCompletion-" +
    String.valueOf(eventIdGenerator.incrementAndGet());

    PlaybackCompletionEvent playbackCompletionEvent = StartConversationRequestEventStream
        .playbackCompletionEventBuilder()
        .eventId(eventId)
        .clientTimestampMillis(System.currentTimeMillis())
        .build();

    eventWriter.writePlaybackFinishedEvent(playbackCompletionEvent);
}

private static class EventWriter implements Runnable {
    private final BlockingQueue<StartConversationRequestEventStream> eventQueue;
    private final AudioInputStream audioInputStream;
    private final AtomicLong demand;
    private final Subscriber subscriber;
    private boolean conversationConfigured;

    public EventWriter(Subscriber subscriber, AudioInputStream audioInputStream) {
        this.eventQueue = new LinkedBlockingQueue<>();
        this.demand = new AtomicLong(0);
        this.subscriber = subscriber;
        this.audioInputStream = audioInputStream;
    }
}
public void writeConfigurationEvent(ConfigurationEvent configurationEvent) {
    eventQueue.add(configurationEvent);
}

public void writeDisconnectEvent(DisconnectionEvent disconnectionEvent) {
    eventQueue.add(disconnectionEvent);
}

public void writePlaybackFinishedEvent(PlaybackCompletionEvent playbackCompletionEvent) {
    eventQueue.add(playbackCompletionEvent);
}

void addDemand(long l) {
    this.demand.addAndGet(l);
}

@Override
public void run() {
    try {
        while (true) {
            long currentDemand = demand.get();
            if (currentDemand > 0) {
                // Try to read from queue of events.
                // If nothing is in queue at this point, read the audio events
directly from audio stream.
                for (long i = 0; i < currentDemand; i++) {
                    if (eventQueue.peek() != null) {
                        subscriber.onNext(eventQueue.take());
                        demand.decrementAndGet();
                    } else {
                        writeAudioEvent();
                    }
                }
            }
        }
    } catch (InterruptedException e) {
        throw new RuntimeException("interrupted when reading data to be sent to
server");
    } catch (Exception e) {
        e.printStackTrace();
    }
}

private void writeAudioEvent() {
    byte[] bytes = new byte[BYTES_IN_AUDIO_CHUNK];
    int numBytesRead = 0;
    try {
        numBytesRead = audioInputStream.read(bytes);
        if (numBytesRead != -1) {
            byte[] byteArrayCopy = Arrays.copyOf(bytes, numBytesRead);
            String eventId = "AudioEvent-" +
            String.valueOf(eventIdGenerator.incrementAndGet());
            AudioInputEvent audioInputEvent =
            StartConversationRequestEventStream.audioInputEventBuilder().
            audioChunk(SdkBytes.fromByteBuffer(ByteBuffer.wrap(byteArrayCopy))).
            contentType(AUDIO_CONTENT_TYPE).
            clientTimestampMillis(System.currentTimeMillis()).
            eventId(eventId).build();
        }
    }
}
The following AWS SDK for Java example configures the Amazon Lex V2 bot to provide an audio response to the input events.

```java
package com.lex.streaming.sample;

import java.io.IOException;
import java.io.InputStream;
import java.io.UncheckedIOException;
import java.util.Optional;
import java.util.concurrent.LinkedBlockingQueue;
import java.util.concurrent.TimeUnit;

public class AudioResponse extends InputStream{

    // Used to convert byte, which is signed in Java, to positive integer (unsigned)
    private static final int UNSIGNED_BYTE_MASK = 0xFF;
    private static final long POLL_INTERVAL_MS = 10;

    private final LinkedBlockingQueue<Integer> byteQueue = new LinkedBlockingQueue<>();
    private volatile boolean closed;

    @Override
    public int read() throws IOException {
        try {
            Optional<Integer> maybeInt;
            while (true) {
                maybeInt = Optional.ofNullable(this.byteQueue.poll(POLL_INTERVAL_MS,
                        TimeUnit.MILLISECONDS));
                // If we get an integer from the queue, return it.
                if (maybeInt.isPresent()) {
                    return maybeInt.get();
                }
                // If the stream is closed and there is nothing queued up, return -1.
                if (this.closed) {
                    return -1;
                }
            }
        } catch (InterruptedException e) {
            throw new IOException(e);
        } catch (IOException e) {
            System.out.println("got an exception when reading from audio stream");
            System.err.println(e);
            subscriber.onError(e);
        }
        return -1;
    }

    @Override
    public void close() throws IOException {
        this.closed = true;
    }
}
```
Event stream encoding

Event stream encoding provides bidirectional communication using messages between a client and a server. Data frames sent to the Amazon Lex V2 streaming service are encoded in this format. The response from Amazon Lex V2 also uses this encoding.

Each message consists of two sections: the prelude and the data. The prelude section contains the total byte length of the message and the combined byte length of all of the headers. The data section contains the headers and a payload.

Each section ends with a 4-byte big-endian integer CRC checksum. The message CRC checksum includes the prelude section and the data section. Amazon Lex V2 uses CRC32 (often referred to as GZIP CRC32) to calculate both CRCs. For more information about CRC32, see [GZIP file format specification version 4.3](#).

Total message overhead, including the prelude and both checksums, is 16 bytes.

The following diagram shows the components that make up a message and a header. There are multiple headers per message.
Enabling your bot to be interrupted

When you start a bidirectional audio stream between an Amazon Lex V2 bot and your application, you can configure the bot to listen for user input while it is sending back a prompt. With this configuration, the user can interrupt the prompt before the bot has finished playing it back. You can use this configuration for
situations where the user might already know the answer to a question, such as when they’re being prompted to provide a CVV code.

A bot knows when the user interrupts a prompt when it detects user input before your application can send back a PlaybackCompletion event. When the user interrupts a bot, the bot sends a PlaybackInterruptionEvent.

By default, the user can interrupt any prompt that the bot is streaming to your application. You can change this setting in the Amazon Lex V2 console.

You can change how a user can respond to a prompt by editing a slot. A slot is part of an intent, and it is the means by which the user provides you the information you want. Each slot has a prompt for the user to provide you with that information. To learn more about slots, see How it works (p. 3).

To change whether the user can interrupt a prompt (console)

1. Sign in to AWS Management Console and open the Amazon Lex V2 console at Amazon Lex V2 console.
2. Under Bots, select a bot.
3. Under Language, select the language of the bot.
4. Choose View intents.
5. Choose the intent.
6. For Slots, choose a slot.
7. Under Advanced options, choose Slot prompts.
9. Select or deselect Users can interrupt the prompt when it is being read.

You can test this functionality by creating a bot with two slots and specifying that users can’t interrupt a prompt for one slot. If you interrupt an interruptible prompt, the bot sends a playback interruption event. If you interrupt an uninterruptible, the prompt continues to play.

Enabling the bot to wait for the user to provide more information

When you start a bidirectional stream from an Amazon Lex V2 bot to your application, you can configure the bot to wait for the user to provide additional information. There are circumstances when a user might not be ready to respond to a prompt. For example, a user might not be ready to provide their credit card information because their wallet is in another room.

By using the Wait and continue behavior of the Amazon Lex V2 bot, users can say phrases such as "hold on a second" to make the bot wait for them to find the information and provide it. When you enable this behavior, the bot sends periodic reminders to the user to provide the information. It does not send back transcript events because there are no user utterances for it to transcribe.

The Amazon Lex V2 bot automatically manages a streaming conversation. You don’t have to write any additional code to enable this functionality. When a bot is prompted to wait by the user, the state of the Intent is Waiting and the type of the DialogAction is ElicitSlot. You can use this information to help customize your application for your needs. For example, you can configure your application to play music when the user is looking for their credit card.

You enable the wait and continue behavior for an individual slot. To learn more about slots, see How it works (p. 3).
To enable wait and continue

1. Sign in to AWS Management Console and open the Amazon Lex V2 console at Amazon Lex V2 console.
2. Under Bots, select a bot.
3. Under Language, select the language of the bot.
4. Choose View intents.
5. Choose the intent.
6. Under Slots, choose a slot.
7. Under Advanced options, choose Wait and continue.
8. Under Wait and continue specify the following fields:
   - Response when user wants the bot to wait – This is how the bot responds when the user asks it to wait for the additional information.
   - Response if the user needs the bot to continue waiting – This is the response the bot sends to remind the user that it's still waiting for the information. You can change how frequently the bot reminds the user.
   - Response when the user wants to continue – This is the bot's response when the user has the requested information.

For every bot response, you can give multiple variations of the response, and one is presented to the user at random. You can also choose whether these responses can be interrupted by the user.

To test the wait and continue functionality, configure your bot to wait for user input and start a stream to an Amazon Lex V2 bot. For information on streaming to a bot, see Using the API to start a streaming conversation (p. 206).

You may need to turn off the wait and continue responses. Use the Active toggle to set whether or not the wait and continue responses are used.

Wait and continue

You can use the responses below to manage a conversation if the user needs to time to provide information requested functionality is available only in streaming conversations.

Configuring fulfillment progress updates

When the fulfillment Lambda function for an intent is called, the bot doesn't send a response until the function completes. If the Lambda function takes more than a few seconds to complete, the user may think that the bot is unresponsive. To address this, you can configure your bot to send updates to the user while the fulfillment Lambda function is running so that the user knows that the bot is still working on their request.

When you add fulfillment updates to an intent, the bot responds at the start of fulfillment and periodically while fulfillment is in progress. When you configure the start response, you can specify a delay before the bot sends the response. With this, you can support cases where the fulfillment doesn't finish relatively quickly. When you configure an update response, you specify the frequency that you want the updates sent. You also configure a timeout to limit the time that the fulfillment function has to run.

You can also add post-fulfillment responses to a bot. This enables the bot to send a different response depending on whether fulfillment succeeds, fails, or times out.
Fulfillment updates

Fulfillment updates are used only when interacting with a bot using the `StartConversation (p. 642)` operation. You can use the post-fulfillment update when interacting with the bot using the `StartConversation (p. 642), RecognizeText (p. 628), and RecognizeUtterance (p. 635)` operations.

**Fulfillment updates**

Fulfillment updates are sent while your Lambda function is fulfilling an intent. When you turn on fulfillment updates, you provide a start response that is sent at the beginning of fulfillment and an update response that is sent periodically while fulfillment is in progress.

When you specify an update response, you also specify a timeout that determines how long the fulfillment function can run. You can specify a timeout length of up to 15 minutes (900 seconds).

If you turn off fulfillment updates by setting `active` to false in the console or using the `CreateIntent (p. 337) or UpdateIntent (p. 572)` operation, the timeout specified for the fulfillment updates isn't used and the default timeout of 30 seconds is used instead.

If the fulfillment function times out, Amazon Lex V2 does one of three things:

- Post-fulfillment response is configured and active – returns the timeout response.
- Post-fulfillment response is configured and not active – returns an exception.
- Post-fulfillment response isn’t configured – returns an exception.

**Start response**

Amazon Lex V2 returns the start response when the Lambda fulfillment function is called during a streaming conversation. It typically tells the user that fulfilling the intent takes some time and that they should wait. The start response isn't returned when you use the `RecognizeText` or `RecognizeUtterance` operations.

You can specify up to five response messages. Amazon Lex V2 chooses one of the messages to play to the user.

You can configure a delay between when the Lambda function is called and when the start response is returned. The start response isn't returned if the Lambda function completes its work before the delay is complete.

You can use the `active` toggle in the console or the `FulfillmentUpdatesSpecification (p. 714)` structure to turn the start response on and off. When `active` is false, the start response isn't played.

**Update response**

Amazon Lex returns the update response periodically during a streaming conversation while the Lambda fulfillment function is running. The update response isn't played when you use the `RecognizeText` or `RecognizeUtterance` operations. You can configure how often the update response plays. For example, you can play an update response every 30 seconds while the fulfillment function runs to let the user know that the process is running and that they should continue to wait.

You can specify up to five update messages. Amazon Lex V2 chooses a message to play to the user. Using multiple messages keeps the updates from being repetitive.

If the user provides input via voice, DTMF, or text while the fulfillment Lambda function is running, Amazon Lex V2 returns the update response to the user.

If the Lambda function completes its work before the first update period ends, the update response isn't returned.
You can use the `active` toggle in the console or the `FulfillmentUpdatesSpecification (p. 714)` structure to turn the update response on and off. When `active` is false, the update response isn't returned.

### Post-fulfillment response

Amazon Lex V2 returns a post-fulfillment response when the fulfillment function ends. A post-fulfillment response can be used when fulfilling any intent, not just streaming conversations. The post-fulfillment response lets the user know that the function is complete and the result.

You can use the `active` toggle in the console or the `PostFulfillmentStatusSpecification (p. 742)` structure to turn the post-fulfillment response on and off. When `active` is false, the response is not played.

There are three types of post-fulfillment responses:

- **Success** – returned when the fulfillment Lambda function completes its work successfully. If post-fulfillment responses aren't active, Amazon Lex V2 takes the next configured action.
- **Timeout** – returned if the Lambda function doesn't complete its work before the configured timeout period elapses. If post-fulfillment responses aren't active, Amazon Lex V2 returns an exception.
- **Failure** – returned when the Lambda function returns the status `Failed` in the response or when Amazon Lex V2 encounters an error while fulfilling the intent. If post-fulfillment responses aren't active, Amazon Lex V2 returns an exception.

You can specify up to five messages for each type. Amazon Lex V2 chooses one of the messages to play to the user. If the Lambda function returns a post-fulfillment message, that message is used instead of the configured messages.

**Note**

If the intent has a closing response, the response is returned after the post-fulfillment response.

### Configuring timeouts for capturing user input

The Amazon Lex V2 streaming API enables a bot to automatically detect utterances in user input. When you create an intent or a slot, you can configure aspects of an utterance, such as maximum duration of an utterance, timeout while waiting for user input, or the end character for DTMF input. You can customize a bot’s behavior for your use case. For example, you can limit the number of digits for a credit card number to 16.

You can also configure timeouts through session attributes when starting a conversation with a bot, and overwrite them in your Lambda function if necessary.

The configuration keys for an attribute use the following syntax:

```
x-amz-lex:<InputType>:<BehaviorName>:<IntentName>:<SlotName>
```

**InputType** can be `audio`, `dtmf`, or `text`.

You can configure default settings for all intents or slots in a bot by specifying `*` as the intent or slot name. Any intent- or slot-specific settings take precedence over default settings.

Amazon Lex V2 provides predefined session attributes for managing the way the `StartConversation (p. 642)` operations works with text, voice, or DTMF input to your bot. All predefined attributes are in the `x-amz-lex` namespace.
Timeouts for voice input

You can set time-out values for voice interaction with your bot using session attributes. The attributes are defined by Amazon Lex V2. These attributes enable you to specify how long Amazon Lex V2 waits for a customer to finish speaking before collecting input speech.

All of these attributes are in the `x-amz-lex:audio` namespace.

### Maximum utterance length

```
x-amz-lex:audio:max-length-ms:<intentName>:<slotName>
```

Defines how long Amazon Lex V2 waits before speech input is truncated and the speech is returned to your application. You can increase the length of the input when you expect long responses, or if you want to give customers more time to provide information.

**Default:** 13,000 milliseconds (13 seconds). The maximum value is 15,000 milliseconds (15 seconds)

If you set the `max-length-ms` attribute to more than 15,000 milliseconds, an error is returned at runtime.

### Voice timeout

```
x-amz-lex:audio:start-timeout-ms:<intentName>:<slotName>
```

How long a bot waits before assuming that the customer isn't going to speak. You can increase the time in situations where the customer may need more time to find or recall information before speaking. For example, you might want to give customers time to get out their credit card so they can enter the number.

**Default:** 4,000 milliseconds (4 seconds)

### Silence timeout

```
x-amz-lex:audio:end-timeout-ms:<intentName>:<slotName>
```

How long a bot waits after the customer stops speaking to assume the utterance is finished. You can increase the time in situations where periods of silence are expected while providing input.

**Default:** 600 milliseconds (0.6 seconds)

### Timeouts for text input

Use the following session attribute to specify how your bot behaves with the text conversation mode.

This attribute is in the `x-amz-lex:text` namespace.
Start timeout threshold

\texttt{x-amz-lex:text:start-timeout-ms:<intentName>:<slotName>}

How long the bot waits before re-prompting a customer for text input. You can increase the time in situations where you'd like to allow the customer more time to find or recall information before providing text input. For example, you might want to give customers more time to find details on their order. Alternatively, you may reduce the threshold to prompt customers earlier.

\textbf{Default:} 30,000 milliseconds (30 seconds)

Configuration for DTMF input

Use the following session attributes to specify how your Amazon Lex V2 bot responds to DTMF input when using an audio conversation.

All of these attributes are in the \texttt{x-amz-lex:dtmf} namespace.

Deletion character

\texttt{x-amz-lex:dtmf:deletion-character:<intentName>:<slotName>}

The DTMF character that clears the accumulated DTMF digits and immediately ends the input.

\textbf{Default:} *

End character

\texttt{x-amz-lex:dtmf:end-character:<intentName>:<slotName>}

The DTMF character that immediately ends input. If the user does not press this character, the input ends after the end timeout.

\textbf{Default:} #

End timeout

\texttt{x-amz-lex:dtmf:end-timeout-ms:<intentName>:<slotName>}

How long the bot should wait from the last DTMF character input before assuming that the input has concluded.

\textbf{Default:} 5000 milliseconds (5 seconds)

Maximum number of DTMF digits per utterance

\texttt{x-amz-lex:dtmf:max-length:<intentName>:<slotName>}

The maximum number of DTMF digits allowed in an utterance. For example, you could set this value to 16 to limit the number of characters that can be input for a credit card number. This value can't be increased.

\textbf{Default:} 1024 characters
Monitoring Amazon Lex V2

Monitoring is important for maintaining the reliability, availability, and performance of your Amazon Lex V2 chatbots. This topic describes using conversation logs to monitor conversations between your users and your chatbots, using utterance statistics to determine the utterances that your bots detect and miss, and how to use Amazon CloudWatch Logs and AWS CloudTrail to monitor Amazon Lex V2 and describes the Amazon Lex V2 runtime and channel association metrics.

Topics
- Monitoring with conversation logs (p. 225)
- Obscuring slot values in logs (p. 231)
- Viewing utterance statistics (p. 232)
- Logging Amazon Lex V2 API calls with AWS CloudTrail (p. 233)
- Monitoring Amazon Lex V2 with Amazon CloudWatch (p. 235)

Monitoring with conversation logs

You enable conversation logs to store bot interactions. You can use these logs to review the performance of your bot and to troubleshoot issues with conversations. You can log text for the RecognizeText (p. 628) operation. You can log both text and audio for the RecognizeUtterance (p. 635) operation. By enabling conversation logs you get a detailed view of conversations that users have with your bot.

For example, a session with your bot has a session ID. You can use this ID to get the transcript of the conversation including user utterances and the corresponding bot responses. You also get metadata such as intent name and slot values for an utterance.

Note
You can't use conversation logs with a bot subject to the Children's Online Privacy Protection Act (COPPA).

Conversation logs are configured for an alias. Each alias can have different settings for their text and audio logs. You can enable text logs, audio logs, or both for each alias. Text logs store text input, transcripts of audio input, and associated metadata in CloudWatch Logs. Audio logs store audio input in Amazon S3. You can enable encryption of text and audio logs using AWS KMS customer managed CMKs.

To configure logging, use the console or the CreateBotAlias (p. 318) or UpdateBotAlias (p. 553) operation. After enabling conversation logs for an alias, RecognizeText (p. 628) or RecognizeUtterance (p. 635) operation for that alias logs the text or audio utterances in the configured CloudWatch Logs log group or S3 bucket.

Topics
- Configuring conversation logs (p. 226)
- Viewing text logs in Amazon CloudWatch Logs (p. 227)
- Accessing audio logs in Amazon S3 (p. 230)
- Monitoring conversation log status with CloudWatch metrics (p. 230)
Configuring conversation logs

You enable and disable conversation logs using the console or the `conversationLogSettings` field of the `CreateBotAlias` or `UpdateBotAlias` operation. You can turn on or turn off audio logs, text logs, or both. Logging starts on new bot sessions. Changes to log settings aren’t reflected for active sessions.

To store text logs, use an Amazon CloudWatch Logs log group in your AWS account. You can use any valid log group. The log group must be in the same region as the Amazon Lex V2 bot. For more information about creating a CloudWatch Logs log group, see Working with Log Groups and Log Streams in the Amazon CloudWatch Logs User Guide.

To store audio logs, use an Amazon S3 bucket in your AWS account. You can use any valid S3 bucket. The bucket must be in the same region as the Amazon Lex V2 bot. For more information about creating an S3 bucket, see Creating a bucket in the Amazon Simple Storage Service Getting Started Guide.

When you manage conversation logs using the console, the console updates your service role so that it has access to the log group and S3 bucket. If you are not using the console, you must provide an IAM role with policies that enable Amazon Lex V2 to write to the configured log group or bucket.

The IAM role that you use to enable conversation logs must have the `iam:PassRole` permission. The following policy should be attached to the role.

```
{
   "Version": "2012-10-17",
   "Statement": [
      {
         "Effect": "Allow",
         "Action": "iam:PassRole",
         "Resource": "arn:aws:iam::account:role/role"
      }
   ]
}
```

Enabling conversation logs

To turn on logs using the console

1. Open the Amazon Lex V2 console https://console.aws.amazon.com/lexv2.
2. From the list, choose a bot.
3. From the left menu, choose Aliases.
4. In the list of aliases, choose the alias for which you want to configure conversation logs.
5. In the Conversation logs section, choose Manage conversation logs.
6. For text logs, choose Enable then enter the Amazon CloudWatch Logs log group name.
7. For audio logs, choose Enable then enter the S3 bucket information.
8. Optional. To encrypt audio logs, choose the AWS KMS key to use for encryption.
9. Choose Save to start logging conversations. If necessary, Amazon Lex V2 will update your service role with permissions to access the CloudWatch Logs log group and selected S3 bucket.

Disabling conversation logs

To turn off logs using the console

1. Open the Amazon Lex V2 console https://console.aws.amazon.com/lexv2.
2. From the list, choose a bot.
3. From the left menu, choose **Aliases**.
4. In the list of aliases, choose the alias for which you want to configure conversation logs.
5. In the **Conversation logs** section, choose **Manage conversation logs**.
6. Disable text logging, audio logging, or both to turn off logging.
7. Choose **Save** to stop logging conversations.

### Viewing text logs in Amazon CloudWatch Logs

Amazon Lex V2 stores text logs for your conversations in Amazon CloudWatch Logs. To view the logs, you can use the CloudWatch Logs console or API. For more information, see [Search Log Data Using Filter Patterns](#) and [CloudWatch Logs Insights Query Syntax](#) in the Amazon CloudWatch Logs User Guide.

#### To view logs using the Amazon Lex V2 console

1. Open the Amazon Lex V2 console https://console.aws.amazon.com/lexv2.
2. From the list, choose a bot.
3. From the left menu, choose **Analytics** and then choose **CloudWatch metrics**.
4. View metrics for your bot on the **CloudWatch metrics** page.

You can also use the CloudWatch console or API to view your log entries. To find the log entries, navigate to the log group that you configured for the alias. You find the log stream prefix for your logs in the Amazon Lex V2 console or by using the DescribeBotAlias (p. 424) operation.

Log entries for a user utterance is in multiple log streams. An utterance in the conversation has an entry in one of the log streams with the specified prefix. An entry in the log stream contains the following information.

```json
{
  "message-version": "2.0",
  "bot": {
    "id": "string",
    "name": "string",
    "aliasId": "string",
    "aliasName": "string",
    "localeId": "string",
    "version": "string"
  },
  "messages": [{
    "contentType": "PlainText | SSML | CustomPayload | ImageResponseCard",
    "content": "string",
    "imageResponseCard": {
      "title": "string",
      "subtitle": "string",
      "imageUrl": "string",
      "buttonsList": [{
        "text": "string",
        "value": "string"
      }]
    }
  }],
  "sessionState": {
    "dialogAction": {
```
"type": "Close | ConfirmIntent | Delegate | ElicitIntent | ElicitSlot",
"slotToElicit": "string"
},
"intent": {
"name": "string",
"slots": {
"string": {
"value": {
"interpretedValue": "string",
"originalValue": "string",
"resolvedValues": [ "string" ]
}
},
"string": {
"shape": "List",
"value": {
"originalValue": "string",
"interpretedValue": "string",
"resolvedValues": [ "string" ]
}
},
"values": [
{
"shape": "Scalar",
"value": {
"originalValue": "string",
"interpretedValue": "string",
"resolvedValues": [ "string" ]
}
}
],
"kendraResponse": {
// Only present when intent is KendraSearchIntent. For details, see
// https://docs.aws.amazon.com/kendra/latest/dg/API_Query.html#API_Query_ResponseSyntax
},
"state": "InProgress | ReadyForFulfillment | Fulfilled | Failed",
"confirmationState": "Confirmed | Denied | None"
},
"originatingRequestId": "string",
"sessionAttributes": {
"string": "string"
},
"interpretations": [
{
"nluConfidence": "string",
"intent": {
"name": "string",
"slots": {
"string": {
"value": {
"originalValue": "string",
"interpretedValue": "string",
"resolvedValues": [ "string" ]
}
}
},
}
The contents of the log entry depends on the result of a transaction and the configuration of the bot and request.

- The `intent`, `slots`, and `slotToElicit` fields don't appear in an entry if the `missedUtterance` field is true.
- The `s3PathForAudio` field doesn't appear if audio logs are disabled or if the `inputDialogMode` field is `Text`.
- The `responseCard` field only appears when you have defined a response card for the bot.
- The `requestAttributes` map only appears if you have specified request attributes in the request.
- The `kendraResponse` field is only present when the `AMAZON.KendraSearchIntent` makes a request to search an Amazon Kendra index.
- The `developerOverride` field is true when an alternative intent was specified in the bot's Lambda function.
- The `sessionAttributes` map only appears if you have specified session attributes in the request.
- The `sentimentResponse` map only appears if you configure the bot to return sentiment values.

**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.

## Accessing audio logs in Amazon S3

Amazon Lex V2 stores audio logs for your conversations in an S3 bucket.

You can use the Amazon S3 console or API to access audio logs. You can see the S3 object key prefix of the audio files in the Amazon Lex V2 console, or in the `conversationLogSettings` field in the `DescribeBotAlias` operation response.

## Monitoring conversation log status with CloudWatch metrics

Use Amazon CloudWatch to monitor delivery metrics of your conversation logs. You can set alarms on metrics so that you are aware of issues with logging if they should occur.

Amazon Lex V2 provides four metrics in the `AWS/Lex` namespace for conversation logs:

- `ConversationLogsAudioDeliverySuccess`
- `ConversationLogsAudioDeliveryFailure`
- `ConversationLogsTextDeliverySuccess`
- `ConversationLogsTextDeliveryFailure`

The success metrics show that Amazon Lex V2 has successfully written your audio or text logs to their destinations.

The failure metrics show that Amazon Lex V2 couldn't deliver audio or text logs to the specified destination. Typically, this is a configuration error. When your failure metrics are above zero, check the following:
Obscuring slot values in logs

Amazon Lex V2 enables you to obfuscate, or hide, the contents of slots so that the content is not visible. To protect sensitive data captured as slot values, you can enable slot obfuscation to mask those values for logging.

When you choose to obfuscate slot values, Amazon Lex V2 replaces the value of the slot with the name of the slot in conversation logs. For a slot called `full_name`, the value of the slot would be obfuscated as follows:

<table>
<thead>
<tr>
<th>Before:</th>
<th>After:</th>
</tr>
</thead>
<tbody>
<tr>
<td>My name is John Stiles</td>
<td>My name is {full_name}</td>
</tr>
</tbody>
</table>

If an utterance contains bracket characters (`) Amazon Lex V2 escapes the bracket characters with two back slashes (``). For example, the text `{John Stiles}` is obfuscated as follows:

<table>
<thead>
<tr>
<th>Before:</th>
<th>After:</th>
</tr>
</thead>
<tbody>
<tr>
<td>My name is {John Stiles}</td>
<td>My name is <code>\{full_name\}</code></td>
</tr>
</tbody>
</table>

Slot values are obfuscated in conversation logs. The slot values are still available in the response from the `RecognizeText` and `RecognizeUtterance` operations, and the slot values are available to your validation and fulfillment Lambda functions. If you are using slot values in your prompts or responses, those slot values are not obfuscated in conversation logs.

In the first turn of a conversation, Amazon Lex V2 obfuscates slot values if it recognizes a slot and slot value in the utterance. If no slot value is recognized, Amazon Lex V2 does not obfuscate the utterance.

On the second and later turns, Amazon Lex V2 knows the slot to elicit and if the slot value should be obfuscated. If Amazon Lex V2 recognizes the slot value, the value is obfuscated. If Amazon Lex V2 does not recognize a value, the entire utterance is obfuscated. Any slot values in missed utterances won't be obfuscated.

Amazon Lex V2 also doesn't obfuscate slot values that you store in request or session attributes. If you are storing slot values that should be obfuscated as an attribute, you must encrypt or otherwise obfuscate the value.

Amazon Lex V2 doesn't obfuscate the slot value in audio. It does obfuscate the slot value in the audio transcription.

All slots are obfuscated by default. However, you don't need to obfuscate all of the slots in a bot. You can choose which slots obfuscate using the console or by using the Amazon Lex V2 API. In the console, choose **Slot obfuscation** in the settings for a slot. If you are using the API, set the `obfuscationSetting` field of the slot to `DEFAULT_OBFUSCATION` when you call the `CreateSlot (p. 365)` or `UpdateSlot (p. 596)` operation.
Viewing utterance statistics

You can use utterance statistics to determine the utterances that your users are sending to your bot. You can see both the utterances that Amazon Lex V2 successfully detects as well as the utterances that it does not. You can use this information to help tune your bot.

For example, if you find that your users are sending an utterance that Amazon Lex V2 is missing, you can add the utterance to an intent. The Draft version of the intent is updated with the new utterance and you can test it before deploying it to your bot.

An utterance is detected when Amazon Lex V2 recognizes the utterance as an attempt to invoke an intent configured for a bot. An utterance is missed when Amazon Lex V2 doesn't recognize the utterance and invokes the AMAZON.FallbackIntent instead.

Utterance statistics are not generated under the following conditions:

- The Child Online Privacy Protection Act setting was set to Yes when the bot was created with the console, or the childDirected field was set to true when the bot was created with the CreateBot operation.
- You are using slot obfuscation with one or more slots.
- You opted out of participating in improving Amazon Lex.

Using utterance statistics, you can see whether a specific utterance was detected or missed, as well as the last time that the utterance was used in a bot interaction.

Amazon Lex V2 stores utterances continuously while users interact with your bot. You can query the statistics using the console or the ListAggregatedUtterances operation. Utterances are stored for 15 days for use with utterance statistics, and then indefinitely for use in improving the ability of your bot to respond to user input. You can delete utterances using the DeleteUtterances operation or by opting out of data storage. All utterances are deleted if you closes your AWS account. Stored utterances are encrypted with a server-managed key.

Utterance statistics are generated for two combinations:

- Bot, language, version
- Bot, language, alias

When you delete a bot version, utterance statistics are available for the version for up to 15 days. You can't see statistics for deleted version in the Amazon Lex V2 console. To see the statistics for deleted versions, use the ListAggregatedUtterances (p. 476) operation.

If an alias points to a deleted version, you still get statistics from the deleted version when you get statistics for the alias. If the statistics from the ListAggregatedUtterances operation contains information from a deleted version, the containsDataFromDeletedResources field is set to true.

You can choose the time period that you want to see statistics for. You can choose to see hours, days, or weeks of information.

- **Hours** – You can request utterance statistics for 1, 3, 6, 12, or 24 hour time windows. Statistics are refreshed every half hour for 1 hour time windows, and hourly for the other time windows.
- **Days** – You can request utterance statistics for 3 days. Statistics are refreshed every 6 hours.
- **Weeks** – You can see statistics for one or two weeks. Statistics are refreshed every 12 hours for one week time windows, and once per day for two week time windows.
Utterances are aggregated by the text of the utterance. For example, all instances where the customer used the phrase "I want to order a pizza" are aggregated into the same line in a response. When you use the RecognizeUtterance (p. 635) operation, the text used is the input transcript.

To list aggregated utterances, apply the following policy to a role.

```json
{
   "Version": "2012-10-17",
   "Statement": [
      {
         "Sid": "ListAggregatedUtterancesPolicy",
         "Effect": "Allow",
         "Action": "lex:ListAggregatedUtterances",
         "Resource": "*"
      }
   ]
}
```

Logging Amazon Lex V2 API calls with AWS CloudTrail

Amazon Lex V2 is integrated with AWS CloudTrail, a service that provides a record of actions taken by a user, role, or an AWS service in Amazon Lex V2. CloudTrail captures API calls for Amazon Lex V2 as events. The calls captured include calls from the Amazon Lex V2 console and code calls to the Amazon Lex V2 API operations. If you create a trail, you can enable continuous delivery of CloudTrail events to an Amazon S3 bucket, including events for Amazon Lex V2. If you don't configure a trail, you can still view the most recent events in the CloudTrail console in Event history. Using the information collected by CloudTrail, you can determine the request that was made to Amazon Lex V2, the IP address from which the request was made, who made the request, when it was made, and additional details.

To learn more about CloudTrail, see the AWS CloudTrail User Guide.

Amazon Lex V2 information in CloudTrail

CloudTrail is enabled on your AWS account when you create the account. When activity occurs in Amazon Lex V2, that activity is recorded in a CloudTrail event along with other AWS service events in Event history. You can view, search, and download recent events in your AWS account. For more information, see Viewing events with CloudTrail Event history.

For an ongoing record of events in your AWS account, including events for Amazon Lex V2, create a trail. A trail enables CloudTrail to deliver log files to an Amazon S3 bucket. By default, when you create a trail in the console, the trail applies to all AWS Regions. The trail logs events from all Regions in the AWS partition and delivers the log files to the Amazon S3 bucket that you specify. Additionally, you can configure other AWS services to further analyze and act upon the event data collected in CloudTrail logs. For more information, see the following:

- Overview for creating a trail
- CloudTrail supported services and integrations
- Configuring Amazon SNS notifications for CloudTrail
- Receiving CloudTrail log files from multiple regions and Receiving CloudTrail log files from multiple accounts

Amazon Lex V2 supports logging for all of the actions listed in Amazon Lex Model Building V2 (p. 308).
Every event or log entry contains information about who generated the request. The identity information helps you determine the following:

- Whether the request was made with root or AWS Identity and Access Management (IAM) user credentials.
- Whether the request was made with temporary security credentials for a role or federated user.
- Whether the request was made by another AWS service.

For more information, see the CloudTrail userIdentity element.

Understanding Amazon Lex V2 log file entries

A trail is a configuration that enables delivery of events as log files to an Amazon S3 bucket that you specify. CloudTrail log files contain one or more log entries. An event represents a single request from any source and includes information about the requested action, the date and time of the action, request parameters, and so on. CloudTrail log files aren't an ordered stack trace of the public API calls, so they don't appear in any specific order.

The following example shows a CloudTrail log entry that demonstrates the `CreateBotAlias` (p. 318) action.

```json
{
    "eventVersion": "1.05",
    "userIdentity": {
        "type": "AssumedRole",
        "principalId": "ID of caller:temporary credentials",
        "arn": "arn:aws:sts::111122223333:assumed-role/role name/role ARN",
        "accountId": "111122223333",
        "accessKeyId": "AKIAIOSFODNN7EXAMPLE",
        "sessionContext": {
            "sessionIssuer": {
                "type": "Role",
                "principalId": "ID of caller",
                "arn": "arn:aws:iam::111122223333:role/role name",
                "accountId": "111122223333",
                "userName": "role name"
            },
            "webIdFederationData": {},
            "attributes": {
                "mfaAuthenticated": "false",
                "creationDate": "creation date"
            }
        },
        "eventTime": "event timestamp",
        "eventSource": "lex.amazonaws.com",
        "eventName": "CreateBotAlias",
        "awsRegion": "Region",
        "sourceIPAddress": "192.0.2.0",
        "userAgent": "user agent",
        "requestParameters": {
            "botAliasLocaleSettingsMap": {
                "en_US": {
                    "enabled": true
                }
            },
            "botId": "bot ID",
            "botAliasName": "bot aliase name",
            "botVersion": "1"
        },
        "responseElements": {
```
Monitoring Amazon Lex V2 with Amazon CloudWatch

You can monitor Amazon Lex V2 using CloudWatch, which collects raw data and processes it into readable, near real-time metrics. These statistics are kept for 15 months, so that you can access historical information and gain a better perspective on how your web application or service is performing. You can also set alarms that watch for certain thresholds, and send notifications or take actions when those thresholds are met. For more information, see the Amazon CloudWatch User Guide.

The Amazon Lex V2 service reports the following metrics in the AWS/Lex namespace.

<table>
<thead>
<tr>
<th>Metric</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>KendraIndexAccessError</td>
<td>The number of times that Amazon Lex V2 could not access your Amazon Kendra index.</td>
</tr>
<tr>
<td></td>
<td>Valid dimensions: BotId, BotAliasId, LocaleId</td>
</tr>
<tr>
<td></td>
<td>Unit: Count</td>
</tr>
<tr>
<td>KendraLatency</td>
<td>The amount of time that it takes Amazon Kendra to respond to a request from the AMAZON.KendraSearchIntent.</td>
</tr>
<tr>
<td></td>
<td>Valid dimensions: BotId, BotVersion, LocaleId, BotAliasId, LocaleId</td>
</tr>
<tr>
<td></td>
<td>Unit: Milliseconds</td>
</tr>
<tr>
<td>KendraSuccess</td>
<td>The number of times that Amazon Lex V2 couldn't access your Amazon Kendra index.</td>
</tr>
<tr>
<td></td>
<td>Valid dimensions: BotId, BotVersion, LocaleId, BotAliasId, LocaleId</td>
</tr>
<tr>
<td>Metric</td>
<td>Description</td>
</tr>
<tr>
<td>------------------------</td>
<td>-----------------------------------------------------------------------------------------------------------------------------------------------</td>
</tr>
<tr>
<td>KendraSystemErrors</td>
<td>The number of times that Amazon Lex V2 couldn't query the Amazon Kendra index.</td>
</tr>
<tr>
<td></td>
<td>Valid dimensions:</td>
</tr>
<tr>
<td></td>
<td>• Operation, BotId, BotAliasId, InputMode, LocaleId</td>
</tr>
<tr>
<td></td>
<td>Unit: Count</td>
</tr>
<tr>
<td>KendraThrottledEvents</td>
<td>The number of times Amazon Kendra throttled requests from the AMAZON.KendraSearchIntent.</td>
</tr>
<tr>
<td></td>
<td>Valid dimensions:</td>
</tr>
<tr>
<td></td>
<td>• Operation, BotId, BotAliasId, InputMode, LocaleId</td>
</tr>
<tr>
<td></td>
<td>Unit: Count</td>
</tr>
<tr>
<td>MissedUtteranceCount</td>
<td>The number of utterances that were not recognized in the specified time period.</td>
</tr>
<tr>
<td></td>
<td>Valid dimensions:</td>
</tr>
<tr>
<td></td>
<td>• Operation, BotId, BotVersion, InputMode, LocaleId</td>
</tr>
<tr>
<td></td>
<td>• Operation, BotId, BotAliasId, InputMode, LocaleId</td>
</tr>
<tr>
<td></td>
<td>Unit: Count</td>
</tr>
<tr>
<td>RuntimeConcurrency</td>
<td>The number of concurrent connections in the specified time period. RuntimeConcurrency is reported as a StatisticSet.</td>
</tr>
<tr>
<td></td>
<td>Valid dimensions for the RecognizeUtterance or StartConversation operations:</td>
</tr>
<tr>
<td></td>
<td>• Operation, BotId, BotVersion, InputMode, LocaleId</td>
</tr>
<tr>
<td></td>
<td>• Operation, BotId, BotAliasId, InputMode, LocaleId</td>
</tr>
<tr>
<td></td>
<td>Valid dimensions for other operations:</td>
</tr>
<tr>
<td></td>
<td>• Operation, BotId, BotVersion, LocaleId</td>
</tr>
<tr>
<td></td>
<td>• Operation, BotId, BotAliasId, LocaleId</td>
</tr>
<tr>
<td></td>
<td>Unit: Count</td>
</tr>
<tr>
<td>RuntimeInvalidLambdaResponses</td>
<td>The number of invalid AWS Lambda responses in the specified period.</td>
</tr>
<tr>
<td></td>
<td>Valid dimensions:</td>
</tr>
<tr>
<td></td>
<td>• Operation, BotId, BotAliasId, InputMode, LocaleId</td>
</tr>
<tr>
<td></td>
<td>Unit: Count</td>
</tr>
<tr>
<td>Metric</td>
<td>Description</td>
</tr>
<tr>
<td>---------------------------</td>
<td>------------------------------------------------------------------------------</td>
</tr>
<tr>
<td><strong>RuntimeLambdaErrors</strong></td>
<td>The number of Lambda runtime errors in the specified time period.</td>
</tr>
<tr>
<td></td>
<td></td>
</tr>
<tr>
<td><strong>RuntimePollyErrors</strong></td>
<td>The number of invalid Amazon Polly responses in the specified time period.</td>
</tr>
<tr>
<td></td>
<td></td>
</tr>
<tr>
<td><strong>RuntimeRequestCount</strong></td>
<td>The number of runtime requests in the specified time period.</td>
</tr>
<tr>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
</tr>
<tr>
<td><strong>RuntimeRequestLength</strong></td>
<td>Total length of a conversation with a Amazon Lex V2 bot. Only applicable to</td>
</tr>
<tr>
<td></td>
<td>the StartConversation (p. 642) operation.</td>
</tr>
<tr>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
</tr>
<tr>
<td>Metric</td>
<td>Description</td>
</tr>
<tr>
<td>---------------------------------</td>
<td>-------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------</td>
</tr>
<tr>
<td>RuntimeSuccessfulRequestLatency</td>
<td>The latency for successful requests between the time the request was made and the response was passed back. Valid dimensions for the RecognizeUtterance and StartConversation operations:</td>
</tr>
</tbody>
</table>
|                                 | • Operation, BotId, BotVersion, InputMode, LocaleId  
|                                 | • Operation, BotId, BotAliasId, InputMode, LocaleId  
|                                 | Valid dimensions for other operations:  |
|                                 | • Operation, BotId, BotVersion, LocaleId  
|                                 | • Operation, BotId, BotAliasId, LocaleId  
|                                 | Unit: milliseconds                                                                                                                                  |
| RuntimeSystemErrors             | The number of system errors in the specified period. The response code range for a system error is 500 to 599. Valid dimensions for the RecognizeUtterance and StartConversation operations:  |
|                                 | • Operation, BotId, BotVersion, InputMode, LocaleId  
|                                 | • Operation, BotId, BotAliasId, InputMode, LocaleId  
|                                 | Valid dimensions for other operations:  |
|                                 | • Operation, BotId, BotVersion, LocaleId  
|                                 | • Operation, BotId, BotAliasId, LocaleId  
|                                 | Unit: Count                                                                                                                                   |
| RuntimeThrottledEvents          | The number of throttled events. Amazon Lex V2 throttles an event 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 AWS service limits. Valid dimensions for the RecognizeUtterance and StartConversation operations:  |
|                                 | • Operation, BotId, BotVersion, InputMode, LocaleId  
|                                 | • Operation, BotId, BotAliasId, InputMode, LocaleId  
|                                 | Valid dimensions for other operations:  |
|                                 | • Operation, BotId, BotVersion, LocaleId  
|                                 | • Operation, BotId, BotAliasId, LocaleId  
|                                 | Unit: Count                                                                                                                                   |
### Metric | Description
---|---
RuntimeUserErrors | The number of user errors in the specified period. The response code range for a user error is 400 to 499.

Valid dimensions for the RecognizeUtterance and StartConversation operations:
- Operation, BotId, BotVersion, InputMode, LocaleId
- Operation, BotId, BotAliasId, InputMode, LocaleId

Valid dimensions for other operations:
- Operation, BotId, BotVersion, LocaleId
- Operation, BotId, BotAliasId, LocaleId

Unit: Count

---

The following dimensions are supported for the Amazon Lex V2 metrics.

<table>
<thead>
<tr>
<th>Dimension</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Operation</td>
<td>The name of the Amazon Lex V2 operation – RecognizeText, RecognizeUtterance, StartConversation, GetSession, PutSession, DeleteSession – that generated the entry.</td>
</tr>
<tr>
<td>BotId</td>
<td>The alphanumeric unique identifier for the bot.</td>
</tr>
<tr>
<td>BotAliasId</td>
<td>The alphanumeric unique identifier for the bot alias.</td>
</tr>
<tr>
<td>BotVersion</td>
<td>The numeric version of the bot.</td>
</tr>
<tr>
<td>InputMode</td>
<td>The type of input to the bot – speech, text, or DTMF.</td>
</tr>
<tr>
<td>LocaleId</td>
<td>The identifier of the bot's locale, such as en-US or fr-CA.</td>
</tr>
</tbody>
</table>
Importing and exporting

You can export a bot definition, a bot locale or a custom vocabulary and then import it back to create a new resource or to overwrite an existing resource in an AWS account. For example, you can export a bot from a test account and then create a copy of the bot in your production account. You can also copy a bot from one AWS Region to another Region.

You can change the resources of the exported resource before importing it. For example, you can export a bot and then edit the JSON file for a slot to add or remove slot value elicitation utterances from a specific slot. After you finish editing the definition, you can import the modified file.

Topics
- Exporting (p. 240)
- Importing (p. 242)
- Using a password when importing or exporting (p. 245)
- JSON format for importing and exporting (p. 245)

Exporting

You export a bot, bot locale, or custom vocabulary using the console or the StartExport operation. You specify the resource to export, and you can provide an optional password to help protect the .zip file when you start an export. After you download the .zip file, you must use the password to access the file before you can use it. For more information, see Using a password when importing or exporting (p. 245).

Exporting is an asynchronous operation. Once you have started the export, you can use the console or the DescribeExport operation to monitor the progress of the export. Once the export is complete, the console or the DescribeExport operation shows a status of COMPLETED, and the console downloads the export .zip file to your browser. If you use the DescribeExport operation, Amazon Lex V2 provides a pre-signed Amazon S3 URL where you can download the results of the export. The download URL is available for only five minutes, but you can get a new URL by calling the DescribeExport operation again.

You can see the history of exports for a resource with the console or with the ListExports operation. The results show the exports along with their current status. An export is available in the history for seven days.

When you export the Draft version of a bot or a bot locale, it is possible for the definition in the JSON file to be in an inconsistent state because the Draft version of a bot or bot locale can be changed while an export is in progress. If the Draft version is changed while it is being exported, the changes may not be included in the export file.

When you export a bot locale, Amazon Lex exports all of the information that defines the locale, including the locale, custom vocabulary, intents, slot types, and slots.

When you export a bot, Amazon Lex exports all of the locales defined for the bot, including the intents, slot types, and slots. The following items are not exported with a bot:

- Bot aliases
- Role ARN associated with a bot
- Tags associated with bots and bot aliases
• Lambda code hooks associated with a bot alias

The role ARN and tags are entered as request parameters when you import a bot. You need to create bot aliases and assign Lambda code hooks after importing, if necessary.

You can remove an export and the associated .zip file using the console or the `DeleteExport` operation.

For an example of exporting a bot using the console, see Exporting a bot (console) (p. 241).

IAM permissions required to export

To export bots, bot locales, and custom vocabularies, the user running the export must have the following IAM permissions.

<table>
<thead>
<tr>
<th>API</th>
<th>Required IAM actions</th>
<th>Resource</th>
</tr>
</thead>
<tbody>
<tr>
<td>CreateExport (p. 333)</td>
<td>• CreateExport</td>
<td>Bot</td>
</tr>
<tr>
<td>UpdateExport (p. 569)</td>
<td>• UpdateExport</td>
<td>Bot</td>
</tr>
<tr>
<td>DescribeExport (p. 443)</td>
<td>• DescribeExport</td>
<td>Bot</td>
</tr>
<tr>
<td></td>
<td>• DescribeBot</td>
<td></td>
</tr>
<tr>
<td></td>
<td>• DescribeCustomVocabulary</td>
<td></td>
</tr>
<tr>
<td></td>
<td>• DescribeLocale</td>
<td></td>
</tr>
<tr>
<td></td>
<td>• DescribeIntent</td>
<td></td>
</tr>
<tr>
<td></td>
<td>• DescribeSlot</td>
<td></td>
</tr>
<tr>
<td></td>
<td>• DescribeSlotType</td>
<td></td>
</tr>
<tr>
<td></td>
<td>• ListLocale</td>
<td></td>
</tr>
<tr>
<td></td>
<td>• ListIntent</td>
<td></td>
</tr>
<tr>
<td></td>
<td>• ListSlot</td>
<td></td>
</tr>
<tr>
<td></td>
<td>• ListSlotType</td>
<td></td>
</tr>
<tr>
<td>DescribeExport (p. 443) for custom vocabularies</td>
<td>• DescribeExport</td>
<td>bot</td>
</tr>
<tr>
<td></td>
<td>• DescribeCustomVocabulary</td>
<td></td>
</tr>
<tr>
<td>DeleteExport (p. 401)</td>
<td>• DeleteExport</td>
<td>Bot</td>
</tr>
<tr>
<td>ListExports (p. 505)</td>
<td>• ListExports</td>
<td>*</td>
</tr>
</tbody>
</table>

For an example IAM policy, see Allow a user to export bots and bot locales (p. 270).

Exporting a bot (console)

You can export a bot from the bot list, from the list of versions, or from the version details page. When you choose a version, Amazon Lex V2 exports that version. The following instructions assume that you start exporting the bot from the list of bots, but when you start with a version the steps are the same.

To export a bot using the console

1. Sign in to the AWS Management Console and open the Amazon Lex V2 console at https://console.aws.amazon.com/lexv2/.
2. From the list of bots, choose the bot to export.
3. From **Action**, choose **Export**.
4. Choose the bot version, platform, and export format.
5. (Optional) Enter a password for the .zip file. Providing a password helps protect the output archive.
6. Choose **Export**.

After you start the export, you return to the list of bots. To monitor the progress of the export, use the Import/export history list. When the status of the export is **Complete**, the console automatically downloads the .zip file to your computer.

To download the export again, on the import/export list, choose the export, and then choose **Download**. You can provide a password for the downloaded .zip file.

**To export a bot language**

1. Sign in to the AWS Management Console and open the Amazon Lex V2 console at https://console.aws.amazon.com/lexv2/.
2. From the list of bots, choose the bot whose language you want to export.
3. From **Add languages**, choose **View languages**.
4. From the **All languages** list, choose the language to export.
5. From **Action**, choose **Export**.
6. Choose the bot version, platform, and format.
7. (Optional) Enter a password for the .zip file. Providing a password helps protect the output archive.
8. Choose **Export**.

After you start the export, you return to the list of languages. To monitor the progress of the export, use the Import/export history list. When the status of the export is **Complete**, the console automatically downloads the .zip file to your computer.

To download the export again, on the import/export list, choose the export, and then choose **Download**. You can provide a password for the downloaded .zip file.

**Importing**

To use the console to import a previously exported bot, bot locale or custom vocabulary, you provide the file location on your local computer and the optional password to unlock the file. For an example, see Importing a bot (console) (p. 244).

When you use the API, importing a resource is a three step process:

1. Create an upload URL using the CreateUploadUrl operation. You don't need to create an upload URL when you are using the console.
2. Upload the .zip file that contains the resource definition.
3. Start the import with the StartImport operation.

The upload URL is a pre-signed Amazon S3 URL with write permission. The URL is available for five minutes after it is generated. If you password protect the .zip file, you must provide the password when you start the import. For more information, see Using a password when importing or exporting (p. 245).

An import is an asynchronous process. You can monitor the progress of an import using the console or the DescribeImport operation.
When you import a bot or bot locale, there may be conflicts between resource names in the import file and existing resource names in Amazon Lex V2. Amazon Lex V2 can handle the conflict in two ways:

- **Fail on conflict** – The import stops and no resources are imported from the import .zip file.
- **Overwrite** – Amazon Lex V2 imports all of the resources from the import .zip file, replacing any existing resource with the definition from the import file.

You can see a list of the imports to a resource using the console or the `ListImports` operation. Imports remain in the list for seven days. You can use the console or the `DescribeImport` operation to see details about a specific import.

You can also remove an import and the associated .zip file using the console or the `DeleteImport` operation.

For an example of importing a bot using the console, see Importing a bot (console) (p. 244).

### IAM permissions required to import

To import bots, bot locales, and custom vocabularies, the user running the import must have the following IAM permissions.

<table>
<thead>
<tr>
<th>API in the console (p. 384)</th>
<th>Required IAM actions</th>
<th>Resource</th>
</tr>
</thead>
<tbody>
<tr>
<td>CreateUploadUrl</td>
<td>CreateUploadUrl</td>
<td>*</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>StartImport (p. 541) for bot and bot locale</th>
<th>Required IAM actions</th>
<th>Resource</th>
</tr>
</thead>
<tbody>
<tr>
<td>StartImport</td>
<td>iam:PassRole, CreateBot, CreateCustomVocabulary, CreateLocale, CreateIntent, CreateSlot, CreateSlotType, UpdateBot, UpdateCustomVocabulary, UpdateLocale, UpdateIntent, UpdateSlot, UpdateSlotType, DeleteBot, DeleteCustomVocabulary, DeleteLocale, DeleteIntent, DeleteSlot, DeleteSlotType</td>
<td>1. To import a new bot: bot, bot alias. 2. To overwrite an existing bot: bot. 3. To import a new locale: bot.</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>StartImport (p. 541) for custom vocabularies</th>
<th>Required IAM actions</th>
<th>Resource</th>
</tr>
</thead>
<tbody>
<tr>
<td>StartImport, CreateCustomVocabulary, DeleteCustomVocabulary, UpdateCustomVocabulary</td>
<td>bot</td>
<td></td>
</tr>
</tbody>
</table>
Importing a bot (console)

To import a bot using the console

1. Sign in to the AWS Management Console and open the Amazon Lex V2 console at https://console.aws.amazon.com/lexv2/.
2. From Action, choose Import.
3. In Input file, give the bot a name and then choose the .zip file that contains the JSON files that define the bot.
4. If the .zip file is password protected, enter the password for the .zip file. Password protecting the archive is optional, but it helps protect the contents.
5. Create or enter the IAM role that defines permissions for your bot.
6. Indicate whether your bot is subject to the Children's Online Privacy Protection Act (COPPA).
7. Provide an idle timeout setting for your bot. If you don't provide a value, the value from the zip file is used. If the .zip file does not contain a timeout setting, Amazon Lex V2 uses the default of 300 seconds (five minutes).
8. (Optional) Add tags for your bot.
9. Choose whether to warn about overwriting existing bots with the same name. If you enable warnings, if the bot you are importing would overwrite an existing bot, you receive a warning and the bot is not imported. If you disable warnings, the imported bot replaces the existing bot with the same name.
10. Choose Import.

After you start the import, you return to the list of bots. To monitor the progress of the import, use the Import/export history list. When the status of the import is Complete, you can choose the bot from the list of bots to modify or build the bot.

To import a bot language

1. Sign in to the AWS Management Console and open the Amazon Lex V2 console at https://console.aws.amazon.com/lexv2/.
2. From the list of bots, choose the bot to which you want to import a language.
3. From Add languages, choose View languages.
4. From Action, choose import.
5. In Input file, choose the file that contains the language to import. If you protected the .zip file, provide the password in Password.
6. In Language, choose the language to import as. The language doesn't have to match the language in the import file. You can copy the intents from one language to another.
7. In Voice, choose the Amazon Polly voice to use for voice interaction, or choose None for a text-only bot.
8. In **Confidence score threshold**, enter the threshold where Amazon Lex V2 inserts the `AMAZON.FallbackIntent`, the `AMAZON.KendraSearchIntent`, or both when returning alternative intents.

9. Choose whether to warn about overwriting an existing language. If you enable warnings, if the language you are importing would overwrite an existing language, you receive a warning and the language is not imported. If you disable warnings, the imported language replaces the existing language.

10. Choose **Import** to start importing the language.

After you start the import, you return to the list of languages. To monitor the progress of the import, use the **Import/export history** list. When the status of the import is **Complete**, you can choose the language from the list of bots to modify or build the bot.

---

### Using a password when importing or exporting

Amazon Lex V2 can password protect your export archives or read your protected import archives using standard .zip file compression. You should always password protect your import and export archives.

Amazon Lex V2 sends your export archive to an S3 bucket, and it is available to you with a pre-signed S3 URL. The URL is only available for five minutes. The archive is available to anyone with access to the download URL. To help protect the data in the archive, provide a password when you export the resource. If you need to get the archive after the URL expires, you can use the console or the `DescribeExport` operation to get a new URL.

If you lose the password for an export archive, you can create a new password for an existing file by choosing **Download** from the import/export history table or by using the `UpdateExport` operation. If you choose **Download** in the history table for an export and you don't provide a password, Amazon Lex V2 downloads an unprotected zip file.

---

### JSON format for importing and exporting

You import and export bots, bot locales, or custom vocabularies from Amazon Lex V2 using a .zip file that contains JSON structures that describe the parts of the resource. When you export a resource, Amazon Lex V2 creates the .zip file and makes it available to you using an Amazon S3 pre-signed URL. When you import a resource, you must create a .zip file that contains the JSON structures and upload it to an S3 pre-signed URL.

Amazon Lex creates the following directory structure in the .zip file when you export a bot. When you export a bot locale, only the structure under the locale is exported. When you export a custom vocabulary, only the structure under the custom vocabulary is exported.

```plaintext
BotName_BotVersion_ExportID_LexJson.zip
-or-
BotName_BotVersion_LocaleId_ExportId_LEX_JSON.zip

--> manifest.json
---> BotName
-------> Bot.json
-------> BotLocales
---------> Locale_A
----------> BotLocale.json
----------> Intents
-----------> Intent_A
-----------> Intent.json
```
Manifest file structure

The manifest file contains metadata for the export file.

```json
{  
  "metadata": {  
    "schemaVersion": "1.0",  
    "fileFormat": "LexJson",  
    "resourceType": "Bot | BotLocale | CustomVocabulary"  
  }  
}
```

Bot file structure

The bot file contains the configuration information for the bot.

```json
{  
  "name": "BotName",  
  "identifier": "identifier",  
  "version": "number",  
  "description": "description",  
  "dataPrivacy": {  
    "childDirected": true | false  
  },  
  "idleSessionTTLInSeconds": seconds  
}
```

Bot locale file structure

The bot locale file contains a description of the locale or language of a bot. When you export a bot, there can be more than one bot locale file in the .zip file. When you export a bot locale, there is only one locale in the zip file.

```json
{  
  "name": "locale name",  
  "identifier": "locale ID",  
  "version": "number",  
  "description": "description",  
  "voiceSettings": {  
```
Intent file structure

The intent file contains the configuration information for an intent. There is one intent file in the .zip file for each intent in a specific locale.

The following is an example of a JSON structure for the BookCar intent in the sample BookTrip bot. For a complete example of the JSON structure for an intent, see the CreateIntent (p. 337) operation.

```json
{
  "name": "BookCar",
  "identifier": "891RWHHICO",
  "description": "Intent to book a car.",
  "parentIntentSignature": null,
  "sampleUtterances": [
    {
      "utterance": "Book a car"
    },
    {
      "utterance": "Reserve a car"
    },
    {
      "utterance": "Make a car reservation"
    }
  ],
  "intentConfirmationSetting": {
    "confirmationPrompt": {
      "messageGroupList": [
        {
          "message": {
            "plainTextMessage": {
              "value": "OK, I have you down for a {CarType} hire in {PickUpCity} from {PickUpDate} to {ReturnDate}. Should I book the reservation?"
            },
            "ssmlMessage": null,
            "customPayload": null,
            "imageResponseCard": null
          },
          "variations": null
        },
        {
          "maxRetries": 2
        },
        "declinationResponse": {
          "messageGroupList": [
            {
              "message": {
                "plainTextMessage": {
                  "value": "OK, I have cancelled your reservation in progress."
                },
                "ssmlMessage": null,
                "customPayload": null,
                "imageResponseCard": null
              },
              "variations": null
            }
          ]
        }
      ]
    }
  }
}
```
Slot file structure

The slot file contains the configuration information for a slot in an intent. There is one slot file in the .zip file for each slot defined for an intent in a specific locale.

The following example is the JSON structure of a slot that enables the customer to choose the type of car they wish to rent in the BookCar intent in the BookTrip example bot. For a complete example of the JSON structure for an slot, see the CreateSlot (p. 365) operation.

```
{
    "name": "CarType",
    "identifier": "KDHJWNGZGC",
    "description": "Type of car being reserved.",
    "multipleValuesSetting": {
        "allowMutlipleValues": false
    },
    "slotTypeName": "CarTypeValues",
    "obfuscationSetting": null,
    "slotConstraint": "Required",
    "defaultValueSpec": null,
    "slotValueElicitationSetting": {
        "promptSpecification": {
            "messageGroupList": [
                {
                    "message": {
                        "plainTextMessage": {
                            "value": "What type of car would you like to rent? Our most popular options are economy, midsize, and luxury"
                        },
                        "ssmlMessage": null,
                        "customPayload": null,
                        "imageResponseCard": null
                    }
                }
            ]
        }
    }
}
```
Slot type file structure

The slot type file contains the configuration information for a custom slot type used in a language or locale. There is one slot type file in the .zip file for each custom slot type in a specific locale.

The following is the JSON structure for the slot type that lists the types of cars available in the BookTrip example bot. For a complete example of the JSON structure for a slot type, see the CreateSlotType (p. 378) operation.

```json
{
    "name": "CarTypeValues",
    "identifier": "T1YUHG9Z9R",
    "description": "Enumeration representing possible types of cars available for hire",
    "slotTypeValues": [{
        "synonyms": null,
        "sampleValue": {
            "value": "economy"
        }
    }, {
        "synonyms": null,
        "sampleValue": {
            "value": "standard"
        }
    }, {
        "synonyms": null,
        "sampleValue": {
            "value": "midsize"
        }
    }, {
        "synonyms": null,
        "sampleValue": {
            "value": "full size"
        }
    }, {
        "synonyms": null,
        "sampleValue": {
            "value": "luxury"
        }
    }, {
        "synonyms": null,
        "sampleValue": {
            "value": "minivan"
        }
    }],
    "parentSlotTypeSignature": null,
    "valueSelectionSetting": {
        "resolutionStrategy": "TOP_RESOLUTION",
        "advancedRecognitionSetting": {
            "audioRecognitionStrategy": "UseSlotValuesAsCustomVocabulary"
        },
        "regexFilter": null
    }
}
```
The following is a slot type that uses a custom grammar to understand the customer's utterances. For more information, see Using a custom grammar slot type (p. 43).

```json
{
  "name": "custom_grammar",
  "identifier": "7KEAQIQKPX",
  "description": "Slot type using a custom grammar",
  "slotTypeValues": null,
  "parentSlotTypeSignature": null,
  "valueSelectionSetting": null,
  "externalSourceSetting": {
    "grammarSlotTypeSetting": {
      "source": {
        "kmsKeyArn": "arn:aws:kms:Region:123456789012:alias/customer-grxml-key",
        "s3BucketName": "grxml-test",
        "s3ObjectKey": "grxml_files/grammar.grxml"
      }
    }
  }
}
```

**Custom vocabulary file structure.**

The custom vocabulary file contains the entries in a custom vocabulary for single language or locale. There is one custom vocabulary file in the .zip file for each locale that has a custom vocabulary.

The following is a custom vocabulary file for a bot that takes restaurant orders. There is one file per locale in the bot.

```json
{
  "customVocabularyItems": [
    {
      "weight": 3,
      "phrase": "wafers"
    },
    {
      "weight": null,
      "phrase": "extra large"
    },
    {
      "weight": null,
      "phrase": "cremini mushroom soup"
    },
    {
      "weight": null,
      "phrase": "ramen"
    },
    {
      "weight": null,
      "phrase": "orzo"
    }
  ]
}
```
Tagging resources

To help you manage your Amazon Lex V2 bots and bot aliases, you can assign metadata to each resource as *tags*. A tag is a label that you assign to an AWS resource. Each tag consists of a key and a value.

Tags enable you to categorize your AWS resources in different ways, for example, by purpose, owner, or application. Tags help you to:

- Identify and organize your AWS resources. Many AWS resources support tagging, so you can assign the same tag to resources in different services to indicate that the resources are the same. For example, you can tag a bot and the Lambda functions that it uses with the same tag.
- Allocate costs. You activate tags on the AWS Billing and Cost Management dashboard. AWS uses the tags to categorize your costs and deliver a monthly cost allocation report to you. For Amazon Lex V2, you can allocate costs for each alias using tags specific to the alias. For more information, see Use cost allocation tags in the *AWS Billing and Cost Management User Guide*.
- Control access to your resources. You can use tags with Amazon Lex V2 to create policies to control access to Amazon Lex V2 resources. These policies can be attached to an IAM role or user to enable tag-based access control.

You can work with tags using the AWS Management Console, the AWS Command Line Interface, or the Amazon Lex V2 API.

Tagging your resources

If you are using the Amazon Lex V2 console, you can tag resources when you create them, or you can add the tags later. You can also use the console to update or remove existing tags.

If you are using the AWS CLI or Amazon Lex V2 API, you use the following operations to manage tags for your resource:

- `CreateBot` (p. 313) and `CreateBotAlias` (p. 318) – apply tags when you create a bot or a bot alias.
- `ListTagsForResource` (p. 530) – view the tags associated with a resource.
- `TagResource` (p. 545) – add and modify tags on an existing resource.
- `UntagResource` (p. 547) – remove tags from a resource.

The following resources in Amazon Lex V2 support tagging:

- **Bots** – use an Amazon Resource Name (ARN) like the following:
  - `arn:aws:lex:{Region}:{account}:bot/{bot-id}`
- **Bot aliases** – use an ARN like the following:
  - `arn:aws:lex:{Region}:{account}:bot-alias/{bot-id}/bot-alias-id`

The `{bot-id}` and `{bot-alias-id}` values are capitalized alphanumeric strings 10 characters long.

Tag restrictions

The following basic restrictions apply to tags on Amazon Lex V2 resources:
Tagging resources (console)

You can use the console to manage tags on a bot or bot alias. You can add tags when you create the resource, or you can add, modify, or remove tags from existing resources.

To add a tag when you create a bot

1. Sign in to the AWS Management Console and open the Amazon Lex V2 console at https://console.aws.amazon.com/lexv2/.
2. Choose Create bot.
3. In the Advanced settings section of Configure bot settings, choose Add new tag. You can add tags to the bot and to the TestBotAlias alias.
4. Choose Next to continue creating your bot.

To add a tag when you create a bot alias

1. Sign in to the AWS Management Console and open the Amazon Lex V2 console at https://console.aws.amazon.com/lexv2/.
2. Choose the bot that you want to add the bot alias to.
3. From the left menu, choose Aliases and then choose Create alias.
4. In General info, choose Add new tag from Tags.
5. Choose Create.

To add, remove, or modify a tag on an existing bot

1. Sign in to the AWS Management Console and open the Amazon Lex V2 console at https://console.aws.amazon.com/lexv2/.
2. Choose the bot that you want to modify.
3. From the left menu, choose Settings, and then choose Edit.
4. In Tags, make your changes.
5. Choose Save to save your changes to the bot.

To add, remove, or modify a tag on an existing alias

1. Sign in to the AWS Management Console and open the Amazon Lex V2 console at https://console.aws.amazon.com/lexv2/.
2. Choose the bot that you want to modify.
3. From the left menu, choose Aliases and then from the list of aliases, choose the alias to modify.
4. From Alias details, in Tags, choose Modify tags.
5. In Manage tags, make your changes.
6. Choose Save to save your changes to the alias.
Security in Amazon Lex V2

Cloud security at AWS is the highest priority. As an AWS customer, you benefit from data centers and network architectures that are built to meet the requirements of the most security-sensitive organizations.

Security is a shared responsibility between AWS and you. The shared responsibility model describes this as security of the cloud and security in the cloud:

- **Security of the cloud** – AWS is responsible for protecting the infrastructure that runs AWS services in the AWS Cloud. AWS also provides you with services that you can use securely. Third-party auditors regularly test and verify the effectiveness of our security as part of the AWS Compliance Programs. To learn about the compliance programs that apply to Amazon Lex V2, see AWS Services in Scope by Compliance Program.

- **Security in the cloud** – Your responsibility is determined by the AWS service that you use. You are also responsible for other factors including the sensitivity of your data, your company’s requirements, and applicable laws and regulations.

This documentation helps you understand how to apply the shared responsibility model when using Amazon Lex V2. The following topics show you how to configure Amazon Lex V2 to meet your security and compliance objectives. You also learn how to use other AWS services that help you to monitor and secure your Amazon Lex V2 resources.

**Topics**
- Data protection in Amazon Lex V2 (p. 253)
- Identity and access management for Amazon Lex V2 (p. 255)
- Using service-linked roles for Amazon Lex V2 (p. 288)
- Logging and monitoring in Amazon Lex V2 (p. 291)
- Compliance validation for Amazon Lex V2 (p. 291)
- Resilience in Amazon Lex V2 (p. 292)
- Infrastructure security in Amazon Lex V2 (p. 292)
- Amazon Lex V2 and interface VPC endpoints (AWS PrivateLink) (p. 292)

Data protection in Amazon Lex V2

Amazon Lex V2 conforms to the AWS shared responsibility model, which includes regulations and guidelines for data protection. AWS is responsible for protecting the global infrastructure that runs all the AWS services. AWS maintains control over data hosted on this infrastructure, including the security configuration controls for handling customer content and personal data. AWS customers and APN partners, acting either as data controllers or data processors, are responsible for any personal data that they put in the AWS Cloud.

For data protection purposes, we recommend that you protect AWS account credentials and set up individual user accounts with AWS Identity and Access Management (IAM), so that each user is given only the permissions necessary to fulfill their job duties. We also recommend that you secure your data in the following ways:

- Use multi-factor authentication (MFA) with each account.
- Use SSL/TLS to communicate with AWS resources.
Encryption at rest

Amazon Lex V2 encrypts user utterances and other information that it stores.

Topics
- Sample utterances (p. 254)
- Session attributes (p. 254)
- Request attributes (p. 254)

Sample utterances

When you develop a bot, you can provide sample utterances for each intent and slot. You can also provide custom values and synonyms for slots. This information is encrypted at rest, and it is used only to build the bot and create the customer experience.

Session attributes

Session attributes contain application-specific information that is passed between Amazon Lex V2 and client applications. Amazon Lex V2 passes session attributes to all AWS Lambda functions configured for a bot. If a Lambda function adds or updates session attributes, Amazon Lex V2 passes the new information back to the client application.

Session attributes persist in an encrypted store for the duration of the session. You can configure the session to remain active for a minimum of 1 minute and up to 24 hours after the last user utterance. The default session duration is 5 minutes.

Request attributes

Request attributes contain request-specific information and apply only to the current request. A client application uses request attributes to send information to Amazon Lex V2 at runtime.

You use request attributes to pass information that doesn't need to persist for the entire session. Because request attributes don't persist across requests, they aren't stored.

Encryption in transit

Amazon Lex V2 uses the HTTPS protocol to communicate with your client application. It uses HTTPS and AWS signatures to communicate with other services, such as Amazon Polly and AWS Lambda on your application's behalf.
Identity and access management for Amazon Lex V2

AWS Identity and Access Management (IAM) is an AWS service that helps an administrator securely control access to AWS resources. IAM administrators control who can be *authenticated* (signed in) and *authorized* (have permissions) to use Amazon Lex V2 resources. IAM is an AWS service that you can use with no additional charge.

**Topics**
- Audience (p. 255)
- Authenticating with identities (p. 255)
- Managing access using policies (p. 257)
- How Amazon Lex V2 works with IAM (p. 259)
- Identity-based policy examples for Amazon Lex V2 (p. 266)
- Resource-based policy examples for Amazon Lex V2 (p. 274)
- AWS managed policies for Amazon Lex V2 (p. 280)
- Troubleshooting Amazon Lex V2 identity and access (p. 287)

**Audience**

How you use AWS Identity and Access Management (IAM) differs, depending on the work that you do in Amazon Lex V2.

**Service user** – If you use the Amazon Lex V2 service to do your job, then your administrator provides you with the credentials and permissions that you need. As you use more Amazon Lex V2 features to do your work, you might need additional permissions. Understanding how access is managed can help you request the right permissions from your administrator. If you cannot access a feature in Amazon Lex V2, see Troubleshooting Amazon Lex V2 identity and access (p. 287).

**Service administrator** – If you're in charge of Amazon Lex V2 resources at your company, you probably have full access to Amazon Lex V2. It's your job to determine which Amazon Lex V2 features and resources your employees should access. You must then submit requests to your IAM administrator to change the permissions of your service users. Review the information on this page to understand the basic concepts of IAM. To learn more about how your company can use IAM with Amazon Lex V2, see How Amazon Lex V2 works with IAM (p. 259).

**IAM administrator** – If you're an IAM administrator, you might want to learn details about how you can write policies to manage access to Amazon Lex V2. To view example Amazon Lex V2 identity-based policies that you can use in IAM, see Identity-based policy examples for Amazon Lex V2 (p. 266).

**Authenticating with identities**

Authentication is how you sign in to AWS using your identity credentials. For more information about signing in using the AWS Management Console, see Signing in to the AWS Management Console as an IAM user or root user in the IAM User Guide.

You must be *authenticated* (signed in to AWS) as the AWS account root user, an IAM user, or by assuming an IAM role. You can also use your company's single sign-on authentication or even sign in using Google or Facebook. In these cases, your administrator previously set up identity federation using IAM roles. When you access AWS using credentials from another company, you are assuming a role indirectly.
To sign in directly to the AWS Management Console, use your password with your root user email address or your IAM user name. You can access AWS programmatically using your root user or IAM users access keys. AWS provides SDK and command line tools to cryptographically sign your request using your credentials. If you don't use AWS tools, you must sign the request yourself. Do this using 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.

Regardless of the authentication method that you use, you might also be required to provide additional security information. For example, AWS recommends that you use multi-factor authentication (MFA) to increase the security of your account. To learn more, see Using multi-factor authentication (MFA) in AWS in the IAM User Guide.

AWS account root user

When you first create an AWS account, you begin with a single sign-in identity that has complete access to all AWS services and resources in the account. This identity is called the AWS account root user and is accessed by signing in with the email address and password that you used to create the account. We strongly recommend that you do not use the root user for your everyday tasks, even the administrative ones. Instead, adhere to the best practice of using the root user only to create your first IAM user. Then securely lock away the root user credentials and use them to perform only a few account and service management tasks.

IAM users and groups

An IAM user is an identity within your AWS account that has specific permissions for a single person or application. An IAM user can have long-term credentials such as a user name and password or a set of access keys. To learn how to generate access keys, see Managing access keys for IAM users in the IAM User Guide. When you generate access keys for an IAM user, make sure you view and securely save the key pair. You cannot recover the secret access key in the future. Instead, you must generate a new access key pair.

An IAM group is an identity that specifies a collection of IAM users. You can't sign in as a group. You can use groups to specify permissions for multiple users at a time. Groups make permissions easier to manage for large sets of users. For example, you could have a group named IAMAdmins and give that group permissions to administer IAM resources.

Users are different from roles. A user is uniquely associated with one person or application, but a role is intended to be assumable by anyone who needs it. Users have permanent long-term credentials, but roles provide temporary credentials. To learn more, see When to create an IAM user (instead of a role) in the IAM User Guide.

IAM roles

An IAM role is an identity within your AWS account that has specific permissions. It is similar to an IAM user, but is not associated with a specific person. You can temporarily assume an IAM role in the AWS Management Console by switching roles. You can assume a role by calling an AWS CLI or AWS API operation or by using a custom URL. For more information about methods for using roles, see Using IAM roles in the IAM User Guide.

IAM roles with temporary credentials are useful in the following situations:

- **Temporary IAM user permissions** – An IAM user can assume an IAM role to temporarily take on different permissions for a specific task.
- **Federated user access** – Instead of creating an IAM user, you can use existing 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 to allow someone (a trusted principal) in a different account to access resources in your account. Roles are the primary way to grant cross-account access. However, with some AWS services, you can attach a policy directly to a resource (instead of using a role as a proxy). To learn the difference between roles and resource-based policies for cross-account access, see [How IAM roles differ from resource-based policies](#) in the IAM User Guide.

• **Cross-service access** – Some AWS services use features in other AWS services. For example, when you make a call in a service, it's common for that service to run applications in Amazon EC2 or store objects in Amazon S3. A service might do this using the calling principal's permissions, using a service role, or using a service-linked role.

• **Principal permissions** – When you use an IAM user or role to perform actions in AWS, you are considered a principal. Policies grant permissions to a principal. When you use some services, you might perform an action that then triggers another action in a different service. In this case, you must have permissions to perform both actions. To see whether an action requires additional dependent actions in a policy, see [Actions, resources, and condition keys for Amazon Lex V2](#) in the Service Authorization Reference.

• **Service role** – A service role is an IAM role that a service assumes to perform actions on your behalf. An IAM administrator can create, modify, and delete a service role from within IAM. For more information, see [Creating a role to delegate permissions to an AWS service](#) in the IAM User Guide.

• **Service-linked role** – A service-linked role is a type of service role that is linked to an AWS service. The service can assume the role to perform an action on your behalf. Service-linked roles appear in your IAM account and are owned by the service. An IAM administrator can view, but not edit the permissions for service-linked roles.

• **Applications running on Amazon EC2** – You can use an IAM role to manage temporary credentials for applications that are running on an EC2 instance and making AWS CLI or 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 that are running on the EC2 instance to get temporary credentials. For more information, see [Using an IAM role to grant permissions to applications running on Amazon EC2 instances](#) in the IAM User Guide.

To learn whether to use IAM roles or IAM users, see [When to create an IAM role (instead of a user)](#) in the IAM User Guide.

### Managing access using policies

You control access in AWS by creating policies and attaching them to IAM identities or AWS resources. A policy is an object in AWS that, when associated with an identity or resource, defines their permissions. You can sign in as the root user or an IAM user, or you can assume an IAM role. When you then make a request, AWS evaluates the related identity-based or resource-based policies. Permissions in the policies determine whether the request is allowed or denied. Most policies are stored in AWS as JSON documents. For more information about the structure and contents of JSON policy documents, see [Overview of JSON policies](#) in the IAM User Guide.

Administrators can use AWS JSON policies to specify who has access to what. That is, which principal can perform actions on what resources, and under what conditions.

Every IAM entity (user or role) starts with no permissions. In other words, by default, users can do nothing, not even change their own password. To give a user permission to do something, an administrator must attach a permissions policy to a user. Or the administrator can add the user to a group that has the intended permissions. When an administrator gives permissions to a group, all users in that group are granted those permissions.

IAM policies define permissions for an action regardless of the method that you use to perform the operation. For example, suppose that you have a policy that allows the `iam:GetRole` action. A user with that policy can get role information from the AWS Management Console, the AWS CLI, or the AWS API.
Identity-based policies

Identity-based policies are JSON permissions policy documents that you can attach to an identity, such as an IAM user, group of users, or role. These policies control what actions users and roles can perform, on which resources, and under what conditions. To learn how to create an identity-based policy, see Creating IAM policies in the IAM User Guide.

Identity-based policies can be further categorized as inline policies or managed policies. Inline policies are embedded directly into a single user, group, or role. Managed policies are standalone policies that you can attach to multiple users, groups, and roles in your AWS account. Managed policies include AWS managed policies and customer managed policies. To learn how to choose between a managed policy or an inline policy, see Choosing between managed policies and inline policies in the IAM User Guide.

Resource-based policies

Resource-based policies are JSON policy documents that you attach to a resource. Examples of resource-based policies are IAM role trust policies and Amazon S3 bucket policies. In services that support resource-based policies, service administrators can use them to control access to a specific resource. For the resource where the policy is attached, the policy defines what actions a specified principal can perform on that resource and under what conditions. You must specify a principal in a resource-based policy. Principals can include accounts, users, roles, federated users, or AWS services.

Resource-based policies are inline policies that are located in that service. You can't use AWS managed policies from IAM in a resource-based policy.

Access control lists (ACLs)

Access control lists (ACLs) control which principals (account members, users, or roles) have permissions to access a resource. ACLs are similar to resource-based policies, although they do not use the JSON policy document format.

Amazon S3, AWS WAF, and Amazon VPC are examples of services that support ACLs. To learn more about ACLs, see Access control list (ACL) overview in the Amazon Simple Storage Service Developer Guide.

Other policy types

AWS supports additional, less-common policy types. These policy types can set the maximum permissions granted to you by the more common policy types.

- **Permissions boundaries** – A permissions boundary is an advanced feature in which you set the maximum permissions that an identity-based policy can grant to an IAM entity (IAM user or role). You can set a permissions boundary for an entity. The resulting permissions are the intersection of entity's identity-based policies and its permissions boundaries. Resource-based policies that specify the user or role in the Principal field are not limited by the permissions boundary. An explicit deny in any of these policies overrides the allow. For more information about permissions boundaries, see Permissions boundaries for IAM entities in the IAM User Guide.

- **Service control policies (SCPs)** – SCPs are JSON policies that specify the maximum permissions for an organization or organizational unit (OU) in AWS Organizations. AWS Organizations is a service for grouping and centrally managing multiple AWS accounts that your business owns. If you enable all features in an organization, then you can apply service control policies (SCPs) to any or all of your accounts. The SCP limits permissions for entities in member accounts, including each AWS account root user. For more information about Organizations and SCPs, see How SCPs work in the AWS Organizations User Guide.

- **Session policies** – Session policies are advanced policies that you pass as a parameter when you programmatically create a temporary session for a role or federated user. The resulting session's permissions are the intersection of the user or role's identity-based policies and the session policies.
Permissions can also come from a resource-based policy. An explicit deny in any of these policies overrides the allow. For more information, see Session policies in the IAM User Guide.

### Multiple policy types

When multiple types of policies apply to a request, the resulting permissions are more complicated to understand. To learn how AWS determines whether to allow a request when multiple policy types are involved, see Policy evaluation logic in the IAM User Guide.

### How Amazon Lex V2 works with IAM

Before you use IAM to manage access to Amazon Lex V2, learn what IAM features are available to use with Amazon Lex V2.

#### IAM features you can use with Amazon Lex V2

<table>
<thead>
<tr>
<th>IAM feature</th>
<th>Amazon Lex V2 support</th>
</tr>
</thead>
<tbody>
<tr>
<td>Identity-based policies (p. 259)</td>
<td>Yes</td>
</tr>
<tr>
<td>Resource-based policies (p. 260)</td>
<td>Yes</td>
</tr>
<tr>
<td>Policy actions (p. 262)</td>
<td>Yes</td>
</tr>
<tr>
<td>Policy resources (p. 263)</td>
<td>Yes</td>
</tr>
<tr>
<td>Policy condition keys (p. 264)</td>
<td>No</td>
</tr>
<tr>
<td>ACLs (p. 264)</td>
<td>No</td>
</tr>
<tr>
<td>ABAC (tags in policies) (p. 264)</td>
<td>Yes</td>
</tr>
<tr>
<td>Temporary credentials (p. 265)</td>
<td>No</td>
</tr>
<tr>
<td>Principal permissions (p. 265)</td>
<td>Yes</td>
</tr>
<tr>
<td>Service roles (p. 265)</td>
<td>Yes</td>
</tr>
<tr>
<td>Service-linked roles (p. 265)</td>
<td>Partial</td>
</tr>
</tbody>
</table>

To get a high-level view of how Amazon Lex V2 and other AWS services work with most IAM features, see AWS services that work with IAM in the IAM User Guide.

#### Identity-based policies for Amazon Lex V2

<table>
<thead>
<tr>
<th>Supports identity-based policies</th>
<th>Yes</th>
</tr>
</thead>
</table>

Identity-based policies are JSON permissions policy documents that you can attach to an identity, such as an IAM user, group of users, or role. These policies control what actions users and roles can perform, on which resources, and under what conditions. To learn how to create an identity-based policy, see Creating IAM policies in the IAM User Guide.

With IAM identity-based policies, you can specify allowed or denied actions and resources as well as the conditions under which actions are allowed or denied. You can't specify the principal in an identity-based
policy because it applies to the user or role to which it is attached. To learn about all of the elements that you can use in a JSON policy, see IAM JSON policy elements reference in the IAM User Guide.

Identity-based policy examples for Amazon Lex V2

To view examples of Amazon Lex V2 identity-based policies, see Identity-based policy examples for Amazon Lex V2 (p. 266).

Resource-based policies within Amazon Lex V2

<table>
<thead>
<tr>
<th>Supports resource-based policies</th>
<th>Resource</th>
</tr>
</thead>
<tbody>
<tr>
<td>Yes</td>
<td>BotId</td>
</tr>
</tbody>
</table>

Resource-based policies are JSON policy documents that you attach to a resource. Examples of resource-based policies are IAM role trust policies and Amazon S3 bucket policies. In services that support resource-based policies, service administrators can use them to control access to a specific resource. For the resource where the policy is attached, the policy defines what actions a specified principal can perform on that resource and under what conditions. You must specify a principal in a resource-based policy. Principals can include users, roles, federated users, or AWS services.

You can't use cross-account or cross-region policies with Amazon Lex. If you create a policy for a resource with a cross-account or cross-region ARN, Amazon Lex returns an error.

The Amazon Lex service supports resource-based policies called a bot policy and a bot alias policy, which are attached to a bot or a bot alias. These policies define which principals can perform actions on the bot or bot alias.

Actions can only be used on specific resources. For example, the UpdateBot action can only be used on bot resources, the UpdateBotAlias action can only be used on bot alias resources. If you specify an action in a policy that can't be used on the resource specified in the policy, Amazon Lex returns an error. For the list of actions and the resources that they can be used with, see the following table.

<table>
<thead>
<tr>
<th>Action</th>
<th>Supports resource-based policy</th>
<th>Resource</th>
</tr>
</thead>
<tbody>
<tr>
<td>BuildBotLocale</td>
<td>Supported</td>
<td>BotId</td>
</tr>
<tr>
<td>CreateBot</td>
<td>No</td>
<td>BotId</td>
</tr>
<tr>
<td>CreateBotAlias</td>
<td>No</td>
<td>BotId</td>
</tr>
<tr>
<td>CreateBotChannel [permission only]</td>
<td>Supported</td>
<td>BotId</td>
</tr>
<tr>
<td>CreateBotLocale</td>
<td>Supported</td>
<td>BotId</td>
</tr>
<tr>
<td>CreateBotVersion</td>
<td>Supported</td>
<td>BotId</td>
</tr>
<tr>
<td>CreateExport</td>
<td>Supported</td>
<td>BotId</td>
</tr>
<tr>
<td>CreateIntent</td>
<td>Supported</td>
<td>BotId</td>
</tr>
<tr>
<td>CreateResourcePolicy</td>
<td>Supported</td>
<td>BotId, BotAliasId</td>
</tr>
<tr>
<td>CreateSlot</td>
<td>Supported</td>
<td>BotId</td>
</tr>
<tr>
<td>CreateSlotType</td>
<td>Supported</td>
<td>BotId</td>
</tr>
<tr>
<td>CreateUploadUrl</td>
<td>No</td>
<td>BotId</td>
</tr>
<tr>
<td>Action</td>
<td>Supports resource-based policy</td>
<td>Resource</td>
</tr>
<tr>
<td>--------------------------------</td>
<td>--------------------------------</td>
<td>-----------------------</td>
</tr>
<tr>
<td>DeleteBot</td>
<td>Supported</td>
<td>BotId, BotAliasId</td>
</tr>
<tr>
<td>DeleteBotAlias</td>
<td>Supported</td>
<td>BotAliasId</td>
</tr>
<tr>
<td>DeleteBotChannel [permission only]</td>
<td>Supported</td>
<td>BotId</td>
</tr>
<tr>
<td>DeleteBotLocale</td>
<td>Supported</td>
<td>BotId</td>
</tr>
<tr>
<td>DeleteBotVersion</td>
<td>Supported</td>
<td>BotId</td>
</tr>
<tr>
<td>DeleteExport</td>
<td>Supported</td>
<td>BotId</td>
</tr>
<tr>
<td>DeleteImport</td>
<td>Supported</td>
<td>BotId</td>
</tr>
<tr>
<td>DeleteIntent</td>
<td>Supported</td>
<td>BotId</td>
</tr>
<tr>
<td>DeleteResourcePolicy</td>
<td>Supported</td>
<td>BotId, BotAliasId</td>
</tr>
<tr>
<td>DeleteSession</td>
<td>Supported</td>
<td>BotAliasId</td>
</tr>
<tr>
<td>DeleteSlot</td>
<td>Supported</td>
<td>BotId</td>
</tr>
<tr>
<td>DeleteSlotType</td>
<td>Supported</td>
<td>BotId</td>
</tr>
<tr>
<td>DescribeBot</td>
<td>Supported</td>
<td>BotId</td>
</tr>
<tr>
<td>DescribeBotAlias</td>
<td>Supported</td>
<td>BotAliasId</td>
</tr>
<tr>
<td>DescribeBotChannel [permission only]</td>
<td>Supported</td>
<td>BotId</td>
</tr>
<tr>
<td>DescribeBotLocale</td>
<td>Supported</td>
<td>BotId</td>
</tr>
<tr>
<td>DescribeBotVersion</td>
<td>Supported</td>
<td>BotId</td>
</tr>
<tr>
<td>DescribeExport</td>
<td>Supported</td>
<td>BotId</td>
</tr>
<tr>
<td>DescribeImport</td>
<td>Supported</td>
<td>BotId</td>
</tr>
<tr>
<td>DescribeIntent</td>
<td>Supported</td>
<td>BotId</td>
</tr>
<tr>
<td>DescribeResourcePolicy</td>
<td>Supported</td>
<td>BotId, BotAliasId</td>
</tr>
<tr>
<td>DescribeSlot</td>
<td>Supported</td>
<td>BotId</td>
</tr>
<tr>
<td>DescribeSlotType</td>
<td>Supported</td>
<td>BotId</td>
</tr>
<tr>
<td>GetSession</td>
<td>Supported</td>
<td>BotAliasId</td>
</tr>
<tr>
<td>ListBotAliases</td>
<td>Supported</td>
<td>BotAliasId</td>
</tr>
<tr>
<td>ListBotChannels [permission only]</td>
<td>Supported</td>
<td>BotId</td>
</tr>
<tr>
<td>ListBotLocales</td>
<td>Supported</td>
<td>BotId</td>
</tr>
<tr>
<td>ListBots</td>
<td>No</td>
<td></td>
</tr>
<tr>
<td>ListBotVersions</td>
<td>Supported</td>
<td>BotId</td>
</tr>
<tr>
<td>Action</td>
<td>Supports resource-based policy</td>
<td>Resource</td>
</tr>
<tr>
<td>--------------------------------</td>
<td>--------------------------------</td>
<td>---------------</td>
</tr>
<tr>
<td>ListBuiltInIntents</td>
<td>No</td>
<td></td>
</tr>
<tr>
<td>ListBuiltIntSlotTypes</td>
<td>No</td>
<td></td>
</tr>
<tr>
<td>ListExports</td>
<td>No</td>
<td></td>
</tr>
<tr>
<td>ListImports</td>
<td>No</td>
<td></td>
</tr>
<tr>
<td>ListIntents</td>
<td>Supported</td>
<td>BotId</td>
</tr>
<tr>
<td>ListSlots</td>
<td>Supported</td>
<td>BotId</td>
</tr>
<tr>
<td>ListSlotTypes</td>
<td>Supported</td>
<td>BotId</td>
</tr>
<tr>
<td>PutSession</td>
<td>Supported</td>
<td>BotAliasId</td>
</tr>
<tr>
<td>RecognizeText</td>
<td>Supported</td>
<td>BotAliasId</td>
</tr>
<tr>
<td>RecognizeUtterance</td>
<td>Supported</td>
<td>BotAliasId</td>
</tr>
<tr>
<td>StartConversation</td>
<td>Supported</td>
<td>BotAliasId</td>
</tr>
<tr>
<td>StartImport</td>
<td>Supported</td>
<td>BotId, BotAliasId</td>
</tr>
<tr>
<td>TagResource</td>
<td>No</td>
<td></td>
</tr>
<tr>
<td>UpdateBot</td>
<td>Supported</td>
<td>BotId</td>
</tr>
<tr>
<td>UpdateBotAlias</td>
<td>Supported</td>
<td>BotAliasId</td>
</tr>
<tr>
<td>UpdateBotLocale</td>
<td>Supported</td>
<td>BotId</td>
</tr>
<tr>
<td>UpdateBotVersion</td>
<td>Supported</td>
<td>BotId</td>
</tr>
<tr>
<td>UpdateExport</td>
<td>Supported</td>
<td>BotId</td>
</tr>
<tr>
<td>UpdateIntent</td>
<td>Supported</td>
<td>BotId</td>
</tr>
<tr>
<td>UpdateResourcePolicy</td>
<td>Supported</td>
<td>BotId, BotAliasId</td>
</tr>
<tr>
<td>UpdateSlot</td>
<td>Supported</td>
<td>BotId</td>
</tr>
<tr>
<td>UpdateSlotType</td>
<td>Supported</td>
<td>BotId</td>
</tr>
<tr>
<td>UntagResource</td>
<td>No</td>
<td></td>
</tr>
</tbody>
</table>

To learn how to attach a resource-based policy to a bot or bot alias, see Resource-based policy examples for Amazon Lex V2 (p. 274).

**Resource-based policy examples within Amazon Lex V2**

To view examples of Amazon Lex V2 resource-based policies, see Resource-based policy examples for Amazon Lex V2 (p. 274).

**Policy actions for Amazon Lex V2**

<table>
<thead>
<tr>
<th>Supports policy actions</th>
<th>Yes</th>
</tr>
</thead>
</table>

262
Administrators can use AWS JSON policies to specify who has access to what. That is, which principal can perform actions on what resources, and under what conditions.

The Action element of a JSON policy describes the actions that you can use to allow or deny access in a policy. Policy actions usually have the same name as the associated AWS API operation. There are some exceptions, such as permission-only actions that don't have a matching API operation. There are also some operations that require multiple actions in a policy. These additional actions are called dependent actions.

Include actions in a policy to grant permissions to perform the associated operation.

To see a list of Amazon Lex V2 actions, see Actions defined by Amazon Lex V2 in the Service Authorization Reference.

Policy actions in Amazon Lex V2 use the following prefix before the action:

```
lex
```

To specify multiple actions in a single statement, separate them with commas.

```
"Action": [
  "lex:action1",
  "lex:action2"
]
```

To view examples of Amazon Lex V2 identity-based policies, see Identity-based policy examples for Amazon Lex V2 (p. 266).

**Policy resources for Amazon Lex V2**

<table>
<thead>
<tr>
<th>Supports policy resources</th>
<th>Yes</th>
</tr>
</thead>
</table>

Administrators can use AWS JSON policies to specify who has access to what. That is, which principal can perform actions on what resources, and under what conditions.

The Resource JSON policy element specifies the object or objects to which the action applies. Statements must include either a Resource or a NotResource element. As a best practice, specify a resource using its Amazon Resource Name (ARN). You can do this for actions that support a specific resource type, known as resource-level permissions.

For actions that don't support resource-level permissions, such as listing operations, use a wildcard (*) to indicate that the statement applies to all resources.

```
"Resource": "*
```

To see a list of Amazon Lex V2 resource types and their ARNs, see Resources defined by Amazon Lex V2 in the Service Authorization Reference. To learn with which actions you can specify the ARN of each resource, see Actions defined by Amazon Lex V2.

To view examples of Amazon Lex V2 identity-based policies, see Identity-based policy examples for Amazon Lex V2 (p. 266).
Policy condition keys for Amazon Lex V2

| Supports policy condition keys | No |

Administrators can use AWS JSON policies to specify who has access to what. That is, which principal can perform actions on what resources, and under what conditions.

The Condition element (or Condition block) lets you specify conditions in which a statement is in effect. The Condition element is optional. You can create conditional expressions that use condition operators, such as equals or less than, to match the condition in the policy with values in the request.

If you specify multiple Condition elements in a statement, or multiple keys in a single Condition element, AWS evaluates them using a logical AND operation. If you specify multiple values for a single condition key, AWS evaluates the condition using a logical OR operation. All of the conditions must be met before the statement's permissions are granted.

You can also use placeholder variables when you specify conditions. For example, you can grant an IAM user permission to access a resource only if it is tagged with their IAM user name. For more information, see IAM policy elements: variables and tags in the IAM User Guide.

AWS supports global condition keys and service-specific condition keys. To see all AWS global condition keys, see AWS global condition context keys in the IAM User Guide.

To see a list of Amazon Lex V2 condition keys, see Condition keys for Amazon Lex V2 in the Service Authorization Reference. To learn with which actions and resources you can use a condition key, see Actions defined by Amazon Lex V2.

To view examples of Amazon Lex V2 identity-based policies, see Identity-based policy examples for Amazon Lex V2 (p. 266).

Access control lists (ACLs) in Amazon Lex V2

| Supports ACLs | No |

Access control lists (ACLs) control which principals (account members, users, or roles) have permissions to access a resource. ACLs are similar to resource-based policies, although they do not use the JSON policy document format.

Attribute-based access control (ABAC) with Amazon Lex V2

| Supports ABAC (tags in policies) | Yes |

Attribute-based access control (ABAC) is an authorization strategy that defines permissions based on attributes. In AWS, these attributes are called tags. You can attach tags to IAM entities (users or roles) and to many AWS resources. Tagging entities and resources is the first step of ABAC. Then you design ABAC policies to allow operations when the principal's tag matches the tag on the resource that they are trying to access.

ABAC is helpful in environments that are growing rapidly and helps with situations where policy management becomes cumbersome.
To control access based on tags, you provide tag information in the condition element of a policy using the `aws:ResourceTag/key-name`, `aws:RequestTag/key-name`, or `aws:TagKeys` condition keys.

For more information about ABAC, see What is ABAC? in the IAM User Guide. To view a tutorial with steps for setting up ABAC, see Use attribute-based access control (ABAC) in the IAM User Guide.

**Using Temporary credentials with Amazon Lex V2**

| Supports temporary credentials | No |

Some AWS services don't work when you sign in using temporary credentials. For additional information, including which AWS services work with temporary credentials, see AWS services that work with IAM in the IAM User Guide.

You are using temporary credentials if you sign in to the AWS Management Console using any method except a user name and password. For example, when you access AWS using your company's single sign-on (SSO) link, that process automatically creates temporary credentials. You also automatically create temporary credentials when you sign in to the console as a user and then switch roles. For more information about switching roles, see Switching to a role (console) in the IAM User Guide.

You can manually create temporary credentials using the AWS CLI or AWS API. You can then use those temporary credentials to access AWS. AWS recommends that you dynamically generate temporary credentials instead of using long-term access keys. For more information, see Temporary security credentials in IAM.

**Cross-service principal permissions for Amazon Lex V2**

| Supports principal permissions | Yes |

When you use an IAM user or role to perform actions in AWS, you are considered a principal. Policies grant permissions to a principal. When you use some services, you might perform an action that then triggers another action in a different service. In this case, you must have permissions to perform both actions. To see whether an action requires additional dependent actions in a policy, see Actions, resources, and condition keys for Amazon Lex V2 in the Service Authorization Reference.

**Service roles for Amazon Lex V2**

| Supports service roles | Yes |

A service role is an IAM role that a service assumes to perform actions on your behalf. An IAM administrator can create, modify, and delete a service role from within IAM. For more information, see Creating a role to delegate permissions to an AWS service in the IAM User Guide.

**Warning**

Changing the permissions for a service role might break Amazon Lex V2 functionality. Edit service roles only when Amazon Lex V2 provides guidance to do so.

**Service-linked roles for Amazon Lex V2**

| Supports service-linked roles | Partial |
A service-linked role is a type of service role that is linked to an AWS service. The service can assume the role to perform an action on your behalf. Service-linked roles appear in your IAM account and are owned by the service. An IAM administrator can view, but not edit the permissions for service-linked roles.

For details about creating or managing service-linked roles, see AWS services that work with IAM. Find a service in the table that includes a Yes in the Service-linked role column. Choose the Yes link to view the service-linked role documentation for that service.

Identity-based policy examples for Amazon Lex V2

By default, IAM users and roles don't have permission to create or modify Amazon Lex V2 resources. They also can't perform tasks using the AWS Management Console, AWS CLI, or AWS API. An IAM administrator must create IAM policies that grant users and roles permission to perform actions on the resources that they need. The administrator must then attach those policies to the IAM users or groups that require those permissions.

To learn how to create an IAM identity-based policy using these example JSON policy documents, see Creating IAM policies in the IAM User Guide.

Topics
- Policy best practices (p. 266)
- Using the Amazon Lex V2 console (p. 267)
- Allow users to add functions to a bot (p. 267)
- Allow users to add channels to a bot (p. 267)
- Allow users to create and update bots (p. 268)
- Allow users to use the Automated Chatbot Designer (p. 268)
- Allow users to use a AWS KMS key to encrypt and decrypt files (p. 269)
- Allow users to delete bots (p. 269)
- Allow users to have a conversation with a bot (p. 270)
- Allow a specific user to manage resource-based policies (p. 270)
- Allow a user to export bots and bot locales (p. 270)
- Allow a user to export a custom vocabulary (p. 271)
- Allow a user to import bots and bot locales (p. 271)
- Allow a user to import a custom vocabulary (p. 272)
- Allow a user to migrate a bot from Amazon Lex to Amazon Lex V2 (p. 272)
- Allow users to view their own permissions (p. 273)

Policy best practices

Identity-based policies are very powerful. They determine whether someone can create, access, or delete Amazon Lex V2 resources in your account. These actions can incur costs for your AWS account. When you create or edit identity-based policies, follow these guidelines and recommendations:

- Get started using AWS managed policies – To start using Amazon Lex V2 quickly, use AWS managed policies to give your employees the permissions they need. These policies are already available in your account and are maintained and updated by AWS. For more information, see Get started using permissions with AWS managed policies in the IAM User Guide.
- Grant least privilege – When you create custom policies, grant only the permissions required to perform a task. Start with a minimum set of permissions and grant additional permissions as necessary. Doing so is more secure than starting with permissions that are too lenient and then trying to tighten them later. For more information, see Grant least privilege in the IAM User Guide.
Enable MFA for sensitive operations – For extra security, require IAM users to use multi-factor authentication (MFA) to access sensitive resources or API operations. For more information, see Using multi-factor authentication (MFA) in AWS in the IAM User Guide.

Use policy conditions for extra security – To the extent that it's practical, define the conditions under which your identity-based policies allow access to a resource. For example, you can write conditions to specify a range of allowable IP addresses that a request must come from. You can also write conditions to allow requests only within a specified date or time range, or to require the use of SSL or MFA. For more information, see IAM JSON policy elements: Condition in the IAM User Guide.

Using the Amazon Lex V2 console

To access the Amazon Lex V2 console, you must have a minimum set of permissions. These permissions must allow you to list and view details about the Amazon Lex V2 resources in your AWS account. If you create an identity-based policy that is more restrictive than the minimum required permissions, the console won’t function as intended for entities (IAM users or roles) with that policy.

You don't need to allow minimum console permissions for users that are making calls only to the AWS CLI or the AWS API. Instead, allow access to only the actions that match the API operation that you're trying to perform.

To ensure that users and roles can still use the Amazon Lex V2 console, users need to have Console access. For more information about creating a user with Console access, see Creating an IAM user in your AWS account in the IAM User Guide.

Allow users to add functions to a bot

This example shows a policy that allows IAM users to add Amazon Comprehend, sentiment analysis and Amazon Kendra query permissions to an Amazon Lex V2 bot.

```
{
   "Version": "2012-10-17",
   "Statement": [
      {
         "Sid": "Id1",
         "Effect": "Allow",
         "Action": "iam:PutRolePolicy",
         "Resource": "arn:aws:iam::*:role/aws-service-role/lexv2.amazonaws.com/AWSServiceRoleForLexV2Bots"
      },
      {
         "Sid": "Id2",
         "Effect": "Allow",
         "Action": "iam:GetRolePolicy",
         "Resource": "arn:aws:iam::*:role/aws-service-role/lexv2.amazonaws.com/AWSServiceRoleForLexV2Bots"
      }
   ]
}
```

Allow users to add channels to a bot

This example is a policy that allows IAM users to add a messaging channel to a bot. A user must have this policy in place before they can deploy a bot on a messaging platform.

```
{
   "Version": "2012-10-17",
   "Statement": [
      {
         "Sid": "Id3",
         "Effect": "Allow",
         "Action": "iam:GetRolePolicy",
         "Resource": "arn:aws:iam::*:role/aws-service-role/lexv2.amazonaws.com/AWSServiceRoleForLexV2Bots"
      }
   ]
}
```
Identity-based policy examples

```
{
    "Sid": "Id1",
    "Effect": "Allow",
    "Action": "iam:PutRolePolicy",
    "Resource": "arn:aws:iam::*:role/aws-service-role/channels.lexv2.amazonaws.com/AWSServiceRoleForLexV2Channels"
},
{
    "Sid": "Id2",
    "Effect": "Allow",
    "Action": "iam:GetRolePolicy",
    "Resource": "arn:aws:iam::*:role/aws-service-role/channels.lexv2.amazonaws.com/AWSServiceRoleForLexV2Channels"
}
]
```

Allow users to create and update bots

This example shows an example policy that allows IAM users to create and update any bot. The policy includes permissions to complete this action on the console or using the AWS CLI or AWS API.

```
{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Action": [
                "lex:CreateBot",
                "lex:UpdateBot",
                "iam:PassRole"
            ],
            "Effect": "Allow",
            "Resource": ["arn:aws:lex:Region:123412341234:bot/*"]
        }
    ]
}
```

Allow users to use the Automated Chatbot Designer

This example shows an example policy that allows IAM users to run the Automated Chatbot Designer.

```
{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Effect": "Allow",
            "Action": [
                "s3:GetObject",
                "s3:ListBucket"
            ],
            "Resource": [
                "arn:aws:s3::<customer-bucket>/<bucketName>",
                # Resource should point to the bucket or an explicit folder.
                # Provide this to read the entire bucket
                "arn:aws:s3::<customer-bucket>/<bucketName>/*",
                # Provide this to read a specific folder
                "arn:aws:s3::<customer-bucket>/<bucketName>/<pathFormat>/*"
            ]
        },
        {
            "Effect": "Allow",
            "Action": ["s3:PutObject"
            ],
            "Resource": "arn:aws:s3::<customer-bucket>/<bucketName>/<pathFormat>/*"
        }
    ]
}
```

# Use this if your S3 bucket is encrypted with a KMS key.
“Effect”: "Allow",
“Action”: [  
  "kms:Decrypt"
],
“Resource”: [  
  "arn:aws:kms:<Region>:<customerAccountId>:key/<kmsKeyId>"
  
}]

Allow users to use a AWS KMS key to encrypt and decrypt files

This example shows an example policy that allows IAM users to use a AWS KMS customer managed key to encrypt and decrypt data.

```json
{
  "Version": "2012-10-17",
  "Id": "sample-policy",
  "Statement": [
    {
      "Sid": "Allow Lex access",
      "Effect": "Allow",
      "Principal": {  
        "Service": "lexv2.amazonaws.com"
      },
      "Action": [  
        # If the key is for encryption
        "kms:Encrypt",
        "kms:GenerateDataKey"
        # If the key is for decryption
        "kms:Decrypt"
      ],
      "Resource": "*"
    }
  ]
}
```

Allow users to delete bots

This example shows an example policy that allows IAM users to delete any bot. The policy includes permissions to complete this action on the console or using the AWS CLI or AWS API.

```json
{
  "Version": "2012-10-17",
  "Statement": [
    {
      "Action": [  
        "lex:DeleteBot",
        "lex:DeleteBotLocale",
        "lex:DeleteBotAlias",
        "lex:DeleteIntent",
        "lex:DeleteSlot",
        "lex:DeleteSlottype"
      ],
      "Effect": "Allow",
      "Resource": [
        "arn:aws:lex:<Region>:123412341234:bot/*",
        "arn:aws:lex:<Region>:123412341234:bot-alias/*"
      ]
    }
  ]
}
```
Allow users to have a conversation with a bot

This example shows an example policy that allows IAM users have a conversation with any bot. The policy includes permissions to complete this action on the console or using the AWS CLI or AWS API.

```json
{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Action": [
                "lex:StartConversation",
                "lex:RecognizeText",
                "lex:RecognizeUtterance",
                "lex:GetSession",
                "lex:PutSession",
                "lex:DeleteSession"
            ],
            "Effect": "Allow",
            "Resource": "arn:aws:lex:Region:123412341234:bot-alias/*"
        }
    ]
}
```

Allow a specific user to manage resource-based policies

The following example grants permission for a specific user to manage the resource-based policies. It allows console and API access to the policies associated with bots and bot aliases.

```json
{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Sid": "ResourcePolicyEditor",
            "Effect": "Allow",
            "Principal": {
                "AWS": "arn:aws:iam::123456789012:role/ResourcePolicyEditor"
            },
            "Action": [
                "lex:CreateResourcePolicy",
                "lex:UpdateResourcePolicy",
                "lex:DeleteResourcePolicy",
                "lex:DescribeResourcePolicy"
            ]
        }
    ]
}
```

Allow a user to export bots and bot locales

The following IAM permission policy enables a user to create, update, and get an export for a bot or bot locale.

```json
{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Action": [
                "lex:CreateExport",
                "lex:UpdateExport",
                "lex:DescribeExport"
            ]
        }
    ]
}
```
Identity-based policy examples

```
"lex:DescribeBot",
"lex:DescribeBotLocale",
"lex:ListBotLocales",
"lex:DescribeIntent",
"lex:ListIntents",
"lex:DescribeSlotType",
"lex:ListSlotTypes",
"lex:DescribeSlot",
"lex:ListSlots",
"lex:DescribeCustomVocabulary"
],
"Effect": "Allow",
"Resource": ["arn:aws:lex:Region:123456789012:bot/*"]
}
```

Allow a user to export a custom vocabulary

The following IAM permission policy allows a user to export a custom vocabulary from a bot locale.

```
{"Version": "2012-10-17",
 "Statement": [
  {
   "Action": [
    "lex:CreateExport",
    "lex:UpdateExport",
    "lex:DescribeExport",
    "lex:DescribeCustomVocabulary"
   ],
   "Effect": "Allow",
   "Resource": ["arn:aws:lex:Region:123456789012:bot/*"]
  }
 ]
}
```

Allow a user to import bots and bot locales

The following IAM permission policy allows a user to import a bot or bot locale and to check the status of an import.

```
{"Version": "2012-10-17",
 "Statement": [
  {
   "Action": [
    "lex:CreateUploadUrl",
    "lex:StartImport",
    "lex:DescribeImport",
    "lex:CreateBot",
    "lex:UpdateBot",
    "lex:DeleteBot",
    "lex:CreateBotLocale",
    "lex:UpdateBotLocale",
    "lex:DeleteBotLocale",
    "lex:CreateIntent",
    "lex:UpdateIntent",
    "lex:DeleteIntent",
    "lex:CreateSlotType",
    "lex:UpdateSlotType",
    "lex:DeleteSlotType",
    "lex:CreateSlot",
   
```
Allow a user to import a custom vocabulary

The following IAM permission policy allows a user to import a custom vocabulary to a bot locale.

```
{
   "Version": "2012-10-17",
   "Statement": [
      {
         "Action": [
            "lex:CreateUploadUrl",
            "lex:StartImport",
            "lex:DescribeImport",
            "lex:CreateCustomVocabulary",
            "lex:UpdateCustomVocabulary",
            "lex:DeleteCustomVocabulary"
         ],
         "Effect": "Allow",
         "Resource": [
            "arn:aws:lex:Region:123456789012:bot/*",
            "arn:aws:lex:Region:123456789012:bot-alias/*"
         ]
      }
   ]
}
```

Allow a user to migrate a bot from Amazon Lex to Amazon Lex V2

The following IAM permission policy allows a user to start migrating a bot from Amazon Lex to Amazon Lex V2.

```
{
   "Version": "2012-10-17",
   "Statement": [
      {
         "Sid": "startMigration",
         "Effect": "Allow",
         "Action": "lex:StartMigration",
         "Resource": "arn:aws:lex:Region:*:bot:*"
      },
      {
         "Sid": "passRole",
         "Effect": "Allow",
         "Action": "iam:PassRole",
         "Resource": "arn:aws:iam::123456789012:role/v2 bot role"
      }
   ]
}
```
Identity-based policy examples

Allow users to view their own permissions

This example shows how you might create a policy that allows IAM users to view the inline and managed policies that are attached to their user identity. This policy includes permissions to complete this action on the console or programmatically using the AWS CLI or AWS API.
Resource-based policy examples for Amazon Lex V2

A resource-based policy is attached to a resource, such as a bot or a bot alias. With a resource-based policy you can specify who has access to the resource and the actions that they can perform on it. For example, you can add resource-based policies that enable a user to modify a specific bot, or to allow a user to use runtime operations on a specific bot alias.

When you use a resource-based policy you can allow other AWS services to access resources in your account. For example, you can allow Amazon Connect to access an Amazon Lex bot.

To learn how to create a bot or bot alias, see Building bots (p. 17).

Topics
- Use the console to specify a resource-based policy (p. 274)
- Use the API to specify a resource-based policy (p. 276)
- Allow an IAM role to update a bot and list bot aliases (p. 278)
- Allow a user to have a conversation with a bot (p. 278)
- Allow an AWS service to use a specific Amazon Lex V2 bot (p. 279)

Use the console to specify a resource-based policy

You can use the Amazon Lex console to manage the resource-based policies for your bots and bot aliases. You enter the JSON structure of a policy and the console associates it with the resource. If there is a policy already associated with a resource, you can use the console to view and modify the policy.

When you save a policy with the policy editor, the console checks the syntax of the policy. If the policy contains errors, such as a non-existent user or an action that is not supported by the resource, it returns an error and doesn't save the policy.

The following shows the resource-based policy editor for a bot in the console. The policy editor for a bot alias is similar.
Resource-based policy

You can use a resource-based policy to grant access permission to other AWS services, IAM users, and roles.

Resource ARN

```
arn:aws:lex:us-west-2:123456789012:bot/AKWB8PVLD2
```

Policy

```
{
  "Version": "2012-10-17",
  "Statement": [
    {
      "Sid": "botRunners",
      "Effect": "Allow",
      "Principal": {
        "AWS": "arn:aws:iam::123456789012:user/botRunner"
      },
      "Action": [
        "lex:RecognizeText",
        "lex:RecognizeUtterance",
        "lex:StartConversation"
      ],
      "Resource": [
        "arn:aws:lex:us-west-2:123456789012:bot/AKWB8PVLD2"
      ]
    }
  ]
}
```
To open the policy editor for a bot

1. Sign in to the AWS Management Console and open the Amazon Lex console at https://console.aws.amazon.com/lex/.
2. From the Bots list, choose the bot whose policy you want to edit.
3. In the Resource-based policy section, choose Edit.

To open the policy editor for a bot alias

1. Sign in to the AWS Management Console and open the Amazon Lex console at https://console.aws.amazon.com/lex/.
2. From the Bots list, choose the bot that contains the alias that you want to edit.
3. From the left menu, choose Aliases, then choose the alias to edit.
4. In the Resource-based policy section, choose Edit.

Use the API to specify a resource-based policy

You can use API operations to manage the resource-based policies for your bots and bot aliases. There are operations to create, update and delete policies.

CreateResourcePolicy (p. 358)

Adds a new resource policy with the specified policy statements to a bot or bot alias.

CreateResourcePolicyStatement (p. 361)

Adds a new resource policy statement to a bot or bot alias.

DeleteResourcePolicy (p. 408)

Removes a resource policy from a bot or bot alias.

DeleteResourcePolicyStatement (p. 410)

Removes a resource policy statement from a bot or bot alias.

DescribeResourcePolicy (p. 462)

Gets a resource policy and the policy revision.

UpdateResourcePolicy (p. 593)

Replaces the existing resource policy for a bot or bot alias with a new one.

Examples

Java

The following example shows how to use the resource-based policy operations to manage a resource-based policy.

```java
/*
 * Create a new policy for the specified bot alias
 * that allows a role to invoke lex:UpdateBotAlias on it.
 * The created policy will have revision id 1.
 */

```
Resource-based policy examples

```
import boto3

lexmodelsv2 = boto3.client('lex-models-v2', region_name='us-west-2')

# Example 1: Create a policy
create_policy_request = {
    'policy': {
        'Version': '2012-10-17',
        'Statement': [
            {
                'Sid': 'BotAliasEditor',
                'Effect': 'Allow',
                'Principal': {'AWS': 'arn:aws:iam::123456789012:role/BotAliasEditor'},
                'Action': ['lex:UpdateBotAlias'],
            }
        ]
    }
}
lexmodelsv2.put_resource_policy(**create_policy_request)

# Example 2: Update a policy
update_policy_request = {
    'policy': {
        'Version': '2012-10-17',
        'Statement': [
            {
                'Sid': 'BotAliasEditor',
                'Effect': 'Deny',
                'Principal': {'AWS': 'arn:aws:iam::123456789012:role/BotAliasEditor'},
                'Action': ['lex:UpdateBotAlias'],
            }
        ]
    }
}
lexmodelsv2.update_resource_policy(**update_policy_request)

# Example 3: Create a statement in a policy
create_statement_request = {
    'effect': 'Allow',
    'principal': {'aws': 'arn:aws:iam::123456789012:role/BotRunner'},
    'action': ['lex:RecognizeText'],
    'statementId': 'BotRunnerStatement',
    'expectedRevisionId': 2
}
lexmodelsv2.create_resource_policy_statement(**create_statement_request)

# Example 4: Delete a statement from a policy
delete_statement_request = {
    'statementId': 'BotRunnerStatement'
}
lexmodelsv2.delete_resource_policy_statement(**delete_statement_request)
```

277
Resource-based policy examples

```java
lexmodelsv2Client.deleteResourcePolicy(deleteStatementRequest);

/**
 * Describe the current policy for the specified bot alias
 * It always returns the current revision.
 */
DescribeResourcePolicyRequest describePolicyRequest =
    DescribeResourcePolicyRequest.builder()
    .resourceArn("arn:aws:lex:Region:123456789012:bot-alias/MYBOTALIAS/TSTALIASID")
    .build();

lexmodelsv2Client.describeResourcePolicy(describePolicyRequest);

/**
 * Delete the current policy for the specified bot alias
 * This request expects to delete revision 3 of the policy. Since the revision
 * id for this policy is already at 4, this request will fail.
 */
DeleteResourcePolicyRequest deletePolicyRequest =
    DeleteResourcePolicyRequest.builder()
    .resourceArn("arn:aws:lex:Region:123456789012:bot-alias/MYBOTALIAS/TSTALIASID")
    .expectedRevisionId(3);
    .build();

lexmodelsv2Client.deleteResourcePolicy(deletePolicyRequest);
```

Allow an IAM role to update a bot and list bot aliases

The following example grants permissions for a specific IAM role to call Amazon Lex V2 model building API operations to modify an existing bot. The user can list aliases for a bot and update the bot, but can’t delete the bot or bot aliases.

```json
{
   "Version": "2012-10-17",
   "Statement": [
      
      "Sid": "botBuilders",
      "Effect": "Allow",
      "Principal": {
         "AWS": "arn:aws:iam::123456789012:role/BotBuilder"
      },
      "Action": [
         "lex:ListBotAliases",
         "lex:UpdateBot"
      ],
      "Resource": ["arn:aws:lex:Region:123456789012:bot/MYBOT"
      ]
   }
}
```

Allow a user to have a conversation with a bot

The following example grants permission for a specific user to call Amazon Lex V2 runtime API operations on a single alias of a bot.
The user is specifically denied permission to update or delete the bot alias.

```
{  
  "Version": "2012-10-17",
  "Statement": [    
    {   
      "Sid": "botRunners",
      "Effect": "Allow",
      "Principal": {   
        "AWS": "arn:aws:iam::123456789012:user/botRunner"  
      },
      "Action": [   
        "lex:RecognizeText",
        "lex:RecognizeUtterance",
        "lex:StartConversation",
        "lex:DeleteSession",
        "lex:GetSession",
        "lex:PutSession"
      ],
      "Resource": [   
        "arn:aws:lex:Region:123456789012:bot-alias/MYBOT/MYBOTALIAS"
      ]
    },
    {   
      "Sid": "botRunners",
      "Effect": "Deny",
      "Principal": {   
        "AWS": "arn:aws:iam::123456789012:user/botRunner"  
      },
      "Action": [   
        "lex:UpdateBotAlias",
        "lex:DeleteBotAlias"
      ],
      "Resource": [   
        "arn:aws:lex:Region:123456789012:bot-alias/MYBOT/MYBOTALIAS"
      ]
    }
  ]
}
```

**Allow an AWS service to use a specific Amazon Lex V2 bot**

The following example grants permission for AWS Lambda and Amazon Connect to call Amazon Lex V2 runtime API operations.

The condition block is required for service principals, and must use the global context keys `AWS:SourceAccount` and `AWS:SourceArn`.

The `AWS:SourceAccount` is the account ID that is calling the Amazon Lex V2 bot.

The `AWS:SourceArn` is the resource ARN of the Amazon Connect service instance or Lambda function that the call to the Amazon Lex V2 bot alias originates from.

```
{  
  "Version": "2012-10-17",
  "Statement": [    
    {   
      "Sid": "connect-bot-alias",
      "Effect": "Allow",
      "Principal": {   
        "Service": [   
          "connect.amazonaws.com"
        ]  
      }
    }
  ]
}
```
AWS managed policies for Amazon Lex V2

To add permissions to users, groups, and roles, it is easier to use AWS managed policies than to write policies yourself. It takes time and expertise to create IAM customer managed policies that provide your team with only the permissions they need. To get started quickly, you can use our AWS managed policies. These policies cover common use cases and are available in your AWS account. For more information about AWS managed policies, see AWS managed policies in the IAM User Guide.

AWS services maintain and update AWS managed policies. You can't change the permissions in AWS managed policies. Services occasionally add additional permissions to an AWS managed policy to support new features. This type of update affects all identities (users, groups, and roles) where the policy is attached. Services are most likely to update an AWS managed policy when a new feature is launched.
or when new operations become available. Services do not remove permissions from an AWS managed policy, so policy updates won't break your existing permissions.

Additionally, AWS supports managed policies for job functions that span multiple services. For example, the ViewOnlyAccess AWS managed policy provides read-only access to many AWS services and resources. When a service launches a new feature, AWS adds read-only permissions for new operations and resources. For a list and descriptions of job function policies, see AWS managed policies for job functions in the IAM User Guide.

**AWS managed policy: AmazonLexReadOnly**

You can attach the AmazonLexReadOnly policy to your IAM identities.

This policy grants read-only permissions that allow users to view all actions in the Amazon Lex V2 and Amazon Lex model building service.

**Permissions details**

This policy includes the following permissions:

- lex – Read-only access to Amazon Lex V2 and Amazon Lex resources in the model building service.

```
{
  "Version": "2012-10-17",
  "Statement": [
    {
      "Effect": "Allow",
      "Action": [
        "lex:GetBot",
        "lex:GetBotAlias",
        "lex:GetBotAliases",
        "lex:GetBots",
        "lex:GetBotChannelAssociation",
        "lex:GetBotChannelAssociations",
        "lex:GetBotVersions",
        "lex:GetBuiltInIntent",
        "lex:GetBuiltInIntents",
        "lex:GetBuiltInSlotTypes",
        "lex:GetIntent",
        "lex:GetIntents",
        "lex:GetIntentVersions",
        "lex:GetSlotType",
        "lex:GetSlotTypes",
        "lex:GetSlotTypeVersions",
        "lex:GetUtterancesView",
        "lex:DescribeBot",
        "lex:DescribeBotAlias",
        "lex:DescribeBotChannel",
        "lex:DescribeBotLocale",
        "lex:DescribeBotRecommendation",
        "lex:DescribeBotVersion",
        "lex:DescribeCustomVocabulary",
        "lex:DescribeCustomVocabularyMetadata",
        "lex:DescribeExport",
        "lex:DescribeImport",
        "lex:DescribeIntent",
        "lex:DescribeResourcePolicy",
```
AWS managed policies

AWS managed policy: AmazonLexRunBotsOnly

You can attach the AmazonLexRunBotsOnly policy to your IAM identities.

This policy grants read-only permissions that allow access to run Amazon Lex V2 and Amazon Lex conversational bots.

Permissions details

This policy includes the following permissions:

- `lex` – Read-only access to all actions in the Amazon Lex V2 and Amazon Lex runtime.

```json
{
  "Version": "2012-10-17",
  "Statement": [
    {
      "Effect": "Allow",
      "Action": [
        "lex:PostContent",
        "lex:PostText",
        "lex:PutSession",
        "lex:GetSession",
        "lex:DeleteSession",
        "lex:RecognizeText",
        "lex:RecognizeUtterance",
        "lex:StartConversation"
      ],
      "Resource": "*"
    }
  ]
}
```

AWS managed policy: AmazonLexFullAccess

You can attach the AmazonLexFullAccess policy to your IAM identities.
This policy grants administrative permissions that allow the user permission to create, read, update, and delete Amazon Lex V2 and Amazon Lex resources; and to run Amazon Lex V2 and Amazon Lex conversational bots.

Permissions details

This policy includes the following permissions:

- **lex** – Allows principals read and write access to all actions in the Amazon Lex V2 and Amazon Lex model building and runtime services.
- **cloudwatch** – Allows principals to view Amazon CloudWatch metrics and alarms.
- **iam** – Allows principals to create and delete service-linked roles, pass roles, and attach and detach policies to a role. The permissions are restricted to "lex.amazonaws.com" for Amazon Lex operations and to "lexv2.amazonaws.com" for Amazon Lex V2 operations.
- **kendra** – Allows principals to list Amazon Kendra indexes.
- **kms** – Allows principals to describe AWS KMS keys and aliases.
- **lambda** – Allows principals to list AWS Lambda functions and manage permissions attached to any Lambda function.
- **polly** – Allows principals to describe Amazon Polly voices and synthesize speech.

```json
{
    "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",
                "Kendra:ListIndices",
                "iam:ListRoles",
                "s3:ListAllMyBuckets",
                "logs:DescribeLogGroups",
                "s3:GetBucketLocation"
            ],
            "Resource": ["
                "*"
            ]
        },
        {
            "Effect": "Allow",
            "Action": [
                "lambda:AddPermission",
                "lambda:RemovePermission"
            ],
            "Resource": "arn:aws:lambda:*:*:function:AmazonLex*",
            "Condition": {
                "StringEquals": {
                    "lambda:Principal": "lex.amazonaws.com"
                }
            }
        }
    ]
}
```
"Effect": "Allow",
"Action": [
  "iam:GetRole"
],
"Resource": [
  "arn:aws:iam::*:role/aws-service-role/lex.amazonaws.com/AWSServiceRoleForLexBots",
  "arn:aws:iam::*:role/aws-service-role/channels.lex.amazonaws.com/AWSServiceRoleForLexChannels",
  "arn:aws:iam::*:role/aws-service-role/lexv2.amazonaws.com/AWSServiceRoleForLexV2Bots",
  "arn:aws:iam::*:role/aws-service-role/channels.lexv2.amazonaws.com/AWSServiceRoleForLexV2Channels"
],
"Condition": {
  "StringEquals": {
    "iam:AWSServiceName": "lex.amazonaws.com"
  }
}
},
{
  "Effect": "Allow",
  "Action": [
    "iam:CreateServiceLinkedRole"
  ],
  "Resource": [
    "arn:aws:iam::*:role/aws-service-role/lex.amazonaws.com/AWSServiceRoleForLexBots"
  ],
  "Condition": {
    "StringEquals": {
      "iam:AWSServiceName": "channels.lex.amazonaws.com"
    }
  }
},
{
  "Effect": "Allow",
  "Action": [
    "iam:CreateServiceLinkedRole"
  ],
  "Resource": [
    "arn:aws:iam::*:role/aws-service-role/lexv2.amazonaws.com/AWSServiceRoleForLexV2Bots"
  ],
  "Condition": {
    "StringEquals": {
      "iam:AWSServiceName": "lexv2.amazonaws.com"
    }
  }
}
"arn:aws:iam::*:role/aws-service-role/channels.lexv2.amazonaws.com/AWSServiceRoleForLexV2Channels",
],
"Condition": {
"StringEquals": {
"iam:AWSServiceName": "channels.lexv2.amazonaws.com"
}
}
},
{
"Effect": "Allow",
"Action": [
"iam:DeleteServiceLinkedRole",
"iam:GetServiceLinkedRoleDeletionStatus"
],
"Resource": [
"arn:aws:iam::*:role/aws-service-role/lex.amazonaws.com/AWSServiceRoleForLexBots",
"arn:aws:iam::*:role/aws-service-role/channels.lex.amazonaws.com/AWSServiceRoleForLexChannels",
"arn:aws:iam::*:role/aws-service-role/lexv2.amazonaws.com/AWSServiceRoleForLexV2Bots",
"arn:aws:iam::*:role/aws-service-role/channels.lexv2.amazonaws.com/AWSServiceRoleForLexV2Channels"
]
},
{
"Effect": "Allow",
"Action": [
"iam:PassRole"
],
"Resource": [
"arn:aws:iam::*:role/aws-service-role/lex.amazonaws.com/AWSServiceRoleForLexBots"
],
"Condition": {
"StringEquals": {
"iam:PassedToService": [
"lex.amazonaws.com"
]
}
}
},
{
"Effect": "Allow",
"Action": [
"iam:PassRole"
],
"Resource": [
"arn:aws:iam::*:role/aws-service-role/lexv2.amazonaws.com/AWSServiceRoleForLexV2Bots"
],
"Condition": {
"StringEquals": {
"iam:PassedToService": [
"lexv2.amazonaws.com"
]
}
}
}
}
{
"Effect": "Allow",
"Action": [
"iam:PassRole"
],
"Resource": [
]
Amazon Lex V2 updates to AWS managed policies

View details about updates to AWS managed policies for Amazon Lex V2 since this service began tracking these changes. For automatic alerts about changes to this page, subscribe to the RSS feed on the Amazon Lex V2 Document history for Amazon Lex V2 (p. 302) page.

<table>
<thead>
<tr>
<th>Change</th>
<th>Description</th>
<th>Date</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>AmazonLexReadOnly (p. 281)</strong> – Update to an existing policy</td>
<td>Amazon Lex V2 added new permissions to allow read-only access to information about custom vocabularies.</td>
<td>November 16, 2021</td>
</tr>
<tr>
<td><strong>AmazonLexFullAccess (p. 282)</strong> – Update to an existing policy</td>
<td>Amazon Lex V2 added new permissions to allow read-only access to Amazon Lex V2 model building service operations.</td>
<td>August 18, 2021</td>
</tr>
<tr>
<td><strong>AmazonLexReadOnly (p. 281)</strong> – Update to an existing policy</td>
<td>Amazon Lex V2 added new permissions to allow read-only access to Amazon Lex V2 Automated Chatbot Designer operations.</td>
<td>December 1, 2021</td>
</tr>
<tr>
<td><strong>AmazonLexFullAccess (p. 282)</strong> – Update to an existing policy</td>
<td>Amazon Lex V2 added new permissions to allow read-only access to Amazon Lex V2 model building service operations.</td>
<td>August 18, 2021</td>
</tr>
<tr>
<td><strong>AmazonLexReadOnly (p. 281)</strong> – Update to an existing policy</td>
<td>Amazon Lex V2 added new permissions to allow read-only access to Amazon Lex V2 model building service operations.</td>
<td>August 18, 2021</td>
</tr>
<tr>
<td><strong>AmazonLexRunBotsOnly (p. 282)</strong> – Update to an existing policy</td>
<td>Amazon Lex V2 added new permissions to allow read-only access to Amazon Lex V2 runtime service operations.</td>
<td>August 18, 2021</td>
</tr>
<tr>
<td>Amazon Lex V2 started tracking changes</td>
<td>Amazon Lex V2 started tracking changes for its AWS managed policies.</td>
<td>August 18, 2021</td>
</tr>
</tbody>
</table>
Troubleshooting Amazon Lex V2 identity and access

Use the following information to help you diagnose and fix common issues that you might encounter when working with Amazon Lex V2 and IAM.

Topics

- I am not authorized to perform an action in Amazon Lex V2 (p. 287)
- I am not authorized to perform iam:PassRole (p. 287)
- I want to view my access keys (p. 287)
- I’m an administrator and want to allow others to access Amazon Lex V2 (p. 288)
- I want to allow people outside of my AWS account to access my Amazon Lex V2 resources (p. 288)

I am not authorized to perform an action in Amazon Lex V2

If the AWS Management Console tells you that you’re not authorized to perform an action, then you must contact your administrator for assistance. Your administrator is the person that provided you with your user name and password.

The following example error occurs when the mateojackson IAM user tries to use the console to view details about a fictional my-example-widget resource but does not have the fictional lex:GetWidget permissions.

```
User: arn:aws:iam::123456789012:user/mateojackson is not authorized to perform: lex:GetWidget on resource: my-example-widget
```

In this case, Mateo asks his administrator to update his policies to allow him to access the my-example-widget resource using the lex:GetWidget action.

I am not authorized to perform iam:PassRole

If you receive an error that you're not authorized to perform the iam:PassRole action, then you must contact your administrator for assistance. Your administrator is the person that provided you with your user name and password. Ask that person to update your policies to allow you to pass a role to Amazon Lex V2.

Some AWS services allow you to pass an existing role to that service, instead of creating a new service role or service-linked role. To do this, you must have permissions to pass the role to the service.

The following example error occurs when an IAM user named marymajor tries to use the console to perform an action in Amazon Lex V2. However, the action requires the service to have permissions granted by a service role. Mary does not have permissions to pass the role to the service.

```
User: arn:aws:iam::123456789012:user/marymajor is not authorized to perform: iam:PassRole
```

In this case, Mary asks her administrator to update her policies to allow her to perform the iam:PassRole action.

I want to view my access keys

After you create your IAM user access keys, you can view your access key ID at any time. However, you can't view your secret access key again. If you lose your secret key, you must create a new access key pair.

Access keys consist of two parts: an access key ID (for example, AKIAIOSFODNN7EXAMPLE) and a secret access key (for example, wJalrXUtnFEMI/K7MEND/bPxRficyEXAMPLEKEY). Like a user name and
password, you must use both the access key ID and secret access key together to authenticate your requests. Manage your access keys as securely as you do your user name and password.

Important

Do not provide your access keys to a third party, even to help find your canonical user ID. By doing this, you might give someone permanent access to your account.

When you create an access key pair, you are prompted to save the access key ID and secret access key in a secure location. The secret access key is available only at the time you create it. If you lose your secret access key, you must add new access keys to your IAM user. You can have a maximum of two access keys. If you already have two, you must delete one key pair before creating a new one. To view instructions, see Managing access keys in the IAM User Guide.

I'm an administrator and want to allow others to access Amazon Lex V2

To allow others to access Amazon Lex V2, you must create an IAM entity (user or role) for the person or application that needs access. They will use the credentials for that entity to access AWS. You must then attach a policy to the entity that grants them the correct permissions in Amazon Lex V2.

To get started right away, see Creating your first IAM delegated user and group in the IAM User Guide.

I want to allow people outside of my AWS account to access my Amazon Lex V2 resources

You can create a role that users in other accounts or people outside of your organization can use to access your resources. You can specify who is trusted to assume the role. For services that support resource-based policies or access control lists (ACLs), you can use those policies to grant people access to your resources.

To learn more, consult the following:

- To learn whether Amazon Lex V2 supports these features, see How Amazon Lex V2 works with IAM (p. 259).
- To learn how to provide access to your resources across AWS accounts that you own, see Providing access to an IAM user in another AWS account that you own in the IAM User Guide.
- To learn how to provide access to your resources to third-party AWS accounts, see Providing access to AWS accounts owned by third parties in the IAM User Guide.
- To learn how to provide access through identity federation, see Providing access to externally authenticated users (identity federation) in the IAM User Guide.
- To learn the difference between using roles and resource-based policies for cross-account access, see How IAM roles differ from resource-based policies in the IAM User Guide.

Using service-linked roles for Amazon Lex V2

Amazon Lex V2 uses AWS Identity and Access Management (IAM) service-linked roles. A service-linked role is a unique type of IAM role that is linked directly to Amazon Lex V2. Service-linked roles are predefined by Amazon Lex V2 and include all the permissions that the service requires to call other AWS services on your behalf.

A service-linked role makes setting up Amazon Lex V2 easier because you don’t have to manually add the necessary permissions. Amazon Lex V2 defines the permissions of its service-linked roles, and unless
defined otherwise, only Amazon Lex V2 can assume its roles. The defined permissions include the trust policy and the permissions policy, and that permissions policy cannot be attached to any other IAM entity.

You can delete a service-linked role only after first deleting their related resources. This protects your Amazon Lex V2 resources because you can't inadvertently remove permission to access the resources.

For information about other services that support service-linked roles, see AWS Services That Work with IAM and look for the services that have Yes in the Service-Linked Role column. Choose a Yes with a link to view the service-linked role documentation for that service.

**Service-linked role permissions for Amazon Lex V2**

Amazon Lex V2 uses these service-linked roles.

- **AWSServiceRoleForLexV2Bots_** – Gives permission to connect your bot to other required services.
- **AWSServiceRoleForLexV2Channels_** – Gives permission to list bots in an account and to call conversation APIs for a bot.

The AWSServiceRoleForLexV2Bots_ service-linked role trusts the following service to assume the role:

- lexv2.amazonaws.com

The AWSServiceRoleForLexV2Channels_ service linked role trusts the following service to assume the role:

- channels.lexv2.amazonaws.com

The AWSServiceRoleForLexV2Bots_ role allows Amazon Lex V2 to complete the following actions on the specified resources:

- Action: Use Amazon Polly to synthesize speech on all Amazon Lex V2 resources that the action supports.
- Action: If a bot is configured to use Amazon Comprehend sentiment analysis, detect sentiment on all Amazon Lex V2 resources that the action supports.
- Action: If a bot is configured to store audio logs in an S3 bucket, put object in a specified bucket.
- Action: If a bot is configured to store audio and text logs, create a log stream in and put logs into a specified log group.
- Action: If a bot is configured to use a AWS KMS key to encrypt data, generate a specific data key.
- Action: If a bot is configured to use the KendraSearchIntent intent, query access to a specified Amazon Kendra index.

If a bot is configured to use a channel to communicate with a messaging service, the AWSServiceRoleForLexV2Channels_ role permissions policy allows Amazon Lex V2 to complete the following actions on the specified resources:

- Action: List permissions on all bots in an account.
- Action: Recognize text, get session and put session permissions on a specified bot alias.

You must configure permissions to allow an IAM entity (such as a user, group, or role) to create, edit, or delete a service-linked role. For more information, see Service-Linked Role Permissions in the IAM User Guide.
Creating a service-linked role for Amazon Lex V2

You don't need to manually create a service-linked role. When you create or modify or bot, or create a channel integration with a messaging service in the AWS Management Console, the AWS CLI, or the AWS API, Amazon Lex V2 creates the service-linked role for you.

Amazon Lex V2 creates a new service-linked role in your account for each bot. When you create a channel integration to deploy a bot on a messaging platform, Amazon Lex V2 creates a new service-linked role in your account for each channel.

If you delete this service-linked role, and then need to create one again, you can use the same process to create a new role in your account. When you create or modify or bot, or create a channel integration with a messaging service, Amazon Lex V2 creates the service-linked role with a new random suffix.

Editing a service-linked role for Amazon Lex V2

Amazon Lex V2 does not allow you to edit the AWSServiceRoleForLexV2Bots_ or AWSServiceRoleForLexV2Channels_ service-linked roles. After you create a service-linked role, you cannot change the name of the role because various entities might reference the role. For AWSServiceRoleForLexV2Bots_, Amazon Lex V2 modifies the role when you add additional capabilities to a bot. For example, if you add Amazon Comprehend sentiment analysis to a bot, Amazon Lex V2 adds permission for the DetectSentiment action to AWSServiceRoleForLexV2Bots_. You can edit the description of either role using IAM. For more information, see Editing a Service-Linked Role in the IAM User Guide.

Deleting a service-linked role for Amazon Lex V2

If you no longer need to use a feature or service that requires a service-linked role, we recommend that you delete that role. That way you don’t have an unused entity that is not actively monitored or maintained. However, you must clean up the resources for your service-linked role before you can manually delete it.

**Note**

If the Amazon Lex V2 service is using the role when you try to delete the resources, then the deletion might fail. If that happens, wait for a few minutes and try the operation again.

To delete Amazon Lex V2 resources used by AWSServiceRoleForLexV2Bots_

1. Sign in to the AWS Management Console and open the Amazon Lex V2 console at [https://console.aws.amazon.com/lexv2/](https://console.aws.amazon.com/lexv2/)
2. Choose to bot to delete.
3. From the left menu, choose **Settings**.
4. From **IAM permissions**, record the **Runtime role** identifier.
5. From the left menu, choose **Bots**.
6. From the list of bots, choose the radio button next to the bot to delete.
7. From the **Action**, choose **Delete**.

To delete Amazon Lex V2 resources used by AWSServiceRoleForLexV2Channels_

1. Sign in to the AWS Management Console and open the Amazon Lex V2 console at [https://console.aws.amazon.com/lexv2/](https://console.aws.amazon.com/lexv2/)
2. Choose to bot to delete.
3. From the left menu, choose **Bot versions**, then choose **Channel integrations**.
4. Choose the channel to delete.
5. From **General configuration**, choose the role name. The IAM management console opens in a new tab with the role chosen.

6. In the Amazon Lex V2 console, choose **Delete**, then choose **Delete** again to delete the channel.

7. In the IAM management console, choose **DeleteRole** to remove the channel integration role.

To manually delete the service-linked role using IAM

Use the IAM console, the AWS CLI, or the AWS API to delete the AWSServiceRoleForLexV2Bots_ or AWSServiceRoleForLexV2Channels_ service-linked role. For more information, see Deleting a Service-Linked Role in the IAM User Guide.

**Supported regions for Amazon Lex V2 service-linked roles**

Amazon Lex V2 supports using service-linked roles in all of the regions where the service is available. For more information, see AWS Regions and Endpoints.

**Logging and monitoring in Amazon Lex V2**

Monitoring is an important part of maintaining the reliability, availability, and performance of Amazon Lex V2 and your other AWS solutions. AWS provides the following monitoring tools to watch Amazon Lex V2, report when something is wrong, and take automatic actions when appropriate:

- **Amazon CloudWatch** monitors your AWS resources and the applications you run on AWS in real time. You can collect and track metrics, create customized dashboards, and set alarms that notify you or take actions when a specified metric reaches a threshold that you specify. For example, you can have CloudWatch track CPU usage or other metrics of your Amazon EC2 instances and automatically launch new instances when needed. For more information, see the Amazon CloudWatch User Guide.

- **AWS CloudTrail** captures API calls and related events made by or on behalf of your AWS account and delivers the log files to an Amazon S3 bucket that you specify. You can identify which users and accounts called AWS, the source IP address from which the calls were made, and when the calls occurred. For more information, see the AWS CloudTrail User Guide.

**Compliance validation for Amazon Lex V2**

Third-party auditors assess the security and compliance of Amazon Lex V2 as part of multiple AWS compliance programs. Amazon Lex V2 is a HIPAA eligible service. It is PCI, SOC, and ISO compliant.

To learn whether Amazon Lex V2 or other AWS services are in scope of specific compliance programs, see AWS Services in Scope by Compliance Program. For general information, see AWS Compliance Programs.

You can download third-party audit reports using AWS Artifact. For more information, see Downloading Reports in AWS Artifact.

Your compliance responsibility when using AWS services is determined by the sensitivity of your data, your company’s compliance objectives, and applicable laws and regulations. AWS provides the following resources to help with compliance:

- **Security and Compliance Quick Start Guides** – These deployment guides discuss architectural considerations and provide steps for deploying baseline environments on AWS that are security and compliance focused.
Resilience in Amazon Lex V2

The AWS global infrastructure is built around AWS Regions and Availability Zones. AWS Regions provide multiple physically separated and isolated Availability Zones, which are connected with low-latency, high-throughput, and highly redundant networking. With Availability Zones, you can design and operate applications and databases that automatically fail over between zones without interruption. Availability Zones are more highly available, fault tolerant, and scalable than traditional single or multiple data center infrastructures.

For more information about AWS Regions and Availability Zones, see AWS Global Infrastructure.

In addition to the AWS global infrastructure, Amazon Lex V2 offers several features to help support your data resiliency and backup needs.

Infrastructure security in Amazon Lex V2

As a managed service, Amazon Lex V2 is protected by the AWS global network security procedures that are described in the Amazon Web Services: Overview of Security Processes whitepaper.

You use AWS published API calls to access Amazon Lex V2 through the network. Clients must support Transport Layer Security (TLS) 1.0 or later. We recommend TLS 1.2 or later. Clients must also support cipher suites with perfect forward secrecy (PFS) such as Ephemeral Diffie-Hellman (DHE) or Elliptic Curve Ephemeral Diffie-Hellman (ECDHE). Most modern systems such as Java 7 and later support these modes.

Additionally, requests must be signed by using an access key ID and a secret access key that is associated with an IAM principal. Or you can use the AWS Security Token Service (AWS STS) to generate temporary security credentials to sign requests.

Amazon Lex V2 and interface VPC endpoints (AWS PrivateLink)

You can establish a private connection between your VPC and Amazon Lex V2 by creating an interface VPC endpoint. Interface endpoints are powered by AWS PrivateLink, a technology that enables you to privately access Amazon Lex V2 APIs without an internet gateway, NAT device, VPN connection, or AWS
Direct Connect connection. Instances in your VPC don't need public IP addresses to communicate with Amazon Lex V2 APIs. Traffic between your VPC and Amazon Lex V2 does not leave the Amazon network.

Each interface endpoint is represented by one or more Elastic Network Interfaces in your subnets.

For more information, see Interface VPC endpoints (AWS PrivateLink) in the Amazon VPC User Guide.

Considerations for Amazon Lex V2 VPC endpoints

Before you set up an interface VPC endpoint for Amazon Lex V2, ensure that you review Interface endpoint properties and limitations in the Amazon VPC User Guide.

Amazon Lex V2 supports making calls to all of its API actions from your VPC.

Creating an interface VPC endpoint for Amazon Lex V2

You can create a VPC endpoint for the Amazon Lex V2 service using either the Amazon VPC console or the AWS Command Line Interface (AWS CLI). For more information, see Creating an interface endpoint in the Amazon VPC User Guide.

Create a VPC endpoint for Amazon Lex V2 using the following service name:

- com.amazonaws.region.models-v2-lex
- com.amazonaws.region.runtime-v2-lex

If you enable private DNS for the endpoint, you can make API requests to Amazon Lex V2 using its default DNS name for the Region, for example, runtime-v2-lex.us-east-1.amazonaws.com.

For more information, see Accessing a service through an interface endpoint in the Amazon VPC User Guide.

Creating a VPC endpoint policy for Amazon Lex V2

You can attach an endpoint policy to your VPC endpoint that controls access to Amazon Lex V2. The policy specifies the following information:

- The principal that can perform actions.
- The actions that can be performed.
- The resources on which actions can be performed.

For more information, see Controlling access to services with VPC endpoints in the Amazon VPC User Guide.

Example: VPC endpoint policy for Amazon Lex V2 actions

The following is an example of an endpoint policy for Amazon Lex V2. When attached to an endpoint, this policy grants access to the listed Amazon Lex V2 actions for all principals on all resources.

```json
{
  "Statement": [
    {
      "Principal": "*",
      "Effect": "Allow",
      "Action": ["lex:CreateSlotType", "lex:CreateSlotTypeVersion", "lex:CreateIntent", "lex:CreateIntentVersion", "lex:CreateLexComponent", "lex:CreateLexComponentVersion", "lex:CreateLexSessionState"],
      "Resource": "*",
      "Condition": {
        "StringLike": {
          "Lex:ComponentName": ["*"]
        }
      }
    }
  ]
}
```
"Action": [
    "lex:RecognizeText",
    "lex:RecognizeUtterance",
    "lex:StartConversation",
    "lex:DeleteSession",
    "lex:GetSession",
    "lex:DeleteSession"
],
"Resource": "*"
}
Amazon Lex V1 to V2 migration guide

The Amazon Lex V2 console and APIs make it easier to build and manage bots. Use this guide to learn about the improvements in the Amazon Lex V2 API as you migrate bots.

You migrate a bot using the Amazon Lex console or API. For more information see Migrating a bot in the Amazon Lex developer guide.

Amazon Lex V2 overview

Multiple languages can be added to a bot so you can manage them as a single resource. A simplified information architecture lets you efficiently manage your bot versions. Capabilities such as a ‘conversation flow’, partial saving of bot configuration and bulk upload of utterances give you more flexibility.

Multiple languages in a bot

You can add multiple languages with the Amazon Lex V2 API. You add, modify, and build each language independently. Resources such as slot types are scoped at the language level. You can quickly move between different languages to compare and refine the conversations. You can use one dashboard in the console to review utterances for all languages for faster analysis and iterations. A bot operator can manage permissions and logging operations for all languages with one bot configuration. You must provide a language as a runtime parameter to converse with a Amazon Lex V2 bot. For more information, see Languages and locales supported by Amazon Lex V2 (p. 4).

Simplified information architecture

The Amazon Lex V2 API follows a simplified information architecture (IA) with intent and slot types scoped to a language. You version at the bot level so that resources such as intents and slot types aren't versioned individually. By default, a bot is created with a Draft version that is mutable and used for testing changes. You can create numbered snapshots from the draft version. You choose the languages to include in a version. All resources within the bot (languages, intents, slot types) are archived as part of creating a bot version. For more information, see Creating versions (p. 23).

Improved builder productivity

You have additional builder productivity tools and capabilities that give you more flexibility and control of your bot design process.

Save partial configuration

The Amazon Lex V2 API enables you to save partial changes during development. For example, you can save a slot that references a deleted slot type. This flexibility enables you to save your work and return to it later. You can resolve these changes before building the bot. In Amazon Lex V2 the partial save can be applied to slots, versions, and aliases.
Renaming resources

With Amazon Lex V2 you can rename a resource after it's created. Use a resource name to associate user-friendly metadata with each resource. The Amazon Lex V2 API assigns every resource a unique 10-character resource ID. All resources have a resource name. You can rename the following resources:

- Bot
- Intent
- Slot type
- Slot
- Alias

You can use resource IDs to read and modify your resources. If you are using the AWS Command Line Interface or the Amazon Lex V2 API to work with Amazon Lex V2, resource IDs are required for certain commands.

Simplified management of Lambda functions

In the Amazon Lex V2 API you define one Lambda function per language instead of a function for each intent. The Lambda function is configured in the alias for the language and is used for both the dialog and fulfillment code hook. You can still choose to enable or disable the dialog and fulfillment code hooks independently for each intent. For more information, see Using an AWS Lambda function (p. 145).

Granular settings

The Amazon Lex V2 API moves the voice and intent classification confidence score threshold from the bot to the language scope. The sentiment analysis flag moves from bot scope to alias scope. Session timeout and privacy settings at the bot scope, and conversation logs at the alias scope, remain unchanged.

Default fallback intent

The Amazon Lex V2 API adds a default fallback intent when you create a language. Use it to configure error handling for your bot instead of specific error-handling prompts.

Optimized session variable update

With the Amazon Lex V2 API you can update session state directly with the RecognizeText (p. 628) and RecognizeUtterance (p. 635) operations without any dependency on session APIs.
Creating Amazon Lex V2 resources with AWS CloudFormation

Amazon Lex V2 and AWS CloudFormation templates

Amazon Lex V2 is integrated with AWS CloudFormation, a service that helps you to model and set up your AWS resources so that you can spend less time creating and managing your resources and infrastructure. You create a template that describes all the AWS resources that you want (such as Amazon Lex V2 chatbots), and AWS CloudFormation provisions and configures those resources for you.

When you use AWS CloudFormation, you can reuse your template to set up your Amazon Lex V2 resources consistently and repeatedly. Describe your resources once, and then provision the same resources over and over in multiple AWS accounts and Regions.

Amazon Lex V2 and AWS CloudFormation templates

To provision and configure resources for Amazon Lex V2 and related services, you must understand AWS CloudFormation templates. Templates are formatted text files in JSON or YAML. These templates describe the resources that you want to provision in your AWS CloudFormation stacks. If you're unfamiliar with JSON or YAML, you can use AWS CloudFormation Designer to help you get started with AWS CloudFormation templates. For more information, see What is AWS CloudFormation Designer? in the AWS CloudFormation User Guide.

Amazon Lex V2 supports creating the following resources in AWS CloudFormation:

- AWS::Lex::Bot
- AWS::Lex::BotAlias
- AWS::Lex::BotVersion
- AWS::Lex::ResourcePolicy

For more information, including examples of JSON and YAML templates for these resources, see the Amazon Lex V2 resource type reference in the AWS CloudFormation User Guide.

Learn more about AWS CloudFormation

To learn more about AWS CloudFormation, see the following resources:

- AWS CloudFormation
- AWS CloudFormation user guide
- AWS CloudFormation API reference
- AWS CloudFormation Command line interface user guide
Guidelines and quotas

Topics
- Regions (p. 298)
- General guidelines (p. 298)
- Quotas (p. 299)

Regions

For a list of AWS Regions where Amazon Lex V2 is available, see AWS regions and endpoints in the AWS general reference.

General guidelines

This topic describes general guidelines when using Amazon Lex V2.

- **Signing requests** – All Amazon Lex V2 model-building and runtime requests in the API reference (p. 306) 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.

- **Note** the following about how Amazon Lex V2 captures slot values from user utterances:

  Amazon Lex V2 uses the enumeration values that you provide in a slot type definition to train its machine learning models. Suppose that you define an intent called GetPredictionIntent with the following sample utterance:

  "Tell me the prediction for {sign}"

  where {sign} is a slot with the custom type ZodiacSign. It has 12 enumeration values, Aries through Pisces. From the user utterance "Tell me the prediction for ..." Amazon Lex V2 understands that the following is a zodiac sign.

  When the valueSelectionStrategy field is set to OriginalValue using the CreateSlotType (p. 378) operation, or if Expand values is selected in the console, if the user says "Tell me the prediction for earth", Amazon Lex V2 infers that "earth" is a ZodiacSign value and passes it to your client application or Lambda function. You must check that slot values are valid values before using them in your fulfillment activity.

  If you set the valueSelectionStrategy field to TopResolution using the CreateSlotType operation or if Restrict to slot values and synonyms is selected in the console, the values that are returned are limited to the values that you defined for the slot type. For example, if the user says "Tell me the prediction for earth", the value would not be recognized because it is not one of the values defined for the slot type. When you define synonyms for slot values, they are recognized the same as a slot value, however, the slot value is returned instead of the synonym.

  When Amazon Lex V2 calls a Lambda function or returns the result of a speech interaction with your client, the case of the slot values is not guaranteed. In text interactions, the case of the slot values matches the text entered or the slot value, depending on the value of the valueResolutionStrategy field.

- **When defining slot values that contain acronyms**, use the following patterns:
• Capital letters separated by periods (D.V.D.)
• Capital letters separated by spaces (D V D)
• The AMAZON.Date (p. 38) and AMAZON.Time (p. 41) built-on slot types capture both absolute and relative dates and times. Relative dates and times are resolved in the region where Amazon Lex V2 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 V2 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 the ability of Amazon Lex V2 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 V2 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 V2 can't pick an intent with a high degree of confidence.

When you have two intents with the same utterance, use input contexts to help Amazon Lex distinguish between the two intents at runtime. For more information, see Setting intent context.

• When you use the TSTALIASID alias, keep in mind the following:
  • The TSTALIASID alias of your bot points to the Draft version and should only be used for manual testing. Amazon Lex limits the number of runtime requests that you can make to the TSTALIASID alias of the bot.
  • When you update the Draft version of the bot, Amazon Lex terminates any in-progress conversations for any client application using the TSTALIASID alias of the bot. Generally, you should not use the TSTALIASID alias of a bot in production because the Draft version can be updated. You should publish a version and an alias and use them instead.
  • When you update an alias, Amazon Lex takes a few minutes to pick up the changes. When you modify the Draft version of the bot, the change is picked up by the TSTALIASID alias immediately.
  • The runtime API operations RecognizeText (p. 628) and RecognizeUtterance (p. 635) take a session ID as a required parameter. Developers can set this to any value that meets the constraints described in the API. We recommend that 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 a conversation with a bot.

## Quotas

Service quotas, also referred to as limits, are the maximum number of service resources for your AWS account. For more information, see AWS service quotas in the AWS general reference.

Service quotas can be adjusted or increased. Contact AWS customer support to increase a quota. It can take a few days to increase a service quota. If you're increasing your quota as part of a larger project, be sure to add this time to your plan.

### Build-time quotas

The following maximum quotas are enforced when you are creating a bot.

<table>
<thead>
<tr>
<th>Description</th>
<th>Default</th>
</tr>
</thead>
<tbody>
<tr>
<td>Bots per AWS account</td>
<td>100</td>
</tr>
</tbody>
</table>
## Runtime quotas

The following maximum quotas are enforced at runtime.

<table>
<thead>
<tr>
<th>Description</th>
<th>Default</th>
</tr>
</thead>
<tbody>
<tr>
<td>Input text size for RecognizeText (p. 628) and RecognizeUtterance (p. 635)</td>
<td>1024 Unicode characters</td>
</tr>
<tr>
<td>Speech input length for RecognizeUtterance operation</td>
<td>15 seconds</td>
</tr>
<tr>
<td>Description</td>
<td>Default</td>
</tr>
<tr>
<td>----------------------------------------------------------------------------</td>
<td>---------</td>
</tr>
<tr>
<td>Size of RecognizeUtterance headers</td>
<td>16 KB</td>
</tr>
<tr>
<td>Size of combined request and session headers for RecognizeUtterance</td>
<td>12 KB</td>
</tr>
<tr>
<td>Maximum number of concurrent text-mode conversations for RecognizeText, RecognizeUtterance, or StartConversation for the TestBotAlias</td>
<td>2</td>
</tr>
<tr>
<td>Maximum number of concurrent text-mode conversations for RecognizeText, RecognizeUtterance, or StartConversation for other aliases</td>
<td>50</td>
</tr>
<tr>
<td>Maximum number of concurrent voice-mode conversations for RecognizeUtterance for the TestBotAlias</td>
<td>2</td>
</tr>
<tr>
<td>Maximum number of concurrent voice-mode conversations for RecognizeUtterance for other aliases</td>
<td>125</td>
</tr>
<tr>
<td>Maximum number of concurrent voice-mode conversations for StartConversation for the TestBotAlias</td>
<td>2</td>
</tr>
<tr>
<td>Maximum number of concurrent voice-mode conversations for StartConversation for other aliases</td>
<td>200</td>
</tr>
<tr>
<td>Maximum number of concurrent session management operations (PutSession, GetSession, or DeleteSession) when using the TestBotAlias</td>
<td>2</td>
</tr>
<tr>
<td>Maximum number of concurrent session management operations (PutSession, GetSession, or DeleteSession) when using other aliases</td>
<td>50</td>
</tr>
<tr>
<td>Maximum input size to a Lambda function</td>
<td>12 KB</td>
</tr>
<tr>
<td>Maximum output size of a Lambda function</td>
<td>25 KB</td>
</tr>
<tr>
<td>Maximum size of session attributes in Lambda function output</td>
<td>12 KB</td>
</tr>
</tbody>
</table>
## Document history for Amazon Lex V2

- **Latest documentation update:** March 18, 2022

The following table describes important changes in each release of Amazon Lex V2. For notification about updates to this documentation, you can subscribe to an RSS feed.

<table>
<thead>
<tr>
<th>update-history-change</th>
<th>update-history-description</th>
<th>update-history-date</th>
</tr>
</thead>
<tbody>
<tr>
<td>New feature</td>
<td>Added example industry grammars for the built-in grammar slot type. For more information, see <a href="#">Industry grammars</a>.</td>
<td>March 22, 2022</td>
</tr>
<tr>
<td>New feature</td>
<td>Added documentation about integrating Amazon Lex V2 with the Amazon Chime SDK. For more information, see <a href="#">Amazon Chime SDK</a>.</td>
<td>March 18, 2022</td>
</tr>
<tr>
<td>New feature</td>
<td>Amazon Lex V2 now provides confidence scores for voice transcriptions. Use the score to help determine the correct response from the user. For more information, see <a href="#">Using voice transcription confidence scores</a>.</td>
<td>January 27, 2022</td>
</tr>
<tr>
<td>New feature</td>
<td>You can now add contextual and dynamic hints to slots to improve the accuracy of your bot. For more information, see <a href="#">Using hints to improve accuracy</a>.</td>
<td>January 13, 2022</td>
</tr>
<tr>
<td>New feature</td>
<td>Amazon Lex V2 adds support for custom vocabularies to improve speech recognition for audio input. For more information, see <a href="#">Creating a custom vocabulary to improve speech recognition</a>.</td>
<td>January 12, 2022</td>
</tr>
<tr>
<td>New feature</td>
<td>Amazon Lex V2 now supports AWS PrivateLink. For more information, see <a href="#">VPC endpoints (AWS PrivateLink)</a>.</td>
<td>January 7, 2022</td>
</tr>
<tr>
<td>New feature</td>
<td>Amazon Lex V2 now supports the Catalan (Spain) locale. For more information, see <a href="#">Languages and locales supported by Amazon Lex V2</a>.</td>
<td>January 3, 2022</td>
</tr>
<tr>
<td>New feature</td>
<td>Date</td>
<td></td>
</tr>
<tr>
<td>---------------------------------------------------------------------------</td>
<td>-----------------------------</td>
<td></td>
</tr>
<tr>
<td>You can now create slot types using your own custom grammar. For more information, see Using a custom grammar slot type.</td>
<td>December 20, 2021</td>
<td></td>
</tr>
<tr>
<td>AWS CloudFormation now supports Amazon Lex V2. For more information, see AWS CloudFormation resources.</td>
<td>December 20, 2021</td>
<td></td>
</tr>
<tr>
<td>Amazon Lex V2 now supports the Portuguese (Brazil), Portuguese (Portugal), and Mandarin (PRC) locales. For more information, see Languages and locales supported by Amazon Lex V2.</td>
<td>December 16, 2021</td>
<td></td>
</tr>
<tr>
<td>Amazon Lex V2 now provides a preview of the Automated Chatbot Designer to help you get started creating a chatbot from contact center transcripts. For more information, see Using the Automated Chatbot Designer (Preview).</td>
<td>December 1, 2021</td>
<td></td>
</tr>
<tr>
<td>You can now use spell-by-letter and spell-by-word styles for entering slot values that Amazon Lex V2 has difficulty understanding. For more information, see Using spelling styles to capture slot values.</td>
<td>November 19, 2021</td>
<td></td>
</tr>
<tr>
<td>You can now use Amazon Polly neural text to speech (NTTS) voices for audio conversations with your users. For more information, see Voices in Amazon Polly.</td>
<td>November 19, 2021</td>
<td></td>
</tr>
<tr>
<td>Amazon Lex V2 now supports the English (South Africa) locale. For more information, see Languages and locales supported by Amazon Lex V2.</td>
<td>November 9, 2021</td>
<td></td>
</tr>
<tr>
<td>Amazon Lex V2 now supports the German (Austria) locale. For more information, see Languages and locales supported by Amazon Lex V2.</td>
<td>November 5, 2021</td>
<td></td>
</tr>
<tr>
<td>Feature Type</td>
<td>Description</td>
<td>Date</td>
</tr>
<tr>
<td>-------------------</td>
<td>------------------------------------------------------------------------------------------------------------------------------</td>
<td>-------------------</td>
</tr>
<tr>
<td>New feature</td>
<td>You can now provide users with update messages that play at the start of a fulfillment function and periodically while the function runs. You can also create messages that inform the user of status of fulfillment when the function is complete. For more information, see Configuring fulfillment progress updates.</td>
<td>October 7, 2021</td>
</tr>
<tr>
<td>Region expansion</td>
<td>Amazon Lex V2 is now available in Africa (Cape Town) (af-south-1) and Asia Pacific (Seoul) (ap-northeast-2).</td>
<td>September 22, 2021</td>
</tr>
<tr>
<td>New feature</td>
<td>You can now view statistics for the utterances that your users send to your bot. For more information, see Viewing utterance statistics.</td>
<td>September 22, 2021</td>
</tr>
<tr>
<td>New feature</td>
<td>Amazon Lex V2 now supports the Korean (Korea) locale. For more information, see Languages and locales supported by Amazon Lex V2.</td>
<td>September 9, 2021</td>
</tr>
<tr>
<td>New feature</td>
<td>Amazon Lex V2 now provides a built-in slot type for UK postal codes. For more information, see AMAZON.UKPostalCode.</td>
<td>July 27, 2021</td>
</tr>
<tr>
<td>New feature</td>
<td>Amazon Lex V2 now supports the English (Indian) locale. For more information, see Languages and locales supported by Amazon Lex V2.</td>
<td>July 15, 2021</td>
</tr>
<tr>
<td>New feature</td>
<td>Amazon Lex V2 now provides a tool to migrate a bot from Amazon Lex V1 to the Amazon Lex V2 API. For more information, see Migrating a bot in the Amazon Lex Developer Guide.</td>
<td>July 13, 2021</td>
</tr>
<tr>
<td>New feature</td>
<td>Amazon Lex V2 now enables you to accept multiple values for a single slot in the English (US) language. For more information, see Using multiple values in a slot.</td>
<td>June 15, 2021</td>
</tr>
<tr>
<td>New feature</td>
<td>You can now build larger bots for English languages. For more information, see Quotas.</td>
<td>June 11, 2021</td>
</tr>
<tr>
<td>New feature</td>
<td>Use Amazon Lex V2 resource-based policies to manage access to your bots and bot aliases. For more information, see Resource-based policies within Amazon Lex V2.</td>
<td>May 20, 2021</td>
</tr>
<tr>
<td>---</td>
<td>---</td>
<td>---</td>
</tr>
<tr>
<td>New feature</td>
<td>Amazon Lex V2 now enables you to import and export bots and bot locales. You can use this feature to copy bots and bot locales between accounts and AWS Regions. For more information, see Importing and exporting bots.</td>
<td>May 18, 2021</td>
</tr>
<tr>
<td>Region expansion</td>
<td>Amazon Lex V2 is now available in Canada (Central) (ca-central-1).</td>
<td>May 17, 2021</td>
</tr>
<tr>
<td>New feature</td>
<td>Amazon Lex V2 now supports the Japanese (Japan) locale. For more information, see Languages and locales supported by Amazon Lex V2.</td>
<td>April 1, 2021</td>
</tr>
<tr>
<td>New feature</td>
<td>Amazon Lex V2 now supports three new built-in slot types: AMAZON.City, AMAZON.Country, and AMAZON.State.</td>
<td>March 12, 2021</td>
</tr>
<tr>
<td>New guide (p. 302)</td>
<td>This is the first release of the Amazon Lex V2 user guide.</td>
<td>January 21, 2021</td>
</tr>
</tbody>
</table>
API reference

This section contains the API reference documentation. See the left menu for a list of actions and data types.

Note
If you use a custom HTTP client to call Amazon Lex Model Building V2 operations, you must set the "Content-Type" HTTP header to "application/x-amz-json-1.1". Otherwise, you receive an HTTP 404 - UnknownOperationException in the response. Amazon Lex Model Building V2 operations return the responses with the "application/x-amz-json-1.1" content type.

Actions

The following actions are supported by Amazon Lex Model Building V2:

- `BuildBotLocale` (p. 310)
- `CreateBot` (p. 313)
- `CreateBotAlias` (p. 318)
- `CreateBotLocale` (p. 324)
- `CreateBotVersion` (p. 329)
- `CreateExport` (p. 333)
- `CreateIntent` (p. 337)
- `CreateResourcePolicy` (p. 358)
- `CreateResourcePolicyStatement` (p. 361)
- `CreateSlot` (p. 365)
- `CreateSlotType` (p. 378)
- `CreateUploadUrl` (p. 384)
- `DeleteBot` (p. 386)
- `DeleteBotAlias` (p. 389)
- `DeleteBotLocale` (p. 392)
- `DeleteBotVersion` (p. 395)
- `DeleteCustomVocabulary` (p. 398)
- `DeleteExport` (p. 401)
- `DeleteImport` (p. 403)
- `DeleteIntent` (p. 405)
- `DeleteResourcePolicy` (p. 408)
- `DeleteResourcePolicyStatement` (p. 410)
- `DeleteSlot` (p. 413)
- `DeleteSlotType` (p. 416)
- `DeleteUtterances` (p. 419)
- `DescribeBot` (p. 421)
- `DescribeBotAlias` (p. 424)
- `DescribeBotLocale` (p. 428)
- `DescribeBotRecommendation` (p. 432)
The following actions are supported by Amazon Lex Runtime V2:

- DescribeBotVersion (p. 436)
- DescribeCustomVocabularyMetadata (p. 440)
- DescribeExport (p. 443)
- DescribeImport (p. 446)
- DescribeIntent (p. 450)
- DescribeResourcePolicy (p. 462)
- DescribeSlot (p. 464)
- DescribeSlotType (p. 472)
- ListAggregatedUtterances (p. 476)
- ListBotAliases (p. 482)
- ListBotLocales (p. 485)
- ListBotRecommendations (p. 489)
- ListBots (p. 493)
- ListBotVersions (p. 496)
- ListBuiltInIntents (p. 499)
- ListBuiltInSlotTypes (p. 502)
- ListExports (p. 505)
- ListImports (p. 509)
- ListIntents (p. 513)
- ListRecommendedIntents (p. 517)
- ListSlots (p. 521)
- ListSlotTypes (p. 526)
- ListTagsForResource (p. 530)
- SearchAssociatedTranscripts (p. 532)
- StartBotRecommendation (p. 536)
- StartImport (p. 541)
- TagResource (p. 545)
- UntagResource (p. 547)
- UpdateBot (p. 549)
- UpdateBotAlias (p. 553)
- UpdateBotLocale (p. 559)
- UpdateBotRecommendation (p. 564)
- UpdateExport (p. 569)
- UpdateIntent (p. 572)
- UpdateResourcePolicy (p. 593)
- UpdateSlot (p. 596)
- UpdateSlotType (p. 609)
Amazon Lex Model Building V2

The following actions are supported by Amazon Lex Model Building V2:

- BuildBotLocale (p. 310)
- CreateBot (p. 313)
- CreateBotAlias (p. 318)
- CreateBotLocale (p. 324)
- CreateBotVersion (p. 329)
- CreateExport (p. 333)
- CreateIntent (p. 337)
- CreateResourcePolicy (p. 358)
- CreateResourcePolicyStatement (p. 361)
- CreateSlot (p. 365)
- CreateSlotType (p. 378)
- CreateUploadUrl (p. 384)
- DeleteBot (p. 386)
- DeleteBotAlias (p. 389)
- DeleteBotLocale (p. 392)
- DeleteBotVersion (p. 395)
- DeleteCustomVocabulary (p. 398)
- DeleteExport (p. 401)
- DeleteImport (p. 403)
- DeleteIntent (p. 405)
- DeleteResourcePolicy (p. 408)
- DeleteResourcePolicyStatement (p. 410)
- DeleteSlot (p. 413)
- DeleteSlotType (p. 416)
- DeleteUtterances (p. 419)
- DescribeBot (p. 421)
- DescribeBotAlias (p. 424)
- DescribeBotLocale (p. 428)
- DescribeBotRecommendation (p. 432)
- DescribeBotVersion (p. 436)
- DescribeCustomVocabularyMetadata (p. 440)
- DescribeExport (p. 443)
- DescribeImport (p. 446)
- DescribeIntent (p. 450)
- DescribeResourcePolicy (p. 462)
- DescribeSlot (p. 464)
- DescribeSlotType (p. 472)
- ListAggregatedUtterances (p. 476)
- ListBotAliases (p. 482)
- ListBotLocales (p. 485)
- ListBotRecommendations (p. 489)
- ListBots (p. 493)
• ListBotVersions (p. 496)
• ListBuiltInIntents (p. 499)
• ListBuiltInSlotTypes (p. 502)
• ListExports (p. 505)
• ListImports (p. 509)
• ListIntents (p. 513)
• ListRecommendedIntents (p. 517)
• ListSlots (p. 521)
• ListSlotTypes (p. 526)
• ListTagsForResource (p. 530)
• SearchAssociatedTranscripts (p. 532)
• StartBotRecommendation (p. 536)
• StartImport (p. 541)
• TagResource (p. 545)
• UntagResource (p. 547)
• UpdateBot (p. 549)
• UpdateBotAlias (p. 553)
• UpdateBotLocale (p. 559)
• UpdateBotRecommendation (p. 564)
• UpdateExport (p. 569)
• UpdateIntent (p. 572)
• UpdateResourcePolicy (p. 593)
• UpdateSlot (p. 596)
• UpdateSlotType (p. 609)
**BuildBotLocale**

Service: Amazon Lex Model Building V2

Builds a bot, its intents, and its slot types into a specific locale. A bot can be built into multiple locales. At runtime the locale is used to choose a specific build of the bot.

**Request Syntax**

```
POST /bots/botId/botversions/botVersion/botlocales/localeId/ HTTP/1.1
```

**URI Request Parameters**

The request uses the following URI parameters.

**botId (p. 310)**

The identifier of the bot to build. The identifier is returned in the response from the CreateBot operation.

Length Constraints: Fixed length of 10.

Pattern: ^[0-9a-zA-Z]+$

Required: Yes

**botVersion (p. 310)**

The version of the bot to build. This can only be the draft version of the bot.

Length Constraints: Fixed length of 5.

Pattern: ^DRAFT$

Required: Yes

**localeId (p. 310)**

The identifier of the language and locale that the bot will be used in. The string must match one of the supported locales. All of the intents, slot types, and slots used in the bot must have the same locale. For more information, see Supported languages.

Required: Yes

**Request Body**

The request does not have a request body.

**Response Syntax**

```
HTTP/1.1 202
Content-type: application/json

{
    "botId": "string",
    "botLocaleStatus": "string",
    "botVersion": "string",
    "lastBuildSubmittedDateTime": number,
    "localeId": "string"
}
```
Response Elements

If the action is successful, the service sends back an HTTP 202 response.

The following data is returned in JSON format by the service.

**botId (p. 310)**

The identifier of the specified bot.

Type: String

Length Constraints: Fixed length of 10.

Pattern: ^[0-9a-zA-Z]+$

**botLocaleStatus (p. 310)**

The bot's build status. When the status is ReadyExpressTesting you can test the bot using the utterances defined for the intents and slot types. When the status is Built, the bot is ready for use and can be tested using any utterance.

Type: String

Valid Values: Creating | Building | Built | ReadyExpressTesting | Failed | Deleting | NotBuilt | Importing | Processing

**botVersion (p. 310)**

The version of the bot that was built. This is only the draft version of the bot.

Type: String

Length Constraints: Fixed length of 5.

Pattern: ^DRAFT$

**lastBuildSubmittedDateTime (p. 310)**

A timestamp indicating the date and time that the bot was last built for this locale.

Type: Timestamp

**localeId (p. 310)**

The language and locale specified of where the bot can be used.

Type: String

Errors

For information about the errors that are common to all actions, see Common Errors (p. 821).

**ConflictException**

The action that you tried to perform couldn't be completed because the resource is in a conflicting state. For example, deleting a bot that is in the CREATING state. Try your request again.

HTTP Status Code: 409

**InternalServerException**

The service encountered an unexpected condition. Try your request again.
HTTP Status Code: 500
**PreconditionFailedException**

Your request couldn't be completed because one or more request fields aren't valid. Check the fields in your request and try again.

HTTP Status Code: 412
**ServiceQuotaExceededException**

You have reached a quota for your bot.

HTTP Status Code: 402
**ThrottlingException**

Your request rate is too high. Reduce the frequency of requests.

HTTP Status Code: 429
**ValidationException**

One of the input parameters in your request isn't valid. Check the parameters and try your request again.

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 V2
- AWS SDK for JavaScript
- AWS SDK for PHP V3
- AWS SDK for Python
- AWS SDK for Ruby V3
CreateBot
Service: Amazon Lex Model Building V2

Creates an Amazon Lex conversational bot.

Request Syntax

```plaintext
PUT /bots/ HTTP/1.1
Content-type: application/json

{
    "botName": "string",
    "botTags": {
        "string": "string"
    },
    "dataPrivacy": {
        "childDirected": boolean
    },
    "description": "string",
    "idleSessionTTLInSeconds": number,
    "roleArn": "string",
    "testBotAliasTags": {
        "string": "string"
    }
}
```

URI Request Parameters

The request does not use any URI parameters.

Request Body

The request accepts the following data in JSON format.

**botName (p. 313)**

The name of the bot. The bot name must be unique in the account that creates the bot.

Type: String

Length Constraints: Minimum length of 1. Maximum length of 100.

Pattern: ^([0-9a-zA-Z][-_]?)+$

Required: Yes

**botTags (p. 313)**

A list of tags to add to the bot. You can only add tags when you create a bot. You can't use the UpdateBot operation to update tags. To update tags, use the TagResource operation.

Type: String to string map

Map Entries: Minimum number of 0 items. Maximum number of 200 items.

Key Length Constraints: Minimum length of 1. Maximum length of 128.

Value Length Constraints: Minimum length of 0. Maximum length of 256.

Required: No
**dataPrivacy (p. 313)**

Provides information on additional privacy protections Amazon Lex should use with the bot’s data.

Type: DataPrivacy (p. 701) object

Required: Yes

**description (p. 313)**

A description of the bot. It appears in lists to help you identify a particular bot.

Type: String

Length Constraints: Minimum length of 0. Maximum length of 200.

Required: No

**idleSessionTTLInSeconds (p. 313)**

The time, in seconds, that Amazon Lex should keep information about a user’s conversation with the bot.

A user interaction 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.

You can specify between 60 (1 minute) and 86,400 (24 hours) seconds.

Type: Integer

Valid Range: Minimum value of 60. Maximum value of 86400.

Required: Yes

**roleArn (p. 313)**

The Amazon Resource Name (ARN) of an IAM role that has permission to access the bot.

Type: String

Length Constraints: Minimum length of 32. Maximum length of 2048.

Pattern: `^arn:aws:iam::[0-9]{12}:role/.*$`

Required: Yes

**testBotAliasTags (p. 313)**

A list of tags to add to the test alias for a bot. You can only add tags when you create a bot. You can’t use the UpdateAlias operation to update tags. To update tags on the test alias, use the TagResource operation.

Type: String to string map

Map Entries: Minimum number of 0 items. Maximum number of 200 items.

Key Length Constraints: Minimum length of 1. Maximum length of 128.

Value Length Constraints: Minimum length of 0. Maximum length of 256.

Required: No

**Response Syntax**

```plaintext
HTTP/1.1 202
```
Response Elements

If the action is successful, the service sends back an HTTP 202 response.

The following data is returned in JSON format by the service.

botId (p. 314)

A unique identifier for a particular bot. You use this to identify the bot when you call other Amazon Lex API operations.

Type: String

Length Constraints: Fixed length of 10.

Pattern: ^[0-9a-zA-Z]+$"botName (p. 314)

The name specified for the bot.

Type: String

Length Constraints: Minimum length of 1. Maximum length of 100.

Pattern: ^([0-9a-zA-Z][_\-]?)+$botStatus (p. 314)

Shows the current status of the bot. The bot is first in the Creating status. Once the bot is read for use, it changes to the Available status. After the bot is created, you can use the Draft version of the bot.

Type: String

Valid Values: Creating | Available | Inactive | Deleting | Failed | Versioning | ImportingbotTags (p. 314)

A list of tags associated with the bot.

Type: String to string map
Map Entries: Minimum number of 0 items. Maximum number of 200 items.
Key Length Constraints: Minimum length of 1. Maximum length of 128.
Value Length Constraints: Minimum length of 0. Maximum length of 256.

**creationDateTime (p. 314)**
A timestamp indicating the date and time that the bot was created.
Type: Timestamp

**dataPrivacy (p. 314)**
The data privacy settings specified for the bot.
Type: DataPrivacy (p. 701) object

**description (p. 314)**
The description specified for the bot.
Type: String
Length Constraints: Minimum length of 0. Maximum length of 200.

**idleSessionTTLInSeconds (p. 314)**
The session idle time specified for the bot.
Type: Integer
Valid Range: Minimum value of 60. Maximum value of 86400.

**roleArn (p. 314)**
The IAM role specified for the bot.
Type: String
Length Constraints: Minimum length of 32. Maximum length of 2048.
Pattern: ^arn:aws:iam::[0-9]{12}:role/.*$

**testBotAliasTags (p. 314)**
A list of tags associated with the test alias for the bot.
Type: String to string map
Map Entries: Minimum number of 0 items. Maximum number of 200 items.
Key Length Constraints: Minimum length of 1. Maximum length of 128.
Value Length Constraints: Minimum length of 0. Maximum length of 256.

**Errors**
For information about the errors that are common to all actions, see Common Errors (p. 821).

**ConflictException**
The action that you tried to perform couldn't be completed because the resource is in a conflicting state. For example, deleting a bot that is in the CREATING state. Try your request again.
HTTP Status Code: 409

**InternalServerException**

The service encountered an unexpected condition. Try your request again.

HTTP Status Code: 500

**PreconditionFailedException**

Your request couldn't be completed because one or more request fields aren't valid. Check the fields in your request and try again.

HTTP Status Code: 412

**ServiceQuotaExceededException**

You have reached a quota for your bot.

HTTP Status Code: 402

**ThrottlingException**

Your request rate is too high. Reduce the frequency of requests.

HTTP Status Code: 429

**ValidationException**

One of the input parameters in your request isn't valid. Check the parameters and try your request again.

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 V2
- AWS SDK for JavaScript
- AWS SDK for PHP V3
- AWS SDK for Python
- AWS SDK for Ruby V3
CreateBotAlias
Service: Amazon Lex Model Building V2

Creates an alias for the specified version of a bot. Use an alias to enable you to change the version of a bot without updating applications that use the bot.

For example, you can create an alias called "PROD" that your applications use to call the Amazon Lex bot.

Request Syntax

```
PUT /bots/botId/botAliases/ HTTP/1.1
Content-type: application/json

{
  "botAliasLocaleSettings": {
    "string": {
      "codeHookSpecification": {
        "lambdaCodeHook": {
          "codeHookInterfaceVersion": "string",
          "lambdaARN": "string"
        },
        "enabled": boolean
      },
      "enabled": boolean
    },
    "botAliasName": "string",
    "botVersion": "string",
    "conversationLogSettings": {
      "audioLogSettings": [
        {
          "destination": {
            "s3Bucket": {
              "kmsKeyArn": "string",
              "logPrefix": "string",
              "s3BucketArn": "string"
            },
          "enabled": boolean
        }
      ],
      "textLogSettings": [
        {
          "destination": {
            "cloudWatch": {
              "cloudWatchLogGroupArn": "string",
              "logPrefix": "string"
            },
          "enabled": boolean
        }
      ],
      "description": "string",
      "sentimentAnalysisSettings": {
        "detectSentiment": boolean
      },
      "tags": {
        "string": "string"
      }
    }
  }
```


URI Request Parameters

The request uses the following URI parameters.

**botId (p. 318)**

The unique identifier of the bot that the alias applies to.

Length Constraints: Fixed length of 10.

Pattern: ^[0-9a-zA-Z]+$

Required: Yes

Request Body

The request accepts the following data in JSON format.

**botAliasLocaleSettings (p. 318)**

Maps configuration information to a specific locale. You can use this parameter to specify a specific Lambda function to run different functions in different locales.

Type: String to BotAliasLocaleSettings (p. 665) object map

Map Entries: Maximum number of items.

Required: No

**botAliasName (p. 318)**

The alias to create. The name must be unique for the bot.

Type: String

Length Constraints: Minimum length of 1. Maximum length of 100.

Pattern: ^([0-9a-zA-Z][_-]?)+$

Required: Yes

**botVersion (p. 318)**

The version of the bot that this alias points to. You can use the UpdateBotAlias operation to change the bot version associated with the alias.

Type: String


Pattern: ^[0-9]+$

Required: No

**conversationLogSettings (p. 318)**

Specifies whether Amazon Lex logs text and audio for a conversation with the bot. When you enable conversation logs, text logs store text input, transcripts of audio input, and associated metadata in Amazon CloudWatch Logs. Audio logs store audio input in Amazon S3.

Type: ConversationLogSettings (p. 697) object
Required: No
description (p. 318)

A description of the alias. Use this description to help identify the alias.

Type: String

Length Constraints: Minimum length of 0. Maximum length of 200.

Required: No

sentimentAnalysisSettings (p. 318)

Determines whether Amazon Lex will use Amazon Comprehend to detect the sentiment of user utterances.

Type: SentimentAnalysisSettings (p. 753) object

Required: No
tags (p. 318)

A list of tags to add to the bot alias. You can only add tags when you create an alias, you can't use the UpdateBotAlias operation to update the tags on a bot alias. To update tags, use the TagResource operation.

Type: String to string map

Map Entries: Minimum number of 0 items. Maximum number of 200 items.

Key Length Constraints: Minimum length of 1. Maximum length of 128.

Value Length Constraints: Minimum length of 0. Maximum length of 256.

Required: No

Response Syntax

HTTP/1.1 202
Content-type: application/json

```json
{
    "botAliasId": "string",
    "botAliasLocaleSettings": {
        "string": {
            "codeHookSpecification": {
                "lambdaCodeHook": {
                    "codeHookInterfaceVersion": "string",
                    "lambdaARN": "string"
                }
            },
            "enabled": boolean
        }
    },
    "botAliasName": "string",
    "botAliasStatus": "string",
    "botId": "string",
    "botVersion": "string",
    "conversationLogSettings": {
        "audioLogSettings": [
            {
                "destination": {
```

320
Response Elements

If the action is successful, the service sends back an HTTP 202 response.

The following data is returned in JSON format by the service.

**botAliasId (p. 320)**

The unique identifier of the bot alias.

Type: String

Length Constraints: Fixed length of 10.

Pattern: `^\bTSTALIASID\b[0-9a-zA-Z]+$`

**botAliasLocaleSettings (p. 320)**

Configuration information for a specific locale.

Type: String to BotAliasLocaleSettings (p. 665) object map

Map Entries: Maximum number of items.

**botAliasName (p. 320)**

The name specified for the bot alias.

Type: String

Length Constraints: Minimum length of 1. Maximum length of 100.

Pattern: `^[0-9a-zA-Z][\-]+$`
**botAliasStatus (p. 320)**

The current status of the alias. The alias is first put into the Creating state. When the alias is ready to be used, it is put into the Available state. You can use the DescribeBotAlias operation to get the current state of an alias.

Type: String

Valid Values: Creating | Available | Deleting | Failed

**botId (p. 320)**

The unique identifier of the bot that this alias applies to.

Type: String

Length Constraints: Fixed length of 10.

Pattern: ^[0-9a-zA-Z]+$

**botVersion (p. 320)**

The version of the bot associated with this alias.

Type: String


Pattern: ^[0-9]+$

**conversationLogSettings (p. 320)**

The conversation log settings specified for the alias.

Type: ConversationLogSettings (p. 697) object

**creationDateTime (p. 320)**

A Unix timestamp indicating the date and time that the bot alias was created.

Type: Timestamp

**description (p. 320)**

The description specified for the bot alias.

Type: String

Length Constraints: Minimum length of 0. Maximum length of 200.

**sentimentAnalysisSettings (p. 320)**

Determines whether Amazon Lex will use Amazon Comprehend to detect the sentiment of user utterances.

Type: SentimentAnalysisSettings (p. 753) object

**tags (p. 320)**

A list of tags associated with the bot alias.

Type: String to string map

Map Entries: Minimum number of 0 items. Maximum number of 200 items.

Key Length Constraints: Minimum length of 1. Maximum length of 128.
Value Length Constraints: Minimum length of 0. Maximum length of 256.

**Errors**

For information about the errors that are common to all actions, see Common Errors (p. 821).

**ConflictException**

The action that you tried to perform couldn't be completed because the resource is in a conflicting state. For example, deleting a bot that is in the CREATING state. Try your request again.

HTTP Status Code: 409

**InternalServerException**

The service encountered an unexpected condition. Try your request again.

HTTP Status Code: 500

**PreconditionFailedException**

Your request couldn't be completed because one or more request fields aren't valid. Check the fields in your request and try again.

HTTP Status Code: 412

**ServiceQuotaExceededException**

You have reached a quota for your bot.

HTTP Status Code: 402

**ThrottlingException**

Your request rate is too high. Reduce the frequency of requests.

HTTP Status Code: 429

**ValidationException**

One of the input parameters in your request isn't valid. Check the parameters and try your request again.

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 V2
- AWS SDK for JavaScript
- AWS SDK for PHP V3
- AWS SDK for Python
- AWS SDK for Ruby V3
CreateBotLocale
Service: Amazon Lex Model Building V2

Creates a locale in the bot. The locale contains the intents and slot types that the bot uses in conversations with users in the specified language and locale. You must add a locale to a bot before you can add intents and slot types to the bot.

Request Syntax

```
PUT /bots/botId/botversions/botVersion/botlocales/ HTTP/1.1
Content-type: application/json

{
  "description": "string",
  "localeId": "string",
  "nluIntentConfidenceThreshold": number,
  "voiceSettings": {
    "engine": "string",
    "voiceId": "string"
  }
}
```

URI Request Parameters

The request uses the following URI parameters.

**botId (p. 324)**

The identifier of the bot to create the locale for.

Length Constraints: Fixed length of 10.

Pattern: ^[0-9a-zA-Z]+$

Required: Yes

**botVersion (p. 324)**

The version of the bot to create the locale for. This can only be the draft version of the bot.

Length Constraints: Fixed length of 5.

Pattern: ^DRAFT$

Required: Yes

Request Body

The request accepts the following data in JSON format.

**description (p. 324)**

A description of the bot locale. Use this to help identify the bot locale in lists.

Type: String

Length Constraints: Minimum length of 0. Maximum length of 200.

Required: No
localeId (p. 324)

The identifier of the language and locale that the bot will be used in. The string must match one of the supported locales. All of the intents, slot types, and slots used in the bot must have the same locale. For more information, see Supported languages.

Type: String

Required: Yes

nluIntentConfidenceThreshold (p. 324)

Determines the threshold where Amazon Lex will insert the AMAZON.FallbackIntent, AMAZON.KendraSearchIntent, or both when returning alternative intents. AMAZON.FallbackIntent and AMAZON.KendraSearchIntent are only inserted if they are configured for the bot.

For example, suppose a bot is configured with the confidence threshold of 0.80 and the AMAZON.FallbackIntent. Amazon Lex returns three alternative intents with the following confidence scores: IntentA (0.70), IntentB (0.60), IntentC (0.50). The response from the RecognizeText operation would be:
- AMAZON.FallbackIntent
- IntentA
- IntentB
- IntentC

Type: Double

Valid Range: Minimum value of 0. Maximum value of 1.

Required: Yes

voiceSettings (p. 324)

The Amazon Polly voice ID that Amazon Lex uses for voice interaction with the user.

Type: VoiceSettings (p. 778) object

Required: No

Response Syntax

```
HTTP/1.1 202
Content-type: application/json

{
  "botId": "string",
  "botLocaleStatus": "string",
  "botVersion": "string",
  "creationDateTime": number,
  "description": "string",
  "localeId": "string",
  "localeName": "string",
  "nluIntentConfidenceThreshold": number,
  "voiceSettings": {
    "engine": "string",
    "voiceId": "string"
  }
}
```
Response Elements

If the action is successful, the service sends back an HTTP 202 response.

The following data is returned in JSON format by the service.

**botId (p. 325)**

The specified bot identifier.

Type: String

Length Constraints: Fixed length of 10.

Pattern: ^[0-9a-zA-Z]+$

**botLocaleStatus (p. 325)**

The status of the bot.

When the status is *Creating* the bot locale is being configured. When the status is *Building* Amazon Lex is building the bot for testing and use.

If the status of the bot is *ReadyExpressTesting*, you can test the bot using the exact utterances specified in the bot's intents. When the bot is ready for full testing or to run, the status is *Built*.

If there was a problem with building the bot, the status is *Failed*. If the bot was saved but not built, the status is *NotBuilt*.

Type: String

Valid Values: Creating | Building | Built | ReadyExpressTesting | Failed | Deleting | NotBuilt | Importing | Processing

**botVersion (p. 325)**

The specified bot version.

Type: String

Length Constraints: Fixed length of 5.

Pattern: ^DRAFT$*

**creationDateTime (p. 325)**

A timestamp specifying the date and time that the bot locale was created.

Type: Timestamp

**description (p. 325)**

The specified description of the bot locale.

Type: String

Length Constraints: Minimum length of 0. Maximum length of 200.

**localeId (p. 325)**

The specified locale identifier.

Type: String

**localeName (p. 325)**

The specified locale name.
Type: String

**nluIntentConfidenceThreshold (p. 325)**

The specified confidence threshold for inserting the AMAZON.FallbackIntent and AMAZON.KendraSearchIntent intents.

Type: Double

Valid Range: Minimum value of 0. Maximum value of 1.

**voiceSettings (p. 325)**

The Amazon Polly voice ID that Amazon Lex uses for voice interaction with the user.

Type: VoiceSettings (p. 778) object

**Errors**

For information about the errors that are common to all actions, see [Common Errors (p. 821)](#).

**ConflictException**

The action that you tried to perform couldn't be completed because the resource is in a conflicting state. For example, deleting a bot that is in the CREATING state. Try your request again.

HTTP Status Code: 409

**InternalServerErrorException**

The service encountered an unexpected condition. Try your request again.

HTTP Status Code: 500

**PreconditionFailedException**

Your request couldn't be completed because one or more request fields aren't valid. Check the fields in your request and try again.

HTTP Status Code: 412

**ServiceQuotaExceededException**

You have reached a quota for your bot.

HTTP Status Code: 402

**ThrottlingException**

Your request rate is too high. Reduce the frequency of requests.

HTTP Status Code: 429

**ValidationException**

One of the input parameters in your request isn't valid. Check the parameters and try your request again.

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 V2
• AWS SDK for JavaScript
• AWS SDK for PHP V3
• AWS SDK for Python
• AWS SDK for Ruby V3
CreateBotVersion
Service: Amazon Lex Model Building V2

Creates a new version of the bot based on the DRAFT version. If the DRAFT 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.

When you create the first version of a bot, Amazon Lex sets the version to 1. Subsequent versions increment by 1.

Request Syntax

```
PUT /bots/botId/botversions/ HTTP/1.1
Content-type: application/json

{
  "botVersionLocaleSpecification": {
    "string": {
      "sourceBotVersion": "string"
    }
  },
  "description": "string"
}
```

URI Request Parameters

The request uses the following URI parameters.

botId (p. 329)

The identifier of the bot to create the version for.

Length Constraints: Fixed length of 10.

Pattern: `^[0-9a-zA-Z]+$`

Required: Yes

Request Body

The request accepts the following data in JSON format.

botVersionLocaleSpecification (p. 329)

Specifies the locales that Amazon Lex adds to this version. You can choose the Draft version or any other previously published version for each locale. When you specify a source version, the locale data is copied from the source version to the new version.

Type: String to BotVersionLocaleDetails (p. 686) object map

Map Entries: Maximum number of items.

Required: Yes

description (p. 329)

A description of the version. Use the description to help identify the version in lists.

Type: String
Length Constraints: Minimum length of 0. Maximum length of 200.

Required: No

Response Syntax

HTTP/1.1 202
Content-type: application/json

{
  "botId": "string",
  "botStatus": "string",
  "botVersion": "string",
  "botVersionLocaleSpecification": {
    "string": {
      "sourceBotVersion": "string"
    }
  },
  "creationDateTime": number,
  "description": "string"
}

Response Elements

If the action is successful, the service sends back an HTTP 202 response.

The following data is returned in JSON format by the service.

**botId (p. 330)**

The bot identifier specified in the request.

Type: String

Length Constraints: Fixed length of 10.

Pattern: `^[0-9a-zA-Z]+$`

**botStatus (p. 330)**

When you send a request to create or update a bot, Amazon Lex sets the status response element to Creating. After Amazon Lex builds the bot, it sets status to Available. If Amazon Lex can't build the bot, it sets status to Failed.

Type: String

Valid Values: Creating | Available | Inactive | Deleting | Failed | Versioning | Importing

**botVersion (p. 330)**

The version number assigned to the version.

Type: String


Pattern: `^[0-9]+$`

**botVersionLocaleSpecification (p. 330)**

The source versions used for each locale in the new version.
Type: String to `BotVersionLocaleDetails (p. 686)` object map

Map Entries: Maximum number of items.

**creationDateTime (p. 330)**

A timestamp of the date and time that the version was created.

Type: Timestamp

**description (p. 330)**

The description of the version specified in the request.

Type: String

Length Constraints: Minimum length of 0. Maximum length of 200.

### Errors

For information about the errors that are common to all actions, see [Common Errors (p. 821)].

**ConflictException**

The action that you tried to perform couldn't be completed because the resource is in a conflicting state. For example, deleting a bot that is in the CREATING state. Try your request again.

HTTP Status Code: 409

**InternalServerException**

The service encountered an unexpected condition. Try your request again.

HTTP Status Code: 500

**PreconditionFailedException**

Your request couldn't be completed because one or more request fields aren't valid. Check the fields in your request and try again.

HTTP Status Code: 412

**ServiceQuotaExceededException**

You have reached a quota for your bot.

HTTP Status Code: 402

**ThrottlingException**

Your request rate is too high. Reduce the frequency of requests.

HTTP Status Code: 429

**ValidationException**

One of the input parameters in your request isn't valid. Check the parameters and try your request again.

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 V2
• AWS SDK for JavaScript
• AWS SDK for PHP V3
• AWS SDK for Python
• AWS SDK for Ruby V3
CreateExport
Service: Amazon Lex Model Building V2

Creates a zip archive containing the contents of a bot or a bot locale. The archive contains a directory structure that contains JSON files that define the bot.

You can create an archive that contains the complete definition of a bot, or you can specify that the archive contain only the definition of a single bot locale.

For more information about exporting bots, and about the structure of the export archive, see Importing and exporting bots

Request Syntax

```json
PUT /exports/ HTTP/1.1
Content-type: application/json

{
    "fileFormat": "string",
    "filePassword": "string",
    "resourceSpecification": {
        "botExportSpecification": {
            "botId": "string",
            "botVersion": "string"
        },
        "botLocaleExportSpecification": {
            "botId": "string",
            "botVersion": "string",
            "localeId": "string"
        },
        "customVocabularyExportSpecification": {
            "botId": "string",
            "botVersion": "string",
            "localeId": "string"
        }
    }
}
```

URI Request Parameters

The request does not use any URI parameters.

Request Body

The request accepts the following data in JSON format.

**fileFormat (p. 333)**

The file format of the bot or bot locale definition files.

Type: String

Valid Values: LexJson | TSV

Required: Yes

**filePassword (p. 333)**

An password to use to encrypt the exported archive. Using a password is optional, but you should encrypt the archive to protect the data in transit between Amazon Lex and your local computer.

Type: String

Required: No

**resourceSpecification (p. 333)**

Specifies the type of resource to export, either a bot or a bot locale. You can only specify one type of resource to export.

Type: ExportResourceSpecification (p. 706) object

Required: Yes

**Response Syntax**

```
HTTP/1.1 202
Content-type: application/json

{
    "creationDateTime": number,
    "exportId": "string",
    "exportStatus": "string",
    "fileFormat": "string",
    "resourceSpecification": {
        "botExportSpecification": {
            "botId": "string",
            "botVersion": "string"
        },
        "botLocaleExportSpecification": {
            "botId": "string",
            "botVersion": "string",
            "localeId": "string"
        },
        "customVocabularyExportSpecification": {
            "botId": "string",
            "botVersion": "string",
            "localeId": "string"
        }
    }
}
```

**Response Elements**

If the action is successful, the service sends back an HTTP 202 response.

The following data is returned in JSON format by the service.

**creationDateTime (p. 334)**

The date and time that the request to export a bot was created.

Type: Timestamp

**exportId (p. 334)**

An identifier for a specific request to create an export.

Type: String

Length Constraints: Fixed length of 10.

Pattern: ^[0-9a-zA-Z]+$
exportStatus (p. 334)

The status of the export. When the status is Completed, you can use the DescribeExport operation to get the pre-signed S3 URL link to your exported bot or bot locale.

Type: String

Valid Values: InProgress | Completed | Failed | Deleting

fileFormat (p. 334)

The file format used for the bot or bot locale definition files.

Type: String

Valid Values: LexJson | TSV

resourceSpecification (p. 334)

A description of the type of resource that was exported, either a bot or a bot locale.

Type: ExportResourceSpecification (p. 706) object

Errors

For information about the errors that are common to all actions, see Common Errors (p. 821).

ConflictException

The action that you tried to perform couldn't be completed because the resource is in a conflicting state. For example, deleting a bot that is in the CREATING state. Try your request again.

HTTP Status Code: 409

InternalServerException

The service encountered an unexpected condition. Try your request again.

HTTP Status Code: 500

ResourceNotFoundException

You asked to describe a resource that doesn't exist. Check the resource that you are requesting and try again.

HTTP Status Code: 404

ServiceQuotaExceededException

You have reached a quota for your bot.

HTTP Status Code: 402

ThrottlingException

Your request rate is too high. Reduce the frequency of requests.

HTTP Status Code: 429

ValidationException

One of the input parameters in your request isn't valid. Check the parameters and try your request again.

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 V2
- AWS SDK for JavaScript
- AWS SDK for PHP V3
- AWS SDK for Python
- AWS SDK for Ruby V3
CreateIntent
Service: Amazon Lex Model Building V2

Creates an intent.

To define the interaction between the user and your bot, you define one or more intents. For example, for a pizza ordering bot you would create an OrderPizza intent.

When you create an intent, you must provide a name. You can optionally provide the following:

- Sample utterances. For example, "I want to order a pizza" and "Can I order a pizza." You can't provide utterances for built-in intents.
- Information to be gathered. You specify slots for the information that your bot requests from the user. You can specify standard slot types, such as date and time, or custom slot types for your application.
- How the intent is fulfilled. You can provide a Lambda function or configure the intent to return the intent information to your client application. If you use a Lambda function, Amazon Lex invokes the function when all of the intent information is available.
- A confirmation prompt to send to the user to confirm an intent. For example, "Shall I order your pizza?"
- A conclusion statement to send to the user after the intent is fulfilled. For example, "I ordered your pizza."
- A follow-up prompt that asks the user for additional activity. For example, "Do you want a drink with your pizza?"

Request Syntax

```
PUT /bots/botId/botversions/botVersion/botlocales/localeId/intents/ HTTP/1.1
Content-type: application/json

{
    "description": "string",
    "dialogCodeHook": {
        "enabled": boolean
    },
    "fulfillmentCodeHook": {
        "enabled": boolean,
        "fulfillmentUpdatesSpecification": {
            "active": boolean,
            "startResponse": {
                "allowInterrupt": boolean,
                "delayInSeconds": number,
                "messageGroups": [
                    {
                        "message": {
                            "customPayload": {
                                "value": "string"
                            },
                            "imageResponseCard": {
                                "buttons": [
                                    {
                                        "text": "string",
                                        "value": "string"
                                    }
                                ],
                                "imageUrl": "string",
                                "subtitle": "string",
                                "title": "string"
                            }
                        }
                    }
                ]
            }
        }
    }
}
```
"value": "string",
"ssmlMessage": {  
  "value": "string"
},
"variations": [  
  {    
    "customPayload": {    
      "value": "string"
    },    
    "imageResponseCard": {    
      "buttons": [    
        {    
          "text": "string",
          "value": "string"
        }
      ],    
      "imageUrl": "string",
      "subtitle": "string",
      "title": "string"
    },    
    "plainTextMessage": {    
      "value": "string"
    },    
    "ssmlMessage": {    
      "value": "string"
    }
  }
]
},
"timeoutInSeconds": number,
"updateResponse": {  
  "allowInterrupt": boolean,
  "frequencyInSeconds": number,
  "messageGroups": [  
    {  
      "message": {  
        "customPayload": {  
          "value": "string"
        },  
        "imageResponseCard": {  
          "buttons": [  
            {  
              "text": "string",
              "value": "string"
            }
          ],  
          "imageUrl": "string",
          "subtitle": "string",
          "title": "string"
        },  
        "plainTextMessage": {  
          "value": "string"
        },  
        "ssmlMessage": {  
          "value": "string"
        }
      },  
      "variations": [  
        {  
          "customPayload": {  
            "value": "string"
          }
        }  
      ]
    }
  ]
}
"imageResponseCard": { "buttons": [ { "text": "string", "value": "string" } ], "imageUrl": "string", "subtitle": "string", "title": "string" }, "plainTextMessage": { "value": "string" }, "ssmlMessage": { "value": "string" } }, ] } }, "postFulfillmentStatusSpecification": { "failureResponse": { "allowInterrupt": boolean, "messageGroups": [ { "message": { "customPayload": { "value": "string" } }, "imageResponseCard": { "buttons": [ { "text": "string", "value": "string" } ], "imageUrl": "string", "subtitle": "string", "title": "string" }, "plainTextMessage": { "value": "string" }, "ssmlMessage": { "value": "string" } }, ] }, "variations": [ { "customPayload": { "value": "String" } }, "imageResponseCard": { "buttons": [ { "text": "string", "value": "string" } ], "imageUrl": "string", "subtitle": "string", "title": "string" } ], "imageUrl": "string", "subtitle": "string", "title": "string" } }
"plainTextMessage": {
  "value": "string"
},
"ssmlMessage": {
  "value": "string"
}
}
},
"successResponse": {
  "allowInterrupt": boolean,
  "messageGroups": [
    {
      "message": {
        "customPayload": {
          "value": "string"
        },
        "imageResponseCard": {
          "buttons": [
            {
              "text": "string",
              "value": "string"
            }
          ],
          "imageUrl": "string",
          "subtitle": "string",
          "title": "string"
        },
        "plainTextMessage": {
          "value": "string"
        },
        "ssmlMessage": {
          "value": "string"
        }
      },
      "variations": [
        {
          "customPayload": {
            "value": "string"
          },
          "imageResponseCard": {
            "buttons": [
              {
                "text": "string",
                "value": "string"
              }
            ],
            "imageUrl": "string",
            "subtitle": "string",
            "title": "string"
          },
          "plainTextMessage": {
            "value": "string"
          },
          "ssmlMessage": {
            "value": "string"
          }
        }
      ]
    }
  ]
},
"timeoutResponse": {
  "allowInterrupt": boolean,

"messageGroups": [
  {
    "message": {
      "customPayload": {
        "value": "string"
      },
      "imageResponseCard": {
        "buttons": [
          {
            "text": "string",
            "value": "string"
          }
        ],
        "imageUrl": "string",
        "subtitle": "string",
        "title": "string"
      },
      "plainTextMessage": {
        "value": "string"
      },
      "ssmlMessage": {
        "value": "string"
      }
    },
    "variations": [
      {
        "customPayload": {
          "value": "string"
        },
        "imageResponseCard": {
          "buttons": [
            {
              "text": "string",
              "value": "string"
            }
          ],
          "imageUrl": "string",
          "subtitle": "string",
          "title": "string"
        },
        "plainTextMessage": {
          "value": "string"
        },
        "ssmlMessage": {
          "value": "string"
        }
      }
    ]
  }
],
"inputContexts": [
  {
    "name": "string"
  }
],
"intentClosingSetting": {
  "active": boolean,
  "closingResponse": {
    "allowInterrupt": boolean,
    "messageGroups": [
      {
        "message": {
          "customPayload": {
            "value": "string"
          }
        }
      }
    ]
  }
}
"value": "string",
},
"imageResponseCard": {
  "buttons": [
    {
      "text": "string",
      "value": "string"
    },
    "imageUrl": "string",
    "subtitle": "string",
    "title": "string"
  ],
  "PlainTextMessage": {
    "value": "string"
  },
  "SsmlMessage": {
    "value": "string"
  }
},
"variations": [
  {
    "customPayload": {
      "value": "string"
    },
    "imageResponseCard": {
      "buttons": [
        {
          "text": "string",
          "value": "string"
        },
      ],
      "imageUrl": "string",
      "subtitle": "string",
      "title": "string"
    },
    "PlainTextMessage": {
      "value": "string"
    },
    "SsmlMessage": {
      "value": "string"
    }
  }
],

"intentConfirmationSetting": {
  "active": boolean,
  "declinationResponse": {
    "allowInterrupt": boolean,
    "messageGroups": [
      {
        "message": {
          "customPayload": {
            "value": "string"
          },
          "imageResponseCard": {
            "buttons": [
              {
                "text": "string",
                "value": "string"
              },
          ],
          "imageUrl": "string",
          "subtitle": "string",
          "title": "string"
        },
      }]}
"subtitle": "string",
"title": "string"
},
"plainTextMessage": {
"value": "string"
},
"ssmlMessage": {
"value": "string"
}
},
"variations": [
{
"customPayload": {
"value": "string"
},
"imageResponseCard": {
"buttons": [
{
"text": "string",
"value": "string"
}
],
"imageUrl": "string",
"subtitle": "string",
"title": "string"
},
"plainTextMessage": {
"value": "string"
},
"ssmlMessage": {
"value": "string"
}
}
]
}
,"promptSpecification": {
"allowInterrupt": boolean,
"maxRetries": number,
"messageGroups": [
{
"message": {
"customPayload": {
"value": "string"
},
"imageResponseCard": {
"buttons": [
{
"text": "string",
"value": "string"
}
],
"imageUrl": "string",
"subtitle": "string",
"title": "string"
},
"plainTextMessage": {
"value": "string"
},
"ssmlMessage": {
"value": "string"
}
},
"variations": [
{

URI Request Parameters

The request uses the following URI parameters.

botId (p. 337)

The identifier of the bot associated with this intent.

Length Constraints: Fixed length of 10.

Pattern: ^[0-9a-zA-Z]+$

Required: Yes

botVersion (p. 337)

The identifier of the version of the bot associated with this intent.
Length Constraints: Fixed length of 5.

Pattern: ^DRAFT$

Required: Yes

localeId (p. 337)

The identifier of the language and locale where this intent is used. All of the bots, slot types, and slots used by the intent must have the same locale. For more information, see Supported languages.

Required: Yes

Request Body

The request accepts the following data in JSON format.

description (p. 337)

A description of the intent. Use the description to help identify the intent in lists.

Type: String

Length Constraints: Minimum length of 0. Maximum length of 200.

Required: No

dialogCodeHook (p. 337)

Specifies that Amazon Lex invokes the alias Lambda function for each user input. You can invoke this Lambda function to personalize user interaction.

For example, suppose that your bot determines that the user's name is John. You Lambda function might retrieve John's information from a backend database and prepupulate 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: DialogCodeHookSettings (p. 703) object

Required: No

fulfillmentCodeHook (p. 337)

Specifies that Amazon Lex invokes the alias Lambda function when the intent is ready for fulfillment. You can invoke this function to complete the bot's transaction with the user.

For example, in a pizza ordering bot, the Lambda function can look up the closest pizza restaurant to the customer's location and then place an order on the customer's behalf.

Type: FulfillmentCodeHookSettings (p. 711) object

Required: No

inputContexts (p. 337)

A list of contexts that must be active for this intent to be considered by Amazon Lex.

When an intent has an input context list, Amazon Lex only considers using the intent in an interaction with the user when the specified contexts are included in the active context list for the session. If the contexts are not active, then Amazon Lex will not use the intent.

A context can be automatically activated using the outputContexts property or it can be set at runtime.
For example, if there are two intents with different input contexts that respond to the same utterances, only the intent with the active context will respond.

An intent may have up to 5 input contexts. If an intent has multiple input contexts, all of the contexts must be active to consider the intent.

Type: Array of InputContext (p. 724) objects

Array Members: Minimum number of 0 items. Maximum number of 5 items.

Required: No

intentClosingSetting (p. 337)

Sets the response that Amazon Lex sends to the user when the intent is closed.

Type: IntentClosingSetting (p. 725) object

Required: No

intentConfirmationSetting (p. 337)

Provides prompts that Amazon Lex sends to the user to confirm the completion of an intent. If the user answers "no," the settings contain a statement that is sent to the user to end the intent.

Type: IntentConfirmationSetting (p. 726) object

Required: No

intentName (p. 337)

The name of the intent. Intent names must be unique in the locale that contains the intent and cannot match the name of any built-in intent.

Type: String

Length Constraints: Minimum length of 1. Maximum length of 100.

Pattern: ^([0-9a-zA-Z][_-]?)+$

Required: Yes

kendraConfiguration (p. 337)

Configuration information required to use the AMAZON.KendraSearchIntent intent to connect to an Amazon Kendra index. The AMAZON.KendraSearchIntent intent is called when Amazon Lex can’t determine another intent to invoke.

Type: KendraConfiguration (p. 732) object

Required: No

outputContexts (p. 337)

A lists of contexts that the intent activates when it is fulfilled.

You can use an output context to indicate the intents that Amazon Lex should consider for the next turn of the conversation with a customer.

When you use the outputContextsList property, all of the contexts specified in the list are activated when the intent is fulfilled. You can set up to 10 output contexts. You can also set the number of conversation turns that the context should be active, or the length of time that the context should be active.
Type: Array of `OutputContext (p. 739)` objects

Array Members: Minimum number of 0 items. Maximum number of 10 items.

Required: No

**parentIntentSignature (p. 337)**

A unique identifier for the built-in intent to base this intent on.

Type: `String`

Required: No

**sampleUtterances (p. 337)**

An array of strings that a user might say to signal the intent. For example, "I want a pizza", or "I want a {PizzaSize} pizza".

In an utterance, slot names are enclosed in curly braces ("{", "}") to indicate where they should be displayed in the utterance shown to the user..

Type: Array of `SampleUtterance (p. 751)` objects

Required: No

**Response Syntax**

```
HTTP/1.1 200
Content-type: application/json

{
  "botId": "string",
  "botVersion": "string",
  "creationDateTime": number,
  "description": "string",
  "dialogCodeHook": {
    "enabled": boolean
  },
  "fulfillmentCodeHook": {
    "enabled": boolean,
    "fulfillmentUpdatesSpecification": {
      "active": boolean,
      "startResponse": {
        "allowInterrupt": boolean,
        "delayInSeconds": number,
        "messageGroups": [
          {
            "message": {
              "customPayload": {
                "value": "string"
              },
              "imageResponseCard": {
                "buttons": [
                  {
                    "text": "string",
                    "value": "string"
                  }
                ],
                "imageUrl": "string",
                "subtitle": "string",
                "title": "string"
              }
            },
            "plainTextMessage": {
```
"value": "string"
},
"ssmlMessage": {
  "value": "string"
}
},
"variations": [
  {
    "customPayload": {
      "value": "string"
    },
    "imageResponseCard": {
      "buttons": [
        {
          "text": "string",
          "value": "string"
        }
      ],
      "imageUrl": "string",
      "subtitle": "string",
      "title": "string"
    },
    "plainTextMessage": {
      "value": "string"
    },
    "ssmlMessage": {
      "value": "string"
    }
  }
],
"timeoutInSeconds": number,
"updateResponse": {
  "allowInterrupt": boolean,
  "frequencyInSeconds": number,
  "messageGroups": [
    {
      "message": {
        "customPayload": {
          "value": "string"
        },
        "imageResponseCard": {
          "buttons": [
            {
              "text": "string",
              "value": "string"
            }
          ],
          "imageUrl": "string",
          "subtitle": "string",
          "title": "string"
        },
        "plainTextMessage": {
          "value": "string"
        },
        "ssmlMessage": {
          "value": "string"
        }
      }
    }
  ],
  "variations": [
    {
      "customPayload": {
        "value": "string"
      },
      "imageResponseCard": {
        "buttons": [
          {
            "text": "string",
            "value": "string"
          }
        ],
        "imageUrl": "string",
        "subtitle": "string",
        "title": "string"
      },
      "plainTextMessage": {
        "value": "string"
      },
      "ssmlMessage": {
        "value": "string"
      }
    }
  ]
}
"imageResponseCard": {
   "buttons": [
      {
         "text": "string",
         "value": "string"
      }
   ],
   "ImageUrl": "string",
   "subtitle": "string",
   "title": "string"
},
"plainTextMessage": {
   "value": "string"
},
"ssmlMessage": {
   "value": "string"
}
},
"postFulfillmentStatusSpecification": {
   "failureResponse": {
      "allowInterrupt": boolean,
      "messageGroups": [
         {
            "message": {
               "customPayload": {
                  "value": "string"
               },
               "imageResponseCard": {
                  "buttons": [
                     {
                        "text": "string",
                        "value": "string"
                     }
                  ],
                  "ImageUrl": "string",
                  "subtitle": "string",
                  "title": "string"
               },
               "plainTextMessage": {
                  "value": "string"
               },
               "ssmlMessage": {
                  "value": "string"
               }
            }
         },
         {
            "customPayload": {
               "value": "string"
            },
            "imageResponseCard": {
               "buttons": [
                  {
                     "text": "string",
                     "value": "string"
                  }
               ],
               "ImageUrl": "string",
               "subtitle": "string",
               "title": "string"
            },
            "plainTextMessage": {
               "value": "string"
            },
            "ssmlMessage": {
               "value": "string"
            }
         }
      ],
      "variations": [
         {
            "customPayload": {
               "value": "string"
            },
            "imageResponseCard": {
               "buttons": [
                  {
                     "text": "string",
                     "value": "string"
                  }
               ],
               "ImageUrl": "string",
               "subtitle": "string",
               "title": "string"
            }
         }
      ]
   }
}
"successResponse": {
    "allowInterrupt": true,
    "messageGroups": [
    {
        "message": {
            "customPayload": {
                "value": "string"
            },
            "imageResponseCard": {
                "buttons": [
                {
                    "text": "string",
                    "value": "string"
                }
                ],
                "imageUrl": "string",
                "subtitle": "string",
                "title": "string"
            },
            "plainTextMessage": {
                "value": "string"
            },
            "ssmlMessage": {
                "value": "string"
            }
        }
    },
    {"customPayload": {
            "value": "string"
        },
        "imageResponseCard": {
            "buttons": [
            {
                "text": "string",
                "value": "string"
            }
            ],
            "imageUrl": "string",
            "subtitle": "string",
            "title": "string"
        },
        "plainTextMessage": {
            "value": "string"
        },
        "ssmlMessage": {
            "value": "string"
        }},
        "variations": [
        {
            "customPayload": {
                "value": "string"
            },
            "imageResponseCard": {
                "buttons": [
                {
                    "text": "string",
                    "value": "string"
                }
                ],
                "imageUrl": "string",
                "subtitle": "string",
                "title": "string"
            },
            "plainTextMessage": {
                "value": "string"
            },
            "ssmlMessage": {
                "value": "string"
            }
        }
    ],
    }]},
"messageGroups": [
    {
        "message": {
            "customPayload": {},
            "imageResponseCard": {
                "buttons": [
                    {
                        "text": "string",
                        "value": "string"
                    }
                ],
                "imageUrl": "string",
                "subtitle": "string",
                "title": "string"
            },
            "plainTextMessage": {
                "value": "string"
            },
            "ssmlMessage": {
                "value": "string"
            }
        },
        "variations": [
            {
                "customPayload": {},
                "imageResponseCard": {
                    "buttons": [
                        {
                            "text": "string",
                            "value": "string"
                        }
                    ],
                    "imageUrl": "string",
                    "subtitle": "string",
                    "title": "string"
                },
                "plainTextMessage": {
                    "value": "string"
                },
                "ssmlMessage": {
                    "value": "string"
                }
            }
        ]
    }
],
"inputContexts": [
    {
        "name": "string"
    }
],
"intentClosingSetting": {
    "active": boolean,
    "closingResponse": {
        "allowInterrupt": boolean,
        "messageGroups": [
            {
                "message": {
                    "customPayload": {
                        "value": "string"
                    }
                },
                "imageResponseCard": {
                    "buttons": [
                        {
                            "text": "string",
                            "value": "string"
                        }
                    ],
                    "imageUrl": "string",
                    "subtitle": "string",
                    "title": "string"
                },
                "plainTextMessage": {
                    "value": "string"
                },
                "ssmlMessage": {
                    "value": "string"
                }
            }
        ]
    }
}
"value": "string"
},
"imageResponseCard": {
  "buttons": [
    {
      "text": "string",
      "value": "string"
    }
  ],
  "imageUrl": "string",
  "subtitle": "string",
  "title": "string"
},
"plainTextMessage": {
  "value": "string"
},
"ssmlMessage": {
  "value": "string"
}
},
"variations": [
  {
    "customPayload": {
      "value": "string"
    },
    "imageResponseCard": {
      "buttons": [
        {
          "text": "string",
          "value": "string"
        }
      ],
      "imageUrl": "string",
      "subtitle": "string",
      "title": "string"
    },
    "plainTextMessage": {
      "value": "string"
    },
    "ssmlMessage": {
      "value": "string"
    }
  }
]
}
]
}
"intentConfirmationSetting": {
  "active": boolean,
  "declinationResponse": {
    "allowInterrupt": boolean,
    "messageGroups": [
      {
        "message": {
          "customPayload": {
            "value": "string"
          },
          "imageResponseCard": {
            "buttons": [
              {
                "text": "string",
                "value": "string"
              }
            ],
            "imageUrl": "string",
            "subtitle": "string",
            "title": "string"
          }
        }
      }
    ]
  }
}
"promptSpecification": {
  "allowInterrupt": boolean,
  "maxRetries": number,
  "messageGroups": [
    {
      "message": {
        "customPayload": {
          "value": "string"
        },
        "imageResponseCard": {
          "buttons": [
            { "text": "string", "value": "string" }
          ],
          "imageUrl": "string",
          "subtitle": "string",
          "title": "string"
        },
        "plainTextMessage": {
          "value": "string"
        },
        "ssmlMessage": {
          "value": "string"
        }
      }
    }
  ]
},
"variations": [
  {
    "customPayload": {
      "value": "string"
    },
    "imageResponseCard": {
      "buttons": [
        { "text": "string", "value": "string" }
      ],
      "imageUrl": "string",
      "subtitle": "string",
      "title": "string"
    },
    "plainTextMessage": {
      "value": "string"
    },
    "ssmlMessage": {
      "value": "string"
    }
  }
]
"customPayload": {  
  "value": "string"
},
"imageResponseCard": {  
  "buttons": [  
    {  
      "text": "string",
      "value": "string"
    },  
    "imageUrl": "string",
    "subtitle": "string",
    "title": "string"
  ],
  "plainTextMessage": {  
    "value": "string"
  },
  "ssmlMessage": {  
    "value": "string"
  }

  }  
  }
  }  
  "intentId": "string",
  "intentName": "string",
  "kendraConfiguration": {  
    "kendraIndex": "string",
    "queryFilterString": "string",
    "queryFilterStringEnabled": boolean
  },
  "localeId": "string",
  "outputContexts": [  
    {  
      "name": "string",
      "timeToLiveInSeconds": number,
      "turnsToLive": number
    },
    "parentIntentSignature": "string",
    "sampleUtterances": [  
      {  
        "utterance": "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.

**botId (p. 347)**

The identifier of the bot associated with the intent.

Type: String

Length Constraints: Fixed length of 10.

Pattern: ^[0-9a-zA-Z]+$
botVersion (p. 347)
The identifier of the version of the bot associated with the intent.
Type: String
Length Constraints: Fixed length of 5.
Pattern: ^DRAFT$
creationDateTime (p. 347)
A timestamp of the date and time that the intent was created.
Type: Timestamp
description (p. 347)
The description specified for the intent.
Type: String
Length Constraints: Minimum length of 0. Maximum length of 200.
dialogCodeHook (p. 347)
The dialog Lambda function specified for the intent.
Type: DialogCodeHookSettings (p. 703) object
fulfillmentCodeHook (p. 347)
The fulfillment Lambda function specified for the intent.
Type: FulfillmentCodeHookSettings (p. 711) object
inputContexts (p. 347)
The list of input contexts specified for the intent.
Type: Array of InputContext (p. 724) objects
Array Members: Minimum number of 0 items. Maximum number of 5 items.
intentClosingSetting (p. 347)
The closing setting specified for the intent.
Type: IntentClosingSetting (p. 725) object
intentConfirmationSetting (p. 347)
The confirmation setting specified for the intent.
Type: IntentConfirmationSetting (p. 726) object
intentId (p. 347)
A unique identifier for the intent.
Type: String
Length Constraints: Fixed length of 10.
Pattern: ^[0-9a-zA-Z]+$
intentName (p. 347)
The name specified for the intent.
Type: String

Length Constraints: Minimum length of 1. Maximum length of 100.

Pattern: ^([0-9a-zA-Z][_-]?)+$

**kendraConfiguration (p. 347)**

Configuration for searching an Amazon Kendra index specified for the intent.

Type: KendraConfiguration (p. 732) object

**localeId (p. 347)**

The locale that the intent is specified to use.

Type: String

**outputContexts (p. 347)**

The list of output contexts specified for the intent.

Type: Array of OutputContext (p. 739) objects

Array Members: Minimum number of 0 items. Maximum number of 10 items.

**parentIntentSignature (p. 347)**

The signature of the parent intent specified for the intent.

Type: String

**sampleUtterances (p. 347)**

The sample utterances specified for the intent.

Type: Array of SampleUtterance (p. 751) objects

**Errors**

For information about the errors that are common to all actions, see Common Errors (p. 821).

**ConflictException**

The action that you tried to perform couldn't be completed because the resource is in a conflicting state. For example, deleting a bot that is in the CREATING state. Try your request again.

HTTP Status Code: 409

**InternalServerErrorException**

The service encountered an unexpected condition. Try your request again.

HTTP Status Code: 500

**PreconditionFailedException**

Your request couldn't be completed because one or more request fields aren't valid. Check the fields in your request and try again.

HTTP Status Code: 412

**ServiceQuotaExceededException**

You have reached a quota for your bot.

HTTP Status Code: 402
ThrottlingException

Your request rate is too high. Reduce the frequency of requests.

HTTP Status Code: 429

ValidationException

One of the input parameters in your request isn't valid. Check the parameters and try your request again.

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 V2
• AWS SDK for JavaScript
• AWS SDK for PHP V3
• AWS SDK for Python
• AWS SDK for Ruby V3
CreateResourcePolicy
Service: Amazon Lex Model Building V2

Creates a new resource policy with the specified policy statements.

Request Syntax

```
POST /policy/resourceArn/ HTTP/1.1
Content-type: application/json

{
  "policy": "string"
}
```

URI Request Parameters

The request uses the following URI parameters.

resourceArn (p. 358)

The Amazon Resource Name (ARN) of the bot or bot alias that the resource policy is attached to.

Length Constraints: Minimum length of 1. Maximum length of 1011.

Required: Yes

Request Body

The request accepts the following data in JSON format.

policy (p. 358)

A resource policy to add to the resource. The policy is a JSON structure that contains one or more statements that define the policy. The policy must follow the IAM syntax. For more information about the contents of a JSON policy document, see IAM JSON policy reference.

If the policy isn't valid, Amazon Lex returns a validation exception.

Type: String

Length Constraints: Minimum length of 2.

Required: Yes

Response Syntax

```
HTTP/1.1 200
Content-type: application/json

{
  "resourceArn": "string",
  "revisionId": "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.

**resourceArn (p. 358)**

The Amazon Resource Name (ARN) of the bot or bot alias that the resource policy was attached to.

Type: String

Length Constraints: Minimum length of 1. Maximum length of 1011.

**revisionId (p. 358)**

The current revision of the resource policy. Use the revision ID to make sure that you are updating the most current version of a resource policy when you add a policy statement to a resource, delete a resource, or update a resource.

Type: String


Pattern: `^[0-9]+$`

**Errors**

For information about the errors that are common to all actions, see Common Errors (p. 821).

**InternalServerException**

The service encountered an unexpected condition. Try your request again.

HTTP Status Code: 500

**PreconditionFailedException**

Your request couldn't be completed because one or more request fields aren't valid. Check the fields in your request and try again.

HTTP Status Code: 412

**ResourceNotFoundException**

You asked to describe a resource that doesn't exist. Check the resource that you are requesting and try again.

HTTP Status Code: 404

**ServiceQuotaExceededException**

You have reached a quota for your bot.

HTTP Status Code: 402

**ThrottlingException**

Your request rate is too high. Reduce the frequency of requests.

HTTP Status Code: 429

**ValidationException**

One of the input parameters in your request isn't valid. Check the parameters and try your request again.

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 V2
- AWS SDK for JavaScript
- AWS SDK for PHP V3
- AWS SDK for Python
- AWS SDK for Ruby V3
CreateResourcePolicyStatement
Service: Amazon Lex Model Building V2

Adds a new resource policy statement to a bot or bot alias. If a resource policy exists, the statement is added to the current resource policy. If a policy doesn't exist, a new policy is created.

You can't create a resource policy statement that allows cross-account access.

Request Syntax

```
POST /policy/resourceArn/statements/?expectedRevisionId=expectedRevisionId HTTP/1.1
Content-type: application/json

{
    "action": [ "string" ],
    "condition": {
        "string": {
            "string": "string"
        }
    },
    "effect": "string",
    "principal": [
        { "arn": "string", "service": "string" }
    ],
    "statementId": "string"
}
```

URI Request Parameters

The request uses the following URI parameters.

**expectedRevisionId (p. 361)**

The identifier of the revision of the policy to edit. If this revision ID doesn't match the current revision ID, Amazon Lex throws an exception.

If you don't specify a revision, Amazon Lex overwrites the contents of the policy with the new values.


Pattern: ^[0-9]+$  

**resourceArn (p. 361)**

The Amazon Resource Name (ARN) of the bot or bot alias that the resource policy is attached to.

Length Constraints: Minimum length of 1. Maximum length of 1011.

Required: Yes

Request Body

The request accepts the following data in JSON format.

**action (p. 361)**

The Amazon Lex action that this policy either allows or denies. The action must apply to the resource type of the specified ARN. For more information, see Actions, resources, and condition keys for Amazon Lex V2.
**condition** (p. 361)

 Specifies a condition when the policy is in effect. If the principal of the policy is a service principal, you must provide two condition blocks, one with a SourceAccount global condition key and one with a SourceArn global condition key.

 For more information, see IAM JSON policy elements: Condition.

 Type: String to string to string map

 Map Entries: Minimum number of 0 items. Maximum number of 10 items.

 Key Length Constraints: Minimum length of 1.

 Map Entries: Minimum number of 0 items. Maximum number of 10 items.

 Key Length Constraints: Minimum length of 1. Maximum length of 1024.


 Required: No

**effect** (p. 361)

 Determines whether the statement allows or denies access to the resource.

 Type: String

 Valid Values: Allow | Deny

 Required: Yes

**principal** (p. 361)

 An IAM principal, such as an IAM users, IAM roles, or AWS services that is allowed or denied access to a resource. For more information, see AWS JSON policy elements: Principal.

 Type: Array of Principal (p. 743) objects

 Required: Yes

**statementId** (p. 361)

 The name of the statement. The ID is the same as the Sid IAM property. The statement name must be unique within the policy. For more information, see IAM JSON policy elements: Sid.

 Type: String

 Length Constraints: Minimum length of 1. Maximum length of 100.

 Pattern: ^([0-9a-zA-Z][_\-]?)+$

 Required: Yes

**Response Syntax**

```
HTTP/1.1 200
```
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.

**resourceArn (p. 362)**

The Amazon Resource Name (ARN) of the bot or bot alias that the resource policy is attached to.

Type: String

Length Constraints: Minimum length of 1. Maximum length of 1011.

**revisionId (p. 362)**

The current revision of the resource policy. Use the revision ID to make sure that you are updating the most current version of a resource policy when you add a policy statement to a resource, delete a resource, or update a resource.

Type: String


Pattern: ^[0-9]+$

Errors

For information about the errors that are common to all actions, see Common Errors (p. 821).

**ConflictException**

The action that you tried to perform couldn't be completed because the resource is in a conflicting state. For example, deleting a bot that is in the CREATING state. Try your request again.

HTTP Status Code: 409

**InternalServerException**

The service encountered an unexpected condition. Try your request again.

HTTP Status Code: 500

**PreconditionFailedException**

Your request couldn't be completed because one or more request fields aren't valid. Check the fields in your request and try again.

HTTP Status Code: 412

**ResourceNotFoundException**

You asked to describe a resource that doesn't exist. Check the resource that you are requesting and try again.

HTTP Status Code: 404
ServiceQuotaExceededException

You have reached a quota for your bot.

HTTP Status Code: 402

ThrottlingException

Your request rate is too high. Reduce the frequency of requests.

HTTP Status Code: 429

ValidationException

One of the input parameters in your request isn’t valid. Check the parameters and try your request again.

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 V2
- AWS SDK for JavaScript
- AWS SDK for PHP V3
- AWS SDK for Python
- AWS SDK for Ruby V3
CreateSlot
Service: Amazon Lex Model Building V2

Creates a slot in an intent. A slot is a variable needed to fulfill an intent. For example, an OrderPizza intent might need slots for size, crust, and number of pizzas. For each slot, you define one or more utterances that Amazon Lex uses to elicit a response from the user.

Request Syntax

```
PUT /bots/botId/botversions/botVersion/botlocales/localeId/intents/intentId/slots/ HTTP/1.1
Content-type: application/json

{
    "description": "string",
    "multipleValuesSetting": {
        "allowMultipleValues": boolean
    },
    "obfuscationSetting": {
        "obfuscationSettingType": "string"
    },
    "slotName": "string",
    "slotTypeId": "string",
    "valueElicitationSetting": {
        "defaultValueSpecification": {
            "defaultValueList": [
                "defaultValue": "string"
            ]
        },
        "promptSpecification": {
            "allowInterrupt": boolean,
            "maxRetries": number,
            "messageGroups": [
                "message": {
                    "customPayload": {
                        "value": "string"
                    },
                    "imageResponseCard": {
                        "buttons": [
                            "text": "string",
                            "value": "string"
                        ]
                    },
                    "imageUrl": "string",
                    "subtitle": "string",
                    "title": "string"
                },
                "plainTextMessage": {
                    "value": "string"
                },
                "ssmlMessage": {
                    "value": "string"
                }
            ]
        }
    },
    "variations": [
        "customPayload": {
            "value": "string"
        },
        "imageResponseCard": {
            "buttons": [
                "text": "string",
                "value": "string"
            ]
        }
    ]
}
```
{  
  "text": "string",
  "value": "string"
}
],
"imageUrl": "string",
"subtitle": "string",
"title": "string"
},
"plainTextMessage": {
  "value": "string"
},
"ssmlMessage": {
  "value": "string"
}
}
],
"sampleUtterances": [
  {
    "utterance": "string"
  }
],
"slotConstraint": "string",
"waitAndContinueSpecification": {
  "active": boolean,
  "continueResponse": {
    "allowInterrupt": boolean,
    "messageGroups": [
      {
        "message": {
          "customPayload": {
            "value": "string"
          },
          "imageResponseCard": {
            "buttons": [
              {
                "text": "string",
                "value": "string"
              }
            ],
            "imageUrl": "string",
            "subtitle": "string",
            "title": "string"
          },
          "plainTextMessage": {
            "value": "string"
          },
          "ssmlMessage": {
            "value": "string"
          }
        },
        "variations": [
          {
            "customPayload": {
              "value": "string"
            },
            "imageResponseCard": {
              "buttons": [
                {
                  "text": "string",
                  "value": "string"
                }
              ],
              "imageUrl": "string",
              "subtitle": "string",
              "title": "string"
            },
            "plainTextMessage": {
              "value": "string"
            },
            "ssmlMessage": {
              "value": "string"
            }
          }
        ]
      }
    ]
  }
}
"imageUrl": "string",
"subtitle": "string",
"title": "string"
},
"plainTextMessage": {
"value": "string"
},
"ssmlMessage": {
"value": "string"
}
]
}
,"stillWaitingResponse": {
"allowInterrupt": boolean,
"frequencyInSeconds": number,
"messageGroups": [
{
"message": {
"customPayload": {
"value": "string"
},
"imageResponseCard": {
"buttons": [
{
"text": "string",
"value": "string"
}
],
"imageUrl": "string",
"subtitle": "string",
"title": "string"
},
"plainTextMessage": {
"value": "string"
},
"ssmlMessage": {
"value": "string"
}
},
"variations": [
{
"customPayload": {
"value": "string"
},
"imageResponseCard": {
"buttons": [
{
"text": "string",
"value": "string"
}
],
"imageUrl": "string",
"subtitle": "string",
"title": "string"
},
"plainTextMessage": {
"value": "string"
},
"ssmlMessage": {
"value": "string"
}
}]
}
URI Request Parameters

The request uses the following URI parameters.
botId (p. 365)
The identifier of the bot associated with the slot.
Length Constraints: Fixed length of 10.
Pattern: ^[0-9a-zA-Z]+$
Required: Yes

botVersion (p. 365)
The version of the bot associated with the slot.
Length Constraints: Fixed length of 5.
Pattern: ^DRAFT$
Required: Yes

intentId (p. 365)
The identifier of the intent that contains the slot.
Length Constraints: Fixed length of 10.
Pattern: ^[0-9a-zA-Z]+$
Required: Yes

localeId (p. 365)
The identifier of the language and locale that the slot will be used in. The string must match one of the supported locales. All of the bots, intents, slot types used by the slot must have the same locale. For more information, see Supported languages.
Required: Yes

**Request Body**
The request accepts the following data in JSON format.

description (p. 365)
A description of the slot. Use this to help identify the slot in lists.
Type: String
Length Constraints: Minimum length of 0. Maximum length of 200.
Required: No

multipleValuesSetting (p. 365)
Indicates whether the slot returns multiple values in one response. Multi-value slots are only available in the en-US locale. If you set this value to true in any other locale, Amazon Lex throws a ValidationException.

If the multipleValuesSetting is not set, the default value is false.

Type: MultipleValuesSetting (p. 737) object
Required: No
**obfuscationSetting (p. 365)**

Determines how slot values are used in Amazon CloudWatch logs. If the value of the `obfuscationSetting` parameter is `DefaultObfuscation`, slot values are obfuscated in the log output. If the value is `None`, the actual value is present in the log output.

The default is to obfuscate values in the CloudWatch logs.

Type: `ObfuscationSetting (p. 738)` object

Required: No

**slotName (p. 365)**

The name of the slot. Slot names must be unique within the bot that contains the slot.

Type: String

Length Constraints: Minimum length of 1. Maximum length of 100.

Pattern: `^[0-9a-zA-Z][_-]?]+$`

Required: Yes

**slotTypeId (p. 365)**

The unique identifier for the slot type associated with this slot. The slot type determines the values that can be entered into the slot.

Type: String


Pattern: `^((AMAZON\.)[a-zA-Z_]\+\?|[0-9a-zA-Z])+$`

Required: Yes

**valueElicitationSetting (p. 365)**

Specifies prompts that Amazon Lex sends to the user to elicit a response that provides the value for the slot.

Type: `SlotValueElicitationSetting (p. 767)` object

Required: Yes

**Response Syntax**

```json
HTTP/1.1 200
Content-type: application/json

{
    "botId": "string",
    "botVersion": "string",
    "creationDateTime": number,
    "description": "string",
    "intentId": "string",
    "localeId": "string",
    "multipleValuesSetting": {
        "allowMultipleValues": boolean
    },
    "obfuscationSetting": {
        "obfuscationSettingType": "string"
    }
}
```
{},
"slotId": "string",
"slotName": "string",
"slotTypeId": "string",
"valueElicitationSetting": {
  "defaultValueSpecification": {
    "defaultValueList": [
      {
        "defaultValue": "string"
      }
    ]
  },
  "promptSpecification": {
    "allowInterrupt": boolean,
    "maxRetries": number,
    "messageGroups": [
      {
        "message": {
          "customPayload": {
            "value": "string"
          },
          "imageResponseCard": {
            "buttons": [
              {
                "text": "string",
                "value": "string"
              }
            ],
            "imageUrl": "string",
            "subtitle": "string",
            "title": "string"
          },
          "plainTextMessage": {
            "value": "string"
          },
          "ssmlMessage": {
            "value": "string"
          }
        },
        "variations": [
          {
            "customPayload": {
              "value": "string"
            },
            "imageResponseCard": {
              "buttons": [
                {
                  "text": "string",
                  "value": "string"
                }
              ],
              "imageUrl": "string",
              "subtitle": "string",
              "title": "string"
            },
            "plainTextMessage": {
              "value": "string"
            },
            "ssmlMessage": {
              "value": "string"
            }
          }
        ]
      }
    ]
  }
}}
"sampleUtterances": [
  {
    "utterance": "string"
  }
],
"slotConstraint": "string",
"waitForAndContinueSpecification": {
  "active": boolean,
  "continueResponse": {
    "allowInterrupt": boolean,
    "messageGroups": [
      {
        "message": {
          "customPayload": {
            "value": "string"
          },
          "imageResponseCard": {
            "buttons": [
              {
                "text": "string",
                "value": "string"
              }
            ],
            "imageUrl": "string",
            "subtitle": "string",
            "title": "string"
          },
          "plainTextMessage": {
            "value": "string"
          },
          "ssmlMessage": {
            "value": "string"
          }
        }
      }
    ]
  }
},
"stillWaitingResponse": {
  "allowInterrupt": boolean,
  "frequencyInSeconds": number,
  "messageGroups": [
    {
    ...
  }
]
"message": { 
  "customPayload": { 
    "value": "string"
  },
  "imageResponseCard": { 
    "buttons": [ 
      { 
        "text": "string",
        "value": "string"
      }
    ],
    "imageUrl": "string",
    "subtitle": "string",
    "title": "string"
  },
  "plainTextMessage": { 
    "value": "string"
  },
  "ssmlMessage": { 
    "value": "string"
  }
},
"variations": [ 
  { 
    "customPayload": { 
      "value": "string"
    },
    "imageResponseCard": { 
      "buttons": [ 
        { 
          "text": "string",
          "value": "string"
        }
      ],
      "imageUrl": "string",
      "subtitle": "string",
      "title": "string"
    },
    "plainTextMessage": { 
      "value": "string"
    },
    "ssmlMessage": { 
      "value": "string"
    }
  }
],
"timeoutInSeconds": number
},
"waitingResponse": { 
  "allowInterrupt": boolean, 
  "messageGroups": [ 
    { 
      "message": { 
        "customPayload": { 
          "value": "string"
        },
        "imageResponseCard": { 
          "buttons": [ 
            { 
              "text": "string",
              "value": "string"
            }
          ],
          "imageUrl": "string",
        },
        "plainTextMessage": { 
          "value": "string"
        },
        "ssmlMessage": { 
          "value": "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.

**botId (p. 370)**

The unique identifier of the bot associated with the slot.

Type: String

Length Constraints: Fixed length of 10.

Pattern: `^[0-9a-zA-Z]+$`

**botVersion (p. 370)**

The version of the bot associated with the slot.

Type: String

Length Constraints: Fixed length of 5.
Pattern: ^DRAFT$  

**creationDateTime (p. 370)**  
The timestamp of the date and time that the slot was created.  
Type: Timestamp  

**description (p. 370)**  
The description associated with the slot.  
Type: String  
Length Constraints: Minimum length of 0. Maximum length of 200.  

**intentId (p. 370)**  
The unique identifier of the intent associated with the slot.  
Type: String  
Length Constraints: Fixed length of 10.  
Pattern: ^[0-9a-zA-Z]+$  

**localeId (p. 370)**  
The language and local specified for the slot.  
Type: String  

**multipleValuesSetting (p. 370)**  
Indicates whether the slot returns multiple values in one response.  
Type: `MultipleValuesSetting (p. 737)` object  

**obfuscationSetting (p. 370)**  
Indicates whether the slot is configured to obfuscate values in Amazon CloudWatch logs.  
Type: `ObfuscationSetting (p. 738)` object  

**slotId (p. 370)**  
The unique identifier associated with the slot. Use this to identify the slot when you update or delete it.  
Type: String  
Length Constraints: Fixed length of 10.  
Pattern: ^[0-9a-zA-Z]+$  

**slotName (p. 370)**  
The name specified for the slot.  
Type: String  

Length Constraints: Minimum length of 1. Maximum length of 100.  
Pattern: ^([0-9a-zA-Z][_-]?)+$  

**slotTypeId (p. 370)**  
The unique identifier of the slot type associated with this slot.
Type: String


Pattern: ^((AMAZON\.|)[a-zA-Z_]+?|\[0-9a-zA-Z\]+)$

valueElicitationSetting (p. 370)

The value elicitation settings specified for the slot.

Type: SlotValueElicitationSetting (p. 767) object

Errors

For information about the errors that are common to all actions, see Common Errors (p. 821).

ConflictException

The action that you tried to perform couldn't be completed because the resource is in a conflicting state. For example, deleting a bot that is in the CREATING state. Try your request again.

HTTP Status Code: 409

InternalServerException

The service encountered an unexpected condition. Try your request again.

HTTP Status Code: 500

PreconditionFailedException

Your request couldn't be completed because one or more request fields aren't valid. Check the fields in your request and try again.

HTTP Status Code: 412

ServiceQuotaExceeded Exception

You have reached a quota for your bot.

HTTP Status Code: 402

ThrottlingException

Your request rate is too high. Reduce the frequency of requests.

HTTP Status Code: 429

ValidationException

One of the input parameters in your request isn't valid. Check the parameters and try your request again.

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 V2
- AWS SDK for JavaScript
- AWS SDK for PHP V3
- AWS SDK for Python
- AWS SDK for Ruby V3
CreateSlotType
Service: Amazon Lex Model Building V2

Creates a custom slot type

To create a custom slot type, specify a name for the slot type and a set of enumeration values, the values that a slot of this type can assume.

Request Syntax

```
PUT /bots/botId/botversions/botVersion/botlocales/localeId/slottypes/ HTTP/1.1
Content-type: application/json

{
    "description": "string",
    "externalSourceSetting": {
        "grammarSlotTypeSetting": {
            "source": {
                "kmsKeyArn": "string",
                "s3BucketName": "string",
                "s3ObjectKey": "string"
            }
        }
    },
    "parentSlotTypeSignature": "string",
    "slotTypeName": "string",
    "slotTypeValues": [
        {
            "sampleValue": {
                "value": "string"
            },
            "synonyms": [
                {
                    "value": "string"
                }
            ]
        }
    ],
    "valueSelectionSetting": {
        "advancedRecognitionSetting": {
            "audioRecognitionStrategy": "string"
        },
        "regexFilter": {
            "pattern": "string"
        },
        "resolutionStrategy": "string"
    }
}
```

URI Request Parameters

The request uses the following URI parameters.

**botId (p. 378)**

The identifier of the bot associated with this slot type.

Length Constraints: Fixed length of 10.

Pattern: ^[0-9a-zA-Z]+$

Required: Yes
**botVersion (p. 378)**

The identifier of the bot version associated with this slot type.

Length Constraints: Fixed length of 5.

Pattern: ^DRAFT$

Required: Yes

**localeId (p. 378)**

The identifier of the language and locale that the slot type will be used in. The string must match one of the supported locales. All of the bots, intents, and slots used by the slot type must have the same locale. For more information, see Supported languages.

Required: Yes

**Request Body**

The request accepts the following data in JSON format.

**description (p. 378)**

A description of the slot type. Use the description to help identify the slot type in lists.

Type: String

Length Constraints: Minimum length of 0. Maximum length of 200.

Required: No

**externalSourceSetting (p. 378)**

Sets the type of external information used to create the slot type.

Type: `ExternalSourceSetting (p. 710)` object

Required: No

**parentSlotTypeSignature (p. 378)**

The built-in slot type used as a parent of this slot type. When you define a parent slot type, the new slot type has the configuration of the parent slot type.

Only `AMAZON.AlphaNumeric` is supported.

Type: String

Required: No

**slotTypeName (p. 378)**

The name for the slot. A slot type name must be unique within the account.

Type: String

Length Constraints: Minimum length of 1. Maximum length of 100.

Pattern: ^([0-9a-zA-Z][-_]?)+$

Required: Yes
slotTypeValues (p. 378)

A list of SlotTypeValue objects that defines the values that the slot type can take. Each value can have a list of synonyms, additional values that help train the machine learning model about the values that it resolves for a slot.

Type: Array of SlotTypeValue (p. 766) objects

Array Members: Minimum number of 1 item. Maximum number of 10000 items.

Required: No

valueSelectionSetting (p. 378)

Determines the strategy that Amazon Lex uses to select a value from the list of possible values. The field can be set to one of the following values:

- **OriginalValue** - Returns the value entered by the user, if the user value is similar to the slot value.
- **TopResolution** - If there is a resolution list for the slot, return the first value in the resolution list. If there is no resolution list, return null.

If you don't specify the valueSelectionSetting parameter, the default is OriginalValue.

Type: SlotValueSelectionSetting (p. 770) object

Required: No

Response Syntax

```json
HTTP/1.1 200
Content-type: application/json

{
    "botId": "string",
    "botVersion": "string",
    "creationDateTime": number,
    "description": "string",
    "externalSourceSetting": {
        "grammarSlotTypeSetting": {
            "source": {
                "kmsKeyArn": "string",
                "s3BucketName": "string",
                "s3ObjectKey": "string"
            }
        }
    },
    "localeId": "string",
    "parentSlotTypeSignature": "string",
    "slotTypeId": "string",
    "slotTypeName": "string",
    "slotTypeValues": [
        {
            "sampleValue": {
                "value": "string"
            },
            "synonyms": [
                {
                    "value": "string"
                }
            ]
        }
    ]
}
```
"valueSelectionSetting": {
    "advancedRecognitionSetting": {
        "audioRecognitionStrategy": "string"
    },
    "regexFilter": {
        "pattern": "string"
    },
    "resolutionStrategy": "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.

**botId (p. 380)**

The identifier for the bot associated with the slot type.

Type: String

Length Constraints: Fixed length of 10.

Pattern: ^[0-9a-zA-Z]+$  

**botVersion (p. 380)**

The version of the bot associated with the slot type.

Type: String

Length Constraints: Fixed length of 5.

Pattern: ^DRAFT$  

**creationDateTime (p. 380)**

A timestamp of the date and time that the slot type was created.

Type: Timestamp

**description (p. 380)**

The description specified for the slot type.

Type: String

Length Constraints: Minimum length of 0. Maximum length of 200.

**externalSourceSetting (p. 380)**

The type of external information used to create the slot type.

Type: ExternalSourceSetting (p. 710) object

**localeId (p. 380)**

The specified language and local specified for the slot type.

Type: String

**parentSlotTypeSignature (p. 380)**

The signature of the base slot type specified for the slot type.
Type: String

**slotTypeId (p. 380)**

The unique identifier assigned to the slot type. Use this to identify the slot type in the `UpdateSlotType` and `DeleteSlotType` operations.

Type: String

Length Constraints: Fixed length of 10.

Pattern: `^[0-9a-zA-Z]+$`

**slotTypeName (p. 380)**

The name specified for the slot type.

Type: String

Length Constraints: Minimum length of 1. Maximum length of 100.

Pattern: `^[0-9a-zA-Z][\-_]?]+$`

**slotTypeValues (p. 380)**

The list of values that the slot type can assume.

Type: Array of `SlotTypeValue (p. 766)` objects

Array Members: Minimum number of 1 item. Maximum number of 10000 items.

**valueSelectionSetting (p. 380)**

The strategy that Amazon Lex uses to select a value from the list of possible values.

Type: `SlotValueSelectionSetting (p. 770)` object

---

**Errors**

For information about the errors that are common to all actions, see `Common Errors (p. 821)`.

**ConflictException**

The action that you tried to perform couldn't be completed because the resource is in a conflicting state. For example, deleting a bot that is in the CREATING state. Try your request again.

HTTP Status Code: 409

**InternalServerException**

The service encountered an unexpected condition. Try your request again.

HTTP Status Code: 500

**PreconditionFailedException**

Your request couldn't be completed because one or more request fields aren't valid. Check the fields in your request and try again.

HTTP Status Code: 412

**ServiceQuotaExceededException**

You have reached a quota for your bot.

HTTP Status Code: 402
ThrottlingException

Your request rate is too high. Reduce the frequency of requests.

HTTP Status Code: 429

ValidationException

One of the input parameters in your request isn't valid. Check the parameters and try your request again.

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 V2
- AWS SDK for JavaScript
- AWS SDK for PHP V3
- AWS SDK for Python
- AWS SDK for Ruby V3
CreateUploadUrl
Service: Amazon Lex Model Building V2

Gets a pre-signed S3 write URL that you use to upload the zip archive when importing a bot or a bot locale.

Request Syntax

POST /createuploadurl/ HTTP/1.1

URI Request Parameters

The request does not use any URI parameters.

Request Body

The request does not have a request body.

Response Syntax

HTTP/1.1 200
Content-type: application/json

{  
   "importId": "string",
   "uploadUrl": "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.

importId (p. 384)

An identifier for a unique import job. Use it when you call the StartImport operation.

Type: String

Length Constraints: Fixed length of 10.

Pattern: ^[0-9a-zA-Z]+$

uploadUrl (p. 384)

A pre-signed S3 write URL. Upload the zip archive file that contains the definition of your bot or bot locale.

Type: String


Errors

For information about the errors that are common to all actions, see Common Errors (p. 821).
ConflictException

The action that you tried to perform couldn't be completed because the resource is in a conflicting state. For example, deleting a bot that is in the CREATING state. Try your request again.

HTTP Status Code: 409

InternalServerException

The service encountered an unexpected condition. Try your request again.

HTTP Status Code: 500

ResourceNotFoundException

You asked to describe a resource that doesn't exist. Check the resource that you are requesting and try again.

HTTP Status Code: 404

ThrottlingException

Your request rate is too high. Reduce the frequency of requests.

HTTP Status Code: 429

ValidationException

One of the input parameters in your request isn't valid. Check the parameters and try your request again.

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 V2
- AWS SDK for JavaScript
- AWS SDK for PHP V3
- AWS SDK for Python
- AWS SDK for Ruby V3
DeleteBot
Service: Amazon Lex Model Building V2

Deletes all versions of a bot, including the Draft version. To delete a specific version, use the DeleteBotVersion operation.

When you delete a bot, all of the resources contained in the bot are also deleted. Deleting a bot removes all locales, intents, slot, and slot types defined for the bot.

If a bot has an alias, the DeleteBot operation returns a ResourceInUseException exception. If you want to delete the bot and the alias, set the skipResourceInUseCheck parameter to true.

Request Syntax

```
DELETE /bots/botId/?skipResourceInUseCheck=skipResourceInUseCheck HTTP/1.1
```

URI Request Parameters

The request uses the following URI parameters.

**botId** (p. 386)

The identifier of the bot to delete.

Length Constraints: Fixed length of 10.

Pattern: `^[0-9a-zA-Z]+$`

Required: Yes

**skipResourceInUseCheck** (p. 386)

When true, Amazon Lex doesn't check to see if another resource, such as an alias, is using the bot before it is deleted.

Request Body

The request does not have a request body.

Response Syntax

```
HTTP/1.1 202
Content-type: application/json

{  
   "botId": "string",
   "botStatus": "string"
}
```

Response Elements

If the action is successful, the service sends back an HTTP 202 response.

The following data is returned in JSON format by the service.

**botId** (p. 386)

The unique identifier of the bot that Amazon Lex is deleting.
Type: String

Length Constraints: Fixed length of 10.

Pattern: ^[0-9a-zA-Z]+$  

**botStatus (p. 386)**

The current status of the bot. The status is *Deleting* while the bot and its associated resources are being deleted.

Type: String

Valid Values: Creating | Available | Inactive | Deleting | Failed | Versioning | Importing

**Errors**

For information about the errors that are common to all actions, see [Common Errors (p. 821)](#).

**ConflictException**

The action that you tried to perform couldn't be completed because the resource is in a conflicting state. For example, deleting a bot that is in the CREATING state. Try your request again.

HTTP Status Code: 409

**InternalServerException**

The service encountered an unexpected condition. Try your request again.

HTTP Status Code: 500

**PreconditionFailedException**

Your request couldn't be completed because one or more request fields aren't valid. Check the fields in your request and try again.

HTTP Status Code: 412

**ServiceQuotaExceededException**

You have reached a quota for your bot.

HTTP Status Code: 402

**ThrottlingException**

Your request rate is too high. Reduce the frequency of requests.

HTTP Status Code: 429

**ValidationException**

One of the input parameters in your request isn't valid. Check the parameters and try your request again.

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 V2
- AWS SDK for JavaScript
- AWS SDK for PHP V3
- AWS SDK for Python
- AWS SDK for Ruby V3
DeleteBotAlias
Service: Amazon Lex Model Building V2

Deletes the specified bot alias.

Request Syntax

DELETE /bots/botId/botAliases/botAliasId/?skipResourceInUseCheck=skipResourceInUseCheck
HTTP/1.1

URI Request Parameters

The request uses the following URI parameters.

botAliasId (p. 389)

The unique identifier of the bot alias to delete.
Length Constraints: Fixed length of 10.
Pattern: ^\bTSTALIASID\b\|[0-9a-zA-Z]+$
Required: Yes

botId (p. 389)

The unique identifier of the bot associated with the alias to delete.
Length Constraints: Fixed length of 10.
Pattern: ^[0-9a-zA-Z]+$
Required: Yes

skipResourceInUseCheck (p. 389)

When this parameter is true, Amazon Lex doesn't check to see if any other resource is using the alias before it is deleted.

Request Body

The request does not have a request body.

Response Syntax

HTTP/1.1 202
Content-type: application/json

{
  "botAliasId": "string",
  "botAliasStatus": "string",
  "botId": "string"
}

Response Elements

If the action is successful, the service sends back an HTTP 202 response.

The following data is returned in JSON format by the service.
botAliasId (p. 389)

The unique identifier of the bot alias to delete.

Type: String

Length Constraints: Fixed length of 10.

Pattern: ^(\bTSTALIASID\b|[0-9a-zA-Z]+)$

botAliasStatus (p. 389)

The current status of the alias. The status is Deleting while the alias is in the process of being deleted. Once the alias is deleted, it will no longer appear in the list of aliases returned by the ListBotAliases operation.

Type: String

Valid Values: Creating | Available | Deleting | Failed

botId (p. 389)

The unique identifier of the bot that contains the alias to delete.

Type: String

Length Constraints: Fixed length of 10.

Pattern: ^[0-9a-zA-Z]+$

Errors

For information about the errors that are common to all actions, see Common Errors (p. 821).

ConflictException

The action that you tried to perform couldn't be completed because the resource is in a conflicting state. For example, deleting a bot that is in the CREATING state. Try your request again.

HTTP Status Code: 409

InternalServerException

The service encountered an unexpected condition. Try your request again.

HTTP Status Code: 500

PreconditionFailedException

Your request couldn't be completed because one or more request fields aren't valid. Check the fields in your request and try again.

HTTP Status Code: 412

ServiceQuotaExceededException

You have reached a quota for your bot.

HTTP Status Code: 402

ThrottlingException

Your request rate is too high. Reduce the frequency of requests.

HTTP Status Code: 429
ValidationException

One of the input parameters in your request isn't valid. Check the parameters and try your request again.

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 V2
- AWS SDK for JavaScript
- AWS SDK for PHP V3
- AWS SDK for Python
- AWS SDK for Ruby V3
DeleteBotLocale
Service: Amazon Lex Model Building V2

Removes a locale from a bot.

When you delete a locale, all intents, slots, and slot types defined for the locale are also deleted.

Request Syntax

DELETE /bots/botId/botversions/botVersion/botlocales/localeId/ HTTP/1.1

URI Request Parameters

The request uses the following URI parameters.

botId (p. 392)

The unique identifier of the bot that contains the locale.

Length Constraints: Fixed length of 10.

Pattern: ^[0-9a-zA-Z]+$

Required: Yes

botVersion (p. 392)

The version of the bot that contains the locale.

Length Constraints: Fixed length of 5.

Pattern: ^DRAFT$

Required: Yes

localeId (p. 392)

The identifier of the language and locale that will be deleted. The string must match one of the supported locales. For more information, see Supported languages.

Required: Yes

Request Body

The request does not have a request body.

Response Syntax

HTTP/1.1 202
Content-type: application/json

{
  "botId": "string",
  "botLocaleStatus": "string",
  "botVersion": "string",
  "localeId": "string"
}
Response Elements

If the action is successful, the service sends back an HTTP 202 response.

The following data is returned in JSON format by the service.

**botId** *(p. 392)*

The identifier of the bot that contained the deleted locale.

Type: String

Length Constraints: Fixed length of 10.

Pattern: `^[0-9a-zA-Z]+$`

**botLocaleStatus** *(p. 392)*

The status of deleting the bot locale. The locale first enters the Deleting status. Once the locale is deleted it no longer appears in the list of locales for the bot.

Type: String

Valid Values: Creating | Building | Built | ReadyExpressTesting | Failed | Deleting | NotBuilt | Importing | Processing

**botVersion** *(p. 392)*

The version of the bot that contained the deleted locale.

Type: String

Length Constraints: Fixed length of 5.

Pattern: `^DRAFT$`

**localeId** *(p. 392)*

The language and locale of the deleted locale.

Type: String

Errors

For information about the errors that are common to all actions, see Common Errors *(p. 821).*

**ConflictException**

The action that you tried to perform couldn't be completed because the resource is in a conflicting state. For example, deleting a bot that is in the CREATING state. Try your request again.

HTTP Status Code: 409

**InternalServerException**

The service encountered an unexpected condition. Try your request again.

HTTP Status Code: 500

**PreconditionFailedException**

Your request couldn't be completed because one or more request fields aren't valid. Check the fields in your request and try again.
HTTP Status Code: 412

**ServiceQuotaExceededException**

You have reached a quota for your bot.

HTTP Status Code: 402

**ThrottlingException**

Your request rate is too high. Reduce the frequency of requests.

HTTP Status Code: 429

**ValidationException**

One of the input parameters in your request isn't valid. Check the parameters and try your request again.

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 V2
- AWS SDK for JavaScript
- AWS SDK for PHP V3
- AWS SDK for Python
- AWS SDK for Ruby V3
DeleteBotVersion
Service: Amazon Lex Model Building V2

Deletes a specific version of a bot. To delete all version of a bot, use the DeleteBot operation.

Request Syntax

```
DELETE /bots/botId/botversions/botVersion/?skipResourceInUseCheck=skipResourceInUseCheck
HTTP/1.1
```

URI Request Parameters

The request uses the following URI parameters.

**botId (p. 395)**

The identifier of the bot that contains the version.

Length Constraints: Fixed length of 10.

Pattern: ^[0-9a-zA-Z]+$

Required: Yes

**botVersion (p. 395)**

The version of the bot to delete.


Pattern: ^[0-9]+$

Required: Yes

**skipResourceInUseCheck (p. 395)**

By default, the DeleteBotVersion operations throws a ResourceInUseException exception if you try to delete a bot version that has an alias pointing at it. Set the skipResourceInUseCheck parameter to true to skip this check and remove the version even if an alias points to it.

Request Body

The request does not have a request body.

Response Syntax

```
HTTP/1.1 202
Content-type: application/json

{
  "botId": "string",
  "botStatus": "string",
  "botVersion": "string"
}
```

Response Elements

If the action is successful, the service sends back an HTTP 202 response.
The following data is returned in JSON format by the service.

**botId (p. 395)**

The identifier of the bot that is being deleted.

Type: String

Length Constraints: Fixed length of 10.

Pattern: `^[0-9a-zA-Z]+$`

**botStatus (p. 395)**

The current status of the bot.

Type: String

Valid Values: Creating | Available | Inactive | Deleting | Failed | Versioning | Importing

**botVersion (p. 395)**

The version of the bot that is being deleted.

Type: String


Pattern: `^[0-9]+$`

**Errors**

For information about the errors that are common to all actions, see [Common Errors (p. 821)](#).

**ConflictException**

The action that you tried to perform couldn't be completed because the resource is in a conflicting state. For example, deleting a bot that is in the CREATING state. Try your request again.

HTTP Status Code: 409

**InternalServerException**

The service encountered an unexpected condition. Try your request again.

HTTP Status Code: 500

**PreconditionFailedException**

Your request couldn't be completed because one or more request fields aren't valid. Check the fields in your request and try again.

HTTP Status Code: 412

**ServiceQuotaExceededException**

You have reached a quota for your bot.

HTTP Status Code: 402

**ThrottlingException**

Your request rate is too high. Reduce the frequency of requests.
HTTP Status Code: 429

ValidationException

One of the input parameters in your request isn't valid. Check the parameters and try your request again.

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 V2
- AWS SDK for JavaScript
- AWS SDK for PHP V3
- AWS SDK for Python
- AWS SDK for Ruby V3
DeleteCustomVocabulary
Service: Amazon Lex Model Building V2

Removes a custom vocabulary from the specified locale in the specified bot.

Request Syntax

```
DELETE /bots/botId/botversions/botVersion/botlocales/localeId/customvocabulary HTTP/1.1
```

URI Request Parameters

The request uses the following URI parameters.

**botId (p. 398)**

The unique identifier of the bot to remove the custom vocabulary from.

- Pattern: `^[0-9a-zA-Z]+$`
- Required: Yes

**botVersion (p. 398)**

The version of the bot to remove the custom vocabulary from.

- Length Constraints: Fixed length of 5.
- Pattern: `^DRAFT$`
- Required: Yes

**localeId (p. 398)**

The locale identifier for the locale that contains the custom vocabulary to remove.

- Required: Yes

Request Body

The request does not have a request body.

Response Syntax

```
HTTP/1.1 202
Content-type: application/json

{
    "botId": "string",
    "botVersion": "string",
    "customVocabularyStatus": "string",
    "localeId": "string"
}
```

Response Elements

If the action is successful, the service sends back an HTTP 202 response.
The following data is returned in JSON format by the service.

**botId (p. 398)**

The identifier of the bot that the custom vocabulary was removed from.

Type: String

Length Constraints: Fixed length of 10.

Pattern: ^[0-9a-zA-Z]+$  

**botVersion (p. 398)**

The version of the bot that the custom vocabulary was removed from.

Type: String

Length Constraints: Fixed length of 5.

Pattern: ^DRAFT$  

**customVocabularyStatus (p. 398)**

The status of removing the custom vocabulary.

Type: String

Valid Values: Ready | Deleting | Exporting | Importing | Creating  

**localeId (p. 398)**

The locale identifier for the locale that the custom vocabulary was removed from.

Type: String

Errors

For information about the errors that are common to all actions, see Common Errors (p. 821).

**ConflictException**

The action that you tried to perform couldn't be completed because the resource is in a conflicting state. For example, deleting a bot that is in the CREATING state. Try your request again.

HTTP Status Code: 409

**InternalServerException**

The service encountered an unexpected condition. Try your request again.

HTTP Status Code: 500

**PreconditionFailedException**

Your request couldn't be completed because one or more request fields aren't valid. Check the fields in your request and try again.

HTTP Status Code: 412

**ServiceQuotaExceededException**

You have reached a quota for your bot.

HTTP Status Code: 402
ThrottlingException

Your request rate is too high. Reduce the frequency of requests.

HTTP Status Code: 429

ValidationException

One of the input parameters in your request isn't valid. Check the parameters and try your request again.

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 V2
- AWS SDK for JavaScript
- AWS SDK for PHP V3
- AWS SDK for Python
- AWS SDK for Ruby V3
DeleteExport
Service: Amazon Lex Model Building V2

Removes a previous export and the associated files stored in an S3 bucket.

Request Syntax

DELETE /exports/exportId/ HTTP/1.1

URI Request Parameters

The request uses the following URI parameters.

exportId (p. 401)

The unique identifier of the export to delete.

Length Constraints: Fixed length of 10.

Pattern: ^[0-9a-zA-Z]+$

Required: Yes

Request Body

The request does not have a request body.

Response Syntax

HTTP/1.1 202
Content-type: application/json

{  "exportId": "string",
   "exportStatus": "string"
}

Response Elements

If the action is successful, the service sends back an HTTP 202 response.

The following data is returned in JSON format by the service.

exportId (p. 401)

The unique identifier of the deleted export.

Type: String

Length Constraints: Fixed length of 10.

Pattern: ^[0-9a-zA-Z]+$

exportStatus (p. 401)

The current status of the deletion. When the deletion is complete, the export will no longer be returned by the ListExports operation and calls to the DescribeExport operation with the export identifier will fail.
Type: String

Valid Values: InProgress | Completed | Failed | Deleting

Errors

For information about the errors that are common to all actions, see Common Errors (p. 821).

**InternalServerException**

The service encountered an unexpected condition. Try your request again.

HTTP Status Code: 500

**PreconditionFailedException**

Your request couldn't be completed because one or more request fields aren't valid. Check the fields in your request and try again.

HTTP Status Code: 412

**ServiceQuotaExceededException**

You have reached a quota for your bot.

HTTP Status Code: 402

**ThrottlingException**

Your request rate is too high. Reduce the frequency of requests.

HTTP Status Code: 429

**ValidationException**

One of the input parameters in your request isn't valid. Check the parameters and try your request again.

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 V2
- AWS SDK for JavaScript
- AWS SDK for PHP V3
- AWS SDK for Python
- AWS SDK for Ruby V3
DeleteImport
Service: Amazon Lex Model Building V2
Removes a previous import and the associated file stored in an S3 bucket.

Request Syntax

```
DELETE /imports/importId/ HTTP/1.1
```

URI Request Parameters

The request uses the following URI parameters.

importId (p. 403)
The unique identifier of the import to delete.
Length Constraints: Fixed length of 10.
Pattern: ^[0-9a-zA-Z]+$
Required: Yes

Request Body

The request does not have a request body.

Response Syntax

```
HTTP/1.1 202
Content-type: application/json
{
   "importId": "string",
   "importStatus": "string"
}
```

Response Elements

If the action is successful, the service sends back an HTTP 202 response.
The following data is returned in JSON format by the service.

importId (p. 403)
The unique identifier of the deleted import.
Type: String
Length Constraints: Fixed length of 10.
Pattern: ^[0-9a-zA-Z]+$

importStatus (p. 403)
The current status of the deletion. When the deletion is complete, the import will no longer be returned by the ListImports operation and calls to the DescribeImport operation with the import identifier will fail.
Type: String

Valid Values: InProgress | Completed | Failed | Deleting

Errors

For information about the errors that are common to all actions, see Common Errors (p. 821).

**InternalServerException**

The service encountered an unexpected condition. Try your request again.

HTTP Status Code: 500

**PreconditionFailedException**

Your request couldn't be completed because one or more request fields aren't valid. Check the fields in your request and try again.

HTTP Status Code: 412

**ServiceQuotaExceededException**

You have reached a quota for your bot.

HTTP Status Code: 402

**ThrottlingException**

Your request rate is too high. Reduce the frequency of requests.

HTTP Status Code: 429

**ValidationException**

One of the input parameters in your request isn't valid. Check the parameters and try your request again.

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 V2
- AWS SDK for JavaScript
- AWS SDK for PHP V3
- AWS SDK for Python
- AWS SDK for Ruby V3
DeleteIntent
Service: Amazon Lex Model Building V2

Removes the specified intent.
Deleting an intent also deletes the slots associated with the intent.

Request Syntax

```
DELETE /bots/botId/botversions/botVersion/botlocales/localeId/intents/intentId/ HTTP/1.1
```

URI Request Parameters

The request uses the following URI parameters.

**botId** *(p. 405)*

The identifier of the bot associated with the intent.
Length Constraints: Fixed length of 10.
Pattern: `^[0-9a-zA-Z]+$`
Required: Yes

**botVersion** *(p. 405)*

The version of the bot associated with the intent.
Length Constraints: Fixed length of 5.
Pattern: `^DRAFT$`
Required: Yes

**intentId** *(p. 405)*

The unique identifier of the intent to delete.
Length Constraints: Fixed length of 10.
Pattern: `^[0-9a-zA-Z]+$`
Required: Yes

**localeId** *(p. 405)*

The identifier of the language and locale where the bot will be deleted. The string must match one of the supported locales. For more information, see Supported languages.
Required: Yes

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

For information about the errors that are common to all actions, see Common Errors (p. 821).

ConflictException

The action that you tried to perform couldn't be completed because the resource is in a conflicting state. For example, deleting a bot that is in the CREATING state. Try your request again.

HTTP Status Code: 409

InternalServerException

The service encountered an unexpected condition. Try your request again.

HTTP Status Code: 500

PreconditionFailedException

Your request couldn't be completed because one or more request fields aren't valid. Check the fields in your request and try again.

HTTP Status Code: 412

ServiceQuotaExceededException

You have reached a quota for your bot.

HTTP Status Code: 402

ThrottlingException

Your request rate is too high. Reduce the frequency of requests.

HTTP Status Code: 429

ValidationException

One of the input parameters in your request isn't valid. Check the parameters and try your request again.

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 C++
- AWS SDK for Go
- AWS SDK for Java V2
- AWS SDK for JavaScript
- AWS SDK for PHP V3
- AWS SDK for Python
- AWS SDK for Ruby V3
DeleteResourcePolicy
Service: Amazon Lex Model Building V2

Removes an existing policy from a bot or bot alias. If the resource doesn't have a policy attached, Amazon Lex returns an exception.

Request Syntax

```
DELETE /policy/resourceArn/?expectedRevisionId=expectedRevisionId HTTP/1.1
```

URI Request Parameters

The request uses the following URI parameters.

- **expectedRevisionId (p. 408)**
  - The identifier of the revision to edit. If this ID doesn't match the current revision number, Amazon Lex returns an exception.
  - If you don't specify a revision ID, Amazon Lex will delete the current policy.
  - Pattern: `^[0-9]+$`

- **resourceArn (p. 408)**
  - The Amazon Resource Name (ARN) of the bot or bot alias that has the resource policy attached.
  - Required: Yes

Request Body

The request does not have a request body.

Response Syntax

```
HTTP/1.1 204
Content-type: application/json

{
  "resourceArn": "string",
  "revisionId": "string"
}
```

Response Elements

If the action is successful, the service sends back an HTTP 204 response.

The following data is returned in JSON format by the service.

- **resourceArn (p. 408)**
  - The Amazon Resource Name (ARN) of the bot or bot alias that the resource policy was deleted from.
  - Type: String
Length Constraints: Minimum length of 1. Maximum length of 1011.

**revisionId (p. 408)**

The current revision of the resource policy. Use the revision ID to make sure that you are updating the most current version of a resource policy when you add a policy statement to a resource, delete a resource, or update a resource.

Type: String


Pattern: ^\[0-9]*$ 

**Errors**

For information about the errors that are common to all actions, see Common Errors (p. 821).

**InternalServerException**

The service encountered an unexpected condition. Try your request again.

HTTP Status Code: 500

**PreconditionFailedException**

Your request couldn't be completed because one or more request fields aren't valid. Check the fields in your request and try again.

HTTP Status Code: 412

**ResourceNotFoundException**

You asked to describe a resource that doesn't exist. Check the resource that you are requesting and try again.

HTTP Status Code: 404

**ThrottlingException**

Your request rate is too high. Reduce the frequency of requests.

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 V2
- AWS SDK for JavaScript
- AWS SDK for PHP V3
- AWS SDK for Python
- AWS SDK for Ruby V3
**DeleteResourcePolicyStatement**  
Service: Amazon Lex Model Building V2  

Deletes a policy statement from a resource policy. If you delete the last statement from a policy, the policy is deleted. If you specify a statement ID that doesn't exist in the policy, or if the bot or bot alias doesn't have a policy attached, Amazon Lex returns an exception.

**Request Syntax**

DELETE /policy/{resourceArn}/statements/{statementId}/?expectedRevisionId={expectedRevisionId}
HTTP/1.1

**URI Request Parameters**

The request uses the following URI parameters.

- **expectedRevisionId** (p. 410)
  
  The identifier of the revision of the policy to delete the statement from. If this revision ID doesn't match the current revision ID, Amazon Lex throws an exception.

  If you don't specify a revision, Amazon Lex removes the current contents of the statement.


  Pattern: `^[0-9]+$`

- **resourceArn** (p. 410)
  
  The Amazon Resource Name (ARN) of the bot or bot alias that the resource policy is attached to.

  Length Constraints: Minimum length of 1. Maximum length of 1011.

  Required: Yes

- **statementId** (p. 410)
  
  The name of the statement (SID) to delete from the policy.

  Length Constraints: Minimum length of 1. Maximum length of 100.

  Pattern: `^[0-9a-zA-Z\[\_\-]+$`

  Required: Yes

**Request Body**

The request does not have a request body.

**Response Syntax**

HTTP/1.1 204
Content-type: application/json

```json
{
  "resourceArn": "string",
  "revisionId": "string"
}
```
Response Elements

If the action is successful, the service sends back an HTTP 204 response. The following data is returned in JSON format by the service.

resourceArn (p. 410)

The Amazon Resource Name (ARN) of the bot or bot alias that the resource policy statement was removed from.

Type: String

Length Constraints: Minimum length of 1. Maximum length of 1011.

revisionId (p. 410)

The current revision of the resource policy. Use the revision ID to make sure that you are updating the most current version of a resource policy when you add a policy statement to a resource, delete a resource, or update a resource.

Type: String


Pattern: ^[0-9]+$

Errors

For information about the errors that are common to all actions, see Common Errors (p. 821).

InternalServerErrorException

The service encountered an unexpected condition. Try your request again.

HTTP Status Code: 500

PreconditionFailedException

Your request couldn't be completed because one or more request fields aren't valid. Check the fields in your request and try again.

HTTP Status Code: 412

ResourceNotFoundException

You asked to describe a resource that doesn't exist. Check the resource that you are requesting and try again.

HTTP Status Code: 404

ThrottlingException

Your request rate is too high. Reduce the frequency of requests.

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 V2
• AWS SDK for JavaScript
• AWS SDK for PHP V3
• AWS SDK for Python
• AWS SDK for Ruby V3
DeleteSlot
Service: Amazon Lex Model Building V2

Deletes the specified slot from an intent.

Request Syntax

```
DELETE /bots/botId/botversions/botVersion/botlocales/localeId/intents/intentId/
slots/slotId/ HTTP/1.1
```

URI Request Parameters

The request uses the following URI parameters.

`botId (p. 413)`

The identifier of the bot associated with the slot to delete.

Length Constraints: Fixed length of 10.

Pattern: `^[0-9a-zA-Z]+$`

Required: Yes

`botVersion (p. 413)`

The version of the bot associated with the slot to delete.

Length Constraints: Fixed length of 5.

Pattern: `^[DRAFT]+$`

Required: Yes

`intentId (p. 413)`

The identifier of the intent associated with the slot.

Length Constraints: Fixed length of 10.

Pattern: `^[0-9a-zA-Z]+$`

Required: Yes

`localeId (p. 413)`

The identifier of the language and locale that the slot will be deleted from. The string must match one of the supported locales. For more information, see Supported languages.

Required: Yes

`s alotId (p. 413)`

The identifier of the slot to delete.

Length Constraints: Fixed length of 10.

Pattern: `^[0-9a-zA-Z]+$`

Required: Yes
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

For information about the errors that are common to all actions, see Common Errors (p. 821).

ConflictException

The action that you tried to perform couldn't be completed because the resource is in a conflicting state. For example, deleting a bot that is in the CREATING state. Try your request again.

HTTP Status Code: 409

InternalServerErrorException

The service encountered an unexpected condition. Try your request again.

HTTP Status Code: 500

PreconditionFailedException

Your request couldn't be completed because one or more request fields aren't valid. Check the fields in your request and try again.

HTTP Status Code: 412

ServiceQuotaExceededException

You have reached a quota for your bot.

HTTP Status Code: 402

ThrottlingException

Your request rate is too high. Reduce the frequency of requests.

HTTP Status Code: 429

ValidationException

One of the input parameters in your request isn't valid. Check the parameters and try your request again.

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 V2
• AWS SDK for JavaScript
• AWS SDK for PHP V3
• AWS SDK for Python
• AWS SDK for Ruby V3
DeleteSlotType
Service: Amazon Lex Model Building V2
Deletes a slot type from a bot locale.

If a slot is using the slot type, Amazon Lex throws a ResourceInUseException exception. To avoid the exception, set the skipResourceInUseCheck parameter to true.

Request Syntax

DELETE /bots/botId/botversions/botVersion/botlocales/localeId/slottypes/slotTypeId/?
skipResourceInUseCheck=skipResourceInUseCheck HTTP/1.1

URI Request Parameters
The request uses the following URI parameters.

botId (p. 416)
The identifier of the bot associated with the slot type.
Length Constraints: Fixed length of 10.
Pattern: ^[0-9a-zA-Z]+$
Required: Yes

botVersion (p. 416)
The version of the bot associated with the slot type.
Length Constraints: Fixed length of 5.
Pattern: ^DRAFT$
Required: Yes

localeId (p. 416)
The identifier of the language and locale that the slot type will be deleted from. The string must match one of the supported locales. For more information, see Supported languages.
Required: Yes

skipResourceInUseCheck (p. 416)
By default, the DeleteSlotType operations throws a ResourceInUseException exception if you try to delete a slot type used by a slot. Set the skipResourceInUseCheck parameter to true to skip this check and remove the slot type even if a slot uses it.

slotTypeId (p. 416)
The identifier of the slot type to delete.
Length Constraints: Fixed length of 10.
Pattern: ^[0-9a-zA-Z]+$
Required: Yes

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

For information about the errors that are common to all actions, see Common Errors (p. 821).

ConflictException

The action that you tried to perform couldn't be completed because the resource is in a conflicting state. For example, deleting a bot that is in the CREATING state. Try your request again.

HTTP Status Code: 409

InternalServerErrorException

The service encountered an unexpected condition. Try your request again.

HTTP Status Code: 500

PreconditionFailedException

Your request couldn't be completed because one or more request fields aren't valid. Check the fields in your request and try again.

HTTP Status Code: 412

ServiceQuotaExceededException

You have reached a quota for your bot.

HTTP Status Code: 402

ThrottlingException

Your request rate is too high. Reduce the frequency of requests.

HTTP Status Code: 429

ValidationException

One of the input parameters in your request isn't valid. Check the parameters and try your request again.

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 V2
- AWS SDK for JavaScript
- AWS SDK for PHP V3
- AWS SDK for Python
- AWS SDK for Ruby V3
DeleteUtterances
Service: Amazon Lex Model Building V2

Deletes stored utterances.

Amazon Lex stores the utterances that users send to your bot. Utterances are stored for 15 days for use with the ListAggregatedUtterances operation, and then stored indefinitely for use in improving the ability of your bot to respond to user input.

Use the DeleteUtterances operation to manually delete utterances for a specific session. When you use the DeleteUtterances operation, utterances stored for improving your bot's ability to respond to user input are deleted immediately. Utterances stored for use with the ListAggregatedUtterances operation are deleted after 15 days.

Request Syntax

```
DELETE /bots/botId/utterances/?localeId=localeId&sessionId=sessionId HTTP/1.1
```

URI Request Parameters

The request uses the following URI parameters.

**botId (p. 419)**

The unique identifier of the bot that contains the utterances.

Length Constraints: Fixed length of 10.

Pattern: `^[0-9a-zA-Z]+$`

Required: Yes

**localeId (p. 419)**

The identifier of the language and locale where the utterances were collected. The string must match one of the supported locales. For more information, see Supported languages.

**sessionId (p. 419)**

The unique identifier of the session with the user. The ID is returned in the response from the RecognizeText and RecognizeUtterance operations.


Pattern: `^[0-9a-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

For information about the errors that are common to all actions, see Common Errors (p. 821).

InternalServerException

The service encountered an unexpected condition. Try your request again.

  HTTP Status Code: 500

ThrottlingException

Your request rate is too high. Reduce the frequency of requests.

  HTTP Status Code: 429

ValidationException

One of the input parameters in your request isn't valid. Check the parameters and try your request again.

  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 V2
  • AWS SDK for JavaScript
  • AWS SDK for PHP V3
  • AWS SDK for Python
  • AWS SDK for Ruby V3
DescribeBot
Service: Amazon Lex Model Building V2

Provides metadata information about a bot.

Request Syntax

GET /bots/{botId} HTTP/1.1

URI Request Parameters

The request uses the following URI parameters.

**botId (p. 421)**

The unique identifier of the bot to describe.

Length Constraints: Fixed length of 10.

Pattern: ^[0-9a-zA-Z]+$

Required: Yes

Request Body

The request does not have a request body.

Response Syntax

HTTP/1.1 200
Content-type: application/json

```
{
  "botId": "string",
  "botName": "string",
  "botStatus": "string",
  "creationDateTime": number,
  "dataPrivacy": {
    "childDirected": boolean
  },
  "description": "string",
  "idleSessionTTLInSeconds": number,
  "lastUpdatedDateTime": number,
  "roleArn": "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.

**botId (p. 421)**

The unique identifier of the bot.

Type: String
Length Constraints: Fixed length of 10.
Pattern: ^[0-9a-zA-Z]+$  

**botName (p. 421)**
The name of the bot.
Type: String

Length Constraints: Minimum length of 1. Maximum length of 100.
Pattern: ^([0-9a-zA-Z][_-]?)+$  

**botStatus (p. 421)**
The current status of the bot. When the status is `Available` the bot is ready to be used in conversations with users.
Type: String
Valid Values: Creating | Available | Inactive | Deleting | Failed | Versioning | Importing

**creationDateTime (p. 421)**
A timestamp of the date and time that the bot was created.
Type: Timestamp

**dataPrivacy (p. 421)**
Settings for managing data privacy of the bot and its conversations with users.
Type: DataPrivacy (p. 701) object

**description (p. 421)**
The description of the bot.
Type: String
Length Constraints: Minimum length of 0. Maximum length of 200.

**idleSessionTTLInSeconds (p. 421)**
The maximum time in seconds that Amazon Lex retains the data gathered in a conversation.
Type: Integer
Valid Range: Minimum value of 60. Maximum value of 86400.

**lastUpdatedDateTime (p. 421)**
A timestamp of the date and time that the bot was last updated.
Type: Timestamp

**roleArn (p. 421)**
The Amazon Resource Name (ARN) of an IAM role that has permission to access the bot.
Type: String
Length Constraints: Minimum length of 32. Maximum length of 2048.
Pattern: ^arn:aws:iam::[0-9]{12}:role/.*$
Errors

For information about the errors that are common to all actions, see Common Errors (p. 821).

InternalServerException

The service encountered an unexpected condition. Try your request again.

HTTP Status Code: 500

ResourceNotFoundException

You asked to describe a resource that doesn't exist. Check the resource that you are requesting and try again.

HTTP Status Code: 404

ServiceQuotaExceededException

You have reached a quota for your bot.

HTTP Status Code: 402

ThrottlingException

Your request rate is too high. Reduce the frequency of requests.

HTTP Status Code: 429

ValidationException

One of the input parameters in your request isn't valid. Check the parameters and try your request again.

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 V2
- AWS SDK for JavaScript
- AWS SDK for PHP V3
- AWS SDK for Python
- AWS SDK for Ruby V3
DescribeBotAlias
Service: Amazon Lex Model Building V2

Get information about a specific bot alias.

Request Syntax

GET /bots/botId/botAliases/botAliasId/ HTTP/1.1

URI Request Parameters

The request uses the following URI parameters.

botAliasId (p. 424)

The identifier of the bot alias to describe.

Length Constraints: Fixed length of 10.

Pattern: ^\bTSTALIASID\b|[0-9a-zA-Z]+$

Required: Yes

botId (p. 424)

The identifier of the bot associated with the bot alias to describe.

Length Constraints: Fixed length of 10.

Pattern: ^[0-9a-zA-Z]+$

Required: Yes

Request Body

The request does not have a request body.

Response Syntax

HTTP/1.1 200
Content-type: application/json

{  "botAliasHistoryEvents": [  {  "botVersion": "string",  "endDate": number,  "startDate": number  }  ],  "botAliasId": "string",  "botAliasLocaleSettings": {  "string": {  "codeHookSpecification": {  "lambdaCodeHook": {  "codeHookInterfaceVersion": "string",  "lambdaARN": "string"  }  },  "enabled": boolean  }  }

424
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.

**botAliasHistoryEvents (p. 424)**

A list of events that affect a bot alias. For example, an event is recorded when the version that the alias points to changes.

Type: Array of BotAliasHistoryEvent (p. 664) objects

**botAliasId (p. 424)**

The identifier of the bot alias.

Type: String

Length Constraints: Fixed length of 10.

Pattern: ^\bTSTALIASID\b\[0-9a-zA-Z]+$
Type: String to BotAliasLocaleSettings (p. 665) object map

Map Entries: Maximum number of items.

**botAliasName (p. 424)**

The name of the bot alias.

Type: String

Length Constraints: Minimum length of 1. Maximum length of 100.

Pattern: `^([0-9a-zA-Z][_-]?)+$`

**botAliasStatus (p. 424)**

The current status of the alias. When the alias is Available, the alias is ready for use with your bot.

Type: String

Valid Values: Creating | Available | Deleting | Failed

**botId (p. 424)**

The identifier of the bot associated with the bot alias.

Type: String

Length Constraints: Fixed length of 10.

Pattern: `^[0-9a-zA-Z]+$`

**botVersion (p. 424)**

The version of the bot associated with the bot alias.

Type: String


Pattern: `^(DRAFT|[0-9]+)$`

**conversationLogSettings (p. 424)**

Specifics of how Amazon Lex logs text and audio conversations with the bot associated with the alias.

Type: ConversationLogSettings (p. 697) object

**creationDateTime (p. 424)**

A timestamp of the date and time that the alias was created.

Type: Timestamp

**description (p. 424)**

The description of the bot alias.

Type: String

Length Constraints: Minimum length of 0. Maximum length of 200.

**lastUpdatedDateTime (p. 424)**

A timestamp of the date and time that the alias was last updated.

Type: Timestamp
sentimentAnalysisSettings (p. 424)

Determines whether Amazon Lex will use Amazon Comprehend to detect the sentiment of user utterances.

Type: SentimentAnalysisSettings (p. 753) object

Errors

For information about the errors that are common to all actions, see Common Errors (p. 821).

**InternalServerException**

The service encountered an unexpected condition. Try your request again.

HTTP Status Code: 500

**ResourceNotFoundException**

You asked to describe a resource that doesn’t exist. Check the resource that you are requesting and try again.

HTTP Status Code: 404

**ServiceQuotaExceededException**

You have reached a quota for your bot.

HTTP Status Code: 402

**ThrottlingException**

Your request rate is too high. Reduce the frequency of requests.

HTTP Status Code: 429

**ValidationException**

One of the input parameters in your request isn’t valid. Check the parameters and try your request again.

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 V2
- AWS SDK for JavaScript
- AWS SDK for PHP V3
- AWS SDK for Python
- AWS SDK for Ruby V3
DescribeBotLocale
Service: Amazon Lex Model Building V2

Describes the settings that a bot has for a specific locale.

Request Syntax

GET /bots/botId/botversions/botVersion/botlocales/localeId/ HTTP/1.1

URI Request Parameters

The request uses the following URI parameters.

**botId (p. 428)**

The identifier of the bot associated with the locale.

Length Constraints: Fixed length of 10.

Pattern: ^[0-9a-zA-Z]+$

Required: Yes

**botVersion (p. 428)**

The identifier of the version of the bot associated with the locale.


Pattern: ^(DRAFT|[0-9]+)$

Required: Yes

**localeId (p. 428)**

The unique identifier of the locale to describe. The string must match one of the supported locales. For more information, see Supported languages.

Required: Yes

Request Body

The request does not have a request body.

Response Syntax

HTTP/1.1 200
Content-type: application/json

```json
{
    "botId": "string",
    "botLocaleHistoryEvents": [
        {
            "event": "string",
            "eventDate": number
        }
    ],
    "botLocaleStatus": "string",
    "botVersion": "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.

**botId (p. 428)**

The identifier of the bot associated with the locale.

Type: String

Length Constraints: Fixed length of 10.

Pattern: ^[0-9a-zA-Z]+$

**botLocaleHistoryEvents (p. 428)**

History of changes, such as when a locale is used in an alias, that have taken place for the locale.

Type: Array of BotLocaleHistoryEvent (p. 674) objects

**botLocaleStatus (p. 428)**

The status of the bot. If the status is Failed, the reasons for the failure are listed in the failureReasons field.

Type: String

Valid Values: Creating | Building | Built | ReadyExpressTesting | Failed | Deleting | NotBuilt | Importing | Processing

**botVersion (p. 428)**

The identifier of the version of the bot associated with the locale.

Type: String


Pattern: ^(DRAFT|\d+)$

**creationDateTime (p. 428)**

The date and time that the locale was created.

Type: Timestamp
**description (p. 428)**

The description of the locale.

Type: String

Length Constraints: Minimum length of 0. Maximum length of 200.

**failureReasons (p. 428)**

if botLocaleStatus is Failed, Amazon Lex explains why it failed to build the bot.

Type: Array of strings

**intentsCount (p. 428)**

The number of intents defined for the locale.

Type: Integer

**lastBuildSubmittedDateTime (p. 428)**

The date and time that the locale was last submitted for building.

Type: Timestamp

**lastUpdatedDateTime (p. 428)**

The date and time that the locale was last updated.

Type: Timestamp

**localeId (p. 428)**

The unique identifier of the described locale.

Type: String

**localeName (p. 428)**

The name of the locale.

Type: String

**nluIntentConfidenceThreshold (p. 428)**

The confidence threshold where Amazon Lex inserts the `AMAZON.FallbackIntent` and `AMAZON.KendraSearchIntent` intents in the list of possible intents for an utterance.

Type: Double

Valid Range: Minimum value of 0. Maximum value of 1.

**recommendedActions (p. 428)**

Recommended actions to take to resolve an error in the `failureReasons` field.

Type: Array of strings

**slotTypesCount (p. 428)**

The number of slot types defined for the locale.

Type: Integer

**voiceSettings (p. 428)**

The Amazon Polly voice Amazon Lex uses for voice interaction with the user.
Type: VoiceSettings (p. 778) object

Errors

For information about the errors that are common to all actions, see Common Errors (p. 821).

InternalServerException
The service encountered an unexpected condition. Try your request again.

HTTP Status Code: 500

ResourceNotFoundException
You asked to describe a resource that doesn't exist. Check the resource that you are requesting and try again.

HTTP Status Code: 404

ServiceQuotaExceededException
You have reached a quota for your bot.

HTTP Status Code: 402

ThrottlingException
Your request rate is too high. Reduce the frequency of requests.

HTTP Status Code: 429

ValidationException
One of the input parameters in your request isn't valid. Check the parameters and try your request again.

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 V2
- AWS SDK for JavaScript
- AWS SDK for PHP V3
- AWS SDK for Python
- AWS SDK for Ruby V3
DescribeBotRecommendation

Service: Amazon Lex Model Building V2

Provides metadata information about a bot recommendation. This information will enable you to get a description on the request inputs, to download associated transcripts after processing is complete, and to download intents and slot-types generated by the bot recommendation.

Request Syntax

```
GET /bots/botId/botversions/botVersion/botlocales/localeId/botrecommendations/botRecommendationId/ HTTP/1.1
```

URI Request Parameters

The request uses the following URI parameters.

**botId (p. 432)**

The unique identifier of the bot associated with the bot recommendation.

- Pattern: `^[0-9a-zA-Z]+$`
- Required: Yes

**botRecommendationId (p. 432)**

The identifier of the bot recommendation to describe.

- Pattern: `^[0-9a-zA-Z]+$`
- Required: Yes

**botVersion (p. 432)**

The version of the bot associated with the bot recommendation.

- Length Constraints: Fixed length of 5.
- Pattern: `^[0-9a-zA-Z]+$`
- Required: Yes

**localeId (p. 432)**

The identifier of the language and locale of the bot recommendation to describe. The string must match one of the supported locales. For more information, see Supported languages.

- Required: Yes

Request Body

The request does not have a request body.

Response Syntax

```
HTTP/1.1 200
Content-type: application/json
```
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.

`botId (p. 432)`

The identifier of the bot associated with the bot recommendation.

Type: String

Length Constraints: Fixed length of 10.

Pattern: `^[0-9a-zA-Z]+$`

`botRecommendationId (p. 432)`

The identifier of the bot recommendation being described.


Type: String

Length Constraints: Fixed length of 10.

Pattern: ^[0-9a-zA-Z]+$  

**botRecommendationResults** *(p. 432)*

The object representing the URL of the bot definition, the URL of the associated transcript and a statistical summary of the bot recommendation results.

Type: **BotRecommendationResults** *(p. 680)* object

**botRecommendationStatus** *(p. 432)*

The status of the bot recommendation. If the status is Failed, then the reasons for the failure are listed in the failureReasons field.

Type: String

Valid Values: Processing | Deleting | Deleted | Downloading | Updating | Available | Failed

**botVersion** *(p. 432)*

The version of the bot associated with the bot recommendation.

Type: String

Length Constraints: Fixed length of 5.

Pattern: ^DRAFT$

**creationDateTime** *(p. 432)*

The date and time that the bot recommendation was created.

Type: Timestamp

**encryptionSetting** *(p. 432)*

The object representing the passwords that were used to encrypt the data related to the bot recommendation results, as well as the KMS key ARN used to encrypt the associated metadata.

Type: **EncryptionSetting** *(p. 704)* object

**failureReasons** *(p. 432)*

If botRecommendationStatus is Failed, Amazon Lex explains why.

Type: Array of strings

**lastUpdatedDateTime** *(p. 432)*

The date and time that the bot recommendation was last updated.

Type: Timestamp

**localeId** *(p. 432)*

The identifier of the language and locale of the bot recommendation to describe.

Type: String

**transcriptSourceSetting** *(p. 432)*

The object representing the Amazon S3 bucket containing the transcript, as well as the associated metadata.
Errors

For information about the errors that are common to all actions, see Common Errors (p. 821).

**InternalServerException**

The service encountered an unexpected condition. Try your request again.

HTTP Status Code: 500

**ResourceNotFoundException**

You asked to describe a resource that doesn’t exist. Check the resource that you are requesting and try again.

HTTP Status Code: 404

**ThrottlingException**

Your request rate is too high. Reduce the frequency of requests.

HTTP Status Code: 429

**ValidationException**

One of the input parameters in your request isn’t valid. Check the parameters and try your request again.

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 V2
- AWS SDK for JavaScript
- AWS SDK for PHP V3
- AWS SDK for Python
- AWS SDK for Ruby V3
DescribeBotVersion
Service: Amazon Lex Model Building V2

Provides metadata about a version of a bot.

Request Syntax

GET /bots/botId/botversions/botVersion/ HTTP/1.1

URI Request Parameters

The request uses the following URI parameters.

botId (p. 436)
The identifier of the bot containing the version to return metadata for.
Length Constraints: Fixed length of 10.
Pattern: ^[0-9a-zA-Z]+$
Required: Yes

botVersion (p. 436)
The version of the bot to return metadata for.
Pattern: ^[0-9]+$
Required: Yes

Request Body

The request does not have a request body.

Response Syntax

HTTP/1.1 200
Content-type: application/json

{  
  "botId": "string",
  "botName": "string",
  "botStatus": "string",
  "botVersion": "string",
  "creationDateTime": number,
  "dataPrivacy": {
    "childDirected": boolean
  },
  "description": "string",
  "failureReasons": [ "string" ],
  "idleSessionTTLInSeconds": number,
  "roleArn": "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.

**botId (p. 436)**

The identifier of the bot that contains the version.

Type: String

Length Constraints: Fixed length of 10.

Pattern: ^[0-9a-zA-Z]+$

**botName (p. 436)**

The name of the bot that contains the version.

Type: String

Length Constraints: Minimum length of 1. Maximum length of 100.

Pattern: ^([0-9a-zA-Z][_-]?)+$

**botStatus (p. 436)**

The current status of the bot. When the status is Available, the bot version is ready for use.

Type: String

Valid Values: Creating | Available | Inactive | Deleting | Failed | Versioning | Importing

**botVersion (p. 436)**

The version of the bot to describe.

Type: String


Pattern: ^[0-9]+$

**creationDateTime (p. 436)**

A timestamp of the date and time that the bot version was created.

Type: Timestamp

**dataPrivacy (p. 436)**

Data privacy settings for the bot version.

Type: DataPrivacy (p. 701) object

**description (p. 436)**

The description specified for the bot.

Type: String

Length Constraints: Minimum length of 0. Maximum length of 200.

**failureReasons (p. 436)**

If the botStatus is Failed, this contains a list of reasons that the version couldn't be built.

Type: Array of strings
idleSessionTTLInSeconds (p. 436)

The number of seconds that a session with the bot remains active before it is discarded by Amazon Lex.

Type: Integer

Valid Range: Minimum value of 60. Maximum value of 86400.

roleArn (p. 436)

The Amazon Resource Name (ARN) of an IAM role that has permission to access the bot version.

Type: String

Length Constraints: Minimum length of 32. Maximum length of 2048.

Pattern: ^arn:aws:iam::[0-9]{12}:role/.*$

Errors

For information about the errors that are common to all actions, see Common Errors (p. 821).

InternalServerException

The service encountered an unexpected condition. Try your request again.

HTTP Status Code: 500

ResourceNotFoundException

You asked to describe a resource that doesn't exist. Check the resource that you are requesting and try again.

HTTP Status Code: 404

ServiceQuotaExceededException

You have reached a quota for your bot.

HTTP Status Code: 402

ThrottlingException

Your request rate is too high. Reduce the frequency of requests.

HTTP Status Code: 429

ValidationException

One of the input parameters in your request isn't valid. Check the parameters and try your request again.

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 V2
- AWS SDK for JavaScript
- AWS SDK for PHP V3
- AWS SDK for Python
- AWS SDK for Ruby V3
DescribeCustomVocabularyMetadata
Service: Amazon Lex Model Building V2

Provides metadata information about a custom vocabulary.

Request Syntax

GET /bots/botId/botversions/botVersion/botlocales/localeId/customvocabulary/DEFAULT/metadata HTTP/1.1

URI Request Parameters

The request uses the following URI parameters.

botId (p. 440)

The unique identifier of the bot that contains the custom vocabulary.

Length Constraints: Fixed length of 10.

Pattern: ^[0-9a-zA-Z]+$

Required: Yes

botVersion (p. 440)

The bot version of the bot to return metadata for.


Pattern: ^(DRAFT|[0-9]+)$

Required: Yes

localeId (p. 440)

The locale to return the custom vocabulary information for. The locale must be en_GB.

Required: Yes

Request Body

The request does not have a request body.

Response Syntax

HTTP/1.1 200
Content-type: application/json

{
    "botId": "string",
    "botVersion": "string",
    "creationDateTime": number,
    "customVocabularyStatus": "string",
    "lastUpdatedDateTime": number,
    "localeId": "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.

**botId (p. 440)**

The identifier of the bot that contains the custom vocabulary.

Type: String

Length Constraints: Fixed length of 10.

Pattern: ^[0-9a-zA-Z]+$  

**botVersion (p. 440)**

The version of the bot that contains the custom vocabulary to describe.

Type: String


Pattern: ^(DRAFT|\d+)$

**creationDateTime (p. 440)**

The date and time that the custom vocabulary was created.

Type: Timestamp

**customVocabularyStatus (p. 440)**

The status of the custom vocabulary. If the status is Ready the custom vocabulary is ready to use.

Type: String

Valid Values: Ready | Deleting | Exporting | Importing | Creating

**lastUpdatedDateTime (p. 440)**

The date and time that the custom vocabulary was last updated.

Type: Timestamp

**localeId (p. 440)**

The locale that contains the custom vocabulary to describe.

Type: String

**Errors**

For information about the errors that are common to all actions, see Common Errors (p. 821).

**InternalServerException**

The service encountered an unexpected condition. Try your request again.

HTTP Status Code: 500

**ResourceNotFoundException**

You asked to describe a resource that doesn't exist. Check the resource that you are requesting and try again.

HTTP Status Code: 404
ServiceQuotaExceededException

You have reached a quota for your bot.

HTTP Status Code: 402

ThrottlingException

Your request rate is too high. Reduce the frequency of requests.

HTTP Status Code: 429

ValidationException

One of the input parameters in your request isn’t valid. Check the parameters and try your request again.

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 V2
- AWS SDK for JavaScript
- AWS SDK for PHP V3
- AWS SDK for Python
- AWS SDK for Ruby V3
DescribeExport
Service: Amazon Lex Model Building V2

Gets information about a specific export.

Request Syntax

GET /exports/exportId/ HTTP/1.1

URI Request Parameters

The request uses the following URI parameters.

exportId (p. 443)

The unique identifier of the export to describe.

Length Constraints: Fixed length of 10.

Pattern: ^[0-9a-zA-Z]+$

Required: Yes

Request Body

The request does not have a request body.

Response Syntax

HTTP/1.1 200
Content-type: application/json

{  "creationDateTime": number,
   "downloadUrl": "string",
   "exportId": "string",
   "exportStatus": "string",
   "failureReasons": [ "string" ],
   "fileFormat": "string",
   "lastUpdatedDateTime": number,
   "resourceSpecification": {
      "botExportSpecification": {
         "botId": "string",
         "botVersion": "string"
      },
      "botLocaleExportSpecification": {
         "botId": "string",
         "botVersion": "string",
         "localeId": "string"
      },
      "customVocabularyExportSpecification": {
         "botId": "string",
         "botVersion": "string",
         "localeId": "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.

**creationDateTime (p. 443)**

The date and time that the export was created.
Type: Timestamp

**downloadUrl (p. 443)**

A pre-signed S3 URL that points to the bot or bot locale archive. The URL is only available for 5 minutes after calling the `DescribeExport` operation.
Type: String

**exportId (p. 443)**

The unique identifier of the described export.
Type: String
Length Constraints: Fixed length of 10.
Pattern: `^[0-9a-zA-Z]+$`

**exportStatus (p. 443)**

The status of the export. When the status is `Complete` the export archive file is available for download.
Type: String
Valid Values: InProgress | Completed | Failed | Deleting

**failureReasons (p. 443)**

If the `exportStatus` is failed, contains one or more reasons why the export could not be completed.
Type: Array of strings

**fileFormat (p. 443)**

The file format used in the files that describe the resource.
Type: String
Valid Values: LexJson | TSV

**lastUpdatedDateTime (p. 443)**

The last date and time that the export was updated.
Type: Timestamp

**resourceSpecification (p. 443)**

The bot, bot ID, and optional locale ID of the exported bot or bot locale.
Type: `ExportResourceSpecification (p. 706)` object
Errors

For information about the errors that are common to all actions, see Common Errors (p. 821).

InternalServerErrorException

The service encountered an unexpected condition. Try your request again.

HTTP Status Code: 500

ResourceNotFoundException

You asked to describe a resource that doesn’t exist. Check the resource that you are requesting and try again.

HTTP Status Code: 404

ThrottlingException

Your request rate is too high. Reduce the frequency of requests.

HTTP Status Code: 429

ValidationException

One of the input parameters in your request isn’t valid. Check the parameters and try your request again.

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 V2
- AWS SDK for JavaScript
- AWS SDK for PHP V3
- AWS SDK for Python
- AWS SDK for Ruby V3
DescribeImport

Service: Amazon Lex Model Building V2

Gets information about a specific import.

Request Syntax

GET /imports/importId/ HTTP/1.1

URI Request Parameters

The request uses the following URI parameters.

**importId (p. 446)**

The unique identifier of the import to describe.

Length Constraints: Fixed length of 10.

Pattern: \^[0-9a-zA-Z]+\$

Required: Yes

Request Body

The request does not have a request body.

Response Syntax

HTTP/1.1 200
Content-type: application/json

```json
{
    "creationDateTime": number,
    "failureReasons": [ "string" ],
    "importedResourceId": "string",
    "importedResourceName": "string",
    "importId": "string",
    "importStatus": "string",
    "lastUpdatedDateTime": number,
    "mergeStrategy": "string",
    "resourceSpecification": {
        "botImportSpecification": {
            "botName": "string",
            "botTags": {
                "string": "string"
            },
            "dataPrivacy": {
                "childDirected": boolean
            },
            "idleSessionTTLInSeconds": number,
            "roleArn": "string",
            "testBotAliasTags": {
                "string": "string"
            }
        },
        "botLocaleImportSpecification": {
            "botId": "string",
            "botVersion": "string",
            "localeId": "string",
            "localeTags": {
                "string": "string"
            },
            "dataPrivacy": {
                "childDirected": boolean
            }
        }
    }
}
```
"nluIntentConfidenceThreshold": number,
"voiceSettings": {
  "engine": "string",
  "voiceId": "string"
}
},
"customVocabularyImportSpecification": {
  "botId": "string",
  "botVersion": "string",
  "localeId": "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.

creationDateTime (p. 446)

The date and time that the import was created.

Type: Timestamp

failureReasons (p. 446)

If the importStatus field is Failed, this provides one or more reasons for the failure.

Type: Array of strings

importedResourceId (p. 446)

The unique identifier that Amazon Lex assigned to the resource created by the import.

Type: String


Pattern: ^([0-9a-zA-Z_]+)$

importedResourceName (p. 446)

The name of the imported resource.

Type: String

Length Constraints: Minimum length of 1. Maximum length of 100.

Pattern: ^([0-9a-zA-Z][_-]?)+$ 

importId (p. 446)

The unique identifier of the described import.

Type: String

Length Constraints: Fixed length of 10.

Pattern: ^[0-9a-zA-Z]+$ 

importStatus (p. 446)

The status of the import process. When the status is Completed the resource is imported and ready for use.
Type: String
Valid Values: InProgress | Completed | Failed | Deleting

lastUpdatedDateTime (p. 446)
The date and time that the import was last updated.
Type: Timestamp

mergeStrategy (p. 446)
The strategy used when there was a name conflict between the imported resource and an existing resource. When the merge strategy is FailOnConflict existing resources are not overwritten and the import fails.
Type: String
Valid Values: Overwrite | FailOnConflict | Append

resourceSpecification (p. 446)
The specifications of the imported bot, bot locale, or custom vocabulary.
Type: ImportResourceSpecification (p. 720) object

Errors
For information about the errors that are common to all actions, see Common Errors (p. 821).

InternalServerException
The service encountered an unexpected condition. Try your request again.
HTTP Status Code: 500

ResourceNotFoundException
You asked to describe a resource that doesn't exist. Check the resource that you are requesting and try again.
HTTP Status Code: 404

ThrottlingException
Your request rate is too high. Reduce the frequency of requests.
HTTP Status Code: 429

ValidationException
One of the input parameters in your request isn't valid. Check the parameters and try your request again.
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 V2
- AWS SDK for JavaScript
- AWS SDK for PHP V3
- AWS SDK for Python
- AWS SDK for Ruby V3
DescribeIntent
Service: Amazon Lex Model Building V2
Returns metadata about an intent.

Request Syntax

```
GET /bots/botId/botversions/botVersion/botlocales/localeId/intents/intentId/ HTTP/1.1
```

URI Request Parameters

The request uses the following URI parameters.

**botId (p. 450)**

- The identifier of the bot associated with the intent.
- Pattern: `^[0-9a-zA-Z]+$`
- Required: Yes

**botVersion (p. 450)**

- The version of the bot associated with the intent.
- Pattern: `^(DRAFT|[0-9]+$`
- Required: Yes

**intentId (p. 450)**

- The identifier of the intent to describe.
- Pattern: `^[0-9a-zA-Z]+$`
- Required: Yes

**localeId (p. 450)**

- The identifier of the language and locale of the intent to describe. The string must match one of the supported locales. For more information, see Supported languages.
- Required: Yes

Request Body

The request does not have a request body.

Response Syntax

```
HTTP/1.1 200
Content-type: application/json
{
```
"botId": "string",
"botVersion": "string",
"creationDateTime": number,
"description": "string",
"dialogCodeHook": {
    "enabled": boolean
},
"fulfillmentCodeHook": {
    "enabled": boolean,
    "fulfillmentUpdatesSpecification": {
        "active": boolean,
        "startResponse": {
            "allowInterrupt": boolean,
            "delayInSeconds": number,
            "messageGroups": [
                {
                    "message": {
                        "customPayload": {
                            "value": "string"
                        },
                        "imageResponseCard": {
                            "buttons": [
                                {
                                    "text": "string",
                                    "value": "string"
                                }
                            ],
                            "imageUrl": "string",
                            "subtitle": "string",
                            "title": "string"
                        },
                        "plainTextMessage": {
                            "value": "string"
                        },
                        "ssmlMessage": {
                            "value": "string"
                        }
                    }
                }
            ],
            "timeoutInSeconds": number,
        }
    }
}
"updateResponse": {  
  "allowInterrupt": boolean,
  "frequencyInSeconds": number,
  "messageGroups": [
    {
      "message": {
        "customPayload": {
          "value": "string"
        },
        "imageResponseCard": {
          "buttons": [
            {
              "text": "string",
              "value": "string"
            }
          ],
          "imageUrl": "string",
          "subtitle": "string",
          "title": "string"
        },
        "plainTextMessage": {
          "value": "string"
        },
        "ssmlMessage": {
          "value": "string"
        }
      },
      "variations": [
        {
          "customPayload": {
            "value": "string"
          },
          "imageResponseCard": {
            "buttons": [
              {
                "text": "string",
                "value": "string"
              }
            ],
            "imageUrl": "string",
            "subtitle": "string",
            "title": "string"
          },
          "plainTextMessage": {
            "value": "string"
          },
          "ssmlMessage": {
            "value": "string"
          }
        }
      ]
    }
  ],
  "postFulfillmentStatusSpecification": {
    "failureResponse": {
      "allowInterrupt": boolean,
      "messageGroups": [
        {
          "message": {
            "customPayload": {
              "value": "string"
            },
            "imageResponseCard": {
              "buttons": [
                ]
            }
          }
        }
      ]
    }
  }
}


```json
{
    "text": "string",
    "value": "string"
}
,"imageUrl": "string",
"subtitle": "string",
"title": "string"
},
"plainTextMessage": {
    "value": "string"
},
"ssmlMessage": {
    "value": "string"
}
},
"variations": [
{
    "customPayload": {
        "value": "string"
    },
    "imageResponseCard": {
        "buttons": [
            {
                "text": "string",
                "value": "string"
            }
        ],
        "imageUrl": "string",
        "subtitle": "string",
        "title": "string"
    },
    "plainTextMessage": {
        "value": "string"
    },
    "ssmlMessage": {
        "value": "string"
    }
}
],
"successResponse": {
    "allowInterrupt": boolean,
    "messageGroups": [
        {
            "message": {
                "customPayload": {
                    "value": "string"
                },
                "imageResponseCard": {
                    "buttons": [
                        {
                            "text": "string",
                            "value": "string"
                        }
                    ],
                    "imageUrl": "string",
                    "subtitle": "string",
                    "title": "string"
                },
                "plainTextMessage": {
                    "value": "string"
                },
                "ssmlMessage": {
                    "value": "string"
                }
            }
        }
    ]
}
```
"value": "string"
"
"variations": [
{
"customPayload": {
"value": "string"
"imageResponseCard": {
"buttons": [
{
"text": "string",
"value": "string"
}],
"imageUrl": "string",
"subtitle": "string",
"title": "string"
"plainTextMessage": {
"value": "string"
"ssmlMessage": {
"value": "string"
"variations": [
{
"customPayload": {
"value": "string"
"imageResponseCard": {
"buttons": [
{
"text": "string",
"value": "string"
}],
"imageUrl": "string",
"subtitle": "string",
"title": "string"
"plainTextMessage": {
"value": "string"
"ssmlMessage": {
"value": "string"
"variations": [
{
"customPayload": {
"value": "string"
"imageResponseCard": {
"buttons": [
{
"text": "string",
"value": "string"


},
  "imageUrl": "string",
  "subtitle": "string",
  "title": "string"
},
  "plaintextMessage": {
    "value": "string"
  },
  "ssmlMessage": {
    "value": "string"
  }
],
  "inputContexts": [
    {
      "name": "string"
    }
  ],
  "intentClosingSetting": {
    "active": boolean,
    "closingResponse": {
      "allowInterrupt": boolean,
      "messageGroups": [
        {
          "message": {
            "customPayload": {
              "value": "string"
            },
            "imageResponseCard": {
              "buttons": [
                {
                  "text": "string",
                  "value": "string"
                }
              ],
              "imageUrl": "string",
              "subtitle": "string",
              "title": "string"
            },
            "plaintextMessage": {
              "value": "string"
            },
            "ssmlMessage": {
              "value": "string"
            }
          },
          "variations": [
            {
              "customPayload": {
                "value": "string"
              },
              "imageResponseCard": {
                "buttons": [
                  {
                    "text": "string",
                    "value": "string"
                  }
                ],
                "imageUrl": "string",
                "subtitle": "string",
                "title": "string"
              },
              "plaintextMessage": {
                "value": "string"
              },
              "ssmlMessage": {
                "value": "string"
              }
            }
          ]
        }
      ]
    }
  }
}
}
"title": "string",
"intentConfirmationSetting": {
  "active": boolean,
  "declinationResponse": {
    "allowInterrupt": boolean,
    "messageGroups": [
      {
        "message": {
          "customPayload": {
            "value": "string"
          },
          "imageResponseCard": {
            "buttons": [
              {
                "text": "string",
                "value": "string"
              }
            ],
            "imageUrl": "string",
            "subtitle": "string",
            "title": "string"
          },
          "plainTextMessage": {
            "value": "string"
          },
          "ssmlMessage": {
            "value": "string"
          }
        }
      }
    ]
  }
}
}
"promptSpecification": {
  "allowInterrupt": boolean,
  "maxRetries": number,
  "messageGroups": [
    {
      "message": {
        "customPayload": {
          "value": "string"
        },
        "imageResponseCard": {
          "buttons": [
            {
              "text": "string",
              "value": "string"
            }
          ],
          "imageUrl": "string",
          "subtitle": "string",
          "title": "string"
        },
        "plainTextMessage": {
          "value": "string"
        },
        "ssmlMessage": {
          "value": "string"
        }
      },
      "variations": [
        {
          "customPayload": {
            "value": "string"
          },
          "imageResponseCard": {
            "buttons": [
              {
                "text": "string",
                "value": "string"
              }
            ],
            "imageUrl": "string",
            "subtitle": "string",
            "title": "string"
          },
          "plainTextMessage": {
            "value": "string"
          },
          "ssmlMessage": {
            "value": "string"
          }
        }
      ]
    }
  ]
},
"intentId": "string",
"intentName": "string",
"kendraConfiguration": {
  "kendraIndex": "string",
  "queryFilterString": "string",
  "queryFilterStringEnabled": boolean
},
"lastUpdatedDateTime": number,
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.

**botId (p. 450)**

The identifier of the bot associated with the intent.
Type: String
Length Constraints: Fixed length of 10.
Pattern: ^[0-9a-zA-Z]+$  

**botVersion (p. 450)**

The version of the bot associated with the intent.
Type: String
Length Constraints: Fixed length of 5.
Pattern: ^DRAFT$  

**creationDateTime (p. 450)**

A timestamp of the date and time that the intent was created.
Type: Timestamp

**description (p. 450)**

The description of the intent.
Type: String
Length Constraints: Minimum length of 0. Maximum length of 200.

**dialogCodeHook (p. 450)**

The Lambda function called during each turn of a conversation with the intent.
Type: DialogCodeHookSettings (p. 703) object

fulfillmentCodeHook (p. 450)

The Lambda function called when the intent is complete and ready for fulfillment.

Type: FulfillmentCodeHookSettings (p. 711) object

inputContexts (p. 450)

A list of contexts that must be active for the intent to be considered for sending to the user.

Type: Array of InputContext (p. 724) objects

Array Members: Minimum number of 0 items. Maximum number of 5 items.

intentClosingSetting (p. 450)

The response that Amazon Lex sends to when the intent is closed.

Type: IntentClosingSetting (p. 725) object

intentConfirmationSetting (p. 450)

Prompts that Amazon Lex sends to the user to confirm completion of an intent.

Type: IntentConfirmationSetting (p. 726) object

intentId (p. 450)

The unique identifier assigned to the intent when it was created.

Type: String

Length Constraints: Fixed length of 10.

Pattern: ^[0-9a-zA-Z]+$

intentName (p. 450)

The name specified for the intent.

Type: String

Length Constraints: Minimum length of 1. Maximum length of 100.

Pattern: ^([0-9a-zA-Z][- ]?)+$

kendraConfiguration (p. 450)

Configuration information required to use the AMAZON.KendraSearchIntent intent.

Type: KendraConfiguration (p. 732) object

lastUpdatedDateTime (p. 452)

A timestamp of the date and time that the intent was last updated.

Type: Timestamp

localeId (p. 450)

The language and locale specified for the intent.

Type: String

outputContexts (p. 450)

A list of contexts that are activated when the intent is fulfilled.
Type: Array of OutputContext (p. 739) objects

Array Members: Minimum number of 0 items. Maximum number of 10 items.

parentIntentSignature (p. 450)

The identifier of the built-in intent that this intent is derived from, if any.

Type: String

sampleUtterances (p. 450)

User utterances that trigger this intent.

Type: Array of SampleUtterance (p. 751) objects

slotPriorities (p. 450)

The list that determines the priority that slots should be elicited from the user.

Type: Array of SlotPriority (p. 757) objects

Errors

For information about the errors that are common to all actions, see Common Errors (p. 821).

InternalServerException

The service encountered an unexpected condition. Try your request again.

HTTP Status Code: 500

ResourceNotFoundException

You asked to describe a resource that doesn't exist. Check the resource that you are requesting and try again.

HTTP Status Code: 404

ServiceQuotaExceededException

You have reached a quota for your bot.

HTTP Status Code: 402

ThrottlingException

Your request rate is too high. Reduce the frequency of requests.

HTTP Status Code: 429

ValidationException

One of the input parameters in your request isn't valid. Check the parameters and try your request again.

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 V2
• AWS SDK for JavaScript
• AWS SDK for PHP V3
• AWS SDK for Python
• AWS SDK for Ruby V3
**DescribeResourcePolicy**  
Service: Amazon Lex Model Building V2

Gets the resource policy and policy revision for a bot or bot alias.

**Request Syntax**

```
GET /policy/resourceArn/ HTTP/1.1
```

**URI Request Parameters**

The request uses the following URI parameters.

**resourceArn (p. 462)**  
The Amazon Resource Name (ARN) of the bot or bot alias that the resource policy is attached to.  
Length Constraints: Minimum length of 1. Maximum length of 1011.  
Required: Yes

**Request Body**

The request does not have a request body.

**Response Syntax**

```
HTTP/1.1 200  
Content-type: application/json  
{  
    "policy": "string",  
    "resourceArn": "string",  
    "revisionId": "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.

**policy (p. 462)**

The JSON structure that contains the resource policy. For more information about the contents of a JSON policy document, see IAM JSON policy reference.

Type: String  
Length Constraints: Minimum length of 2.

**resourceArn (p. 462)**

The Amazon Resource Name (ARN) of the bot or bot alias that the resource policy is attached to.

Type: String  
Length Constraints: Minimum length of 1. Maximum length of 1011.
revisionId (p. 462)

The current revision of the resource policy. Use the revision ID to make sure that you are updating the most current version of a resource policy when you add a policy statement to a resource, delete a resource, or update a resource.

Type: String


Pattern: ^[0-9]+$

Errors

For information about the errors that are common to all actions, see Common Errors (p. 821).

InternalServerErrorException

The service encountered an unexpected condition. Try your request again.

HTTP Status Code: 500

ResourceNotFoundException

You asked to describe a resource that doesn’t exist. Check the resource that you are requesting and try again.

HTTP Status Code: 404

ThrottlingException

Your request rate is too high. Reduce the frequency of requests.

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 V2
- AWS SDK for JavaScript
- AWS SDK for PHP V3
- AWS SDK for Python
- AWS SDK for Ruby V3
DescribeSlot
Service: Amazon Lex Model Building V2
Gets metadata information about a slot.

Request Syntax

GET /bots/botId/botversions/botVersion/botlocales/localeId/intents/intentId/slots/slotId/
HTTP/1.1

URI Request Parameters

The request uses the following URI parameters.

botId (p. 464)

The identifier of the bot associated with the slot.
Length Constraints: Fixed length of 10.
Pattern: ^[0-9a-zA-Z]+$
Required: Yes

botVersion (p. 464)

The version of the bot associated with the slot.
Pattern: ^(DRAFT|[0-9]+$)
Required: Yes

intentId (p. 464)

The identifier of the intent that contains the slot.
Length Constraints: Fixed length of 10.
Pattern: ^[0-9a-zA-Z]+$
Required: Yes

localeId (p. 464)

The identifier of the language and locale of the slot to describe. The string must match one of the supported locales. For more information, see Supported languages.
Required: Yes

slotId (p. 464)

The unique identifier for the slot.
Length Constraints: Fixed length of 10.
Pattern: ^[0-9a-zA-Z]+$
Required: Yes

Request Body

The request does not have a request body.
Response Syntax

HTTP/1.1 200
Content-type: application/json

{
  "botId": "string",
  "botVersion": "string",
  "creationDateTime": number,
  "description": "string",
  "intentId": "string",
  "lastUpdatedDateTime": number,
  "localeId": "string",
  "multipleValuesSetting": {
    "allowMultipleValues": boolean
  },
  "obfuscationSetting": {
    "obfuscationSettingType": "string"
  },
  "slotId": "string",
  "slotName": "string",
  "slotTypeId": "string",
  "valueElicitationSetting": {
    "defaultValueSpecification": {
      "defaultValueList": [
        {
          "defaultValue": "string"
        }
      ],
    },
    "promptSpecification": {
      "allowInterrupt": boolean,
      "maxRetries": number,
      "messageGroups": [
        {
          "message": {
            "customPayload": { "value": "string" },
            "imageResponseCard": { "buttons": [ { "text": "string", "value": "string" } ] },
            "imageUrl": "string",
            "subtitle": "string",
            "title": "string"
          },
          "plainTextMessage": {
            "value": "string"
          },
          "ssm1Message": {
            "value": "string"
          }
        },
        { "customPayload": { "value": "string" },
          "imageResponseCard": { "buttons": [ ] }
        }
      ]
    }
  }
}
"text": "string",
"value": "string"
},
"imageUrl": "string",
"subtitle": "string",
"title": "string"
},
"plainTextMessage": {
"value": "string"
},
"ssmlMessage": {
"value": "string"
}
]}
},
"sampleUtterances": [
{
"utterance": "string"
}
],
"slotConstraint": "string",
"waitAndContinueSpecification": {
"active": boolean,
"continueResponse": {
"allowInterrupt": boolean,
"messageGroups": [
{
"message": {
"customPayload": {
"value": "string"
},
"imageResponseCard": {
"buttons": [
{
"text": "string",
"value": "string"
}
],
"imageUrl": "string",
"subtitle": "string",
"title": "string"
},
"plainTextMessage": {
"value": "string"
},
"ssmlMessage": {
"value": "string"
}
},
"variations": [
{
"customPayload": {
"value": "string"
},
"imageResponseCard": {
"buttons": [
{
"text": "string",
"value": "string"
}
],
"imageUrl": "string",
"subtitle": "string",
"title": "string"
},
"plainTextMessage": {
"value": "string"
},
"ssmlMessage": {
"value": "string"
}
}]
}
"subtitle": "string",
"title": "string"
},
"plainTextMessage": {
 "value": "string"
},
"ssmlMessage": {
 "value": "string"

"stillWaitingResponse": {
 "allowInterrupt": boolean,
 "frequencyInSeconds": number,
 "messageGroups": [
 { "message": {
 "customPayload": {
 "value": "string"
 },
 "imageResponseCard": {
 "buttons": [
 { "text": "string",
 "value": "string"
 }
 ],
 "imageUrl": "string",
 "subtitle": "string",
 "title": "string"
 },
"plainTextMessage": {
 "value": "string"
 },
"ssmlMessage": {
 "value": "string"
 }
 },
"variations": [ {
 "customPayload": {
 "value": "string"
 },
 "imageResponseCard": {
 "buttons": [
 { "text": "string",
 "value": "string"
 }
 ],
 "imageUrl": "string",
 "subtitle": "string",
 "title": "string"
 },
"plainTextMessage": {
 "value": "string"
 },
"ssmlMessage": {
 "value": "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.
botId (p. 465)

The identifier of the bot associated with the slot.

Type: String

Length Constraints: Fixed length of 10.

Pattern: ^[0-9a-zA-Z]+$

botVersion (p. 465)

The version of the bot associated with the slot.

Type: String


Pattern: ^(DRAFT|[0-9]+)$

creationDateTime (p. 465)

A timestamp of the date and time that the slot was created.

Type: Timestamp

description (p. 465)

The description specified for the slot.

Type: String

Length Constraints: Minimum length of 0. Maximum length of 200.

intentId (p. 465)

The identifier of the intent associated with the slot.

Type: String

Length Constraints: Fixed length of 10.

Pattern: ^[0-9a-zA-Z]+$

lastUpdatedDateTime (p. 465)

A timestamp of the date and time that the slot was last updated.

Type: Timestamp

localeId (p. 465)

The language and locale specified for the slot.

Type: String

multipleValuesSetting (p. 465)

Indicates whether the slot accepts multiple values in a single utterance.

If the multipleValuesSetting is not set, the default value is false.

Type: MultipleValuesSetting (p. 737) object

obfuscationSetting (p. 465)

Whether slot values are shown in Amazon CloudWatch logs. If the value is None, the actual value of the slot is shown in logs.
Type: **ObfuscationSetting (p. 738)** object

---

### slotId (p. 465)

The unique identifier generated for the slot.

Type: **String**

Length Constraints: Fixed length of 10.

Pattern: `^[0-9a-zA-Z]+$`

---

### slotName (p. 465)

The name specified for the slot.

Type: **String**

Length Constraints: Minimum length of 1. Maximum length of 100.

Pattern: `^[0-9a-zA-Z][_-]?]+$`

---

### slotTypeId (p. 465)

The identifier of the slot type that determines the values entered into the slot.

Type: **String**


Pattern: `^((AMAZON\.)?[a-zA-Z_]+?|0-9a-zA-Z)+$`

---

### valueElicitationSetting (p. 465)

Prompts that Amazon Lex uses to elicit a value for the slot.

Type: **SlotValueElicitationSetting (p. 767)** object

---

### Errors

For information about the errors that are common to all actions, see [Common Errors (p. 821)](#).

---

**InternalServerException**

The service encountered an unexpected condition. Try your request again.

HTTP Status Code: 500

---

**ResourceNotFoundException**

You asked to describe a resource that doesn't exist. Check the resource that you are requesting and try again.

HTTP Status Code: 404

---

**ServiceQuotaExceededException**

You have reached a quota for your bot.

HTTP Status Code: 402

---

**ThrottlingException**

Your request rate is too high. Reduce the frequency of requests.

HTTP Status Code: 429
ValidationException

One of the input parameters in your request isn't valid. Check the parameters and try your request again.

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 V2
- AWS SDK for JavaScript
- AWS SDK for PHP V3
- AWS SDK for Python
- AWS SDK for Ruby V3
**DescribeSlotType**  
Service: Amazon Lex Model Building V2  

Gets metadata information about a slot type.  

**Request Syntax**  

```plaintext
GET /bots/botId/botversions/botVersion/botlocales/localeId/slottypes/slotTypeId/ HTTP/1.1
```

**URI Request Parameters**  
The request uses the following URI parameters.  

**botId (p. 472)**  
The identifier of the bot associated with the slot type.  

Length Constraints: Fixed length of 10.  

Pattern: ^[0-9a-zA-Z]+$  

Required: Yes  

**botVersion (p. 472)**  
The version of the bot associated with the slot type.  


Pattern: ^(DRAFT|[0-9]+)$  

Required: Yes  

**localeId (p. 472)**  
The identifier of the language and locale of the slot type to describe. The string must match one of the supported locales. For more information, see Supported languages.  

Required: Yes  

**slotTypeId (p. 472)**  
The identifier of the slot type.  

Length Constraints: Fixed length of 10.  

Pattern: ^[0-9a-zA-Z]+$  

Required: Yes  

**Request Body**  
The request does not have a request body.  

**Response Syntax**  

```plaintext
HTTP/1.1 200  
Content-type: application/json  

{  
```
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.

botId (p. 472)

The identifier of the bot associated with the slot type.

Type: String

Length Constraints: Fixed length of 10.

Pattern: ^[0-9a-zA-Z]+$ botVersion (p. 472)

The version of the bot associated with the slot type.

Type: String

Pattern: ^\(DRAFT | [0-9]+\)$

**creationDateTime (p. 472)**

A timestamp of the date and time that the slot type was created.

Type: Timestamp

**description (p. 472)**

The description specified for the slot type.

Type: String

Length Constraints: Minimum length of 0. Maximum length of 200.

**externalSourceSetting (p. 472)**

Provides information about the external source of the slot type's definition.

Type: `ExternalSourceSetting (p. 710)` object

**lastUpdatedDateTime (p. 472)**

A timestamp of the date and time that the slot type was last updated.

Type: Timestamp

**localeId (p. 472)**

The language and locale specified for the slot type.

Type: String

**parentSlotTypeSignature (p. 472)**

The built in slot type used as a parent to this slot type.

Type: String

**slotTypeId (p. 472)**

The unique identifier for the slot type.

Type: String

Length Constraints: Fixed length of 10.

Pattern: ^\[0-9a-zA-Z]+$  

**slotTypeName (p. 472)**

The name specified for the slot type.

Type: String

Length Constraints: Minimum length of 1. Maximum length of 100.

Pattern: ^([0-9a-zA-Z][-_]?)+$  

**slotTypeValues (p. 472)**

The values that the slot type can take. Includes any synonyms for the slot type values.

Type: Array of `SlotTypeValue (p. 766)` objects

Array Members: Minimum number of 1 item. Maximum number of 10000 items.
valueSelectionSetting (p. 472)

The strategy that Amazon Lex uses to choose a value from a list of possible values.

Type: SlotValueSelectionSetting (p. 770) object

Errors

For information about the errors that are common to all actions, see Common Errors (p. 821).

InternalServerException

The service encountered an unexpected condition. Try your request again.

HTTP Status Code: 500

ResourceNotFoundException

You asked to describe a resource that doesn’t exist. Check the resource that you are requesting and try again.

HTTP Status Code: 404

ServiceQuotaExceededException

You have reached a quota for your bot.

HTTP Status Code: 402

ThrottlingException

Your request rate is too high. Reduce the frequency of requests.

HTTP Status Code: 429

ValidationException

One of the input parameters in your request isn't valid. Check the parameters and try your request again.

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 V2
- AWS SDK for JavaScript
- AWS SDK for PHP V3
- AWS SDK for Python
- AWS SDK for Ruby V3
ListAggregatedUtterances
Service: Amazon Lex Model Building V2

Provides a list of utterances that users have sent to the bot.

Utterances are aggregated by the text of the utterance. For example, all instances where customers used the phrase "I want to order pizza" are aggregated into the same line in the response.

You can see both detected utterances and missed utterances. A detected utterance is where the bot properly recognized the utterance and activated the associated intent. A missed utterance was not recognized by the bot and didn't activate an intent.

Utterances can be aggregated for a bot alias or for a bot version, but not both at the same time.

Utterances statistics are not generated under the following conditions:

- The childDirected field was set to true when the bot was created.
- You are using slot obfuscation with one or more slots.
- You opted out of participating in improving Amazon Lex.

Request Syntax

```json
POST /bots/botId/aggregatedutterances/ HTTP/1.1
Content-type: application/json

{
   "aggregationDuration": {
      "relativeAggregationDuration": {
         "timeDimension": "string",
         "timeValue": number
      }
   },
   "botAliasId": "string",
   "botVersion": "string",
   "filters": [
      {
         "name": "string",
         "operator": "string",
         "values": [ "string" ]
      }
   ],
   "localeId": "string",
   "maxResults": number,
   "nextToken": "string",
   "sortBy": {
      "attribute": "string",
      "order": "string"
   }
}
```

URI Request Parameters

The request uses the following URI parameters.

**botId (p. 476)**

The unique identifier of the bot associated with this request.

Length Constraints: Fixed length of 10.
Pattern: ^[0-9a-zA-Z]+$  
Required: Yes

**Request Body**
The request accepts the following data in JSON format.

**aggregationDuration (p. 476)**
The time window for aggregating the utterance information. You can specify a time between one hour and two weeks.

Type: UtteranceAggregationDuration (p. 777) object

Required: Yes

**botAliasId (p. 476)**
The identifier of the bot alias associated with this request. If you specify the bot alias, you can't specify the bot version.

Type: String

Length Constraints: Fixed length of 10.

Pattern: ^\bTSTALIASID\b[0-9a-zA-Z]+$  
Required: No

**botVersion (p. 476)**
The identifier of the bot version associated with this request. If you specify the bot version, you can't specify the bot alias.

Type: String


Pattern: ^(DRAFT|[0-9]+)$  
Required: No

**filters (p. 476)**
Provides the specification of a filter used to limit the utterances in the response to only those that match the filter specification. You can only specify one filter and one string to filter on.

Type: Array of AggregatedUtterancesFilter (p. 656) objects

Array Members: Fixed number of 1 item.

Required: No

**localeId (p. 476)**
The identifier of the language and locale where the utterances were collected. For more information, see Supported languages.

Type: String

Required: Yes
**maxResults (p. 476)**

The maximum number of utterances to return in each page of results. If there are fewer results than
the maximum page size, only the actual number of results are returned. If you don't specify the
`maxResults` parameter, 1,000 results are returned.

Type: Integer

Valid Range: Minimum value of 1. Maximum value of 1000.

Required: No

**nextToken (p. 476)**

If the response from the `ListAggregatedUtterances` operation contains more results that
specified in the `maxResults` parameter, a token is returned in the response. Use that token in the
`nextToken` parameter to return the next page of results.

Type: String

Required: No

**sortBy (p. 476)**

Specifies sorting parameters for the list of utterances. You can sort by the hit count, the missed
count, or the number of distinct sessions the utterance appeared in.

Type: `AggregatedUtterancesSortBy` (p. 657) object

Required: No

**Response Syntax**

```
HTTP/1.1 200
Content-type: application/json

{
   "aggregatedUtterancesSummaries": [
      {
         "containsDataFromDeletedResources": boolean,
         "hitCount": number,
         "missedCount": number,
         "utterance": string,
         "utteranceFirstRecordedInAggregationDuration": number,
         "utteranceLastRecordedInAggregationDuration": number
      }
   ],
   "aggregationDuration": {
      "relativeAggregationDuration": {
         "timeDimension": string,
         "timeValue": number
      }
   },
   "aggregationLastRefreshedDateTime": number,
   "aggregationWindowEndTime": number,
   "aggregationWindowStartTime": number,
   "botAliasId": string,
   "botId": string,
   "botVersion": string,
   "localeId": 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.

aggregatedUtterancesSummaries (p. 478)

Summaries of the aggregated utterance data. Each response contains information about the number of times that the utterance was seen during the time period, whether it was detected or missed, and when it was seen during the time period.

Type: Array of AggregatedUtterancesSummary (p. 658) objects

aggregationDuration (p. 478)

The time period used to aggregate the utterance data.

Type: UtteranceAggregationDuration (p. 777) object

aggregationLastRefreshedDateTime (p. 478)

The last date and time that the aggregated data was collected. The time period depends on the length of the aggregation window.

• Hours - for 1 hour time window, every half hour; otherwise every hour.
• Days - every 6 hours
• Weeks - for a one week time window, every 12 hours; otherwise, every day

Type: Timestamp

aggregationWindowEndTime (p. 478)

The date and time that the aggregation window ends. Only data collected between the start time and the end time are returned in the results.

Type: Timestamp

aggregationWindowStartTime (p. 478)

The date and time that the aggregation window begins. Only data collected after this time is returned in the results.

Type: Timestamp

botAliasId (p. 478)

The identifier of the bot alias that contains the utterances. If you specified the bot version, the bot alias ID isn't returned.

Type: String

Length Constraints: Fixed length of 10.

Pattern: ^(<bTSTALIASID><b|<0-9a-zA-Z>+)$

botId (p. 478)

The identifier of the bot that contains the utterances.

Type: String

Length Constraints: Fixed length of 10.

Pattern: ^[0-9a-zA-Z]+$
botVersion (p. 478)

The identifier of the bot version that contains the utterances. If you specified the bot alias, the bot version isn't returned.

Type: String


Pattern: ^\(DRAFT|[0-9]+\)$

localeId (p. 478)

The identifier of the language and locale that the utterances are in.

Type: String

nextToken (p. 478)

A token that indicates whether there are more results to return in a response to the ListAggregatedUtterances operation. If the nextToken field is present, you send the contents as the nextToken parameter of a ListAggregatedUtterances operation request to get the next page of results.

Type: String

Errors

For information about the errors that are common to all actions, see Common Errors (p. 821).

InternalServerException

The service encountered an unexpected condition. Try your request again.

HTTP Status Code: 500

PreconditionFailedException

Your request couldn't be completed because one or more request fields aren't valid. Check the fields in your request and try again.

HTTP Status Code: 412

ThrottlingException

Your request rate is too high. Reduce the frequency of requests.

HTTP Status Code: 429

ValidationException

One of the input parameters in your request isn't valid. Check the parameters and try your request again.

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 V2
- AWS SDK for JavaScript
- AWS SDK for PHP V3
- AWS SDK for Python
- AWS SDK for Ruby V3
ListBotAliases
Service: Amazon Lex Model Building V2

Gets a list of aliases for the specified bot.

Request Syntax

```plaintext
POST /bots/botId/botAliases/ HTTP/1.1
Content-type: application/json
{
  "maxResults": number,
  "nextToken": "string"
}
```

URI Request Parameters

The request uses the following URI parameters.

**botId (p. 482)**

The identifier of the bot to list aliases for.

Length Constraints: Fixed length of 10.

Pattern: `^[0-9a-zA-Z]+$`

Required: Yes

Request Body

The request accepts the following data in JSON format.

**maxResults (p. 482)**

The maximum number of aliases to return in each page of results. If there are fewer results than the max page size, only the actual number of results are returned.

Type: Integer

Valid Range: Minimum value of 1. Maximum value of 1000.

Required: No

**nextToken (p. 482)**

If the response from the ListBotAliases operation contains more results than specified in the maxResults parameter, a token is returned in the response. Use that token in the nextToken parameter to return the next page of results.

Type: String

Required: No

Response Syntax

```
HTTP/1.1 200
Content-type: application/json
```
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.

**botAliasSummaries (p. 482)**

Summary information for the bot aliases that meet the filter criteria specified in the request. The length of the list is specified in the maxResults parameter of the request. If there are more aliases available, the nextToken field contains a token to get the next page of results.

Type: Array of BotAliasSummary (p. 666) objects

**botId (p. 482)**

The identifier of the bot associated with the aliases.

Type: String

Length Constraints: Fixed length of 10.

Pattern: ^[0-9a-zA-Z]+$

**nextToken (p. 482)**

A token that indicates whether there are more results to return in a response to the ListBotAliases operation. If the nextToken field is present, you send the contents as the nextToken parameter of a ListBotAliases operation request to get the next page of results.

Type: String

Errors

For information about the errors that are common to all actions, see Common Errors (p. 821).

**InternalServerException**

The service encountered an unexpected condition. Try your request again.

HTTP Status Code: 500

**ServiceQuotaExceededException**

You have reached a quota for your bot.
HTTP Status Code: 402

**ThrottlingException**

Your request rate is too high. Reduce the frequency of requests.

HTTP Status Code: 429

**ValidationException**

One of the input parameters in your request isn't valid. Check the parameters and try your request again.

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 V2](#)
- [AWS SDK for JavaScript](#)
- [AWS SDK for PHP V3](#)
- [AWS SDK for Python](#)
- [AWS SDK for Ruby V3](#)
**ListBotLocales**

Service: Amazon Lex Model Building V2

Gets a list of locales for the specified bot.

**Request Syntax**

```plaintext
POST /bots/botId/botversions/botVersion/botlocales/ HTTP/1.1
Content-type: application/json

{
    "filters": [
        {
            "name": "string",
            "operator": "string",
            "values": [ "string" ]
        }
    ],
    "maxResults": number,
    "nextToken": "string",
    "sortBy": {
        "attribute": "string",
        "order": "string"
    }
}
```

**URI Request Parameters**

The request uses the following URI parameters.

**botId (p. 485)**

The identifier of the bot to list locales for.

- Pattern: `^[0-9a-zA-Z]+$`
  
  Required: Yes

**botVersion (p. 485)**

The version of the bot to list locales for.

  
  Pattern: `^(DRAFT|[0-9]+)$`
  
  Required: Yes

**Request Body**

The request accepts the following data in JSON format.

**filters (p. 485)**

Provides the specification for a filter used to limit the response to only those locales that match the filter specification. You can only specify one filter and one value to filter on.

Type: Array of `BotLocaleFilter (p. 673)` objects
Array Members: Fixed number of 1 item.

Required: No

maxResults (p. 485)

The maximum number of aliases to return in each page of results. If there are fewer results than the max page size, only the actual number of results are returned.

Type: Integer

Valid Range: Minimum value of 1. Maximum value of 1000.

Required: No

nextToken (p. 485)

If the response from the ListBotLocales operation contains more results than specified in the maxResults parameter, a token is returned in the response. Use that token as the nextToken parameter to return the next page of results.

Type: String

Required: No

sortBy (p. 485)

Specifies sorting parameters for the list of locales. You can sort by locale name in ascending or descending order.

Type: BotLocaleSortBy (p. 677) object

Required: No

Response Syntax

HTTP/1.1 200
Content-type: application/json

```
{
    "botId": "string",
    "botLocaleSummaries": [
        {
            "botLocaleStatus": "string",
            "description": "string",
            "lastBuildSubmittedDateTime": number,
            "lastUpdatedDateTime": number,
            "localeId": "string",
            "localeName": "string"
        }
    ],
    "botVersion": "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.

botId (p. 486)

The identifier of the bot to list locales for.
Type: String
Length Constraints: Fixed length of 10.
Pattern: ^[0-9a-zA-Z]+$  
**botLocaleSummaries (p. 486)**

Summary information for the locales that meet the filter criteria specified in the request. The length of the list is specified in the `maxResults` parameter of the request. If there are more locales available, the `nextToken` field contains a token to get the next page of results.

Type: Array of **BotLocaleSummary (p. 678)** objects

**botVersion (p. 486)**

The version of the bot.

Type: String
Pattern: ^(DRAFT|[0-9]+)$

**nextToken (p. 486)**

A token that indicates whether there are more results to return in a response to the `ListBotLocales` operation. If the `nextToken` field is present, you send the contents as the `nextToken` parameter of a `ListBotLocales` operation request to get the next page of results.

Type: String

**Errors**

For information about the errors that are common to all actions, see Common Errors (p. 821).

**InternalServerException**

The service encountered an unexpected condition. Try your request again.

HTTP Status Code: 500

**ServiceQuotaExceededException**

You have reached a quota for your bot.

HTTP Status Code: 402

**ThrottlingException**

Your request rate is too high. Reduce the frequency of requests.

HTTP Status Code: 429

**ValidationException**

One of the input parameters in your request isn't valid. Check the parameters and try your request again.

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 V2
• AWS SDK for JavaScript
• AWS SDK for PHP V3
• AWS SDK for Python
• AWS SDK for Ruby V3
**ListBotRecommendations**  
Service: Amazon Lex Model Building V2

Get a list of bot recommendations that meet the specified criteria.

**Request Syntax**

```
POST /bots/{botId}/botversions/{botVersion}/botlocales/{localeId}/botrecommendations/ HTTP/1.1
Content-type: application/json

{
  "maxResults": number,
  "nextToken": "string"
}
```

**URI Request Parameters**

The request uses the following URI parameters.

**botId** (p. 489)

The unique identifier of the bot that contains the bot recommendation list.

Length Constraints: Fixed length of 10.

Pattern: `^[0-9a-zA-Z]+$`

Required: Yes

**botVersion** (p. 489)

The version of the bot that contains the bot recommendation list.

Length Constraints: Fixed length of 5.

Pattern: `^DRAFT$`

Required: Yes

**localeId** (p. 489)

The identifier of the language and locale of the bot recommendation list.

Required: Yes

**Request Body**

The request accepts the following data in JSON format.

**maxResults** (p. 489)

The maximum number of bot recommendations to return in each page of results. If there are fewer results than the max page size, only the actual number of results are returned.

Type: Integer

Valid Range: Minimum value of 1. Maximum value of 1000.

Required: No
nextToken (p. 489)

If the response from the ListBotRecommendation operation contains more results than specified in the maxResults parameter, a token is returned in the response. Use that token in the nextToken parameter to return the next page of results.

Type: String

Required: No

Response Syntax

HTTP/1.1 200
Content-type: application/json

{   "botId": "string",
"botRecommendationSummaries": [
   {
      "botRecommendationId": "string",
      "botRecommendationStatus": "string",
      "creationDateTime": number,
      "lastUpdatedDateTime": number
   }
   ],
"botVersion": "string",
"localeId": "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.

botId (p. 490)

The unique identifier of the bot that contains the bot recommendation list.

Type: String

Length Constraints: Fixed length of 10.

Pattern: ^[0-9a-zA-Z]+$

botRecommendationSummaries (p. 490)

Summary information for the bot recommendations that meet the filter specified in this request. The length of the list is specified in the maxResults parameter of the request. If there are more bot recommendations available, the nextToken field contains a token to get the next page of results.

Type: Array of BotRecommendationSummary (p. 682) objects

botVersion (p. 490)

The version of the bot that contains the bot recommendation list.

Type: String

Length Constraints: Fixed length of 5.
localeId (p. 490)

The identifier of the language and locale of the bot recommendation list.

Type: String

nextToken (p. 490)

A token that indicates whether there are more results to return in a response to the ListBotRecommendations operation. If the nextToken field is present, you send the contents as the nextToken parameter of a ListBotRecommendations operation request to get the next page of results.

Type: String

Errors

For information about the errors that are common to all actions, see Common Errors (p. 821).

InternalServerException

The service encountered an unexpected condition. Try your request again.

HTTP Status Code: 500

ResourceNotFoundException

You asked to describe a resource that doesn't exist. Check the resource that you are requesting and try again.

HTTP Status Code: 404

ThrottlingException

Your request rate is too high. Reduce the frequency of requests.

HTTP Status Code: 429

ValidationException

One of the input parameters in your request isn't valid. Check the parameters and try your request again.

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 V2
- AWS SDK for JavaScript
- AWS SDK for PHP V3
- AWS SDK for Python
- AWS SDK for Ruby V3
ListBots
Service: Amazon Lex Model Building V2

Gets a list of available bots.

Request Syntax

```
POST /bots/ HTTP/1.1
Content-type: application/json

{
  "filters": [
    {
      "name": "string",
      "operator": "string",
      "values": [ "string" ]
    }
  ],
  "maxResults": number,
  "nextToken": "string",
  "sortBy": {
    "attribute": "string",
    "order": "string"
  }
}
```

URI Request Parameters

The request does not use any URI parameters.

Request Body

The request accepts the following data in JSON format.

**filters (p. 493)**

Provides the specification of a filter used to limit the bots in the response to only those that match the filter specification. You can only specify one filter and one string to filter on.

Type: Array of BotFilter (p. 669) objects

Array Members: Fixed number of 1 item.

Required: No

**maxResults (p. 493)**

The maximum number of bots to return in each page of results. If there are fewer results than the maximum page size, only the actual number of results are returned.

Type: Integer

Valid Range: Minimum value of 1. Maximum value of 1000.

Required: No

**nextToken (p. 493)**

If the response from the ListBots operation contains more results than specified in the maxResults parameter, a token is returned in the response.
Use the returned token in the `nextToken` parameter of a `ListBots` request to return the next page of results. For a complete set of results, call the `ListBots` operation until the `nextToken` returned in the response is null.

Type: String
Required: No

**sortBy (p. 493)**

Specifies sorting parameters for the list of bots. You can specify that the list be sorted by bot name in ascending or descending order.

Type: BotSortBy (p. 683) object
Required: No

**Response Syntax**

```
HTTP/1.1 200
Content-type: application/json

{
  "botSummaries": [
    {
      "botId": "string",
      "botName": "string",
      "botStatus": "string",
      "description": "string",
      "lastUpdatedDateTime": number,
      "latestBotVersion": "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.

**botSummaries (p. 494)**

Summary information for the bots that meet the filter criteria specified in the request. The length of the list is specified in the `maxResults` parameter of the request. If there are more bots available, the `nextToken` field contains a token to the next page of results.

Type: Array of BotSummary (p. 684) objects

**nextToken (p. 494)**

A token that indicates whether there are more results to return in a response to the `ListBots` operation. If the `nextToken` field is present, you send the contents as the `nextToken` parameter of a `ListBots` operation request to get the next page of results.

Type: String

**Errors**

For information about the errors that are common to all actions, see Common Errors (p. 821).
InternalServerException

The service encountered an unexpected condition. Try your request again.

HTTP Status Code: 500

ServiceQuotaExceededException

You have reached a quota for your bot.

HTTP Status Code: 402

ThrottlingException

Your request rate is too high. Reduce the frequency of requests.

HTTP Status Code: 429

ValidationException

One of the input parameters in your request isn’t valid. Check the parameters and try your request again.

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 V2
- AWS SDK for JavaScript
- AWS SDK for PHP V3
- AWS SDK for Python
- AWS SDK for Ruby V3
ListBotVersions
Service: Amazon Lex Model Building V2

Gets information about all of the versions of a bot.

The ListBotVersions operation returns a summary of each version of a bot. For example, if a bot has three numbered versions, the ListBotVersions operation returns for summaries, one for each numbered version and one for the DRAFT version.

The ListBotVersions operation always returns at least one version, the DRAFT version.

Request Syntax

```
POST /bots/botId/botversions/ HTTP/1.1
Content-type: application/json

{
    "maxResults": number,
    "nextToken": "string",
    "sortBy": {
        "attribute": "string",
        "order": "string"
    }
}
```

URI Request Parameters

The request uses the following URI parameters.

**botId (p. 496)**

The identifier of the bot to list versions for.

Length Constraints: Fixed length of 10.

Pattern: `^[0-9a-zA-Z]+$`

Required: Yes

Request Body

The request accepts the following data in JSON format.

**maxResults (p. 496)**

The maximum number of versions to return in each page of results. If there are fewer results than the max page size, only the actual number of results are returned.

Type: Integer

Valid Range: Minimum value of 1. Maximum value of 1000.

Required: No

**nextToken (p. 496)**

If the response to the ListBotVersion operation contains more results than specified in the maxResults parameter, a token is returned in the response. Use that token in the nextToken parameter to return the next page of results.
sortBy (p. 496)

Specifies sorting parameters for the list of versions. You can specify that the list be sorted by version name in either ascending or descending order.

Type: BotVersionSortBy (p. 687) object

Required: No

Response Syntax

HTTP/1.1 200
Content-type: application/json

{
  "botId": "string",
  "botVersionSummaries": [
    {
      "botName": "string",
      "botStatus": "string",
      "botVersion": "string",
      "creationDateTime": number,
      "description": "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.

botId (p. 497)

The identifier of the bot to list versions for.

Type: String

Length Constraints: Fixed length of 10.

Pattern: ^[0-9a-zA-Z]+$

botVersionSummaries (p. 497)

Summary information for the bot versions that meet the filter criteria specified in the request. The length of the list is specified in the maxResults parameter of the request. If there are more versions available, the nextToken field contains a token to get the next page of results.

Type: Array of BotVersionSummary (p. 688) objects

nextToken (p. 497)

A token that indicates whether there are more results to return in a response to the ListBotVersions operation. If the nextToken field is present, you send the contents as the nextToken parameter of a ListBotAliases operation request to get the next page of results.

Type: String
Errors

For information about the errors that are common to all actions, see Common Errors (p. 821).

InternalServerException

The service encountered an unexpected condition. Try your request again.

HTTP Status Code: 500

ServiceQuotaExceededException

You have reached a quota for your bot.

HTTP Status Code: 402

ThrottlingException

Your request rate is too high. Reduce the frequency of requests.

HTTP Status Code: 429

ValidationException

One of the input parameters in your request isn't valid. Check the parameters and try your request again.

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 V2
- AWS SDK for JavaScript
- AWS SDK for PHP V3
- AWS SDK for Python
- AWS SDK for Ruby V3
ListBuiltInIntents
Service: Amazon Lex Model Building V2

Gets a list of built-in intents provided by Amazon Lex that you can use in your bot.

To use a built-in intent as a the base for your own intent, include the built-in intent signature in
the parentIntentSignature parameter when you call the CreateIntent operation. For more
information, see CreateIntent.

Request Syntax

```json
POST /builtins/locales/localeId/intents/ HTTP/1.1
Content-type: application/json
{
  "maxResults": number,
  "nextToken": "string",
  "sortBy": {
    "attribute": "string",
    "order": "string"
  }
}
```

URI Request Parameters

The request uses the following URI parameters.

localeId (p. 499)

The identifier of the language and locale of the intents to list. The string must match one of the
supported locales. For more information, see Supported languages.

Required: Yes

Request Body

The request accepts the following data in JSON format.

maxResults (p. 499)

The maximum number of built-in intents to return in each page of results. If there are fewer results
than the max page size, only the actual number of results are returned.

Type: Integer


Required: No

nextToken (p. 499)

If the response from the ListBuiltInIntents operation contains more results than specified in
the maxResults parameter, a token is returned in the response. Use that token in the nextToken
parameter to return the next page of results.

Type: String

Required: No
sortBy (p. 499)

Specifies sorting parameters for the list of built-in intents. You can specify that the list be sorted by
the built-in intent signature in either ascending or descending order.

Type: BuiltInIntentSortBy (p. 690) object

Required: No

Response Syntax

HTTP/1.1 200
Content-type: application/json

{
   "builtInIntentSummaries": [
      {
         "description": "string",
         "intentSignature": "string"
      }
   ],
   "localeId": "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.

builtInIntentSummaries (p. 500)

Summary information for the built-in intents that meet the filter criteria specified in the request. The
length of the list is specified in the maxResults parameter of the request. If there are more
intents available, the nextToken field contains a token to get the next page of results.

Type: Array of BuiltInIntentSummary (p. 691) objects

localeId (p. 500)

The language and locale of the intents in the list.

Type: String

nextToken (p. 500)

A token that indicates whether there are more results to return in a response to the
ListBuiltInIntents operation. If the nextToken field is present, you send the contents as the
nextToken parameter of a ListBotAliases operation request to get the next page of results.

Type: String

Errors

For information about the errors that are common to all actions, see Common Errors (p. 821).

InternalServerErrorException

The service encountered an unexpected condition. Try your request again.
HTTP Status Code: 500

**ServiceQuotaExceededException**

You have reached a quota for your bot.

HTTP Status Code: 402

**ThrottlingException**

Your request rate is too high. Reduce the frequency of requests.

HTTP Status Code: 429

**ValidationException**

One of the input parameters in your request isn’t valid. Check the parameters and try your request again.

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 V2
- AWS SDK for JavaScript
- AWS SDK for PHP V3
- AWS SDK for Python
- AWS SDK for Ruby V3
ListBuiltInSlotTypes
Service: Amazon Lex Model Building V2

Gets a list of built-in slot types that meet the specified criteria.

Request Syntax

POST /builtins/locales//localeId/slottypes/ HTTP/1.1
Content-type: application/json

{  "maxResults": number,
   "nextToken": "string",
   "sortBy": {
      "attribute": "string",
      "order": "string"
   }
}

URI Request Parameters

The request uses the following URI parameters.

localeId (p. 502)

The identifier of the language and locale of the slot types to list. The string must match one of the supported locales. For more information, see Supported languages.

Required: Yes

Request Body

The request accepts the following data in JSON format.

maxResults (p. 502)

The maximum number of built-in slot types to return in each page of results. If there are fewer results than the max page size, only the actual number of results are returned.

Type: Integer


Required: No

nextToken (p. 502)

If the response from the ListBuiltInSlotTypes operation contains more results than specified in the maxResults parameter, a token is returned in the response. Use that token in the nextToken parameter to return the next page of results.

Type: String

Required: No

sortBy (p. 502)

Determines the sort order for the response from the ListBuiltInSlotTypes operation. You can choose to sort by the slot type signature in either ascending or descending order.

Type: BuiltInSlotTypeSortBy (p. 692) object
Required: No

Response Syntax

```json
HTTP/1.1 200
Content-type: application/json
{
   "builtInSlotTypeSummaries": [
      {
         "description": "string",
         "slotTypeSignature": "string"
      },
      {
         "localeId": "string",
         "nextToken": "string"
      }
   ],
   "localeId": "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.

**builtInSlotTypeSummaries (p. 503)**

Summary information for the built-in slot types that meet the filter criteria specified in the request. The length of the list is specified in the `maxResults` parameter of the request. If there are more slot types available, the `nextToken` field contains a token to get the next page of results.

Type: Array of `BuiltInSlotTypeSummary (p. 693)` objects

**localeId (p. 503)**

The language and locale of the slot types in the list.

Type: String

**nextToken (p. 503)**

A token that indicates whether there are more results to return in a response to the `ListBuiltInSlotTypes` operation. If the `nextToken` field is present, you send the contents as the `nextToken` parameter of a `ListBuiltInSlotTypes` operation request to get the next page of results.

Type: String

Errors

For information about the errors that are common to all actions, see [Common Errors (p. 821)](#).

**InternalServerException**

The service encountered an unexpected condition. Try your request again.

HTTP Status Code: 500

**ServiceQuotaExceededException**

You have reached a quota for your bot.
HTTP Status Code: 402

**ThrottlingException**

Your request rate is too high. Reduce the frequency of requests.

HTTP Status Code: 429

**ValidationException**

One of the input parameters in your request isn’t valid. Check the parameters and try your request again.

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 V2
- AWS SDK for JavaScript
- AWS SDK for PHP V3
- AWS SDK for Python
- AWS SDK for Ruby V3
ListExports
Service: Amazon Lex Model Building V2

Lists the exports for a bot, bot locale, or custom vocabulary. Exports are kept in the list for 7 days.

Request Syntax

```json
POST /exports/ HTTP/1.1
Content-type: application/json

{
  "botId": "string",
  "botVersion": "string",
  "filters": [ 
    {
      "name": "string",
      "operator": "string",
      "values": [ "string" ]
    } ],
  "localeId": "string",
  "maxResults": number,
  "nextToken": "string",
  "sortBy": {
    "attribute": "string",
    "order": "string"
  }
}
```

URI Request Parameters

The request does not use any URI parameters.

Request Body

The request accepts the following data in JSON format.

`botId` (p. 505)

The unique identifier that Amazon Lex assigned to the bot.

Type: String

Length Constraints: Fixed length of 10.

Pattern: ^[0-9a-zA-Z]+$

Required: No

`botVersion` (p. 505)

The version of the bot to list exports for.

Type: String


Pattern: ^(DRAFT|[0-9]+)$

Required: No
filters (p. 505)

Provides the specification of a filter used to limit the exports in the response to only those that match the filter specification. You can only specify one filter and one string to filter on.

Type: Array of ExportFilter (p. 705) objects

Array Members: Fixed number of 1 item.

Required: No

localeId (p. 505)

Specifies the resources that should be exported. If you don’t specify a resource type in the filters parameter, both bot locales and custom vocabularies are exported.

Type: String

Required: No

maxResults (p. 505)

The maximum number of exports to return in each page of results. If there are fewer results than the max page size, only the actual number of results are returned.

Type: Integer

Valid Range: Minimum value of 1. Maximum value of 1000.

Required: No

nextToken (p. 505)

If the response from the ListExports operation contains more results that specified in the maxResults parameter, a token is returned in the response.

Use the returned token in the nextToken parameter of a ListExports request to return the next page of results. For a complete set of results, call the ListExports operation until the nextToken returned in the response is null.

Type: String

Required: No

sortBy (p. 505)

Determines the field that the list of exports is sorted by. You can sort by the LastUpdatedDateTime field in ascending or descending order.

Type: ExportSortBy (p. 707) object

Required: No

Response Syntax

HTTP/1.1 200
Content-type: application/json

{
  "botId": "string",
  "botVersion": "string",
  "exportSummaries": [
    
  ]
}
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.

**botId (p. 506)**

The unique identifier assigned to the bot by Amazon Lex.

Type: String

Length Constraints: Fixed length of 10.

Pattern: ^[0-9a-zA-Z]+$  

**botVersion (p. 506)**

The version of the bot that was exported.

Type: String


Pattern: ^(DRAFT|[0-9]+)\$

**exportSummaries (p. 506)**

Summary information for the exports that meet the filter criteria specified in the request. The length of the list is specified in the maxResults parameter. If there are more exports available, the nextToken field contains a token to get the next page of results.

Type: Array of ExportSummary (p. 708) objects

**localeId (p. 506)**

The locale specified in the request.
Type: String

nextToken (p. 506)

A token that indicates whether there are more results to return in a response to the ListExports operation. If the nextToken field is present, you send the contents as the nextToken parameter of a ListExports operation request to get the next page of results.

Type: String

Errors

For information about the errors that are common to all actions, see Common Errors (p. 821).

InternalServerException

The service encountered an unexpected condition. Try your request again.

HTTP Status Code: 500

ThrottlingException

Your request rate is too high. Reduce the frequency of requests.

HTTP Status Code: 429

ValidationException

One of the input parameters in your request isn't valid. Check the parameters and try your request again.

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 V2
- AWS SDK for JavaScript
- AWS SDK for PHP V3
- AWS SDK for Python
- AWS SDK for Ruby V3
ListImports
Service: Amazon Lex Model Building V2

Lists the imports for a bot, bot locale, or custom vocabulary. Imports are kept in the list for 7 days.

Request Syntax

```json
POST /imports/ HTTP/1.1
Content-type: application/json

{
    "botId": "string",
    "botVersion": "string",
    "filters": [
        {
            "name": "string",
            "operator": "string",
            "values": [ "string" ]
        }
    ],
    "localeId": "string",
    "maxResults": number,
    "nextToken": "string",
    "sortBy": {
        "attribute": "string",
        "order": "string"
    }
}
```

URI Request Parameters

The request does not use any URI parameters.

Request Body

The request accepts the following data in JSON format.

**botId (p. 509)**

The unique identifier that Amazon Lex assigned to the bot.

Type: String

Length Constraints: Fixed length of 10.

Pattern: `^[0-9a-zA-Z]+$`

Required: No

**botVersion (p. 509)**

The version of the bot to list imports for.

Type: String

Length Constraints: Fixed length of 5.

Pattern: `^DRAFT$`

Required: No
filters (p. 509)

Provides the specification of a filter used to limit the bots in the response to only those that match the filter specification. You can only specify one filter and one string to filter on.

Type: Array of ImportFilter (p. 719) objects

Array Members: Fixed number of 1 item.

Required: No

localeId (p. 509)

Specifies the locale that should be present in the list. If you don't specify a resource type in the filters parameter, the list contains both bot locales and custom vocabularies.

Type: String

Required: No

maxResults (p. 509)

The maximum number of imports to return in each page of results. If there are fewer results than the max page size, only the actual number of results are returned.

Type: Integer

Valid Range: Minimum value of 1. Maximum value of 1000.

Required: No

nextToken (p. 509)

If the response from the ListImports operation contains more results than specified in the maxResults parameter, a token is returned in the response.

Use the returned token in the nextToken parameter of a ListImports request to return the next page of results. For a complete set of results, call the ListImports operation until the nextToken returned in the response is null.

Type: String

Required: No

sortBy (p. 509)

Determines the field that the list of imports is sorted by. You can sort by the LastUpdatedDateTime field in ascending or descending order.

Type: ImportSortBy (p. 721) object

Required: No

Response Syntax

HTTP/1.1 200
Content-type: application/json

{
   "botId": "string",
   "botVersion": "string",
   "importSummaries": [
      {
         "creationDateTime": number,
      },
   ]
}
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.

**botId (p. 510)**

The unique identifier assigned by Amazon Lex to the bot.

Type: String

Length Constraints: Fixed length of 10.

Pattern: ^[0-9a-zA-Z]+$  

**botVersion (p. 510)**

The version of the bot that was imported. It will always be DRAFT.

Type: String

Length Constraints: Fixed length of 5.

Pattern: ^DRAFT$

**importSummaries (p. 510)**

Summary information for the imports that meet the filter criteria specified in the request. The length of the list is specified in the maxResults parameter. If there are more imports available, the nextToken field contains a token to get the next page of results.

Type: Array of ImportSummary (p. 722) objects

**localeId (p. 510)**

The locale specified in the request.

Type: String

**nextToken (p. 510)**

A token that indicates whether there are more results to return in a response to the ListImports operation. If the nextToken field is present, you send the contents as the nextToken parameter of a ListImports operation request to get the next page of results.

Type: String

Errors

For information about the errors that are common to all actions, see Common Errors (p. 821).
**InternalServerException**

The service encountered an unexpected condition. Try your request again.

HTTP Status Code: 500

**ThrottlingException**

Your request rate is too high. Reduce the frequency of requests.

HTTP Status Code: 429

**ValidationException**

One of the input parameters in your request isn’t valid. Check the parameters and try your request again.

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 V2
- AWS SDK for JavaScript
- AWS SDK for PHP V3
- AWS SDK for Python
- AWS SDK for Ruby V3
**ListIntents**

Service: Amazon Lex Model Building V2

Get a list of intents that meet the specified criteria.

**Request Syntax**

```
POST /bots/botId/botversions/botVersion/botlocales/localeId/intents/ HTTP/1.1
Content-type: application/json

{
  "filters": [
    {
      "name": "string",
      "operator": "string",
      "values": [ "string" ]
    }
  ],
  "maxResults": number,
  "nextToken": "string",
  "sortBy": {
    "attribute": "string",
    "order": "string"
  }
}
```

**URI Request Parameters**

The request uses the following URI parameters.

**botId (p. 513)**

The unique identifier of the bot that contains the intent.

Length Constraints: Fixed length of 10.

Pattern: `^[0-9a-zA-Z]+$`

Required: Yes

**botVersion (p. 513)**

The version of the bot that contains the intent.


Pattern: `^(DRAFT|[0-9]+)\$`

Required: Yes

**localeId (p. 513)**

The identifier of the language and locale of the intents to list. The string must match one of the supported locales. For more information, see Supported languages.

Required: Yes

**Request Body**

The request accepts the following data in JSON format.
**filters (p. 513)**

Provides the specification of a filter used to limit the intents in the response to only those that match the filter specification. You can only specify one filter and only one string to filter on.

Type: Array of IntentFilter (p. 727) objects

Array Members: Fixed number of 1 item.

Required: No

**maxResults (p. 513)**

The maximum number of intents to return in each page of results. If there are fewer results than the max page size, only the actual number of results are returned.

Type: Integer

Valid Range: Minimum value of 1. Maximum value of 1000.

Required: No

**nextToken (p. 513)**

If the response from the ListIntents operation contains more results than specified in the `maxResults` parameter, a token is returned in the response.

Use the returned token in the `nextToken` parameter of a ListIntents request to return the next page of results. For a complete set of results, call the ListIntents operation until the `nextToken` returned in the response is null.

Type: String

Required: No

**sortBy (p. 513)**

Determines the sort order for the response from the ListIntents operation. You can choose to sort by the intent name or last updated date in either ascending or descending order.

Type: IntentSortBy (p. 728) object

Required: No

**Response Syntax**

```json
HTTP/1.1 200
Content-type: application/json

{
    "botId": "string",
    "botVersion": "string",
    "intentSummaries": [*
        {
            "description": "string",
            "inputContexts": [*
                {
                    "name": "string"
                }*
            ],
            "intentId": "string",
            "intentName": "string",
            "lastUpdatedDateTime": number,
            "outputContexts": [*
```
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.

**botId (p. 514)**

The identifier of the bot that contains the intent.

Type: String

Length Constraints: Fixed length of 10.

Pattern: ^[0-9a-zA-Z]+$  

**botVersion (p. 514)**

The version of the bot that contains the intent.

Type: String


Pattern: ^(DRAFT|[0-9]+)$  

**intentSummaries (p. 514)**

Summary information for the intents that meet the filter criteria specified in the request. The length of the list is specified in the maxResults parameter of the request. If there are more intents available, the nextToken field contains a token to get the next page of results.

Type: Array of IntentSummary (p. 730) objects

**localeId (p. 514)**

The language and locale of the intents in the list.

Type: String

**nextToken (p. 514)**

A token that indicates whether there are more results to return in a response to the ListIntents operation. If the nextToken field is present, you send the contents as the nextToken parameter of a ListIntents operation request to get the next page of results.

Type: String

Errors

For information about the errors that are common to all actions, see Common Errors (p. 821).
InternalServerException

The service encountered an unexpected condition. Try your request again.

HTTP Status Code: 500

ServiceQuotaExceededException

You have reached a quota for your bot.

HTTP Status Code: 402

ThrottlingException

Your request rate is too high. Reduce the frequency of requests.

HTTP Status Code: 429

ValidationException

One of the input parameters in your request isn’t valid. Check the parameters and try your request again.

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 V2
- AWS SDK for JavaScript
- AWS SDK for PHP V3
- AWS SDK for Python
- AWS SDK for Ruby V3
ListRecommendedIntents
Service: Amazon Lex Model Building V2

Gets a list of recommended intents provided by the bot recommendation that you can use in your bot.

Request Syntax

```
POST /bots/{botId}/botversions/{botVersion}/botlocales/{localeId}/botrecommendations/{botRecommendationId}/intents HTTP/1.1
Content-type: application/json

{
  "maxResults": number,
  "nextToken": "string"
}
```

URI Request Parameters

The request uses the following URI parameters.

botId (p. 517)

The unique identifier of the bot associated with the recommended intents.

Length Constraints: Fixed length of 10.

Pattern: ^[0-9a-zA-Z]+$

Required: Yes

botRecommendationId (p. 517)

The identifier of the bot recommendation that contains the recommended intents.

Length Constraints: Fixed length of 10.

Pattern: ^[0-9a-zA-Z]+$

Required: Yes

botVersion (p. 517)

The version of the bot that contains the recommended intents.

Length Constraints: Fixed length of 5.

Pattern: ^DRAFT$

Required: Yes

localeId (p. 517)

The identifier of the language and locale of the recommended intents.

Required: Yes

Request Body

The request accepts the following data in JSON format.
maxResults (p. 517)

The maximum number of bot recommendations to return in each page of results. If there are fewer results than the max page size, only the actual number of results are returned.

Type: Integer

Valid Range: Minimum value of 1. Maximum value of 1000.

Required: No

nextToken (p. 517)

If the response from the ListRecommendedIntents operation contains more results than specified in the maxResults parameter, a token is returned in the response. Use that token in the nextToken parameter to return the next page of results.

Type: String

Required: No

Response Syntax

HTTP/1.1 200
Content-type: application/json

```
{
  "botId": "string",
  "botRecommendationId": "string",
  "botVersion": "string",
  "localeId": "string",
  "nextToken": "string",
  "summaryList": [
    {
      "intentId": "string",
      "intentName": "string",
      "sampleUtterancesCount": number
    }
  ]
}
```

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.

**botId (p. 518)**

The unique identifier of the bot associated with the recommended intent.

Type: String

Length Constraints: Fixed length of 10.

Pattern: ^[0-9a-zA-Z]+$

**botRecommendationId (p. 518)**

The identifier of the bot recommendation that contains the recommended intent.

Type: String
Length Constraints: Fixed length of 10.
Pattern: `^[0-9a-zA-Z]+$`

**botVersion** (p. 518)

The version of the bot that contains the intent.
Type: String
Length Constraints: Fixed length of 5.
Pattern: `^DRAFT$`

**localeId** (p. 518)

The identifier of the language and locale of the intents to list. The string must match one of the supported locales. For more information, see Supported languages.
Type: String

**nextToken** (p. 518)

A token that indicates whether there are more results to return in a response to the ListRecommendedIntents operation. If the nextToken field is present, you send the contents as the nextToken parameter of a ListRecommendedIntents operation request to get the next page of results.
Type: String

**summaryList** (p. 518)

Summary information for the intents that meet the filter criteria specified in the request. The length of the list is specified in the maxResults parameter of the request. If there are more intents available, the nextToken field contains a token to get the next page of results.
Type: Array of RecommendedIntentSummary (p. 745) objects

**Errors**

For information about the errors that are common to all actions, see Common Errors (p. 821).

**InternalServerException**

The service encountered an unexpected condition. Try your request again.

HTTP Status Code: 500

**ResourceNotFoundException**

You asked to describe a resource that doesn't exist. Check the resource that you are requesting and try again.

HTTP Status Code: 404

**ServiceQuotaExceededException**

You have reached a quota for your bot.

HTTP Status Code: 402

**ThrottlingException**

Your request rate is too high. Reduce the frequency of requests.
HTTP Status Code: 429

**ValidationException**

One of the input parameters in your request isn't valid. Check the parameters and try your request again.

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 V2
- AWS SDK for JavaScript
- AWS SDK for PHP V3
- AWS SDK for Python
- AWS SDK for Ruby V3
ListSlots
Service: Amazon Lex Model Building V2

Gets a list of slots that match the specified criteria.

Request Syntax

```
POST /bots/botId/botversions/botVersion/botlocales/localeId/intents/intentId/slots/
HTTP/1.1
Content-type: application/json

{
  "filters": [
    {
      "name": "string",
      "operator": "string",
      "values": [ "string" ]
    }
  ],
  "maxResults": number,
  "nextToken": "string",
  "sortBy": {
    "attribute": "string",
    "order": "string"
  }
}
```

URI Request Parameters

The request uses the following URI parameters.

**botId (p. 521)**

The identifier of the bot that contains the slot.

Length Constraints: Fixed length of 10.

Pattern: `^[0-9a-zA-Z]+$`

Required: Yes

**botVersion (p. 521)**

The version of the bot that contains the slot.


Pattern: `^(DRAFT|[0-9]+$`

Required: Yes

**intentId (p. 521)**

The unique identifier of the intent that contains the slot.

Length Constraints: Fixed length of 10.

Pattern: `^[0-9a-zA-Z]+$`

Required: Yes
localeId (p. 521)

The identifier of the language and locale of the slots to list. The string must match one of the supported locales. For more information, see Supported languages.

Required: Yes

Request Body

The request accepts the following data in JSON format.

filters (p. 521)

Provides the specification of a filter used to limit the slots in the response to only those that match the filter specification. You can only specify one filter and only one string to filter on.

Type: Array of SlotFilter (p. 756) objects

Array Members: Fixed number of 1 item.

Required: No

maxResults (p. 521)

The maximum number of slots to return in each page of results. If there are fewer results than the max page size, only the actual number of results are returned.

Type: Integer

Valid Range: Minimum value of 1. Maximum value of 1000.

Required: No

nextToken (p. 521)

If the response from the ListSlots operation contains more results than specified in the maxResults parameter, a token is returned in the response. Use that token in the nextToken parameter to return the next page of results.

Type: String

Required: No

sortBy (p. 521)

Determines the sort order for the response from the ListSlots operation. You can choose to sort by the slot name or last updated date in either ascending or descending order.

Type: SlotSortBy (p. 758) object

Required: No

Response Syntax

HTTP/1.1 200
Content-type: application/json

```json
{
  "botId": "string",
  "botVersion": "string",
  "intentId": "string",
```
"localeId": "string",
"nextToken": "string",
"slotSummaries": [
 {
 "description": "string",
 "lastUpdatedDateTime": number,
 "slotConstraint": "string",
 "slotId": "string",
 "slotName": "string",
 "slotTypeId": "string",
 "valueElicitationPromptSpecification": {
 "allowInterrupt": boolean,
 "maxRetries": number,
 "messageGroups": [
 {
 "message": {
 "customPayload": {
 "value": "string"
 },
 "imageResponseCard": {
 "buttons": [
 {"text": "string",
 "value": "string"
 }
 ],
 "imageUrl": "string",
 "subtitle": "string",
 "title": "string"
 },
 "plainTextMessage": {
 "value": "string"
 },
 "ssmlMessage": {
 "value": "string"
 }
 }
 },
 "variations": [
 {
 "customPayload": {
 "value": "string"
 },
 "imageResponseCard": {
 "buttons": [
 {"text": "string",
 "value": "string"
 }
 ],
 "imageUrl": "string",
 "subtitle": "string",
 "title": "string"
 },
 "plainTextMessage": {
 "value": "string"
 },
 "ssmlMessage": {
 "value": "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.

botId (p. 522)

The identifier of the bot that contains the slots.
Type: String
Length Constraints: Fixed length of 10.
Pattern: ^[0-9a-zA-Z]+$

botVersion (p. 522)

The version of the bot that contains the slots.
Type: String
Pattern: ^(DRAFT|[0-9]+$)

intentId (p. 522)

The identifier of the intent that contains the slots.
Type: String
Length Constraints: Fixed length of 10.
Pattern: ^[0-9a-zA-Z]+$

localeId (p. 522)

The language and locale of the slots in the list.
Type: String

nextToken (p. 522)

A token that indicates whether there are more results to return in a response to the ListSlots operation. If the nextToken field is present, you send the contents as the nextToken parameter of a ListSlots operation request to get the next page of results.
Type: String

slotSummaries (p. 522)

Summary information for the slots that meet the filter criteria specified in the request. The length of the list is specified in the maxResults parameter of the request. If there are more slots available, the nextToken field contains a token to get the next page of results.
Type: Array of SlotSummary (p. 759) objects

Errors

For information about the errors that are common to all actions, see Common Errors (p. 821).
**InternalServerException**

The service encountered an unexpected condition. Try your request again.

HTTP Status Code: 500

**ServiceQuotaExceededException**

You have reached a quota for your bot.

HTTP Status Code: 402

**ThrottlingException**

Your request rate is too high. Reduce the frequency of requests.

HTTP Status Code: 429

**ValidationException**

One of the input parameters in your request isn't valid. Check the parameters and try your request again.

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 V2
- AWS SDK for JavaScript
- AWS SDK for PHP V3
- AWS SDK for Python
- AWS SDK for Ruby V3
ListSlotTypes
Service: Amazon Lex Model Building V2

Gets a list of slot types that match the specified criteria.

Request Syntax

```json
POST /bots/botId/botversions/botVersion/botlocales/localeId/slottypes/ HTTP/1.1
Content-type: application/json

{  
  "filters": [  
    {  
      "name": "string",
      "operator": "string",
      "values": [ "string" ]
    }
  ],  
  "maxResults": number,
  "nextToken": "string",
  "sortBy": {  
    "attribute": "string",
    "order": "string"
  }
}
```

URI Request Parameters

The request uses the following URI parameters.

**botId (p. 526)**

The unique identifier of the bot that contains the slot types.

Length Constraints: Fixed length of 10.

Pattern: `^[0-9a-zA-Z]+$`

Required: Yes

**botVersion (p. 526)**

The version of the bot that contains the slot type.


Pattern: `^(DRAFT|[0-9]+)$`

Required: Yes

**localeId (p. 526)**

The identifier of the language and locale of the slot types to list. The string must match one of the supported locales. For more information, see Supported languages.

Required: Yes

**Request Body**

The request accepts the following data in JSON format.
**filters (p. 526)**

Provides the specification of a filter used to limit the slot types in the response to only those that match the filter specification. You can only specify one filter and only one string to filter on.

Type: Array of `SlotTypeFilter (p. 761)` objects

Array Members: Fixed number of 1 item.

Required: No

**maxResults (p. 526)**

The maximum number of slot types to return in each page of results. If there are fewer results than the max page size, only the actual number of results are returned.

Type: Integer

Valid Range: Minimum value of 1. Maximum value of 1000.

Required: No

**nextToken (p. 526)**

If the response from the `ListSlotTypes` operation contains more results than specified in the `maxResults` parameter, a token is returned in the response. Use that token in the `nextToken` parameter to return the next page of results.

Type: String

Required: No

**sortBy (p. 526)**

Determines the sort order for the response from the `ListSlotTypes` operation. You can choose to sort by the slot type name or last updated date in either ascending or descending order.

Type: `SlotTypeSortBy (p. 762)` object

Required: No

---

**Response Syntax**

```json
HTTP/1.1 200
Content-type: application/json

{
  "botId": "string",
  "botVersion": "string",
  "localeId": "string",
  "nextToken": "string",
  "slotTypeSummaries": [
    {
      "description": "string",
      "lastUpdatedDateTime": number,
      "parentSlotTypeSignature": "string",
      "slotTypeCategory": "string",
      "slotTypeId": "string",
      "slotTypeName": "string"
    }
  ]
}
```

527
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.

**botId (p. 527)**

The identifier of the bot that contains the slot types.

Type: String

Length Constraints: Fixed length of 10.

Pattern: ^[0-9a-zA-Z]+$

**botVersion (p. 527)**

The version of the bot that contains the slot types.

Type: String


Pattern: ^(DRAFT|[0-9]+$

**localeId (p. 527)**

The language and local of the slot types in the list.

Type: String

**nextToken (p. 527)**

A token that indicates whether there are more results to return in a response to the `ListSlotTypes` operation. If the `nextToken` field is present, you send the contents as the `nextToken` parameter of a `ListSlotTypes` operation request to get the next page of results.

Type: String

**slotTypeSummaries (p. 527)**

Summary information for the slot types that meet the filter criteria specified in the request. The length of the list is specified in the `maxResults` parameter of the request. If there are more slot types available, the `nextToken` field contains a token to get the next page of results.

Type: Array of `SlotTypeSummary (p. 764)` objects

Errors

For information about the errors that are common to all actions, see `Common Errors (p. 821)`.

**InternalServerException**

The service encountered an unexpected condition. Try your request again.

HTTP Status Code: 500

**ServiceQuotaExceededException**

You have reached a quota for your bot.

HTTP Status Code: 402
**ThrottlingException**

Your request rate is too high. Reduce the frequency of requests.

HTTP Status Code: 429

**ValidationException**

One of the input parameters in your request isn't valid. Check the parameters and try your request again.

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 V2
- AWS SDK for JavaScript
- AWS SDK for PHP V3
- AWS SDK for Python
- AWS SDK for Ruby V3
ListTagsForResource
Service: Amazon Lex Model Building V2

Gets a list of tags associated with a resource. Only bots, bot aliases, and bot channels can have tags associated with them.

Request Syntax

GET /tags/resourceARN HTTP/1.1

URI Request Parameters

The request uses the following URI parameters.

resourceARN (p. 530)

The Amazon Resource Name (ARN) of the resource to get a list of tags for.

Length Constraints: Minimum length of 1. Maximum length of 1011.

Required: Yes

Request Body

The request does not have a request body.

Response Syntax

HTTP/1.1 200
Content-type: application/json

{
  "tags": {
    "string" : "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.

tags (p. 530)

The tags associated with a resource.

Type: String to string map

Map Entries: Minimum number of 0 items. Maximum number of 200 items.

Key Length Constraints: Minimum length of 1. Maximum length of 128.

Value Length Constraints: Minimum length of 0. Maximum length of 256.

Errors

For information about the errors that are common to all actions, see Common Errors (p. 821).
InternalServerException

The service encountered an unexpected condition. Try your request again.

HTTP Status Code: 500

ResourceNotFoundException

You asked to describe a resource that doesn't exist. Check the resource that you are requesting and try again.

HTTP Status Code: 404

ThrottlingException

Your request rate is too high. Reduce the frequency of requests.

HTTP Status Code: 429

ValidationException

One of the input parameters in your request isn't valid. Check the parameters and try your request again.

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 V2
- AWS SDK for JavaScript
- AWS SDK for PHP V3
- AWS SDK for Python
- AWS SDK for Ruby V3
SearchAssociatedTranscripts
Service: Amazon Lex Model Building V2

Search for associated transcripts that meet the specified criteria.

Request Syntax

```json
POST /bots/botId/botversions/botVersion/botlocales/localeId/botrecommendations/botRecommendationId/associatedtranscripts HTTP/1.1
Content-type: application/json

{
  "filters": [?
    {?
      "name": "string",
      "values": [ "string" ]
    },
  ],
  "maxResults": number,
  "nextIndex": number,
  "searchOrder": "string"
}
```

URI Request Parameters

The request uses the following URI parameters.

**botId (p. 532)**

The unique identifier of the bot associated with the transcripts that you are searching.

Length Constraints: Fixed length of 10.

Pattern: ^[0-9a-zA-Z]+$

Required: Yes

**botRecommendationId (p. 532)**

The unique identifier of the bot recommendation associated with the transcripts to search.

Length Constraints: Fixed length of 10.

Pattern: ^[0-9a-zA-Z]+$

Required: Yes

**botVersion (p. 532)**

The version of the bot containing the transcripts that you are searching.


Pattern: ^(DRAFT|[0-9]+)$

Required: Yes

**localeId (p. 532)**

The identifier of the language and locale of the transcripts to search. The string must match one of the supported locales. For more information, see Supported languages

Required: Yes
Request Body

The request accepts the following data in JSON format.

filters (p. 532)

A list of filter objects.

Type: Array of AssociatedTranscriptFilter (p. 661) objects

Array Members: Fixed number of 1 item.

Required: Yes

maxResults (p. 532)

The maximum number of bot recommendations to return in each page of results. If there are fewer results than the max page size, only the actual number of results are returned.

Type: Integer

Valid Range: Minimum value of 1. Maximum value of 1000.

Required: No

nextIndex (p. 532)

If the response from the SearchAssociatedTranscriptsRequest operation contains more results than specified in the maxResults parameter, an index is returned in the response. Use that index in the nextIndex parameter to return the next page of results.

Type: Integer

Valid Range: Minimum value of 0. Maximum value of 10000000.

Required: No

searchOrder (p. 532)

How SearchResults are ordered. Valid values are Ascending or Descending. The default is Descending.

Type: String

Valid Values: Ascending | Descending

Required: No

Response Syntax

HTTP/1.1 200
Content-type: application/json

{
    "associatedTranscripts": [  
        {  
            "transcript": "string"
        }
    ],
    "botId": "string",
    "botRecommendationId": "string",
    "botVersion": "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.

associatedTranscripts (p. 533)

The object that contains the associated transcript that meet the criteria you specified.

Type: Array of AssociatedTranscript (p. 660) objects

botId (p. 533)

The unique identifier of the bot associated with the transcripts that you are searching.

Type: String

Length Constraints: Fixed length of 10.

Pattern: ^[0-9a-zA-Z]+$

botRecommendationId (p. 533)

The unique identifier of the bot recommendation associated with the transcripts to search.

Type: String

Length Constraints: Fixed length of 10.

Pattern: ^[0-9a-zA-Z]+$

botVersion (p. 533)

The version of the bot containing the transcripts that you are searching.

Type: String


Pattern: ^(DRAFT|[0-9]+)$

localeId (p. 533)

The identifier of the language and locale of the transcripts to search. The string must match one of the supported locales. For more information, see Supported languages

Type: String

nextIndex (p. 533)

A index that indicates whether there are more results to return in a response to the SearchAssociatedTranscripts operation. If the nextIndex field is present, you send the contents as the nextIndex parameter of a SearchAssociatedTranscriptsRequest operation to get the next page of results.

Type: Integer

Valid Range: Minimum value of 0. Maximum value of 10000000.
**totalResults (p. 533)**

The total number of transcripts returned by the search.

Type: Integer

Valid Range: Minimum value of 1. Maximum value of 1000.

**Errors**

For information about the errors that are common to all actions, see Common Errors (p. 821).

**InternalServerException**

The service encountered an unexpected condition. Try your request again.

HTTP Status Code: 500

**ResourceNotFoundException**

You asked to describe a resource that doesn’t exist. Check the resource that you are requesting and try again.

HTTP Status Code: 404

**ServiceQuotaExceededException**

You have reached a quota for your bot.

HTTP Status Code: 402

**ThrottlingException**

Your request rate is too high. Reduce the frequency of requests.

HTTP Status Code: 429

**ValidationException**

One of the input parameters in your request isn’t valid. Check the parameters and try your request again.

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 V2
- AWS SDK for JavaScript
- AWS SDK for PHP V3
- AWS SDK for Python
- AWS SDK for Ruby V3
StartBotRecommendation
Service: Amazon Lex Model Building V2

Use this to provide your transcript data, and to start the bot recommendation process.

Request Syntax

```plaintext
PUT /bots/botId/botversions/botVersion/botlocales/localeId/botrecommendations/ HTTP/1.1
Content-type: application/json

{
    "encryptionSetting": {
        "associatedTranscriptsPassword": "string",
        "botLocaleExportPassword": "string",
        "kmsKeyArn": "string"
    },
    "transcriptSourceSetting": {
        "s3BucketTranscriptSource": {
            "kmsKeyArn": "string",
            "pathFormat": {
                "objectPrefixes": [ "string" ]
            },
            "s3BucketName": "string",
            "transcriptFilter": {
                "lexTranscriptFilter": {
                    "dateRangeFilter": {
                        "endDateTime": number,
                        "startDateTime": number
                    }
                }
            }
        },
        "transcriptFormat": "string"
    }
}
```

URI Request Parameters

The request uses the following URI parameters.

**botId (p. 536)**

The unique identifier of the bot containing the bot recommendation.

Length Constraints: Fixed length of 10.

Pattern: ^[0-9a-zA-Z]+$

Required: Yes

**botVersion (p. 536)**

The version of the bot containing the bot recommendation.

Length Constraints: Fixed length of 5.

Pattern: ^DRAFT$

Required: Yes

**localeId (p. 536)**

The identifier of the language and locale of the bot recommendation to start. The string must match one of the supported locales. For more information, see Supported languages
Required: Yes

Request Body

The request accepts the following data in JSON format.

**encryptionSetting (p. 536)**

The object representing the passwords that will be used to encrypt the data related to the bot recommendation results, as well as the KMS key ARN used to encrypt the associated metadata.

Type: EncryptionSetting (p. 704) object

Required: No

**transcriptSourceSetting (p. 536)**

The object representing the Amazon S3 bucket containing the transcript, as well as the associated metadata.

Type: TranscriptSourceSetting (p. 776) object

Required: Yes

Response Syntax

```
HTTP/1.1 202
Content-type: application/json

{
   "botId": "string",
   "botRecommendationId": "string",
   "botRecommendationStatus": "string",
   "botVersion": "string",
   "creationDateTime": number,
   "encryptionSetting": {
      "associatedTranscriptsPassword": "string",
      "botLocaleExportPassword": "string",
      "kmsKeyArn": "string"
   },
   "localeId": "string",
   "transcriptSourceSetting": {
      "s3BucketTranscriptSource": {
         "kmsKeyArn": "string",
         "pathFormat": {
            "objectPrefixes": [ "string" ]
         },
         "s3BucketName": "string",
         "transcriptFilter": {
            "lexTranscriptFilter": {
               "dateRangeFilter": {
                  "endDateTime": number,
                  "startDateTime": number
               }
            }
         }
      },
      "transcriptFormat": "string"
   }
}```
**Response Elements**

If the action is successful, the service sends back an HTTP 202 response.

The following data is returned in JSON format by the service.

**botId (p. 537)**

The unique identifier of the bot containing the bot recommendation.

Type: String

Length Constraints: Fixed length of 10.

Pattern: `^[0-9a-zA-Z]+$`

**botRecommendationId (p. 537)**

The identifier of the bot recommendation that you have created.

Type: String

Length Constraints: Fixed length of 10.

Pattern: `^[0-9a-zA-Z]+$`

**botRecommendationStatus (p. 537)**

The status of the bot recommendation.

If the status is Failed, then the reasons for the failure are listed in the failureReasons field.

Type: String

Valid Values: Processing | Deleting | Deleted | Downloading | Updating | Available | Failed

**botVersion (p. 537)**

The version of the bot containing the bot recommendation.

Type: String

Length Constraints: Fixed length of 5.

Pattern: `^DRAFT$`

**creationDateTime (p. 537)**

A timestamp of the date and time that the bot recommendation was created.

Type: Timestamp

**encryptionSetting (p. 537)**

The object representing the passwords that were used to encrypt the data related to the bot recommendation results, as well as the KMS key ARN used to encrypt the associated metadata.

Type: EncryptionSetting (p. 704) object

**localeId (p. 537)**

The identifier of the language and locale of the bot recommendation to start. The string must match one of the supported locales. For more information, see Supported languages

---

538
Type: String

transcriptSourceSetting (p. 537)

The object representing the Amazon S3 bucket containing the transcript, as well as the associated metadata.

Type: TranscriptSourceSetting (p. 776) object

Errors

For information about the errors that are common to all actions, see Common Errors (p. 821).

ConflictException

The action that you tried to perform couldn't be completed because the resource is in a conflicting state. For example, deleting a bot that is in the CREATING state. Try your request again.

HTTP Status Code: 409

ConflictException

The action that you tried to perform couldn't be completed because the resource is in a conflicting state. For example, deleting a bot that is in the CREATING state. Try your request again.

HTTP Status Code: 409

InternalServerException

The service encountered an unexpected condition. Try your request again.

HTTP Status Code: 500

PreconditionFailedException

Your request couldn't be completed because one or more request fields aren't valid. Check the fields in your request and try again.

HTTP Status Code: 412

ResourceNotFoundException

You asked to describe a resource that doesn't exist. Check the resource that you are requesting and try again.

HTTP Status Code: 404

ServiceQuotaExceededException

You have reached a quota for your bot.

HTTP Status Code: 402

ThrottlingException

Your request rate is too high. Reduce the frequency of requests.

HTTP Status Code: 429

ValidationException

One of the input parameters in your request isn't valid. Check the parameters and try your request again.

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 V2
- AWS SDK for JavaScript
- AWS SDK for PHP V3
- AWS SDK for Python
- AWS SDK for Ruby V3
StartImport
Service: Amazon Lex Model Building V2

Starts importing a bot, bot locale, or custom vocabulary from a zip archive that you uploaded to an S3 bucket.

Request Syntax

```
PUT /imports/ HTTP/1.1
Content-type: application/json

{
  "filePassword": "string",
  "importId": "string",
  "mergeStrategy": "string",
  "resourceSpecification": {
    "botImportSpecification": {
      "botName": "string",
      "botTags": {
        "string": "string"
      },
      "dataPrivacy": {
        "childDirected": boolean
      },
      "idleSessionTTLInSeconds": number,
      "roleArn": "string",
      "testBotAliasTags": {
        "string": "string"
      }
    },
    "botLocaleImportSpecification": {
      "botId": "string",
      "botVersion": "string",
      "localeId": "string",
      "nluIntentConfidenceThreshold": number,
      "voiceSettings": {
        "engine": "string",
        "voiceId": "string"
      }
    },
    "customVocabularyImportSpecification": {
      "botId": "string",
      "botVersion": "string",
      "localeId": "string"
    }
  }
}
```

URI Request Parameters

The request does not use any URI parameters.

Request Body

The request accepts the following data in JSON format.

**filePassword (p. 541)**

The password used to encrypt the zip archive that contains the resource definition. You should always encrypt the zip archive to protect it during transit between your site and Amazon Lex.

Type: String

**Required: No**

**importId (p. 541)**

The unique identifier for the import. It is included in the response from the CreateUploadUrl operation.

Type: String

Length Constraints: Fixed length of 10.

Pattern: ^[0-9a-zA-Z]+$

Required: Yes

**mergeStrategy (p. 541)**

The strategy to use when there is a name conflict between the imported resource and an existing resource. When the merge strategy is FailOnConflict existing resources are not overwritten and the import fails.

Type: String

Valid Values: Overwrite | FailOnConflict | Append

Required: Yes

**resourceSpecification (p. 541)**

Parameters for creating the bot, bot locale or custom vocabulary.

Type: ImportResourceSpecification (p. 720) object

Required: Yes

**Response Syntax**

```
HTTP/1.1 202
Content-type: application/json
{
  "creationDateTime": number,
  "importId": "string",
  "importStatus": "string",
  "mergeStrategy": "string",
  "resourceSpecification": {
    "botImportSpecification": {
      "botName": "string",
      "botTags": {
        "string": "string"
      },
      "dataPrivacy": {
        "childDirected": boolean
      },
      "idleSessionTTLInSeconds": number,
      "roleArn": "string",
      "testBotAliasTags": {
        "string": "string"
      }
    },
    "botLocaleImportSpecification": {
    }
  }
}
```
Response Elements

If the action is successful, the service sends back an HTTP 202 response.

The following data is returned in JSON format by the service.

creationDateTime (p. 542)

The date and time that the import request was created.

Type: Timestamp

importId (p. 542)

A unique identifier for the import.

Type: String

Length Constraints: Fixed length of 10.

Pattern: ^[0-9a-zA-Z]+$

importStatus (p. 542)

The current status of the import. When the status is Complete the bot, bot alias, or custom vocabulary is ready to use.

Type: String

Valid Values: InProgress | Completed | Failed | Deleting

mergeStrategy (p. 542)

The strategy used when there was a name conflict between the imported resource and an existing resource. When the merge strategy is FailOnConflict existing resources are not overwritten and the import fails.

Type: String

Valid Values: Overwrite | FailOnConflict | Append

resourceSpecification (p. 542)

The parameters used when importing the resource.

Type: ImportResourceSpecification (p. 720) object
Errors

For information about the errors that are common to all actions, see Common Errors (p. 821).

ConflictException

The action that you tried to perform couldn’t be completed because the resource is in a conflicting state. For example, deleting a bot that is in the CREATING state. Try your request again.

HTTP Status Code: 409

InternalServerException

The service encountered an unexpected condition. Try your request again.

HTTP Status Code: 500

ResourceNotFoundException

You asked to describe a resource that doesn’t exist. Check the resource that you are requesting and try again.

HTTP Status Code: 404

ServiceQuotaExceededException

You have reached a quota for your bot.

HTTP Status Code: 402

ThrottlingException

Your request rate is too high. Reduce the frequency of requests.

HTTP Status Code: 429

ValidationException

One of the input parameters in your request isn’t valid. Check the parameters and try your request again.

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 V2
- AWS SDK for JavaScript
- AWS SDK for PHP V3
- AWS SDK for Python
- AWS SDK for Ruby V3
TagResource
Service: Amazon Lex Model Building V2

Adds the specified tags to the specified resource. If a tag key already exists, the existing value is replaced with the new value.

Request Syntax

POST `/tags/resourceARN` HTTP/1.1
Content-type: application/json

```json
{
  "tags": {
    "string": "string"
  }
}
```

URI Request Parameters

The request uses the following URI parameters.

**resourceARN (p. 545)**

The Amazon Resource Name (ARN) of the bot, bot alias, or bot channel to tag.

Length Constraints: Minimum length of 1. Maximum length of 1011.

Required: Yes

Request Body

The request accepts the following data in JSON format.

**tags (p. 545)**

A list of tag keys to add to the resource. If a tag key already exists, the existing value is replaced with the new value.

Type: String to string map

Map Entries: Minimum number of 0 items. Maximum number of 200 items.

Key Length Constraints: Minimum length of 1. Maximum length of 128.

Value Length Constraints: Minimum length of 0. Maximum length of 256.

Required: Yes

Response Syntax

HTTP/1.1 200

Response Elements

If the action is successful, the service sends back an HTTP 200 response with an empty HTTP body.

545
Errors

For information about the errors that are common to all actions, see Common Errors (p. 821).

InternalServerException

The service encountered an unexpected condition. Try your request again.

HTTP Status Code: 500

ResourceNotFoundException

You asked to describe a resource that doesn't exist. Check the resource that you are requesting and try again.

HTTP Status Code: 404

ThrottlingException

Your request rate is too high. Reduce the frequency of requests.

HTTP Status Code: 429

ValidationException

One of the input parameters in your request isn't valid. Check the parameters and try your request again.

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 V2
- AWS SDK for JavaScript
- AWS SDK for PHP V3
- AWS SDK for Python
- AWS SDK for Ruby V3
UntagResource
Service: Amazon Lex Model Building V2
Removes tags from a bot, bot alias, or bot channel.

Request Syntax

```
DELETE /tags/
resourceARN?tagKeys=tagKeys HTTP/1.1
```

URI Request Parameters

The request uses the following URI parameters.

**resourceARN (p. 547)**

The Amazon Resource Name (ARN) of the resource to remove the tags from.

Length Constraints: Minimum length of 1. Maximum length of 1011.

Required: Yes

**tagKeys (p. 547)**

A list of tag keys to remove from the resource. If a tag key does not exist on the resource, it is ignored.

Array Members: Minimum number of 0 items. Maximum number of 200 items.


Required: Yes

Request Body

The request does not have a request body.

Response Syntax

```
HTTP/1.1 200
```

Response Elements

If the action is successful, the service sends back an HTTP 200 response with an empty HTTP body.

Errors

For information about the errors that are common to all actions, see Common Errors (p. 821).

**InternalServerException**

The service encountered an unexpected condition. Try your request again.

HTTP Status Code: 500

**ResourceNotFoundException**

You asked to describe a resource that doesn't exist. Check the resource that you are requesting and try again.
HTTP Status Code: 404

**ThrottlingException**

Your request rate is too high. Reduce the frequency of requests.

HTTP Status Code: 429

**ValidationException**

One of the input parameters in your request isn't valid. Check the parameters and try your request again.

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 V2
- AWS SDK for JavaScript
- AWS SDK for PHP V3
- AWS SDK for Python
- AWS SDK for Ruby V3
UpdateBot
Service: Amazon Lex Model Building V2

Updates the configuration of an existing bot.

Request Syntax

```
PUT /bots/botId/ HTTP/1.1
Content-type: application/json

{
   "botName": "string",
   "dataPrivacy": {
      "childDirected": boolean
   },
   "description": "string",
   "idleSessionTTLInSeconds": number,
   "roleArn": "string"
}
```

URI Request Parameters

The request uses the following URI parameters.

**botId (p. 549)**

The unique identifier of the bot to update. This identifier is returned by the CreateBot operation.

- Pattern: `^[0-9a-zA-Z]+$`
- Required: Yes

Request Body

The request accepts the following data in JSON format.

**botName (p. 549)**

The new name of the bot. The name must be unique in the account that creates the bot.

- Type: String
- Pattern: `^[0-9a-zA-Z][_-]?]+$`
- Required: Yes

**dataPrivacy (p. 549)**

Provides information on additional privacy protections Amazon Lex should use with the bot's data.

- Type: DataPrivacy (p. 701) object
- Required: Yes

**description (p. 549)**

A description of the bot.
Type: String
Length Constraints: Minimum length of 0. Maximum length of 200.
Required: No

idleSessionTTLInSeconds (p. 549)
The time, in seconds, that Amazon Lex should keep information about a user’s conversation with the bot.
A user interaction 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.
You can specify between 60 (1 minute) and 86,400 (24 hours) seconds.
Type: Integer
Valid Range: Minimum value of 60. Maximum value of 86400.
Required: Yes

roleArn (p. 549)
The Amazon Resource Name (ARN) of an IAM role that has permissions to access the bot.
Type: String
Length Constraints: Minimum length of 32. Maximum length of 2048.
Pattern: ^arn:aws:iam::[0-9]{12}:role/.*$
Required: Yes

Response Syntax
HTTP/1.1 202
Content-type: application/json

{
  "botId": "string",
  "botName": "string",
  "botStatus": "string",
  "creationDateTime": number,
  "dataPrivacy": {
    "childDirected": boolean
  },
  "description": "string",
  "idleSessionTTLInSeconds": number,
  "lastUpdatedDateTime": number,
  "roleArn": "string"
}

Response Elements
If the action is successful, the service sends back an HTTP 202 response.
The following data is returned in JSON format by the service.

botId (p. 550)
The unique identifier of the bot that was updated.
Type: String
Length Constraints: Fixed length of 10.
Pattern: ^[0-9a-zA-Z]+$

botName (p. 550)

The name of the bot after the update.

Type: String
Length Constraints: Minimum length of 1. Maximum length of 100.
Pattern: ^([0-9a-zA-Z][\_\-]?)+$

botStatus (p. 550)

Shows the current status of the bot. The bot is first in the Creating status. Once the bot is read for use, it changes to the Available status. After the bot is created, you can use the DRAFT version of the bot.

Type: String
Valid Values: Creating | Available | Inactive | Deleting | Failed | Versioning | Importing

creationDateTime (p. 550)

A timestamp of the date and time that the bot was created.

Type: Timestamp

dataPrivacy (p. 550)

The data privacy settings for the bot after the update.

Type: DataPrivacy (p. 701) object

description (p. 550)

The description of the bot after the update.

Type: String
Length Constraints: Minimum length of 0. Maximum length of 200.

idleSessionTTLInSeconds (p. 550)

The session timeout, in seconds, for the bot after the update.

Type: Integer
Valid Range: Minimum value of 60. Maximum value of 86400.

lastUpdatedDateTime (p. 550)

A timestamp of the date and time that the bot was last updated.

Type: Timestamp

roleArn (p. 550)

The Amazon Resource Name (ARN) of the IAM role used by the bot after the update.

Type: String
Length Constraints: Minimum length of 32. Maximum length of 2048.
Pattern: ^arn:aws:iam::\[0-9\]{12}:role/.*$

Errors

For information about the errors that are common to all actions, see Common Errors (p. 821).

ConflictException  
The action that you tried to perform couldn't be completed because the resource is in a conflicting state. For example, deleting a bot that is in the CREATING state. Try your request again.

HTTP Status Code: 409

InternalServerException  
The service encountered an unexpected condition. Try your request again.

HTTP Status Code: 500

PreconditionFailedException  
Your request couldn't be completed because one or more request fields aren't valid. Check the fields in your request and try again.

HTTP Status Code: 412

ServiceQuotaExceededException  
You have reached a quota for your bot.

HTTP Status Code: 402

ThrottlingException  
Your request rate is too high. Reduce the frequency of requests.

HTTP Status Code: 429

ValidationException  
One of the input parameters in your request isn't valid. Check the parameters and try your request again.

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 V2
- AWS SDK for JavaScript
- AWS SDK for PHP V3
- AWS SDK for Python
- AWS SDK for Ruby V3
UpdateBotAlias
Service: Amazon Lex Model Building V2

Updates the configuration of an existing bot alias.

Request Syntax

```
PUT /bots/botId/botAliases/botAliasId/ HTTP/1.1
Content-type: application/json

{
    "botAliasLocaleSettings": {
        "string": {
            "codeHookSpecification": {
                "lambdaCodeHook": {
                    "codeHookInterfaceVersion": "string",
                    "lambdaARN": "string"
                }
            },
            "enabled": boolean
        }
    },
    "botAliasName": "string",
    "botVersion": "string",
    "conversationLogSettings": {
        "audioLogSettings": [
            {
                "destination": {
                    "s3Bucket": {
                        "kmsKeyArn": "string",
                        "logPrefix": "string",
                        "s3BucketArn": "string"
                    }
                },
                "enabled": boolean
            }
        ],
        "textLogSettings": [
            {
                "destination": {
                    "cloudWatch": {
                        "cloudWatchLogGroupArn": "string",
                        "logPrefix": "string"
                    }
                },
                "enabled": boolean
            }
        ],
        "description": "string",
        "sentimentAnalysisSettings": {
            "detectSentiment": boolean
        }
    }
}
```

URI Request Parameters

The request uses the following URI parameters.

botAliasId (p. 553)

The unique identifier of the bot alias.
botId (p. 553)
The identifier of the bot with the updated alias.
Length Constraints: Fixed length of 10.
Pattern: ^\bTSTALIASID\b[0-9a-zA-Z]+$  
Required: Yes

Request Body

The request accepts the following data in JSON format.

botAliasLocaleSettings (p. 553)
The new Lambda functions to use in each locale for the bot alias.
Type: String to BotAliasLocaleSettings (p. 665) object map
Map Entries: Maximum number of items.
Required: No

botAliasName (p. 553)
The new name to assign to the bot alias.
Type: String
Length Constraints: Minimum length of 1. Maximum length of 100.
Pattern: ^([0-9a-zA-Z][_\-]?)+$  
Required: Yes

botVersion (p. 553)
The new bot version to assign to the bot alias.
Type: String
Pattern: ^(DRAFT]|[0-9]+)$  
Required: No

collectionLogSettings (p. 553)
The new settings for storing conversation logs in Amazon CloudWatch Logs and Amazon S3 buckets.
Type: ConversationLogSettings (p. 697) object
Required: No

description (p. 553)
The new description to assign to the bot alias.
Type: String

Length Constraints: Minimum length of 0. Maximum length of 200.

Required: No

sentimentAnalysisSettings (p. 553)

Determines whether Amazon Lex will use Amazon Comprehend to detect the sentiment of user utterances.

Type: SentimentAnalysisSettings (p. 753) object

Required: No

Response Syntax

HTTP/1.1 202
Content-type: application/json

{  
  "botAliasId": "string",
  "botAliasLocaleSettings": {
    "string": {
      "codeHookSpecification": {
        "lambdaCodeHook": {
          "codeHookInterfaceVersion": "string",
          "lambdaARN": "string"
        }
      },
      "enabled": boolean
    }
  },
  "botAliasName": "string",
  "botAliasStatus": "string",
  "botId": "string",
  "botVersion": "string",
  "conversationLogSettings": {
    "audioLogSettings": [
      {
        "destination": {
          "s3Bucket": {
            "kmsKeyArn": "string",
            "logPrefix": "string",
            "s3BucketArn": "string"
          }
        },
        "enabled": boolean
      }
    ],
    "textLogSettings": [
      {
        "destination": {
          "cloudWatch": {
            "cloudWatchLogGroupArn": "string",
            "logPrefix": "string"
          }
        },
        "enabled": boolean
      }
    ]
  },
  "creationDateTime": number,
  "description": "string",

"sentimentAnalysisSettings (p. 553)"
Response Elements

If the action is successful, the service sends back an HTTP 202 response.

The following data is returned in JSON format by the service.

botAliasId (p. 555)

The identifier of the updated bot alias.
Type: String
Length Constraints: Fixed length of 10.
Pattern: ^\bTSTALIASID\b\[0-9a-zA-Z]+$ botAliasLocaleSettings (p. 555)

The updated Lambda functions to use in each locale for the bot alias.
Type: String to BotAliasLocaleSettings (p. 665) object map
Map Entries: Maximum number of items.

botAliasName (p. 555)

The updated name of the bot alias.
Type: String
Length Constraints: Minimum length of 1. Maximum length of 100.
Pattern: ^([0-9a-zA-Z][_-]?)+$ botAliasStatus (p. 555)

The current status of the bot alias. When the status is Available the alias is ready for use.
Type: String
Valid Values: Creating | Available | Deleting | Failed

botId (p. 555)

The identifier of the bot with the updated alias.
Type: String
Length Constraints: Fixed length of 10.
Pattern: ^[0-9a-zA-Z]+$ botVersion (p. 555)

The updated version of the bot that the alias points to.
Type: String
Pattern: (^DRAFT | \d\d\d\d$)

**conversationLogSettings (p. 555)**

The updated settings for storing conversation logs in Amazon CloudWatch Logs and Amazon S3 buckets.

Type: ConversationLogSettings (p. 697) object

**creationDateTime (p. 555)**

A timestamp of the date and time that the bot was created.

Type: Timestamp

**description (p. 555)**

The updated description of the bot alias.

Type: String

Length Constraints: Minimum length of 0. Maximum length of 200.

**lastUpdatedDateTime (p. 555)**

A timestamp of the date and time that the bot was last updated.

Type: Timestamp

**sentimentAnalysisSettings (p. 555)**

Determines whether Amazon Lex will use Amazon Comprehend to detect the sentiment of user utterances.

Type: SentimentAnalysisSettings (p. 753) object

**Errors**

For information about the errors that are common to all actions, see Common Errors (p. 821).

**ConflictException**

The action that you tried to perform couldn't be completed because the resource is in a conflicting state. For example, deleting a bot that is in the CREATING state. Try your request again.

HTTP Status Code: 409

**InternalServerException**

The service encountered an unexpected condition. Try your request again.

HTTP Status Code: 500

**PreconditionFailedException**

Your request couldn't be completed because one or more request fields aren't valid. Check the fields in your request and try again.

HTTP Status Code: 412

**ServiceQuotaExceededException**

You have reached a quota for your bot.

HTTP Status Code: 402
ThrottlingException

Your request rate is too high. Reduce the frequency of requests.

HTTP Status Code: 429

ValidationException

One of the input parameters in your request isn't valid. Check the parameters and try your request again.

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 V2
- AWS SDK for JavaScript
- AWS SDK for PHP V3
- AWS SDK for Python
- AWS SDK for Ruby V3
UpdateBotLocale
Service: Amazon Lex Model Building V2

Updates the settings that a bot has for a specific locale.

**Request Syntax**

```
PUT /bots/botId/botversions/botVersion/botlocales/localeId/ HTTP/1.1
Content-type: application/json

{
    "description": "string",
    "nluIntentConfidenceThreshold": number,
    "voiceSettings": {
        "engine": "string",
        "voiceId": "string"
    }
}
```

**URI Request Parameters**

The request uses the following URI parameters.

**botId (p. 559)**

The unique identifier of the bot that contains the locale.

Length Constraints: Fixed length of 10.

Pattern: ^[0-9a-zA-Z]+$

Required: Yes

**botVersion (p. 559)**

The version of the bot that contains the locale to be updated. The version can only be the DRAFT version.

Length Constraints: Fixed length of 5.

Pattern: ^DRAFT$

Required: Yes

**localeId (p. 559)**

The identifier of the language and locale to update. The string must match one of the supported locales. For more information, see Supported languages.

Required: Yes

**Request Body**

The request accepts the following data in JSON format.

**description (p. 559)**

The new description of the locale.

Type: String
Length Constraints: Minimum length of 0. Maximum length of 200.

Required: No

**nluIntentConfidenceThreshold (p. 559)**

The new confidence threshold where Amazon Lex inserts the `AMAZON.FallbackIntent` and `AMAZON.KendraSearchIntent` intents in the list of possible intents for an utterance.

Type: Double

Valid Range: Minimum value of 0. Maximum value of 1.

Required: Yes

**voiceSettings (p. 559)**

The new Amazon Polly voice Amazon Lex should use for voice interaction with the user.

Type: `VoiceSettings (p. 778)` object

Required: No

### Response Syntax

```
HTTP/1.1 202
Content-type: application/json

{
  "botId": "string",
  "botLocaleStatus": "string",
  "botVersion": "string",
  "creationDateTime": number,
  "description": "string",
  "failureReasons": [ "string" ],
  "lastUpdatedDateTime": number,
  "localeId": "string",
  "localeName": "string",
  "nluIntentConfidenceThreshold": number,
  "recommendedActions": [ "string" ],
  "voiceSettings": {
    "engine": "string",
    "voiceId": "string"
  }
}
```

### Response Elements

If the action is successful, the service sends back an HTTP 202 response.

The following data is returned in JSON format by the service.

**botId (p. 560)**

The identifier of the bot that contains the updated locale.

Type: String

Length Constraints: Fixed length of 10.

Pattern: `^[0-9a-zA-Z]+$`
botLocaleStatus (p. 560)

The current status of the locale. When the bot status is Built the locale is ready for use.

Type: String

Valid Values: Creating | Building | Built | ReadyExpressTesting | Failed | Deleting | NotBuilt | Importing | Processing

botVersion (p. 560)

The version of the bot that contains the updated locale.

Type: String

Length Constraints: Fixed length of 5.

Pattern: ^DRAFT$

creationDateTime (p. 560)

A timestamp of the date and time that the locale was created.

Type: Timestamp

description (p. 560)

The updated description of the locale.

Type: String

Length Constraints: Minimum length of 0. Maximum length of 200.

failureReasons (p. 560)

If the botLocaleStatus is Failed, the failureReasons field lists the errors that occurred while building the bot.

Type: Array of strings

lastUpdatedDateTime (p. 560)

A timestamp of the date and time that the locale was last updated.

Type: Timestamp

localeId (p. 560)

The language and locale of the updated bot locale.

Type: String

localeName (p. 560)

The updated locale name for the locale.

Type: String

nluIntentConfidenceThreshold (p. 560)

The updated confidence threshold for inserting the AMAZON.FallbackIntent and AMAZON.KendraSearchIntent intents in the list of possible intents for an utterance.

Type: Double

Valid Range: Minimum value of 0. Maximum value of 1.
recommendedActions (p. 560)

Recommended actions to take to resolve an error in the failureReasons field.
Type: Array of strings

voiceSettings (p. 560)

The updated Amazon Polly voice to use for voice interaction with the user.
Type: VoiceSettings (p. 778) object

Errors

For information about the errors that are common to all actions, see Common Errors (p. 821).

ConflictException

The action that you tried to perform couldn't be completed because the resource is in a conflicting state. For example, deleting a bot that is in the CREATING state. Try your request again.

HTTP Status Code: 409

InternalServerException

The service encountered an unexpected condition. Try your request again.

HTTP Status Code: 500

PreconditionFailedException

Your request couldn't be completed because one or more request fields aren't valid. Check the fields in your request and try again.

HTTP Status Code: 412

ServiceQuotaExceededException

You have reached a quota for your bot.

HTTP Status Code: 402

ThrottlingException

Your request rate is too high. Reduce the frequency of requests.

HTTP Status Code: 429

ValidationException

One of the input parameters in your request isn't valid. Check the parameters and try your request again.

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 V2
- AWS SDK for JavaScript
- AWS SDK for PHP V3
- AWS SDK for Python
- AWS SDK for Ruby V3
UpdateBotRecommendation
Service: Amazon Lex Model Building V2
Updates an existing bot recommendation request.

Request Syntax

```
PUT /bots/botId/botversions/botVersion/botlocales/localeId/botrecommendations/botRecommendationId/ HTTP/1.1
Content-type: application/json

{
   "encryptionSetting": {
      "associatedTranscriptsPassword": "string",
      "botLocaleExportPassword": "string",
      "kmsKeyArn": "string"
   }
}
```

URI Request Parameters

The request uses the following URI parameters.

**botId (p. 564)**

The unique identifier of the bot containing the bot recommendation to be updated.

Length Constraints: Fixed length of 10.

Pattern: `^[0-9a-zA-Z]+$`

Required: Yes

**botRecommendationId (p. 564)**

The unique identifier of the bot recommendation to be updated.

Length Constraints: Fixed length of 10.

Pattern: `^[0-9a-zA-Z]+$`

Required: Yes

**botVersion (p. 564)**

The version of the bot containing the bot recommendation to be updated.

Length Constraints: Fixed length of 5.

Pattern: `^DRAFT$`

Required: Yes

**localeId (p. 564)**

The identifier of the language and locale of the bot recommendation to update. The string must match one of the supported locales. For more information, see Supported languages

Required: Yes

Request Body

The request accepts the following data in JSON format.
encryptionSetting (p. 564)

The object representing the passwords that will be used to encrypt the data related to the bot recommendation results, as well as the KMS key ARN used to encrypt the associated metadata.

Type: EncryptionSetting (p. 704) object

Required: Yes

Response Syntax

```
HTTP/1.1 202
Content-type: application/json

{
  "botId": "string",
  "botRecommendationId": "string",
  "botRecommendationStatus": "string",
  "botVersion": "string",
  "creationDateTime": number,
  "encryptionSetting": {
    "associatedTranscriptsPassword": "string",
    "botLocaleExportPassword": "string",
    "kmsKeyArn": "string"
  },
  "lastUpdatedDateTime": number,
  "localeId": "string",
  "transcriptSourceSetting": {
    "s3BucketTranscriptSource": {
      "kmsKeyArn": "string",
      "pathFormat": {
        "objectPrefixes": ["string"]
      },
      "s3BucketName": "string",
      "transcriptFilter": {
        "lexTranscriptFilter": {
          "dateRangeFilter": {
            "endDate": number,
            "startTime": number
          }
        }
      }
    },
    "transcriptFormat": "string"
  }
}
```

Response Elements

If the action is successful, the service sends back an HTTP 202 response.

The following data is returned in JSON format by the service.

botId (p. 565)

The unique identifier of the bot containing the bot recommendation that has been updated.

Type: String

Length Constraints: Fixed length of 10.

Pattern: ^[0-9a-zA-Z]*$
botRecommendationId (p. 565)

The unique identifier of the bot recommendation to be updated.

Type: String

Length Constraints: Fixed length of 10.

Pattern: ^[0-9a-zA-Z]+$

botRecommendationStatus (p. 565)

The status of the bot recommendation.

If the status is Failed, then the reasons for the failure are listed in the failureReasons field.

Type: String

Valid Values: Processing | Deleting | Deleted | Downloading | Updating | Available | Failed

botVersion (p. 565)

The version of the bot containing the bot recommendation that has been updated.

Type: String

Length Constraints: Fixed length of 5.

Pattern: ^DRAFT$

creationDateTime (p. 565)

A timestamp of the date and time that the bot recommendation was created.

Type: Timestamp

encryptionSetting (p. 565)

The object representing the passwords that were used to encrypt the data related to the bot recommendation results, as well as the KMS key ARN used to encrypt the associated metadata.

Type: EncryptionSetting (p. 704) object

lastUpdatedDateTime (p. 565)

A timestamp of the date and time that the bot recommendation was last updated.

Type: Timestamp

localeId (p. 565)

The identifier of the language and locale of the bot recommendation to update. The string must match one of the supported locales. For more information, see Supported languages

Type: String

transcriptSourceSetting (p. 565)

The object representing the Amazon S3 bucket containing the transcript, as well as the associated metadata.

Type: TranscriptSourceSetting (p. 776) object

Errors

For information about the errors that are common to all actions, see Common Errors (p. 821).
ConflictException

The action that you tried to perform couldn't be completed because the resource is in a conflicting state. For example, deleting a bot that is in the CREATING state. Try your request again.

HTTP Status Code: 409

ConflictException

The action that you tried to perform couldn't be completed because the resource is in a conflicting state. For example, deleting a bot that is in the CREATING state. Try your request again.

HTTP Status Code: 409

InternalServerException

The service encountered an unexpected condition. Try your request again.

HTTP Status Code: 500

PreconditionFailedException

Your request couldn't be completed because one or more request fields aren't valid. Check the fields in your request and try again.

HTTP Status Code: 412

ResourceNotFoundException

You asked to describe a resource that doesn't exist. Check the resource that you are requesting and try again.

HTTP Status Code: 404

ServiceQuotaExceededException

You have reached a quota for your bot.

HTTP Status Code: 402

ThrottlingException

Your request rate is too high. Reduce the frequency of requests.

HTTP Status Code: 429

ValidationException

One of the input parameters in your request isn't valid. Check the parameters and try your request again.

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 V2
- AWS SDK for JavaScript
• AWS SDK for PHP V3
• AWS SDK for Python
• AWS SDK for Ruby V3
UpdateExport
Service: Amazon Lex Model Building V2

Updates the password used to protect an export zip archive.

The password is not required. If you don’t supply a password, Amazon Lex generates a zip file that is not protected by a password. This is the archive that is available at the pre-signed S3 URL provided by the DescribeExport operation.

Request Syntax

```
PUT /exports/exportId/ HTTP/1.1
Content-type: application/json

{
   "filePassword": "string"
}
```

URI Request Parameters

The request uses the following URI parameters.

exportId (p. 569)

The unique identifier Amazon Lex assigned to the export.

Length Constraints: Fixed length of 10.

Pattern: ^[0-9a-zA-Z]+$

Required: Yes

Request Body

The request accepts the following data in JSON format.

filePassword (p. 569)

The new password to use to encrypt the export zip archive.

Type: String


Required: No

Response Syntax

```
HTTP/1.1 202
Content-type: application/json

{
   "creationDateTime": number,
   "exportId": "string",
   "exportStatus": "string",
   "fileFormat": "string",
   "lastUpdatedDateTime": number,
   "resourceSpecification": {
```
"botExportSpecification": {  
  "botId": "string",
  "botVersion": "string"
},
"botLocaleExportSpecification": {  
  "botId": "string",
  "botVersion": "string",
  "localeId": "string"
},
"customVocabularyExportSpecification": {  
  "botId": "string",
  "botVersion": "string",
  "localeId": "string"
}
}

Response Elements

If the action is successful, the service sends back an HTTP 202 response.

The following data is returned in JSON format by the service.

creationDateTime (p. 569)

The date and time that the export was created.

Type: Timestamp

exportId (p. 569)

The unique identifier Amazon Lex assigned to the export.

Type: String

Length Constraints: Fixed length of 10.

Pattern: ^[0-9a-zA-Z]+$

exportStatus (p. 569)

The status of the export. When the status is Completed the export archive is available for download.

Type: String

Valid Values: InProgress | Completed | Failed | Deleting

fileFormat (p. 569)

The file format used for the files that define the resource. The TSV format is required to export a custom vocabulary only; otherwise use LexJson format.

Type: String

Valid Values: LexJson | TSV

lastUpdatedDateTime (p. 569)

The date and time that the export was last updated.

Type: Timestamp

resourceSpecification (p. 569)

A description of the type of resource that was exported, either a bot or a bot locale.
Errors

For information about the errors that are common to all actions, see Common Errors (p. 821).

ConflictingException

The action that you tried to perform couldn't be completed because the resource is in a conflicting state. For example, deleting a bot that is in the CREATING state. Try your request again.

HTTP Status Code: 409

InternalServerException

The service encountered an unexpected condition. Try your request again.

HTTP Status Code: 500

ResourceNotFoundException

You asked to describe a resource that doesn't exist. Check the resource that you are requesting and try again.

HTTP Status Code: 404

ServiceQuotaExceededException

You have reached a quota for your bot.

HTTP Status Code: 402

ThrottlingException

Your request rate is too high. Reduce the frequency of requests.

HTTP Status Code: 429

ValidationException

One of the input parameters in your request isn't valid. Check the parameters and try your request again.

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 V2
- AWS SDK for JavaScript
- AWS SDK for PHP V3
- AWS SDK for Python
- AWS SDK for Ruby V3
UpdateIntent
Service: Amazon Lex Model Building V2
Updates the settings for an intent.

Request Syntax

PUT /bots/botId/botversions/botVersion/botlocales/localeId/intents/intentId/ HTTP/1.1
Content-type: application/json

{
    "description": "string",
    "dialogCodeHook": {
        "enabled": boolean
    },
    "fulfillmentCodeHook": {
        "enabled": boolean,
        "fulfillmentUpdatesSpecification": {
            "active": boolean,
            "startResponse": {
                "allowInterrupt": boolean,
                "delayInSeconds": number,
                "messageGroups": [
                    {
                        "message": {
                            "customPayload": {
                                "value": "string"
                            },
                            "imageResponseCard": {
                                "buttons": [
                                    {
                                        "text": "string",
                                        "value": "string"
                                    }
                                ],
                            "imageUrl": "string",
                            "subtitle": "string",
                            "title": "string"
                        },
                        "plainTextMessage": {
                            "value": "string"
                        },
                        "ssmlMessage": {
                            "value": "string"
                        }
                    },
                    "variations": [
                        {
                            "customPayload": {
                                "value": "string"
                            },
                            "imageResponseCard": {
                                "buttons": [
                                    {
                                        "text": "string",
                                        "value": "string"
                                    }
                                ],
                            "imageUrl": "string",
                            "subtitle": "string",
                            "title": "string"
                        },
                        "plainTextMessage": {
                            "value": "string"
                        }
                    }
                ]
            }
        }
    }
}
"ssmlMessage": {
  "value": "string"
}
]
"timeoutInSeconds": number,
"updateResponse": {
  "allowInterrupt": boolean,
  "frequencyInSeconds": number,
  "messageGroups": [
    {
      "message": {
        "customPayload": {
          "value": "string"
        },
        "imageResponseCard": {
          "buttons": [
            {
              "text": "string",
              "value": "string"
            }
          ],
          "imageUrl": "string",
          "subtitle": "string",
          "title": "string"
        },
        "plainTextMessage": {
          "value": "string"
        },
        "ssmlMessage": {
          "value": "string"
        }
      },
      "variations": [
        {
          "customPayload": {
            "value": "string"
          },
          "imageResponseCard": {
            "buttons": [
              {
                "text": "string",
                "value": "string"
              }
            ],
            "imageUrl": "string",
            "subtitle": "string",
            "title": "string"
          },
          "plainTextMessage": {
            "value": "string"
          },
          "ssmlMessage": {
            "value": "string"
          }
        }
      ]
    }
  ]
},
"postFulfillmentStatusSpecification": {
}
"failureResponse": {  
  "allowInterrupt": boolean,  
  "messageGroups": [  
    {  
      "message": {  
        "customPayload": {  
          "value": "string"  
        },  
        "imageResponseCard": {  
          "buttons": [  
            {  
              "text": "string",  
              "value": "string"  
            }  
          ],  
          "imageUrl": "string",  
          "subtitle": "string",  
          "title": "string"  
        },  
        "plainTextMessage": {  
          "value": "string"  
        },  
        "ssmlMessage": {  
          "value": "string"  
        }  
      },  
      "variations": [  
        {  
          "customPayload": {  
            "value": "string"  
          },  
          "imageResponseCard": {  
            "buttons": [  
              {  
                "text": "string",  
                "value": "string"  
              }  
            ],  
            "imageUrl": "string",  
            "subtitle": "string",  
            "title": "string"  
          },  
          "plainTextMessage": {  
            "value": "string"  
          },  
          "ssmlMessage": {  
            "value": "string"  
          }  
        }  
      ]  
    }  
  ]  
},  
"successResponse": {  
  "allowInterrupt": boolean,  
  "messageGroups": [  
    {  
      "message": {  
        "customPayload": {  
          "value": "string"  
        },  
        "imageResponseCard": {  
          "buttons": [  
            {  
              "text": "string",  
              "value": "string"  
            }  
          ],  
          "imageUrl": "string",  
          "subtitle": "string",  
          "title": "string"  
        },  
        "plainTextMessage": {  
          "value": "string"  
        },  
        "ssmlMessage": {  
          "value": "string"  
        }  
      },  
      "variations": [  
        {  
          "customPayload": {  
            "value": "string"  
          },  
          "imageResponseCard": {  
            "buttons": [  
              {  
                "text": "string",  
                "value": "string"  
              }  
            ],  
            "imageUrl": "string",  
            "subtitle": "string",  
            "title": "string"  
          },  
          "plainTextMessage": {  
            "value": "string"  
          },  
          "ssmlMessage": {  
            "value": "string"  
          }  
        }  
      ]  
    }  
  ]  
}
",
  "imageUrl": "string",
  "subtitle": "string",
  "title": "string"
},
  "plainTextMessage": {
    "value": "string"
  },
  "ssmlMessage": {
    "value": "string"
  }
},
  "variations": [
    {
      "customPayload": {
        "value": "string"
      },
      "imageResponseCard": {
        "buttons": [
          {
            "text": "string",
            "value": "string"
          }
        ],
        "imageUrl": "string",
        "subtitle": "string",
        "title": "string"
      },
      "plainTextMessage": {
        "value": "string"
      },
      "ssmlMessage": {
        "value": "string"
      }
    }
  ]
},
  "timeoutResponse": {
    "allowInterrupt": boolean,
    "messageGroups": [
      {
        "message": {
          "customPayload": {
            "value": "string"
          },
          "imageResponseCard": {
            "buttons": [
              {
                "text": "string",
                "value": "string"
              }
            ],
            "imageUrl": "string",
            "subtitle": "string",
            "title": "string"
          },
          "plainTextMessage": {
            "value": "string"
          },
          "ssmlMessage": {
            "value": "string"
          }
        }
      }
    ]
  }
]
"variations": [
  {
    "customPayload": {
      "value": "string"
    },
    "imageResponseCard": {
      "buttons": [
        {
          "text": "string",
          "value": "string"
        }
      ],
      "imageUrl": "string",
      "subtitle": "string",
      "title": "string"
    },
    "plainTextMessage": {
      "value": "string"
    },
    "ssmlMessage": {
      "value": "string"
    }
  }
],
"inputContexts": [
  {
    "name": "string"
  }
],
"intentClosingSetting": {
  "active": boolean,
  "closingResponse": {
    "allowInterrupt": boolean,
    "messageGroups": [
      {
        "message": {
          "customPayload": {
            "value": "string"
          },
          "imageResponseCard": {
            "buttons": [
              {
                "text": "string",
                "value": "string"
              }
            ],
            "imageUrl": "string",
            "subtitle": "string",
            "title": "string"
          },
          "plainTextMessage": {
            "value": "string"
          },
          "ssmlMessage": {
            "value": "string"
          }
        },
        "variations": [
          {
            "customPayload": {
              "value": "string"
            }
          }
        ]
      }
    ]
  }
]
"intentConfirmationSetting": {
  "active": boolean,
  "declinationResponse": {
    "allowInterrupt": boolean,
    "messageGroups": [
      {
        "message": {
          "customPayload": {
            "value": "string"
          },
          "imageResponseCard": {
            "buttons": [
              {
                "text": "string",
                "value": "string"
              }
            ],
            "imageUrl": "string",
            "subtitle": "string",
            "title": "string"
          },
          "plainTextMessage": {
            "value": "string"
          },
          "ssmlMessage": {
            "value": "string"
          }
        }
      }
    ],
    "variations": [
      {
        "customPayload": {
          "value": "string"
        },
        "imageResponseCard": {
          "buttons": [
            {
              "text": "string",
              "value": "string"
            }
          ],
          "imageUrl": "string",
          "subtitle": "string",
          "title": "string"
        },
        "plainTextMessage": {
          "value": "string"
        },
        "ssmlMessage": {
          "value": "string"
        }
      }
    ]
  }
}
"title": "string",
"plainTextMessage": {
  "value": "string"
},
"ssmlMessage": {
  "value": "string"
}
]
},
"promptSpecification": {
  "allowInterrupt": boolean,
  "maxRetries": number,
  "messageGroups": [
    {
      "message": {
        "customPayload": {
          "value": "string"
        },
        "imageResponseCard": {
          "buttons": [
            {
              "text": "string",
              "value": "string"
            }
          ],
          "imageUrl": "string",
          "subtitle": "string",
          "title": "string"
        },
        "plainTextMessage": {
          "value": "string"
        },
        "ssmlMessage": {
          "value": "string"
        }
      },
      "variations": [
        {
          "customPayload": {
            "value": "string"
          },
          "imageResponseCard": {
            "buttons": [
              {
                "text": "string",
                "value": "string"
              }
            ],
            "imageUrl": "string",
            "subtitle": "string",
            "title": "string"
          },
          "plainTextMessage": {
            "value": "string"
          },
          "ssmlMessage": {
            "value": "string"
          }
        }
      ]
    }
  ]
}
URI Request Parameters

The request uses the following URI parameters.

**botId (p. 572)**

The identifier of the bot that contains the intent.

Length Constraints: Fixed length of 10.

Pattern: `^[0-9a-zA-Z]+$`

Required: Yes

**botVersion (p. 572)**

The version of the bot that contains the intent. Must be **DRAFT**.

Length Constraints: Fixed length of 5.

Pattern: `^[DRAFT]+$`

Required: Yes

**intentId (p. 572)**

The unique identifier of the intent to update.

Length Constraints: Fixed length of 10.

Pattern: `^[0-9a-zA-Z]+$`

Required: Yes

**localeId (p. 572)**

The identifier of the language and locale where this intent is used. The string must match one of the supported locales. For more information, see **Supported languages**.
Required: Yes

**Request Body**

The request accepts the following data in JSON format.

**description (p. 572)**

The new description of the intent.

Type: String

Length Constraints: Minimum length of 0. Maximum length of 200.

Required: No

**dialogCodeHook (p. 572)**

The new Lambda function to use between each turn of the conversation with the bot.

Type: DialogCodeHookSettings (p. 703) object

Required: No

**fulfillmentCodeHook (p. 572)**

The new Lambda function to call when all of the intents required slots are provided and the intent is ready for fulfillment.

Type: FulfillmentCodeHookSettings (p. 711) object

Required: No

**inputContexts (p. 572)**

A new list of contexts that must be active in order for Amazon Lex to consider the intent.

Type: Array of InputContext (p. 724) objects

Array Members: Minimum number of 0 items. Maximum number of 5 items.

Required: No

**intentClosingSetting (p. 572)**

The new response that Amazon Lex sends the user when the intent is closed.

Type: IntentClosingSetting (p. 725) object

Required: No

**intentConfirmationSetting (p. 572)**

New prompts that Amazon Lex sends to the user to confirm the completion of an intent.

Type: IntentConfirmationSetting (p. 726) object

Required: No

**intentName (p. 572)**

The new name for the intent.

Type: String
Length Constraints: Minimum length of 1. Maximum length of 100.

Pattern: ^([0-9a-zA-Z][-_]?)+$

Required: Yes

**kendraConfiguration (p. 572)**

New configuration settings for connecting to an Amazon Kendra index.

Type: KendraConfiguration (p. 732) object

Required: No

**outputContexts (p. 572)**

A new list of contexts that Amazon Lex activates when the intent is fulfilled.

Type: Array of OutputContext (p. 739) objects

Array Members: Minimum number of 0 items. Maximum number of 10 items.

Required: No

**parentIntentSignature (p. 572)**

The signature of the new built-in intent to use as the parent of this intent.

Type: String

Required: No

**sampleUtterances (p. 572)**

New utterances used to invoke the intent.

Type: Array of SampleUtterance (p. 751) objects

Required: No

**slotPriorities (p. 572)**

A new list of slots and their priorities that are contained by the intent.

Type: Array of SlotPriority (p. 757) objects

Required: No

### Response Syntax

```
HTTP/1.1 200
Content-type: application/json

{
   "botId": "string",
   "botVersion": "string",
   "creationDateTime": number,
   "description": "string",
   "dialogCodeHook": {
       "enabled": boolean
   },
   "fulfillmentCodeHook": {
       "enabled": boolean,
       "fulfillmentUpdatesSpecification": {
           "active": boolean,
           ...  
   }
}
```
"startResponse": {
  "allowInterrupt": boolean,
  "delayInSeconds": number,
  "messageGroups": [
    {
      "message": {
        "customPayload": {
          "value": "string"
        },
        "imageResponseCard": {
          "buttons": [
            {
              "text": "string",
              "value": "string"
            }
          ],
          "imageUrl": "string",
          "subtitle": "string",
          "title": "string"
        },
        "plainTextMessage": {
          "value": "string"
        },
        "ssmlMessage": {
          "value": "string"
        }
      }
    },
    {
      "customPayload": {
        "value": "string"
      },
      "imageResponseCard": {
        "buttons": [
          {
            "text": "string",
            "value": "string"
          }
        ],
        "imageUrl": "string",
        "subtitle": "string",
        "title": "string"
      },
      "plainTextMessage": {
        "value": "string"
      },
      "ssmlMessage": {
        "value": "string"
      }
    }
  ],
  "timeoutInSeconds": number,
  "updateResponse": {
    "allowInterrupt": boolean,
    "frequencyInSeconds": number,
    "messageGroups": [
      {
        "message": {
          "customPayload": {
            "value": "string"
          },
          "imageResponseCard": {
            "buttons": [
          ]
        }
      }
"ssmlMessage": {
  "value": "string"
},
"variations": [
  {
    "customPayload": {
      "value": "string"
    },
    "imageResponseCard": {
      "buttons": [
        {
          "text": "string",
          "value": "string"
        }
      ],
      "imageUrl": "string",
      "subtitle": "string",
      "title": "string"
    },
    "plainTextMessage": {
      "value": "string"
    },
    "ssmlMessage": {
      "value": "string"
    }
  }
]}
},
"successResponse": {
  "allowInterrupt": boolean,
  "messageGroups": [
    {
      "message": {
        "customPayload": {
          "value": "string"
        },
        "imageResponseCard": {
          "buttons": [
            {
              "text": "string",
              "value": "string"
            }
          ],
          "imageUrl": "string",
          "subtitle": "string",
          "title": "string"
        },
        "plainTextMessage": {
          "value": "string"
        },
        "ssmlMessage": {
          "value": "string"
        }
      },
      "variations": [
        {
          "customPayload": {
            "value": "string"
          },
          "imageResponseCard": {
            "buttons": [
            
          ]
        }
      ]
    }
  ]
}
"text": "string",
"value": "string"
},
"imageUrl": "string",
"subtitle": "string",
"title": "string"
},
"plainTextMessage": {
"value": "string"
},
"ssmlMessage": {
"value": "string"
}
]
"
"timeoutResponse": {
"allowInterrupt": boolean,
"messageGroups": [
{
"message": {
"customPayload": {
"value": "string"
},
"imageResponseCard": {
"buttons": [
{
"text": "string",
"value": "string"
}
],
"imageUrl": "string",
"subtitle": "string",
"title": "string"
},
"plainTextMessage": {
"value": "string"
},
"ssmlMessage": {
"value": "string"
}
},
"variations": [
{
"customPayload": {
"value": "string"
},
"imageResponseCard": {
"buttons": [
{
"text": "string",
"value": "string"
}
],
"imageUrl": "string",
"subtitle": "string",
"title": "string"
},
"plainTextMessage": {
"value": "string"
},
"ssmlMessage": {
"value": "string"
}
]


```json
{
  "inputContexts": [
    {
      "name": "string"
    }
  ],
  "intentClosingSetting": {
    "active": boolean,
    "closingResponse": {
      "allowInterrupt": boolean,
      "messageGroups": [
        {
          "message": {
            "customPayload": {
              "value": "string"
            },
            "imageResponseCard": {
              "buttons": [
                {
                  "text": "string",
                  "value": "string"
                }
              ],
              "imageUrl": "string",
              "subtitle": "string",
              "title": "string"
            },
            "plainTextMessage": {
              "value": "string"
            },
            "ssmlMessage": {
              "value": "string"
            }
          }
        }
      ]
    }
  }
}
```
"intentConfirmationSetting": {  "active": boolean,  "declinationResponse": {  "allowInterrupt": boolean,  "messageGroups": [  {  "message": {  "customPayload": {  "value": "string"  },  "imageResponseCard": {  "buttons": [  {  "text": "string",  "value": "string"  }  ],  "imageUrl": "string",  "subtitle": "string",  "title": "string"  },  "plainTextMessage": {  "value": "string"  },  "ssmlMessage": {  "value": "string"  }  },  "variations": [  {  "customPayload": {  "value": "string"  },  "imageResponseCard": {  "buttons": [  {  "text": "string",  "value": "string"  }  ],  "imageUrl": "string",  "subtitle": "string",  "title": "string"  },  "plainTextMessage": {  "value": "string"  },  "ssmlMessage": {  "value": "string"  }  }  ]  }  },  "allowInterrupt": boolean,  "maxRetries": number,  "messageGroups": [  {  "message": {  "customPayload": {  "value": "string"  },  "imageResponseCard": {  "buttons": [  {  "text": "string",  "value": "string"  }  ],  "imageUrl": "string",  "subtitle": "string",  "title": "string"  },  "plainTextMessage": {  "value": "string"  },  "ssmlMessage": {  "value": "string"  }  }  }  ]}
},
"imageResponseCard": {
  "buttons": [
    {
      "text": "string",
      "value": "string"
    }
  ],
  "imageUrl": "string",
  "subtitle": "string",
  "title": "string"
},
"plainTextMessage": {
  "value": "string"
},
"ssmlMessage": {
  "value": "string"
},
"variations": [
  {
    "customPayload": {
      "value": "string"
    },
    "imageResponseCard": {
      "buttons": [
        {
          "text": "string",
          "value": "string"
        }
      ],
      "imageUrl": "string",
      "subtitle": "string",
      "title": "string"
    },
    "plainTextMessage": {
      "value": "string"
    },
    "ssmlMessage": {
      "value": "string"
    }
  }
],
"intentId": "string",
"intentName": "string",
"kendraConfiguration": {
  "kendraIndex": "string",
  "queryFilterString": "string",
  "queryFilterStringEnabled": boolean
},
"lastUpdatedDateTime": number,
"localeId": "string",
"outputContexts": [
  {
    "name": "string",
    "timeToLiveInSeconds": number,
    "turnsToLive": number
  }
],
"parentIntentSignature": "string",
"sampleUtterances": [

}
"utterance": "string"
]
"slotPriorities": [
  {
    "priority": number,
    "slotId": "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.

**botId (p. 581)**

The identifier of the bot that contains the intent.

Type: String

Length Constraints: Fixed length of 10.

Pattern: \^[0-9a-zA-Z]+\$

**botVersion (p. 581)**

The version of the bot that contains the intent. Will always be DRAFT.

Type: String

Length Constraints: Fixed length of 5.

Pattern: \^DRAFT\$

**creationDateTime (p. 581)**

A timestamp of when the intent was created.

Type: Timestamp

**description (p. 581)**

The updated description of the intent.

Type: String

Length Constraints: Minimum length of 0. Maximum length of 200.

**dialogCodeHook (p. 581)**

The updated Lambda function called during each turn of the conversation with the user.

Type: [DialogCodeHookSettings (p. 703)]

**fulfillmentCodeHook (p. 581)**

The updated Lambda function called when the intent is ready for fulfillment.

Type: [FulfillmentCodeHookSettings (p. 711)]

**inputContexts (p. 581)**

The updated list of contexts that must be active for the intent to be considered by Amazon Lex.
The updated response that Amazon Lex sends the user when the intent is closed.

Type: IntentClosingSetting (p. 581)

The updated prompts that Amazon Lex sends to the user to confirm the completion of an intent.

Type: IntentConfirmationSetting (p. 581)

The identifier of the intent that was updated.

Type: String

Length Constraints: Fixed length of 10.

Pattern: ^[0-9a-zA-Z]+$

The updated name of the intent.

Type: String

Length Constraints: Minimum length of 1. Maximum length of 100.

Pattern: ^([0-9a-zA-Z][_-]?)+$

The updated configuration for connecting to an Amazon Kendra index with the AMAZON.KendraSearchIntent intent.

Type: KendraConfiguration (p. 581) object

A timestamp of the last time that the intent was modified.

Type: Timestamp

The updated language and locale of the intent.

Type: String

The updated list of contexts that Amazon Lex activates when the intent is fulfilled.

Type: Array of OutputContext (p. 739) objects

Array Members: Minimum number of 0 items. Maximum number of 10 items.

The updated built-in intent that is the parent of this intent.

Type: String
**sampleUtterances (p. 581)**

The updated list of sample utterances for the intent.

Type: Array of SampleUtterance (p. 751) objects

**slotPriorities (p. 581)**

The updated list of slots and their priorities that are elicited from the user for the intent.

Type: Array of SlotPriority (p. 757) objects

**Errors**

For information about the errors that are common to all actions, see Common Errors (p. 821).

**ConflictException**

The action that you tried to perform couldn't be completed because the resource is in a conflicting state. For example, deleting a bot that is in the CREATING state. Try your request again.

HTTP Status Code: 409

**InternalServerException**

The service encountered an unexpected condition. Try your request again.

HTTP Status Code: 500

**PreconditionFailedException**

Your request couldn't be completed because one or more request fields aren't valid. Check the fields in your request and try again.

HTTP Status Code: 412

**ServiceQuotaExceededException**

You have reached a quota for your bot.

HTTP Status Code: 402

**ThrottlingException**

Your request rate is too high. Reduce the frequency of requests.

HTTP Status Code: 429

**ValidationException**

One of the input parameters in your request isn't valid. Check the parameters and try your request again.

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 V2
- AWS SDK for JavaScript
- AWS SDK for PHP V3
- AWS SDK for Python
- AWS SDK for Ruby V3
UpdateResourcePolicy
Service: Amazon Lex Model Building V2
Replaces the existing resource policy for a bot or bot alias with a new one. If the policy doesn't exist, Amazon Lex returns an exception.

Request Syntax

```plaintext
PUT /policy/{resourceArn}/?expectedRevisionId={expectedRevisionId} HTTP/1.1
Content-type: application/json

{
  "policy": "string"
}
```

URI Request Parameters

The request uses the following URI parameters.

expectedRevisionId (p. 593)
The identifier of the revision of the policy to update. If this revision ID doesn't match the current revision ID, Amazon Lex throws an exception.

If you don't specify a revision, Amazon Lex overwrites the contents of the policy with the new values.

Pattern: ^[0-9]+$

resourceArn (p. 593)
The Amazon Resource Name (ARN) of the bot or bot alias that the resource policy is attached to.
Length Constraints: Minimum length of 1. Maximum length of 1011.
Required: Yes

Request Body

The request accepts the following data in JSON format.

policy (p. 593)
A resource policy to add to the resource. The policy is a JSON structure that contains one or more statements that define the policy. The policy must follow the IAM syntax. For more information about the contents of a JSON policy document, see IAM JSON policy reference.

If the policy isn't valid, Amazon Lex returns a validation exception.

Type: String
Length Constraints: Minimum length of 2.
Required: Yes

Response Syntax

```
HTTP/1.1 200
```
Content-type: application/json

{
   "resourceArn": "string",
   "revisionId": "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.

**resourceArn (p. 593)**

The Amazon Resource Name (ARN) of the bot or bot alias that the resource policy is attached to.

Type: String

Length Constraints: Minimum length of 1. Maximum length of 1011.

**revisionId (p. 593)**

The current revision of the resource policy. Use the revision ID to make sure that you are updating
the most current version of a resource policy when you add a policy statement to a resource, delete a
resource, or update a resource.

Type: String


Pattern: ^[0-9]+$

Errors

For information about the errors that are common to all actions, see Common Errors (p. 821).

**InternalServerException**

The service encountered an unexpected condition. Try your request again.

HTTP Status Code: 500

**PreconditionFailedException**

Your request couldn't be completed because one or more request fields aren't valid. Check the fields
in your request and try again.

HTTP Status Code: 412

**ResourceNotFoundException**

You asked to describe a resource that doesn't exist. Check the resource that you are requesting and
try again.

HTTP Status Code: 404

**ServiceQuotaExceededException**

You have reached a quota for your bot.

HTTP Status Code: 402
**ThrottlingException**

Your request rate is too high. Reduce the frequency of requests.

HTTP Status Code: 429

**ValidationException**

One of the input parameters in your request isn't valid. Check the parameters and try your request again.

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 V2
- AWS SDK for JavaScript
- AWS SDK for PHP V3
- AWS SDK for Python
- AWS SDK for Ruby V3
UpdateSlot
Service: Amazon Lex Model Building V2

Updates the settings for a slot.

Request Syntax

PUT /bots/botId/botversions/botVersion/botlocales/localeId/intents/intentId/slots/slotId/
HTTP/1.1
Content-type: application/json

{
    "description": "string",
    "multipleValuesSetting": {
        "allowMultipleValues": boolean
    },
    "obfuscationSetting": {
        "obfuscationSettingType": "string"
    },
    "slotName": "string",
    "slotTypeId": "string",
    "valueElicitationSetting": {
        "defaultValueSpecification": {
            "defaultValueList": [
                {
                    "defaultValue": "string"
                }
            ]
        },
        "promptSpecification": {
            "allowInterrupt": boolean,
            "maxRetries": number,
            "messageGroups": [
                {
                    "message": {
                        "customPayload": {
                            "value": "string"
                        },
                        "imageResponseCard": {
                            "buttons": [
                                {
                                    "text": "string",
                                    "value": "string"
                                }
                            ],
                            "imageUrl": "string",
                            "subtitle": "string",
                            "title": "string"
                        },
                        "plainTextMessage": {
                            "value": "string"
                        },
                        "ssmlMessage": {
                            "value": "string"
                        }
                    }
                },
                {
                    "customPayload": {
                        "value": "string"
                    },
                    "imageResponseCard": {
                        "buttons": [
                            {
                                "text": "string",
                                "value": "string"
                            }
                        ],
                        "imageUrl": "string",
                        "subtitle": "string",
                        "title": "string"
                    }
                }
            ]
        }
    }
}
"text": "string",
"value": "string"
},
"imageUrl": "string",
"subtitle": "string",
"title": "string"
},
"plainTextMessage": {
"value": "string"
},
"ssmlMessage": {
"value": "string"
}
}
}
,"sampleUtterances": [
{
"utterance": "string"
}
],
"slotConstraint": "string",
"waitAndContinueSpecification": {
"active": boolean,
"continueResponse": {
"allowInterrupt": boolean,
"messageGroups": [
{
"message": {
"customPayload": {
"value": "string"
},
"imageResponseCard": {
"buttons": [
{
"text": "string",
"value": "string"
}
],
"imageUrl": "string",
"subtitle": "string",
"title": "string"
},
"plainTextMessage": {
"value": "string"
},
"ssmlMessage": {
"value": "string"
}
}
],
"variations": [
{
"customPayload": {
"value": "string"
},
"imageResponseCard": {
"buttons": [
{
"text": "string",
"value": "string"
}
],
"imageUrl": "string",
"subtitle": "string",
"title": "string"
},
"plainTextMessage": {
"value": "string"
},
"ssmlMessage": {
"value": "string"
}
}]
}
"subtitle": "string",
"title": "string"
},
"plainTextMessage": {
 "value": "string"
},
"ssmlMessage": {
 "value": "string"
 }

"stillWaitingResponse": {
 "allowInterrupt": boolean,
 "frequencyInSeconds": number,
 "messageGroups": [
 { 
 "message": {
 
 "customPayload": {
 "value": "string"
 },
 "imageResponseCard": {
 "buttons": [
 { 
 "text": "string",
 "value": "string"
 }
 ],
 "imageUrl": "string",
 "subtitle": "string",
 "title": "string"
 },
"plainTextMessage": {
 "value": "string"
 },
"ssmlMessage": {
 "value": "string"
 }
 },
"variations": [
 { 
 
 "customPayload": {
 "value": "string"
 },
 "imageResponseCard": {
 "buttons": [
 { 
 "text": "string",
 "value": "string"
 }
 ],
 "imageUrl": "string",
 "subtitle": "string",
 "title": "string"
 },
"plainTextMessage": {
 "value": "string"
 },
"ssmlMessage": {
 "value": "string"
 }
 ]
 ]
}
[ },
   "timeoutInSeconds": number
},
"waitingResponse": {
  "allowInterrupt": boolean,
  "messageGroups": [
    { "message": {
      "customPayload": {
        "value": "string"
      },
      "imageResponseCard": {
        "buttons": [
          {
            "text": "string",
            "value": "string"
          }
        ],
        "imageUrl": "string",
        "subtitle": "string",
        "title": "string"
      },
      "plainTextMessage": {
        "value": "string"
      },
      "ssmlMessage": {
        "value": "string"
      }
    },
    { "customPayload": {
      "value": "string"
    },
    "imageResponseCard": {
      "buttons": [
        {
          "text": "string",
          "value": "string"
        }
      ],
      "imageUrl": "string",
      "subtitle": "string",
      "title": "string"
    },
    "plainTextMessage": {
      "value": "string"
    },
    "ssmlMessage": {
      "value": "string"
    }
    },
    { "customPayload": {
      "value": "string"
    },
    "imageResponseCard": {
      "buttons": [
        {
          "text": "string",
          "value": "string"
        }
      ],
      "imageUrl": "string",
      "subtitle": "string",
      "title": "string"
    },
    "plainTextMessage": {
      "value": "string"
    },
    "ssmlMessage": {
      "value": "string"
    }
    }
  ]
},
"variations": [
  { "customPayload": {
    "value": "string"
  },
  "imageResponseCard": {
    "buttons": [
      { "text": "string",
        "value": "string"
      }
    ],
    "imageUrl": "string",
    "subtitle": "string",
    "title": "string"
  },
  "plainTextMessage": {
    "value": "string"
  },
  "ssmlMessage": {
    "value": "string"
  }
  }
]}

URI Request Parameters

The request uses the following URI parameters.
**botId (p. 596)**

The unique identifier of the bot that contains the slot.

Length Constraints: Fixed length of 10.

Pattern: `^[0-9a-zA-Z]+$`

Required: Yes

**botVersion (p. 596)**

The version of the bot that contains the slot. Must always be DRAFT.

Length Constraints: Fixed length of 5.

Pattern: `^DRAFT$`

Required: Yes

**intentId (p. 596)**

The identifier of the intent that contains the slot.

Length Constraints: Fixed length of 10.

Pattern: `^[0-9a-zA-Z]+$`

Required: Yes

**localeId (p. 596)**

The identifier of the language and locale that contains the slot. The string must match one of the supported locales. For more information, see Supported languages.

Required: Yes

**slotId (p. 596)**

The unique identifier for the slot to update.

Length Constraints: Fixed length of 10.

Pattern: `^[0-9a-zA-Z]+$`

Required: Yes

**Request Body**

The request accepts the following data in JSON format.

**description (p. 596)**

The new description for the slot.

Type: String

Length Constraints: Minimum length of 0. Maximum length of 200.

Required: No

**multipleValuesSetting (p. 596)**

Determines whether the slot accepts multiple values in one response. Multiple value slots are only available in the en-US locale. If you set this value to true in any other locale, Amazon Lex throws a ValidationException.
If the `multipleValuesSetting` is not set, the default value is `false`.

Type: `MultipleValuesSetting (p. 737)` object

Required: No

**obfuscationSetting (p. 596)**

New settings that determine how slot values are formatted in Amazon CloudWatch logs.

Type: `ObfuscationSetting (p. 738)` object

Required: No

**slotName (p. 596)**

The new name for the slot.

Type: `String`

Length Constraints: Minimum length of 1. Maximum length of 100.

Pattern: `^[0-9a-zA-Z][_-]?]+$`

Required: Yes

**slotTypeId (p. 596)**

The unique identifier of the new slot type to associate with this slot.

Type: `String`


Pattern: `^((AMAZON\.|)[a-zA-Z\_]+?|\[0-9a-zA-Z\_]+?)[0-9a-zA-Z\_]+?\[0-9a-zA-Z\_]+?$`

Required: Yes

**valueElicitationSetting (p. 596)**

A new set of prompts that Amazon Lex sends to the user to elicit a response the provides a value for the slot.

Type: `SlotValueElicitationSetting (p. 767)` object

Required: Yes

---

**Response Syntax**

```
HTTP/1.1 200
Content-type: application/json

{
  "botId": "string",
  "botVersion": "string",
  "creationDateTime": number,
  "description": "string",
  "intentId": "string",
  "lastUpdatedDateTime": number,
  "localeId": "string",
  "multipleValuesSetting": {
    "allowMultipleValues": boolean
  },
  "obfuscationSetting": {
    "obfuscationSettingType": "string"
  }
}
```
"slotId": "string",
"slotName": "string",
"slotTypeId": "string",
"valueElicitationSetting": {
  "defaultValueSpecification": {
    "defaultValueList": [
      {
        "defaultValue": "string"
      }
    ],
  },
  "promptSpecification": {
    "allowInterrupt": boolean,
    "maxRetries": number,
    "messageGroups": [{
      "message": {
        "customPayload": {
          "value": "string"
        },
        "imageResponseCard": {
          "buttons": [
            {
              "text": "string",
              "value": "string"
            }
          ],
          "imageUrl": "string",
          "subtitle": "string",
          "title": "string"
        },
        "plainTextMessage": {
          "value": "string"
        },
        "ssmlMessage": {
          "value": "string"
        }
      },
      "variations": [{
        "customPayload": {
          "value": "string"
        },
        "imageResponseCard": {
          "buttons": [
            {
              "text": "string",
              "value": "string"
            }
          ],
          "imageUrl": "string",
          "subtitle": "string",
          "title": "string"
        },
        "plainTextMessage": {
          "value": "string"
        },
        "ssmlMessage": {
          "value": "string"
        }
      }]
    }
  ]
}
"sampleUtterances": [
  {
    "utterance": "string"
  }
],
"slotConstraint": "string",
"waitAndContinueSpecification": {
  "active": boolean,
  "continueResponse": {
    "allowInterrupt": boolean,
    "messageGroups": [
      {
        "message": {
          "customPayload": {
            "value": "string"
          },
          "imageResponseCard": {
            "buttons": [
              {
                "text": "string",
                "value": "string"
              }
            ],
            "imageUrl": "string",
            "subtitle": "string",
            "title": "string"
          },
          "plainTextMessage": {
            "value": "string"
          },
          "ssmlMessage": {
            "value": "string"
          }
        }
      },
      {
        "customPayload": {
          "value": "string"
        },
        "imageResponseCard": {
          "buttons": [
            {
              "text": "string",
              "value": "string"
            }
          ],
          "imageUrl": "string",
          "subtitle": "string",
          "title": "string"
        },
        "plainTextMessage": {
          "value": "string"
        },
        "ssmlMessage": {
          "value": "string"
        }
      }
    ],
    "stillWaitingResponse": {
      "allowInterrupt": boolean,
      "frequencyInSeconds": number,
      "messageGroups": [
        {
        }
      ]
    }
  }
]
"message": {
"customPayload": {
"value": "string"
},
"imageResponseCard": {
"buttons": [
{
"text": "string",
"value": "string"
}
],
"imageUrl": "string",
"subtitle": "string",
"title": "string"
},
"plainTextMessage": {
"value": "string"
},
"ssmlMessage": {
"value": "string"
}
},
"variations": [
{
"customPayload": {
"value": "string"
},
"imageResponseCard": {
"buttons": [
{
"text": "string",
"value": "string"
}
],
"imageUrl": "string",
"subtitle": "string",
"title": "string"
},
"plainTextMessage": {
"value": "string"
},
"ssmlMessage": {
"value": "string"
}
}
],
"timeoutInSeconds": number
},
"waitingResponse": {
"allowInterrupt": boolean,
"messageGroups": [
{
"message": {
"customPayload": {
"value": "string"
},
"imageResponseCard": {
"buttons": [
{
"text": "string",
"value": "string"
}
],
"imageUrl": "string",
"subtitle": "string",
"title": "string"
},
"plainTextMessage": {
"value": "string"
},
"ssmlMessage": {
"value": "string"
}
}
],
"timeoutInSeconds": number
}
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.

**botId (p. 601)**

The identifier of the bot that contains the slot.

Type: String
Length Constraints: Fixed length of 10.

Pattern: ^[0-9a-zA-Z]+$**

**botVersion (p. 601)**

The identifier of the slot version that contains the slot. Will always be **DRAFT**.

Type: String
Length Constraints: Fixed length of 5.
Pattern: ^DRAFT$

**creationDateTime (p. 601)**

The timestamp of the date and time that the slot was created.

Type: Timestamp

**description (p. 601)**

The updated description of the bot.

Type: String

Length Constraints: Minimum length of 0. Maximum length of 200.

**intentId (p. 601)**

The intent that contains the slot.

Type: String

Length Constraints: Fixed length of 10.

Pattern: ^[0-9a-zA-Z]+$

**lastUpdatedDateTime (p. 601)**

The timestamp of the date and time that the slot was last updated.

Type: Timestamp

**localeId (p. 601)**

The locale that contains the slot.

Type: String

**multipleValuesSetting (p. 601)**

Indicates whether the slot accepts multiple values in one response.

Type: MultipleValuesSetting (p. 737) object

**obfuscationSetting (p. 601)**

The updated setting that determines whether the slot value is obfuscated in the Amazon CloudWatch logs.

Type: ObfuscationSetting (p. 738) object

**slotId (p. 601)**

The unique identifier of the slot that was updated.

Type: String

Length Constraints: Fixed length of 10.

Pattern: ^[0-9a-zA-Z]+$
Pattern: ^([0-9a-zA-Z][_-]?)+$

**slotTypeId (p. 601)**

The updated identifier of the slot type that provides values for the slot.

Type: String


Pattern: ^((AMAZON\.)[a-zA-Z_]+?|\[0-9a-zA-Z\]+)$

**valueElicitationSetting (p. 601)**

The updated prompts that Amazon Lex sends to the user to elicit a response that provides a value for the slot.

Type: SlotValueElicitationSetting (p. 767) object

**Errors**

For information about the errors that are common to all actions, see Common Errors (p. 821).

**ConflictException**

The action that you tried to perform couldn't be completed because the resource is in a conflicting state. For example, deleting a bot that is in the CREATING state. Try your request again.

HTTP Status Code: 409

**InternalServerException**

The service encountered an unexpected condition. Try your request again.

HTTP Status Code: 500

**PreconditionFailedException**

Your request couldn't be completed because one or more request fields aren't valid. Check the fields in your request and try again.

HTTP Status Code: 412

**ServiceQuotaExceededException**

You have reached a quota for your bot.

HTTP Status Code: 402

**ThrottlingException**

Your request rate is too high. Reduce the frequency of requests.

HTTP Status Code: 429

**ValidationException**

One of the input parameters in your request isn't valid. Check the parameters and try your request again.

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 V2
• AWS SDK for JavaScript
• AWS SDK for PHP V3
• AWS SDK for Python
• AWS SDK for Ruby V3
UpdateSlotType
Service: Amazon Lex Model Building V2

Updates the configuration of an existing slot type.

Request Syntax

```
PUT /bots/botId/botversions/botVersion/botlocales/localeId/slottypes/slotTypeId/ HTTP/1.1
Content-type: application/json

{
  "description": "string",
  "externalSourceSetting": {
    "grammarSlotTypeSetting": {
      "source": {
        "kmsKeyArn": "string",
        "s3BucketName": "string",
        "s3ObjectKey": "string"
      }
    },
    "parentSlotTypeSignature": "string",
    "slotTypeName": "string",
    "slotTypeValues": [
      {
        "sampleValue": {
          "value": "string"
        },
        "synonyms": [
          {
            "value": "string"
          }
        ]
      }
    ],
    "valueSelectionSetting": {
      "advancedRecognitionSetting": {
        "audioRecognitionStrategy": "string"
      },
      "regexFilter": {
        "pattern": "string"
      },
      "resolutionStrategy": "string"
    }
  }
}
```

URI Request Parameters

The request uses the following URI parameters.

**botId (p. 609)**

The identifier of the bot that contains the slot type.

Length Constraints: Fixed length of 10.

Pattern: ^[0-9a-zA-Z]+$

Required: Yes

**botVersion (p. 609)**

The version of the bot that contains the slot type. Must be DRAFT.
localeId (p. 609)
The identifier of the language and locale that contains the slot type. The string must match one of the supported locales. For more information, see Supported languages.

Required: Yes

slotTypeId (p. 609)
The unique identifier of the slot type to update.

Length Constraints: Fixed length of 10.

Pattern: ^[0-9a-zA-Z]+$

Required: Yes

Request Body
The request accepts the following data in JSON format.

description (p. 609)
The new description of the slot type.

Type: String

Length Constraints: Minimum length of 0. Maximum length of 200.

Required: No

externalSourceSetting (p. 609)
Provides information about the external source of the slot type's definition.

Type: ExternalSourceSetting (p. 710) object

Required: No

parentSlotTypeSignature (p. 609)
The new built-in slot type that should be used as the parent of this slot type.

Type: String

Required: No

slotTypeName (p. 609)
The new name of the slot type.

Type: String

Length Constraints: Minimum length of 1. Maximum length of 100.

Pattern: ^([0-9a-zA-Z][-_]?)+$ 

Required: Yes
slotTypeValues (p. 609)

A new list of values and their optional synonyms that define the values that the slot type can take.

Type: Array of SlotTypeValue (p. 766) objects

Array Members: Minimum number of 1 item. Maximum number of 10000 items.

Required: No

valueSelectionSetting (p. 609)

The strategy that Amazon Lex should use when deciding on a value from the list of slot type values.

Type: SlotValueSelectionSetting (p. 770) object

Required: No

Response Syntax

```
HTTP/1.1 202
Content-type: application/json

{
  "botId": "string",
  "botVersion": "string",
  "creationDateTime": number,
  "description": "string",
  "externalSourceSetting": {
    "grammarSlotTypeSetting": {
      "source": {
        "kmsKeyArn": "string",
        "s3BucketName": "string",
        "s3ObjectKey": "string"
      }
    }
  },
  "lastUpdatedDateTime": number,
  "localeId": "string",
  "parentSlotTypeSignature": "string",
  "slotTypeId": "string",
  "slotTypeName": "string",
  "slotTypeValues": [
    {
      "sampleValue": {
        "value": "string"
      },
      "synonyms": [
        {
          "value": "string"
        }
      ]
    }
  ],
  "valueSelectionSetting": {
    "advancedRecognitionSetting": {
      "audioRecognitionStrategy": "string"
    },
    "regexFilter": {
      "pattern": "string"
    },
    "resolutionStrategy": "string"
  }
}
```
Response Elements

If the action is successful, the service sends back an HTTP 202 response.

The following data is returned in JSON format by the service.

botId (p. 611)

The identifier of the bot that contains the slot type.

Type: String

Length Constraints: Fixed length of 10.

Pattern: ^[0-9a-zA-Z]+$

botVersion (p. 611)

The version of the bot that contains the slot type. This is always DRAFT.

Type: String

Length Constraints: Fixed length of 5.

Pattern: ^DRAFT$

creationDateTime (p. 611)

The timestamp of the date and time that the slot type was created.

Type: Timestamp

description (p. 611)

The updated description of the slot type.

Type: String

Length Constraints: Minimum length of 0. Maximum length of 200.

externalSourceSetting (p. 611)

Provides information about the external source of the slot type's definition.

Type: ExternalSourceSetting (p. 710) object

lastUpdatedDateTime (p. 611)

A timestamp of the date and time that the slot type was last updated.

Type: Timestamp

localeId (p. 611)

The language and locale of the updated slot type.

Type: String

parentSlotTypeSignature (p. 611)

The updated signature of the built-in slot type that is the parent of this slot type.

Type: String
slotTypeId (p. 611)
The unique identifier of the updated slot type.
Type: String
Length Constraints: Fixed length of 10.
Pattern: ^[0-9a-zA-Z]+$ slotTypeName (p. 611)
The updated name of the slot type.
Type: String
Length Constraints: Minimum length of 1. Maximum length of 100.
Pattern: ^([0-9a-zA-Z][\-_]?)+$ slotTypeValues (p. 611)
The updated values that the slot type provides.
Type: Array of SlotTypeValue (p. 766) objects
Array Members: Minimum number of 1 item. Maximum number of 10000 items.
valueSelectionSetting (p. 611)
The updated strategy that Amazon Lex uses to determine which value to select from the slot type.
Type: SlotValueSelectionSetting (p. 770) object

Errors
For information about the errors that are common to all actions, see Common Errors (p. 821).

ConflictException
The action that you tried to perform couldn't be completed because the resource is in a conflicting state. For example, deleting a bot that is in the CREATING state. Try your request again.
HTTP Status Code: 409

InternalServerException
The service encountered an unexpected condition. Try your request again.
HTTP Status Code: 500

PreconditionFailedException
Your request couldn't be completed because one or more request fields aren't valid. Check the fields in your request and try again.
HTTP Status Code: 412

ServiceQuotaExceededException
You have reached a quota for your bot.
HTTP Status Code: 402

ThrottlingException
Your request rate is too high. Reduce the frequency of requests.
HTTP Status Code: 429

**ValidationException**

One of the input parameters in your request isn't valid. Check the parameters and try your request again.

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 V2
- AWS SDK for JavaScript
- AWS SDK for PHP V3
- AWS SDK for Python
- AWS SDK for Ruby V3

**Amazon Lex Runtime V2**

The following actions are supported by Amazon Lex Runtime V2:

- `DeleteSession` (p. 615)
- `GetSession` (p. 618)
- `PutSession` (p. 623)
- `RecognizeText` (p. 628)
- `RecognizeUtterance` (p. 635)
- `StartConversation` (p. 642)
DeleteSession

Service: Amazon Lex Runtime V2

Removes session information for a specified bot, alias, and user ID.

You can use this operation to restart a conversation with a bot. When you remove a session, the entire history of the session is removed so that you can start again.

You don't need to delete a session. Sessions have a time limit and will expire. Set the session time limit when you create the bot. The default is 5 minutes, but you can specify anything between 1 minute and 24 hours.

If you specify a bot or alias ID that doesn't exist, you receive a BadRequestException.

If the locale doesn't exist in the bot, or if the locale hasn't been enables for the alias, you receive a BadRequestException.

Request Syntax

```
DELETE /bots/botId/botAliases/botAliasId/botLocales/localeId/sessions/sessionId HTTP/1.1
```

URI Request Parameters

The request uses the following URI parameters.

**botAliasId (p. 615)**

The alias identifier in use for the bot that contains the session data.

Required: Yes

**botId (p. 615)**

The identifier of the bot that contains the session data.

Length Constraints: Fixed length of 10.

Pattern: `^[0-9a-zA-Z]+$`

Required: Yes

**localeId (p. 615)**

The locale where the session is in use.

Length Constraints: Minimum length of 1.

Required: Yes

**sessionId (p. 615)**

The identifier of the session to delete.


Pattern: `[0-9a-zA-Z_:-]+$`

Required: Yes

Request Body

The request does not have a request body.
Response Syntax

```json
HTTP/1.1 200
Content-type: application/json
{
    "botAliasId": "string",
    "botId": "string",
    "localeId": "string",
    "sessionId": "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.

**botAliasId (p. 616)**

The alias identifier in use for the bot that contained the session data.

Type: String

**botId (p. 616)**

The identifier of the bot that contained the session data.

Type: String

Length Constraints: Fixed length of 10.

Pattern: ^[0-9a-zA-Z]+$

**localeId (p. 616)**

The locale where the session was used.

Type: String

Length Constraints: Minimum length of 1.

**sessionId (p. 616)**

The identifier of the deleted session.

Type: String


Pattern: [0-9a-zA-Z._-]+}

Errors

For information about the errors that are common to all actions, see Common Errors (p. 821).

**AccessDeniedException**

HTTP Status Code: 403

**ConflictException**
HTTP Status Code: 409
InternalServerErrorException

HTTP Status Code: 500
ResourceNotFoundException

HTTP Status Code: 404
ThrottlingException

HTTP Status Code: 429
ValidationException

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 V2
- AWS SDK for JavaScript
- AWS SDK for PHP V3
- AWS SDK for Python
- AWS SDK for Ruby V3
GetSession
Service: Amazon Lex Runtime V2

Returns session information for a specified bot, alias, and user.
For example, you can use this operation to retrieve session information for a user that has left a long-running session in use.

If the bot, alias, or session identifier doesn't exist, Amazon Lex V2 returns a BadRequestException. If the locale doesn't exist or is not enabled for the alias, you receive a BadRequestException.

Request Syntax

```
GET /bots/botId/botAliases/botAliasId/botLocales/localeId/sessions/sessionId HTTP/1.1
```

URI Request Parameters

The request uses the following URI parameters.

botAliasId (p. 618)
The alias identifier in use for the bot that contains the session data.
Required: Yes

botId (p. 618)
The identifier of the bot that contains the session data.
Length Constraints: Fixed length of 10.
Pattern: ^[0-9a-zA-Z]+$
Required: Yes

localeId (p. 618)
The locale where the session is in use.
Length Constraints: Minimum length of 1.
Required: Yes

sessionId (p. 618)
The identifier of the session to return.
Pattern: [0-9a-zA-Z_.-]+$
Required: Yes

Request Body

The request does not have a request body.

Response Syntax

```
HTTP/1.1 200
Content-type: application/json

{
```
"interpretations": [  
  {  
    "intent": {  
      "confirmationState": "string",  
      "name": "string",  
      "slots": {  
        "string": {  
          "shape": "string",  
          "value": {  
            "interpretedValue": "string",  
            "originalValue": "string",  
            "resolvedValues": [ "string" ]  
          },  
          "values": [  
            "Slot"  
          ]  
        }  
      },  
      "state": "string"  
    },  
    "nluConfidence": {  
      "score": number  
    },  
    "sentimentResponse": {  
      "sentiment": "string",  
      "sentimentScore": {  
        "mixed": number,  
        "negative": number,  
        "neutral": number,  
        "positive": number  
      }  
    }  
  }  
],  
"messages": [  
  {  
    "content": "string",  
    "contentType": "string",  
    "imageResponseCard": {  
      "buttons": [  
        {  
          "text": "string",  
          "value": "string"  
        }  
      ],  
      "imageUrl": "string",  
      "subtitle": "string",  
      "title": "string"  
    }  
  }  
],  
"sessionId": "string",  
"sessionState": {  
  "activeContexts": [  
    {  
      "contextAttributes": {  
        "string": "string"  
      },  
      "name": "string",  
      "timeToLive": {  
        "timeToLiveInSeconds": number,  
        "turnsToLive": number  
      }  
    }  
  ],  
  "dialogAction": {  

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.

interpretations (p. 618)

A list of intents that Amazon Lex V2 determined might satisfy the user’s utterance.

Each interpretation includes the intent, a score that indicates how confident Amazon Lex V2 is that the interpretation is the correct one, and an optional sentiment response that indicates the sentiment expressed in the utterance.

Type: Array of Interpretation (p. 801) objects

Array Members: Maximum number of 5 items.

messages (p. 618)

A list of messages that were last sent to the user. The messages are ordered based on the order that your returned the messages from your Lambda function or the order that messages are defined in the bot.
Type: Array of Message (p. 802) objects

Array Members: Maximum number of 10 items.

sessionId (p. 618)

The identifier of the returned session.

Type: String

Length Constraints: Minimum length of 1.

sessionState (p. 618)

Represents the current state of the dialog between the user and the bot.

You can use this to determine the progress of the conversation and what the next action might be.

Type: SessionState (p. 810) object

Errors

For information about the errors that are common to all actions, see Common Errors (p. 821).

AccessDeniedException

HTTP Status Code: 403

InternalServerError

HTTP Status Code: 500

ResourceNotFoundException

HTTP Status Code: 404

ThrottlingException

HTTP Status Code: 429

ValidationException

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 V2
- AWS SDK for JavaScript
- AWS SDK for PHP V3
- AWS SDK for Python
- AWS SDK for Ruby V3
**PutSession**  
Service: Amazon Lex Runtime V2

Creates a new session or modifies an existing session with an Amazon Lex V2 bot. Use this operation to enable your application to set the state of the bot.

**Request Syntax**

```
POST /bots/botId/botAliases/botAliasId/botLocales/localeId/sessions/sessionId HTTP/1.1
ResponseContentType: responseContentType
Content-type: application/json
{
    "messages": [
        {
            "content": "string",
            "contentType": "string",
            "imageResponseCard": {
                "buttons": [
                    {
                        "text": "string",
                        "value": "string"
                    }
                ],
                "imageUrl": "string",
                "subtitle": "string",
                "title": "string"
            }
        }
    ],
    "requestAttributes": {
        "String": "string"
    },
    "sessionState": {
        "activeContexts": [
            {
                "contextAttributes": {
                    "string": "string"
                },
                "name": "string",
                "timeToLive": {
                    "timeToLiveInSeconds": number,
                    "turnsToLive": number
                }
            }
        ],
        "dialogAction": {
            "slotElicitationStyle": "string",
            "slotToElicit": "string",
            "type": "string"
        },
        "intent": {
            "confirmationState": "string",
            "name": "string",
            "slots": {
                "string": {
                    "shape": "string",
                    "value": {
                        "interpretedValue": "string",
                        "originalValue": "string",
                        "resolvedValues": [ "string" ]
                    }
                },
                "values": [ "Slot"
```
URI Request Parameters

The request uses the following URI parameters.

**botAliasId (p. 623)**

The alias identifier of the bot that receives the session data.

Required: Yes

**botId (p. 623)**

The identifier of the bot that receives the session data.

Length Constraints: Fixed length of 10.

Pattern: ^[0-9a-zA-Z]+$  

Required: Yes

**localeId (p. 623)**

The locale where the session is in use.

Length Constraints: Minimum length of 1.

Required: Yes

**responseContentType (p. 623)**

The message that Amazon Lex V2 returns in the response can be either text or speech depending on the value of this parameter.

- If the value is `text/plain; charset=utf-8`, Amazon Lex V2 returns text in the response.

Length Constraints: Minimum length of 1.

**sessionId (p. 623)**

The identifier of the session that receives the session data.

Pattern: [0-9a-zA-Z._:-]+

Required: Yes

Request Body

The request accepts the following data in JSON format.

messages (p. 623)

A list of messages to send to the user. Messages are sent in the order that they are defined in the list.

Type: Array of Message (p. 802) objects

Array Members: Maximum number of 10 items.

Required: No

requestAttributes (p. 623)

Request-specific information passed between Amazon Lex V2 and the client application.

The namespace x-amz-lex: is reserved for special attributes. Don't create any request attributes with the prefix x-amz-lex:.

Type: String to string map

Key Length Constraints: Minimum length of 1.

Required: No

sessionState (p. 623)

Sets the state of the session with the user. You can use this to set the current intent, attributes, context, and dialog action. Use the dialog action to determine the next step that Amazon Lex V2 should use in the conversation with the user.

Type: SessionState (p. 810) object

Required: Yes

Response Syntax

HTTP/1.1 200
Content-Type: contentType
x-amz-lex-messages: messages
x-amz-lex-session-state: sessionState
x-amz-lex-request-attributes: requestAttributes
x-amz-lex-session-id: sessionId
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. 625)

The type of response. Same as the type specified in the responseContentType field in the request.

Length Constraints: Minimum length of 1.

messages (p. 625)

A list of messages that were last sent to the user. The messages are ordered based on how you return the messages from you Lambda function or the order that the messages are defined in the bot.

Length Constraints: Minimum length of 1.

requestAttributes (p. 625)

Request-specific information passed between the client application and Amazon Lex V2. These are the same as the requestAttribute parameter in the call to the PutSession operation.

Length Constraints: Minimum length of 1.

sessionId (p. 625)

The identifier of the session that received the data.


Pattern: [0-9a-zA-Z._:-]+  

sessionState (p. 625)

Represents the current state of the dialog between the user and the bot.

Use this to determine the progress of the conversation and what the next action may be.

Length Constraints: Minimum length of 1.

The response returns the following as the HTTP body.

audioStream (p. 625)

If the requested content type was audio, the audio version of the message to convey to the user.

Errors

For information about the errors that are common to all actions, see Common Errors (p. 821).

AccessDeniedException

HTTP Status Code: 403

BadGatewayException

HTTP Status Code: 502

ConflictException

HTTP Status Code: 409

DependencyFailedException
HTTP Status Code: 424
*InternalServerException*

HTTP Status Code: 500
*ResourceNotFoundException*

HTTP Status Code: 404
*ThrottlingException*

HTTP Status Code: 429
*ValidationException*

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 V2
- AWS SDK for JavaScript
- AWS SDK for PHP V3
- AWS SDK for Python
- AWS SDK for Ruby V3
RecognizeText
Service: Amazon Lex Runtime V2

Sends user input to Amazon Lex V2. Client applications use this API to send requests to Amazon Lex V2
at runtime. Amazon Lex V2 then interprets the user input using the machine learning model that it build
for the bot.

In response, Amazon Lex V2 returns the next message to convey to the user and an optional response
card to display.

If the optional post-fulfillment response is specified, the messages are returned as follows. For more
information, see PostFulfillmentStatusSpecification.

- **Success message** - Returned if the Lambda function completes successfully and the intent state is
  fulfilled or ready fulfillment if the message is present.
- **Failed message** - The failed message is returned if the Lambda function throws an exception or if the
  Lambda function returns a failed intent state without a message.
- **Timeout message** - If you don't configure a timeout message and a timeout, and the Lambda function
doesn't return within 30 seconds, the timeout message is returned. If you configure a timeout, the
timeout message is returned when the period times out.

For more information, see Completion message.

**Request Syntax**

```
POST /bots/botId/botAliases/botAliasId/botLocales/localeId/sessions/sessionId/text HTTP/1.1
Content-type: application/json

{
  "requestAttributes": {
    "string" : "string"
  },
  "sessionState": {
    "activeContexts": [
      {
        "contextAttributes": {
          "string" : "string"
        },
        "name": "string",
        "timeToLive": {
          "timeToLiveInSeconds": number,
          "turnsToLive": number
        }
      }
    ],
    "dialogAction": {
      "slotElicitationStyle": "string",
      "slotToElicit": "string",
      "type": "string"
    },
    "intent": {
      "confirmationState": "string",
      "name": "string",
      "slots": {
        "string" : {
          "shape": "string",
          "value": {
            "interpretedValue": "string",
            "originalValue": "string",
            "resolvedValues": [ "string" ]
          }
        }
      }
    }
  }
}
```
URI Request Parameters

The request uses the following URI parameters.

**botAliasId (p. 628)**

The alias identifier in use for the bot that processes the request.

Required: Yes

**botId (p. 628)**

The identifier of the bot that processes the request.

Length Constraints: Fixed length of 10.

Pattern: ^[0-9a-zA-Z]+$

Required: Yes

**localeId (p. 628)**

The locale where the session is in use.

Length Constraints: Minimum length of 1.

Required: Yes

**sessionId (p. 628)**

The identifier of the user session that is having the conversation.


Pattern: [0-9a-zA-Z_:.-]+
Required: Yes

Request Body

The request accepts the following data in JSON format.

requestAttributes (p. 628)

Request-specific information passed between the client application and Amazon Lex V2

The namespace x-amz-lex: is reserved for special attributes. Don't create any request attributes with the prefix x-amz-lex:

Type: String to string map

Key Length Constraints: Minimum length of 1.

Required: No

sessionState (p. 628)

The current state of the dialog between the user and the bot.

Type: SessionState (p. 810) object

Required: No

text (p. 628)

The text that the user entered. Amazon Lex V2 interprets this text.

Type: String


Required: Yes

Response Syntax

HTTP/1.1 200
Content-type: application/json

{
  "interpretations": [
    {
      "intent": {
        "confirmationState": "string",
        "name": "string",
        "slots": {
          "string": {
            "shape": "string",
            "value": {
              "interpretedValue": "string",
              "originalValue": "string",
              "resolvedValues": [ "string" ]
            }
          },
          "values": [
            "Slot"
          ]
        }
      },
      "state": "string"
    }
  ]
}
Amazon Lex V2 Developer Guide
Amazon Lex Runtime V2

},
  "nluConfidence": {
    "score": number
  },
  "sentimentResponse": {
    "sentiment": "string",
    "sentimentScore": {
      "mixed": number,
      "negative": number,
      "neutral": number,
      "positive": number
    }
  }
},
"messages": [
  {
    "content": "string",
    "contentType": "string",
    "imageResponseCard": {
      "buttons": [
        {
          "text": "string",
          "value": "string"
        }
      ],
      "imageUrl": "string",
      "subtitle": "string",
      "title": "string"
    }
  }
],
"requestAttributes": {
  "string": "string"
},
"sessionId": "string",
"sessionState": {
  "activeContexts": [
    {
      "contextAttributes": {
        "string": "string"
      },
      "name": "string",
      "timeToLive": {
        "timeToLiveInSeconds": number,
        "turnsToLive": number
      }
    }
  ],
  "dialogAction": {
    "slotElicitationStyle": "string",
    "slotToElicit": "string",
    "type": "string"
  },
  "intent": {
    "confirmationState": "string",
    "name": "string",
    "slots": {
      "string": {
        "shape": "string",
        "value": {
          "interpretedValue": "string",
          "originalValue": "string",
          "resolvedValues": [ "string" ]
        }
      },
      "values": [
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.

interceptions (p. 630)

A list of intents that Amazon Lex V2 determined might satisfy the user's utterance.

Each interpretation includes the intent, a score that indicates how confident Amazon Lex V2 is that the interpretation is the correct one, and an optional sentiment response that indicates the sentiment expressed in the utterance.

Type: Array of Interpretation (p. 801) objects

Array Members: Maximum number of 5 items.

messages (p. 630)

A list of messages last sent to the user. The messages are ordered based on the order that you returned the messages from your Lambda function or the order that the messages are defined in the bot.

Type: Array of Message (p. 802) objects

Array Members: Maximum number of 10 items.

requestAttributes (p. 630)

The attributes sent in the request.

Type: String to string map

Key Length Constraints: Minimum length of 1.

sessionId (p. 630)

The identifier of the session in use.
Type: String


Pattern: [0-9a-zA-Z._:-]+  

**sessionState (p. 630)**

Represents the current state of the dialog between the user and the bot.
Use this to determine the progress of the conversation and what the next action may be.
Type: SessionState (p. 810) object

**Errors**

For information about the errors that are common to all actions, see Common Errors (p. 821).

**AccessDeniedException**

HTTP Status Code: 403

**BadGatewayException**

HTTP Status Code: 502

**ConflictException**

HTTP Status Code: 409

**DependencyFailedException**

HTTP Status Code: 424

**InternalServerException**

HTTP Status Code: 500

**ResourceNotFoundException**

HTTP Status Code: 404

**ThrottlingException**

HTTP Status Code: 429

**ValidationException**

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 V2
• AWS SDK for JavaScript
• AWS SDK for PHP V3
• AWS SDK for Python
• AWS SDK for Ruby V3
RecognizeUtterance
Service: Amazon Lex Runtime V2

Sends user input to Amazon Lex V2. You can send text or speech. Clients use this API to send text and audio requests to Amazon Lex V2 at runtime. Amazon Lex V2 interprets the user input using the machine learning model built for the bot.

The following request fields must be compressed with gzip and then base64 encoded before you send them to Amazon Lex V2.

- requestAttributes
- sessionState

The following response fields are compressed using gzip and then base64 encoded by Amazon Lex V2. Before you can use these fields, you must decode and decompress them.

- inputTranscript
- interpretations
- messages
- requestAttributes
- sessionState

The example contains a Java application that compresses and encodes a Java object to send to Amazon Lex V2, and a second that decodes and decompresses a response from Amazon Lex V2.

If the optional post-fulfillment response is specified, the messages are returned as follows. For more information, see PostFulfillmentStatusSpecification.

- **Success message** - Returned if the Lambda function completes successfully and the intent state is fulfilled or ready fulfillment if the message is present.
- **Failed message** - The failed message is returned if the Lambda function throws an exception or if the Lambda function returns a failed intent state without a message.
- **Timeout message** - If you don't configure a timeout message and a timeout, and the Lambda function doesn't return within 30 seconds, the timeout message is returned. If you configure a timeout, the timeout message is returned when the period times out.

For more information, see Completion message.

Request Syntax

```plaintext
POST /bots/botId/botAliases/botAliasId/botLocales/localeId/sessions/sessionId/utterance HTTP/1.1
x-amz-lex-session-state: sessionState
x-amz-lex-request-attributes: requestAttributes
Content-Type: requestContentType
Response-Content-Type: responseContentType

inputStream
```

URI Request Parameters

The request uses the following URI parameters.
**botAliasId (p. 635)**
The alias identifier in use for the bot that should receive the request.

Required: Yes

**botId (p. 635)**
The identifier of the bot that should receive the request.

Length Constraints: Fixed length of 10.

Pattern: ^[0-9a-zA-Z]+$

Required: Yes

**localeId (p. 635)**
The locale where the session is in use.

Length Constraints: Minimum length of 1.

Required: Yes

**requestAttributes (p. 635)**
Request-specific information passed between the client application and Amazon Lex V2

The namespace x-amz-lex: is reserved for special attributes. Don't create any request attributes for prefix x-amz-lex:.

The requestAttributes field must be compressed using gzip and then base64 encoded before sending to Amazon Lex V2.

**requestContentType (p. 635)**
Indicates the format for audio input or that the content is text. The header 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

Length Constraints: Minimum length of 1.

Required: Yes

**responseContentType (p. 635)**
The message that Amazon Lex V2 returns in the response can be either text or speech based on the responseContentType value.
- If the value is text/plain;charset=utf-8, Amazon Lex V2 returns text in the response.
- If the value begins with audio/, Amazon Lex V2 returns speech in the response. Amazon Lex V2 uses Amazon Polly to generate the speech using the configuration that you specified in the requestContentType parameter. For example, if you specify audio/mpeg as the value, Amazon Lex V2 returns speech in the MPEG format.
• If the value is audio/pcm, the speech returned is audio/pcm at 16 KHz in 16-bit, little-endian format.

• The following are the accepted values:
  • audio/mpeg
  • audio/ogg
  • audio/pcm (16 KHz)
  • audio/* (defaults to mpeg)
  • text/plain; charset=utf-8

Length Constraints: Minimum length of 1.

```plaintext
sessionId (p. 635)
```

The identifier of the session in use.


```
Pattern: [0-9a-zA-Z._:-]+
```

Required: Yes

```plaintext
sessionState (p. 635)
```

Sets the state of the session with the user. You can use this to set the current intent, attributes, context, and dialog action. Use the dialog action to determine the next step that Amazon Lex V2 should use in the conversation with the user.

The `sessionState` field must be compressed using gzip and then base64 encoded before sending to Amazon Lex V2.

**Request Body**

The request accepts the following binary data.

```plaintext
inputStream (p. 635)
```

User input in PCM or Opus audio format or text format as described in the requestContentType parameter.

**Response Syntax**

```plaintext
HTTP/1.1 200
x-amz-lex-input-mode: inputMode
Content-Type: contentType
x-amz-lex-messages: messages
x-amz-lex-interpretations: interpretations
x-amz-lex-session-state: sessionState
x-amz-lex-request-attributes: requestAttributes
x-amz-lex-session-id: sessionId
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. 637)

Content type as specified in the responseContentType in the request.

Length Constraints: Minimum length of 1.

inputMode (p. 637)

Indicates whether the input mode to the operation was text or speech.

Length Constraints: Minimum length of 1.

inputTranscript (p. 637)

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 V2 is correctly processing the audio that you send.

The inputTranscript field is compressed with gzip and then base64 encoded. Before you can use the contents of the field, you must decode and decompress the contents. See the example for a simple function to decode and decompress the contents.

Length Constraints: Minimum length of 1.

interpretations (p. 637)

A list of intents that Amazon Lex V2 determined might satisfy the user's utterance.

Each interpretation includes the intent, a score that indicates how confident Amazon Lex V2 is that the interpretation is the correct one, and an optional sentiment response that indicates the sentiment expressed in the utterance.

The interpretations field is compressed with gzip and then base64 encoded. Before you can use the contents of the field, you must decode and decompress the contents. See the example for a simple function to decode and decompress the contents.

Length Constraints: Minimum length of 1.

messages (p. 637)

A list of messages that were last sent to the user. The messages are ordered based on the order that you returned the messages from your Lambda function or the order that the messages are defined in the bot.

The messages field is compressed with gzip and then base64 encoded. Before you can use the contents of the field, you must decode and decompress the contents. See the example for a simple function to decode and decompress the contents.

Length Constraints: Minimum length of 1.

requestAttributes (p. 637)

The attributes sent in the request.

The requestAttributes field is compressed with gzip and then base64 encoded. Before you can use the contents of the field, you must decode and decompress the contents.

Length Constraints: Minimum length of 1.

sessionId (p. 637)

The identifier of the session in use.
sessionState (p. 637)

Represents the current state of the dialog between the user and the bot.

Use this to determine the progress of the conversation and what the next action might be.

The sessionState field is compressed with gzip and then base64 encoded. Before you can use the contents of the field, you must decode and decompress the contents. See the example for a simple function to decode and decompress the contents.

Length Constraints: Minimum length of 1.

The response returns the following as the HTTP body.

audioStream (p. 637)

The prompt or statement to send to the user. This is based on the bot configuration and context. For example, if Amazon Lex V2 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 V2 sends that message in the response.

Errors

For information about the errors that are common to all actions, see Common Errors (p. 821).

AccessDeniedException

HTTP Status Code: 403

BadGatewayException

HTTP Status Code: 502

ConflictException

HTTP Status Code: 409

DependencyFailedException

HTTP Status Code: 424

InternalServerException

HTTP Status Code: 500

ResourceNotFoundException

HTTP Status Code: 404

ThrottlingException
Examples

Encode and decode a field

The following example provides two functions, one to compress a string with gzip and then base64 encode it and one to decode and decompress a string.

```java
package com.amazonaws.deepsense.util;

import com.amazonaws.deepsense.runtime.conversation.serialize.RuntimeSdkSerializer;
import com.fasterxml.jackson.annotation.JsonInclude;
import com.fasterxml.jackson.core.JsonProcessingException;
import com.fasterxml.jackson.core.type.TypeReference;
import com.fasterxml.jackson.databind.DeserializationFeature;
import com.fasterxml.jackson.databind.ObjectMapper;
import com.fasterxml.jackson.databind.SerializationFeature;
import com.google.common.base.Preconditions;
import java.io.IOException;
import java.util.Base64;

public class CompressionAndEncoding {
    private CompressionAndEncoding() {
    }

    /**
     * Given a generic object
     * 1. Serialize the object using jackson
     * 2. Compress the serialized object
     * 3. Base64 encode the compressed data
     */
    public static String compressAndEncodeBase64(Object object) {
        if (object == null) {
            return null;
        }
        String objectAsString = GsonSerializer.serialize(object);
        byte[] compressedString = Compressions.compress(objectAsString);
        return Base64.getEncoder().encodeToString(compressedString);
    }

    /**
     * Given a base64 encoded, compressed, serialized object
     * 1. Base64 decodes the data
     * 2. Decompresses the base64 decoded data
     * 3. Converts the serialized object into a proper POJO
     */
    public static <T> T decodeBase64AndDecompress(String objectAsString, TypeReference<T> typeRef) {
        if (objectAsString == null) {
            return null;
        }
        Preconditions.checkNotNull(typeRef, "Serialization class can't be null.");
        byte[] decodedBytes = Base64.getDecoder().decode(objectAsString);
        ...
String decompressedObjectAsString = Compressions.uncompress(decodedBytes);
try {
    return RuntimeSdkSerializer.instance().readValue(decompressedObjectAsString, typeRef);
} catch (IOException e) {
    throw new RuntimeException(String.format("Unable to deserialize string %s", decompressedObjectAsString), e);
}

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 V2
- AWS SDK for JavaScript
- AWS SDK for PHP V3
- AWS SDK for Python
- AWS SDK for Ruby V3
StartConversation
Service: Amazon Lex Runtime V2

Starts an HTTP/2 bidirectional event stream that enables you to send audio, text, or DTMF input in real time. After your application starts a conversation, users send input to Amazon Lex V2 as a stream of events. Amazon Lex V2 processes the incoming events and responds with streaming text or audio events.

Audio input must be in the following format: audio/lpcm sample-rate=8000 sample-size-bits=16 channel-count=1; is-big-endian=false.

If the optional post-fulfillment response is specified, the messages are returned as follows. For more information, see PostFulfillmentStatusSpecification.

- **Success message** - Returned if the Lambda function completes successfully and the intent state is fulfilled or ready fulfillment if the message is present.
- **Failed message** - The failed message is returned if the Lambda function throws an exception or if the Lambda function returns a failed intent state without a message.
- **Timeout message** - If you don't configure a timeout message and a timeout, and the Lambda function doesn't return within 30 seconds, the timeout message is returned. If you configure a timeout, the timeout message is returned when the period times out.

For more information, see Completion message.

If the optional update message is configured, it is played at the specified frequency while the Lambda function is running and the update message state is active. If the fulfillment update message is not active, the Lambda function runs with a 30 second timeout.

For more information, see Update message

The StartConversation operation is supported only in the following SDKs:

- AWS SDK for C++
- AWS SDK for Java V2
- AWS SDK for Ruby V3

**Request Syntax**

```
POST /bots/botId/botAliases/botAliasId/botLocales/localeId/sessions/sessionId/conversation
HTTP/2
x-amz-lex-conversation-mode: conversationMode
Content-type: application/json

{
  "requestEventStream": {
    "AudioInputEvent": {
      "audioChunk": blob,
      "clientTimestampMillis": number,
      "contentType": "string",
      "eventId": "string"
    },
    "ConfigurationEvent": {
      "clientTimestampMillis": number,
      "disablePlayback": boolean,
      "eventId": "string",
      "requestAttributes": {
        "string" : "String"
      }
    }
  }
}
```
"responseContentType": "string",
"sessionState": {
  "activeContexts": [
    {
      "contextAttributes": {
        "string": "string"
      },
      "name": "string",
      "timeToLive": {
        "timeToLiveInSeconds": number,
        "turnsToLive": number
      }
    }
  ],
  "dialogAction": {
    "slotElicitationStyle": "string",
    "slotToElicit": "string",
    "type": "string"
  },
  "intent": {
    "confirmationState": "string",
    "name": "string",
    "slots": {
      "string": {
        "shape": "string",
        "value": {
          "interpretedValue": "string",
          "originalValue": "string",
          "resolvedValues": [ "string" ]
        },
        "values": [ "Slot" ]
      }
    }},
  "state": "string"
},
"originatingRequestId": "string",
"runtimeHints": {
  "slotHints": {
    "string": {
      "string": {
        "runtimeHintValues": [ {
          "phrase": "string"
        } ]
      }
    }
  }
},
"sessionAttributes": {
  "string": "string"
},
"welcomeMessages": [
  {
    "content": "string",
    "contentType": "string",
    "imageResponseCard": {
      "buttons": [
        {
          "text": "string",
          "value": "string"
        }
      ]
    }
  }
]
URI Request Parameters

The request uses the following URI parameters.

**botAliasId** *(p. 642)*

The alias identifier in use for the bot that processes the request.

Required: Yes

**botId** *(p. 642)*

The identifier of the bot to process the request.

Length Constraints: Fixed length of 10.

Pattern: `^[0-9a-zA-Z]+$`

Required: Yes

**conversationMode** *(p. 642)*

The conversation type that you are using the Amazon Lex V2. If the conversation mode is AUDIO you can send both audio and DTMF information. If the mode is TEXT you can only send text.

Valid Values: AUDIO | TEXT

**localeId** *(p. 642)*

The locale where the session is in use.

Length Constraints: Minimum length of 1.

Required: Yes
sessionId (p. 642)

The identifier of the user session that is having the conversation.


Pattern: [0-9a-zA-Z._:-]+

Required: Yes

Request Body

The request accepts the following data in JSON format.

requestEventStream (p. 642)

Represents the stream of events to Amazon Lex V2 from your application. The events are encoded as HTTP/2 data frames.

Type: StartConversationRequestEventStream (p. 813) object

Required: Yes

Response Syntax

HTTP/2 200
Content-type: application/json

{
  "responseEventStream": {
    "AccessDeniedException": {
    },
    "AudioResponseEvent": {
      "audioChunk": blob,
      "contentType": "string",
      "eventId": "string"
    },
    "BadGatewayException": {
    },
    "ConflictException": {
    },
    "DependencyFailedException": {
    },
    "HeartbeatEvent": {
      "eventId": "string"
    },
    "IntentResultEvent": {
      "eventId": "string",
      "inputMode": "string",
      "interpretations": [
      {
        "intent": {
          "confirmationState": "string",
          "name": "string",
          "slots": {
            "string": {
              "shape": "string",
              "value": {
                "interpretedValue": "string",
                "originalValue": "string",
                "resolvedValues": [ "string" ]
              }
            }
          }
        }
      }
    }
  }
}
"values": [
  "Slot"
],

"state": "string",

"nluConfidence": {
  "score": number
},

"sentimentResponse": {
  "sentiment": "string",
  "sentimentScore": {
    "mixed": number,
    "negative": number,
    "neutral": number,
    "positive": number
  }
}
]

"requestAttributes": {
  "string": "string"
}

"sessionId": "string",

"sessionState": {
  "activeContexts": [ {
    "contextAttributes": {
      "string": "string"
    },
    "name": "string",
    "timeToLive": {
      "timeToLiveInSeconds": number,
      "turnsToLive": number
    }
  }
]

"dialogAction": {
  "slotElicitationStyle": "string",
  "slotToElicit": "string",
  "type": "string"
}

"intent": {
  "confirmationState": "string",
  "name": "string",
  "slots": { 
    "string": {
      "shape": "string",
      "value": {
        "interpretedValue": "string",
        "originalValue": "string",
        "resolvedValues": [ "string" ]
      },
      "values": [
        "Slot"
      ]
    }
  }
}

"state": "string"

"originatingRequestId": "string",

"runtimeHints": {
  "slotHints": {
    "string": {
      "string": {

646
"runtimeHintValues": [  
  {  
    "phrase": "string"  
  }  
],  
"sessionAttributes": {  
  "string": "string"  
},  
"InternalServerException": {  
},  
"PlaybackInterruptionEvent": {  
  "causedByEventId": "string",  
  "eventId": "string",  
  "eventReason": "string"  
},  
"ResourceNotFoundException": {  
},  
"TextResponseEvent": {  
  "eventId": "string",  
  "messages": [  
    {  
      "content": "string",  
      "contentType": "string",  
      "imageResponseCard": {  
        "buttons": [  
          {  
            "text": "string",  
            "value": "string"  
          }  
        ],  
        "imageUrl": "string",  
        "subtitle": "string",  
        "title": "string"  
      }  
    }  
  ]  
},  
"ThrottlingException": {  
},  
"TranscriptEvent": {  
  "eventId": "string",  
  "transcript": "string"  
},  
"ValidationException": {  
}  
}

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.

responseEventStream (p. 645)

Represents the stream of events from Amazon Lex V2 to your application. The events are encoded as HTTP/2 data frames.
Type: StartConversationResponseEventStream (p. 815) object

Errors
For information about the errors that are common to all actions, see Common Errors (p. 821).

AccessDeniedException

HTTP Status Code: 403

InternalServerError

HTTP Status Code: 500

ThrottlingException

HTTP Status Code: 429

ValidationException

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 V2
- AWS SDK for JavaScript
- AWS SDK for PHP V3
- AWS SDK for Python
- AWS SDK for Ruby V3

Data Types
The following data types are supported by Amazon Lex Model Building V2:

- AdvancedRecognitionSetting (p. 655)
- AggregatedUtterancesFilter (p. 656)
- AggregatedUtterancesSortBy (p. 657)
- AggregatedUtterancesSummary (p. 658)
- AssociatedTranscript (p. 660)
- AssociatedTranscriptFilter (p. 661)
- AudioLogDestination (p. 662)
- AudioLogSetting (p. 663)
- BotAliasHistoryEvent (p. 664)
- BotAliasLocaleSettings (p. 665)
- BotAliasSummary (p. 666)
- BotExportSpecification (p. 668)
- BotFilter (p. 669)
- BotImportSpecification (p. 670)
- BotLocaleExportSpecification (p. 672)
- BotLocaleFilter (p. 673)
- BotLocaleHistoryEvent (p. 674)
- BotLocaleImportSpecification (p. 675)
- BotLocaleSortBy (p. 677)
- BotLocaleSummary (p. 678)
- BotRecommendationResults (p. 680)
- BotRecommendationResultStatistics (p. 681)
- BotRecommendationSummary (p. 682)
- BotSortBy (p. 683)
- BotSummary (p. 684)
- BotVersionLocaleDetails (p. 686)
- BotVersionSortBy (p. 687)
- BotVersionSummary (p. 688)
- BuiltInIntentSortBy (p. 690)
- BuiltInIntentSummary (p. 691)
- BuiltInSlotTypeSortBy (p. 692)
- BuiltInSlotTypeSummary (p. 693)
- Button (p. 694)
- CloudWatchLogGroupLogDestination (p. 695)
- CodeHookSpecification (p. 696)
- ConversationLogSettings (p. 697)
- CustomPayload (p. 698)
- CustomVocabularyExportSpecification (p. 699)
- CustomVocabularyImportSpecification (p. 700)
- DataPrivacy (p. 701)
- DateRangeFilter (p. 702)
- DialogCodeHookSettings (p. 703)
- EncryptionSetting (p. 704)
- ExportFilter (p. 705)
- ExportResourceSpecification (p. 706)
- ExportSortBy (p. 707)
- ExportSummary (p. 708)
- ExternalSourceSetting (p. 710)
- FulfillmentCodeHookSettings (p. 711)
- FulfillmentStartResponseSpecification (p. 712)
- FulfillmentUpdateResponseSpecification (p. 713)
- FulfillmentUpdatesSpecification (p. 714)
- GrammarSlotTypeSetting (p. 715)
- GrammarSlotTypeSource (p. 716)
- ImageResponseCard (p. 717)
- ImportFilter (p. 719)
- `ImportResourceSpecification` (p. 720)
- `ImportSortBy` (p. 721)
- `ImportSummary` (p. 722)
- `InputContext` (p. 724)
- `IntentClosingSetting` (p. 725)
- `IntentConfirmationSetting` (p. 726)
- `IntentFilter` (p. 727)
- `IntentSortBy` (p. 728)
- `IntentStatistics` (p. 729)
- `IntentSummary` (p. 730)
- `KendraConfiguration` (p. 732)
- `LambdaCodeHook` (p. 733)
- `LexTranscriptFilter` (p. 734)
- `Message` (p. 735)
- `MessageGroup` (p. 736)
- `MultipleValuesSetting` (p. 737)
- `ObfuscationSetting` (p. 738)
- `OutputContext` (p. 739)
- `PathFormat` (p. 740)
- `PlainTextMessage` (p. 741)
- `PostFulfillmentStatusSpecification` (p. 742)
- `Principal` (p. 743)
- `PromptSpecification` (p. 744)
- `RecommendedIntentSummary` (p. 745)
- `RelativeAggregationDuration` (p. 746)
- `ResponseSpecification` (p. 747)
- `S3BucketLogDestination` (p. 748)
- `S3BucketTranscriptSource` (p. 749)
- `SampleUtterance` (p. 751)
- `SampleValue` (p. 752)
- `SentimentAnalysisSettings` (p. 753)
- `SlotDefaultValue` (p. 754)
- `SlotDefaultValueSpecification` (p. 755)
- `SlotFilter` (p. 756)
- `SlotPriority` (p. 757)
- `SlotSortBy` (p. 758)
- `SlotSummary` (p. 759)
- `SlotTypeFilter` (p. 761)
- `SlotTypeSortBy` (p. 762)
- `SlotTypeStatistics` (p. 763)
- `SlotTypeSummary` (p. 764)
- `SlotTypeValue` (p. 766)
- `SlotValueElicitationSetting` (p. 767)
- `SlotValueRegexFilter` (p. 769)
- `SlotValueSelectionSetting` (p. 770)
- `SSMLMessage` (p. 771)
• StillWaitingResponseSpecification (p. 772)
• TextLogDestination (p. 773)
• TextLogSetting (p. 774)
• TranscriptFilter (p. 775)
• TranscriptSourceSetting (p. 776)
• UtteranceAggregationDuration (p. 777)
• VoiceSettings (p. 778)
• WaitAndContinueSpecification (p. 779)

The following data types are supported by Amazon Lex Runtime V2:

• ActiveContext (p. 781)
• ActiveContextTimeToLive (p. 783)
• AudioInputEvent (p. 784)
• AudioResponseEvent (p. 786)
• Button (p. 787)
• ConfidenceScore (p. 788)
• ConfigurationEvent (p. 789)
• DialogAction (p. 791)
• DisconnectionEvent (p. 793)
• DTMFInputEvent (p. 794)
• HeartbeatEvent (p. 795)
• ImageResponseCard (p. 796)
• Intent (p. 798)
• IntentResultEvent (p. 799)
• Interpretation (p. 801)
• Message (p. 802)
• PlaybackCompletionEvent (p. 803)
• Playback InterruptionEvent (p. 804)
• RuntimeHintDetails (p. 805)
• RuntimeHints (p. 806)
• RuntimeHintValue (p. 807)
• SentimentResponse (p. 808)
• SentimentScore (p. 809)
• SessionState (p. 810)
• Slot (p. 812)
• StartConversationRequestEventStream (p. 813)
• StartConversationResponseEventStream (p. 815)
• TextInputEvent (p. 818)
• TextResponseEvent (p. 819)
• TranscriptEvent (p. 820)
• Value (p. 821)

Amazon Lex Model Building V2

The following data types are supported by Amazon Lex Model Building V2:
• AdvancedRecognitionSetting (p. 655)
• AggregatedUtterancesFilter (p. 656)
• AggregatedUtterancesSortBy (p. 657)
• AggregatedUtterancesSummary (p. 658)
• AssociatedTranscript (p. 660)
• AssociatedTranscriptFilter (p. 661)
• AudioLogDestination (p. 662)
• AudioLogSetting (p. 663)
• BotAliasHistoryEvent (p. 664)
• BotAliasLocaleSettings (p. 665)
• BotAliasSummary (p. 666)
• BotExportSpecification (p. 668)
• BotFilter (p. 669)
• BotImportSpecification (p. 670)
• BotLocaleExportSpecification (p. 672)
• BotLocaleFilter (p. 673)
• BotLocaleHistoryEvent (p. 674)
• BotLocaleImportSpecification (p. 675)
• BotLocaleSortBy (p. 677)
• BotLocaleSummary (p. 678)
• BotRecommendationResults (p. 680)
• BotRecommendationResultStatistics (p. 681)
• BotRecommendationSummary (p. 682)
• BotSortBy (p. 683)
• BotSummary (p. 684)
• BotVersionLocaleDetails (p. 686)
• BotVersionSortBy (p. 687)
• BotVersionSummary (p. 688)
• BuiltInIntentSortBy (p. 690)
• BuiltInIntentSummary (p. 691)
• BuiltInSlotTypeSortBy (p. 692)
• BuiltInSlotTypeSummary (p. 693)
• Button (p. 694)
• CloudWatchLogGroupLogDestination (p. 695)
• CodeHookSpecification (p. 696)
• ConversationLogSettings (p. 697)
• CustomPayload (p. 698)
• CustomVocabularyExportSpecification (p. 699)
• CustomVocabularyImportSpecification (p. 700)
• DataPrivacy (p. 701)
• DateRangeFilter (p. 702)
• DialogCodeHookSettings (p. 703)
• EncryptionSetting (p. 704)
• ExportFilter (p. 705)
• ExportResourceSpecification (p. 706)
• ExportSortBy (p. 707)
• ExportSummary (p. 708)
• ExternalSourceSetting (p. 710)
• FulfillmentCodeHookSettings (p. 711)
• FulfillmentStartResponseSpecification (p. 712)
• FulfillmentUpdateResponseSpecification (p. 713)
• FulfillmentUpdatesSpecification (p. 714)
• GrammarSlotTypeSetting (p. 715)
• GrammarSlotTypeSource (p. 716)
• ImageResponseCard (p. 717)
• ImportFilter (p. 719)
• ImportResourceSpecification (p. 720)
• ImportSortBy (p. 721)
• ImportSummary (p. 722)
• InputContext (p. 724)
• IntentClosingSetting (p. 725)
• IntentConfirmationSetting (p. 726)
• IntentFilter (p. 727)
• IntentSortBy (p. 728)
• IntentStatistics (p. 729)
• IntentSummary (p. 730)
• KendraConfiguration (p. 732)
• LambdaCodeHook (p. 733)
• LexTranscriptFilter (p. 734)
• Message (p. 735)
• MessageGroup (p. 736)
• MultipleValuesSetting (p. 737)
• ObfuscationSetting (p. 738)
• OutputContext (p. 739)
• PathFormat (p. 740)
• PlainTextMessage (p. 741)
• PostFulfillmentStatusSpecification (p. 742)
• Principal (p. 743)
• PromptSpecification (p. 744)
• RecommendedIntentSummary (p. 745)
• RelativeAggregationDuration (p. 746)
• ResponseSpecification (p. 747)
• S3BucketLogDestination (p. 748)
• S3BucketTranscriptSource (p. 749)
• SampleUtterance (p. 751)
• SampleValue (p. 752)
• SentimentAnalysisSettings (p. 753)
• SlotDefaultValue (p. 754)
• SlotDefaultValueSpecification (p. 755)
• SlotFilter (p. 756)
• SlotPriority (p. 757)
• SlotSortBy (p. 758)
• SlotSummary (p. 759)
• SlotTypeFilter (p. 761)
• SlotTypeSortBy (p. 762)
• SlotTypeStatistics (p. 763)
• SlotTypeSummary (p. 764)
• SlotTypeValue (p. 766)
• SlotValueElicitationSetting (p. 767)
• SlotValueRegexFilter (p. 769)
• SlotValueSelectionSetting (p. 770)
• SSMLMessage (p. 771)
• StillWaitingResponseSpecification (p. 772)
• TextLogDestination (p. 773)
• TextLogSetting (p. 774)
• TranscriptFilter (p. 775)
• TranscriptSourceSetting (p. 776)
• UtteranceAggregationDuration (p. 777)
• VoiceSettings (p. 778)
• WaitAndContinueSpecification (p. 779)
AdvancedRecognitionSetting
Service: Amazon Lex Model Building V2

Provides settings that enable advanced recognition settings for slot values.

Contents

audioRecognitionStrategy

   Enables using the slot values as a custom vocabulary for recognizing user utterances.

       Type: String
       Valid Values: UseSlotValuesAsCustomVocabulary
       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 V2
- AWS SDK for Ruby V3
AggregatedUtterancesFilter
Service: Amazon Lex Model Building V2

Filters responses returned by the ListAggregatedUtterances operation.

Contents

name

The name of the field to filter the utterance list.

Type: String

Valid Values: Utterance

Required: Yes

operator

The operator to use for the filter. Specify EQ when the ListAggregatedUtterances operation should return only utterances that equal the specified value. Specify CO when the ListAggregatedUtterances operation should return utterances that contain the specified value.

Type: String

Valid Values: CO | EQ

Required: Yes

values

The value to use for filtering the list of bots.

Type: Array of strings

Array Members: Fixed number of 1 item.

Length Constraints: Minimum length of 1. Maximum length of 100.

Pattern: ^[0-9a-zA-Z_()-\s-]+$

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 V2
- AWS SDK for Ruby V3
AggregatedUtterancesSortBy
Service: Amazon Lex Model Building V2

Specifies attributes for sorting a list of utterances.

Contents

attribute
The utterance attribute to sort by.
Type: String
Valid Values: HitCount | MissedCount
Required: Yes

order
Specifies whether to sort the aggregated utterances in ascending or descending order.
Type: String
Valid Values: Ascending | Descending
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 V2
- AWS SDK for Ruby V3
AggregatedUtterancesSummary
Service: Amazon Lex Model Building V2

Provides summary information for aggregated utterances. The ListAggregatedUtterances operations combines all instances of the same utterance into a single aggregated summary.

Contents

containsDataFromDeletedResources
Aggregated utterance data may contain utterances from versions of your bot that have since been deleted. When the aggregated contains this kind of data, this field is set to true.

Type: Boolean
Required: No

hitCount
The number of times that the utterance was detected by Amazon Lex during the time period. When an utterance is detected, it activates an intent or a slot.

Type: Integer
Required: No

missedCount
The number of times that the utterance was missed by Amazon Lex. An utterance is missed when it doesn’t activate an intent or slot.

Type: Integer
Required: No

utterance
The text of the utterance. If the utterance was used with the RecognizeUtterance operation, the text is the transcription of the audio utterance.

Type: String
Required: No

utteranceFirstRecordedInAggregationDuration
The date and time that the utterance was first recorded in the time window for aggregation. An utterance may have been sent to Amazon Lex before that time, but only utterances within the time window are counted.

Type: Timestamp
Required: No

utteranceLastRecordedInAggregationDuration
The last date and time that an utterance was recorded in the time window for aggregation. An utterance may be sent to Amazon Lex after that time, but only utterances within the time window are counted.

Type: Timestamp
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 V2
- AWS SDK for Ruby V3
**AssociatedTranscript**  
Service: Amazon Lex Model Building V2

The object containing information that associates the recommended intent/slot type with a conversation.

**Contents**

**transcript**

The content of the transcript that meets the search filter criteria. For the JSON format of the transcript, see Output transcript format.

Type: String


Pattern: . *

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 V2
- AWS SDK for Ruby V3
**AssociatedTranscriptFilter**

Service: Amazon Lex Model Building V2

Filters to search for the associated transcript.

**Contents**

**name**

The name of the field to use for filtering. The allowed names are IntentId and SlotTypeId.

Type: String

Valid Values: IntentId | SlotTypeId

Required: Yes

**values**

The values to use to filter the transcript.

Type: Array of strings

Array Members: Fixed number of 1 item.

Length Constraints: Minimum length of 1. Maximum length of 100.

Pattern: ^[0-9a-zA-Z_\(\)\(\)\s-]+$

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 V2
- AWS SDK for Ruby V3
**AudioLogDestination**  
Service: Amazon Lex Model Building V2

The location of audio log files collected when conversation logging is enabled for a bot.

**Contents**

**s3Bucket**

The Amazon S3 bucket where the audio log files are stored. The IAM role specified in the `roleArn` parameter of the `CreateBot` operation must have permission to write to this bucket.

Type: `S3BucketLogDestination (p. 748)` 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 V2
- AWS SDK for Ruby V3
AudioLogSetting
Service: Amazon Lex Model Building V2

Settings for logging audio of conversations between Amazon Lex and a user. You specify whether to log audio and the Amazon S3 bucket where the audio file is stored.

Contents

destination

The location of audio log files collected when conversation logging is enabled for a bot.

Type: AudioLogDestination (p. 662) object

Required: Yes

destination

enabled

Determines whether audio logging in enabled for the bot.

Type: Boolean

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 V2
- AWS SDK for Ruby V3
BotAliasHistoryEvent
Service: Amazon Lex Model Building V2

Provides a record of an event that affects a bot alias. For example, when the version of a bot that the alias points to changes.

Contents

botVersion

The version of the bot that was used in the event.
Type: String
Pattern: ^\(DRAFT|\[0-9\]+\)$
Required: No

endDate

The date and time that the event ended.
Type: Timestamp
Required: No

startDate

The date and time that the event started.
Type: Timestamp
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 V2
- AWS SDK for Ruby V3
BotAliasLocaleSettings
Service: Amazon Lex Model Building V2

Specifies settings that are unique to a locale. For example, you can use different Lambda function depending on the bot's locale.

Contents

codeHookSpecification

  Specifies the Lambda function that should be used in the locale.

  Type: CodeHookSpecification (p. 696) object

  Required: No

enabled

  Determines whether the locale is enabled for the bot. If the value is false, the locale isn't available for use.

  Type: Boolean

  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 V2
- AWS SDK for Ruby V3
BotAliasSummary

Service: Amazon Lex Model Building V2

Summary information about bot aliases returned from the ListBotAliases operation.

Contents

botAliasId

The unique identifier assigned to the bot alias. You can use this ID to get detailed information about
the alias using the DescribeBotAlias operation.

Type: String

Length Constraints: Fixed length of 10.

Pattern: ^\bTSTALIASID\b\[0-9a-zA-Z]+$  

Required: No

botAliasName

The name of the bot alias.

Type: String

Length Constraints: Minimum length of 1. Maximum length of 100.

Pattern: ^\[0-9a-zA-Z]([-_]?)++$

Required: No

botAliasStatus

The current state of the bot alias. If the status is Available, the alias is ready for use.

Type: String

Valid Values: Creating | Available | Deleting | Failed

Required: No

botVersion

The version of the bot that the bot alias references.

Type: String


Pattern: ^(DRAFT|[0-9]+)$

Required: No

creationDateTime

A timestamp of the date and time that the bot alias was created.

Type: Timestamp

Required: No

description

The description of the bot alias.
Type: String
Length Constraints: Minimum length of 0. Maximum length of 200.
Required: No

**lastUpdatedDate Time**

A timestamp of the date and time that the bot alias was last updated.
Type: Timestamp
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 V2
- AWS SDK for Ruby V3
BotExportSpecification
Service: Amazon Lex Model Building V2

Provides the identity of a the bot that was exported.

Contents

botId

The identifier of the bot assigned by Amazon Lex.

Type: String

Length Constraints: Fixed length of 10.

Pattern: ^[0-9a-zA-Z]+$

Required: Yes

botVersion

The version of the bot that was exported. This will be either DRAFT or the version number.

Type: String


Pattern: ^(DRAFT|\[0-9\]+$

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 V2
- AWS SDK for Ruby V3
BotFilter
Service: Amazon Lex Model Building V2

Filters the responses returned by the ListBots operation.

Contents

name

The name of the field to filter the list of bots.

Type: String

Valid Values: BotName

Required: Yes

operator

The operator to use for the filter. Specify EQ when the ListBots operation should return only aliases that equal the specified value. Specify CO when the ListBots operation should return aliases that contain the specified value.

Type: String

Valid Values: CO | EQ

Required: Yes

values

The value to use for filtering the list of bots.

Type: Array of strings

Array Members: Fixed number of 1 item.
Length Constraints: Minimum length of 1. Maximum length of 100.

Pattern: ^[0-9a-zA-Z_\(\)\s-]+$

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 V2
- AWS SDK for Ruby V3
BotImportSpecification
Service: Amazon Lex Model Building V2

Provides the bot parameters required for importing a bot.

Contents

botName

The name that Amazon Lex should use for the bot.

Type: String

Length Constraints: Minimum length of 1. Maximum length of 100.

Pattern: ^([0-9a-zA-Z][-_]?)+$

Required: Yes

botTags

A list of tags to add to the bot. You can only add tags when you import a bot. You can’t use the UpdateBot operation to update tags. To update tags, use the TagResource operation.

Type: String to string map

Map Entries: Minimum number of 0 items. Maximum number of 200 items.

Key Length Constraints: Minimum length of 1. Maximum length of 128.

Value Length Constraints: Minimum length of 0. Maximum length of 256.

Required: No

dataPrivacy

By default, data stored by Amazon Lex is encrypted. The DataPrivacy structure provides settings that determine how Amazon Lex handles special cases of securing the data for your bot.

Type: DataPrivacy (p. 701) object

Required: Yes

idleSessionTTLInSeconds

The time, in seconds, that Amazon Lex should keep information about a user’s conversation with the bot.

A user interaction 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.

You can specify between 60 (1 minute) and 86,400 (24 hours) seconds.

Type: Integer

Valid Range: Minimum value of 60. Maximum value of 86400.

Required: No

roleArn

The Amazon Resource Name (ARN) of the IAM role used to build and run the bot.

Type: String
Length Constraints: Minimum length of 32. Maximum length of 2048.

Pattern: ^arn:aws:iam::[0-9]{12}:role/.*$

Required: Yes

**testBotAliasTags**

A list of tags to add to the test alias for a bot. You can only add tags when you import a bot. You can't use the UpdateAlias operation to update tags. To update tags on the test alias, use the TagResource operation.

Type: String to string map

Map Entries: Minimum number of 0 items. Maximum number of 200 items.

Key Length Constraints: Minimum length of 1. Maximum length of 128.

Value Length Constraints: Minimum length of 0. Maximum length of 256.

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 V2
- AWS SDK for Ruby V3
BotLocaleExportSpecification
Service: Amazon Lex Model Building V2

Provides the bot locale parameters required for exporting a bot locale.

Contents

botId

The identifier of the bot to create the locale for.

Type: String
Length Constraints: Fixed length of 10.
Pattern: ^[0-9a-zA-Z]+$
Required: Yes

botVersion

The version of the bot to export.

Type: String
Pattern: ^(DRAFT|[0-9]+)$
Required: Yes

localeId

The identifier of the language and locale to export. The string must match one of the locales in the bot.

Type: String
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 V2
- AWS SDK for Ruby V3
BotLocaleFilter
Service: Amazon Lex Model Building V2

Filters responses returned by the ListBotLocales operation.

Contents

name
The name of the field to filter the list of bots.
Type: String
Valid Values: BotLocaleName
Required: Yes

operator
The operator to use for the filter. Specify EQ when the ListBotLocales operation should return only aliases that equal the specified value. Specify CO when the ListBotLocales operation should return aliases that contain the specified value.
Type: String
Valid Values: CO | EQ
Required: Yes

values
The value to use for filtering the list of bots.
Type: Array of strings
Array Members: Fixed number of 1 item.
Length Constraints: Minimum length of 1. Maximum length of 100.
Pattern: ^[0-9a-zA-Z_\(-()\s-]+$ 
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 V2
- AWS SDK for Ruby V3
BotLocaleHistoryEvent
Service: Amazon Lex Model Building V2

Provides information about an event that occurred affecting the bot locale.

Contents

event

A description of the event that occurred.

Type: String

Required: Yes

eventDate

A timestamp of the date and time that the event occurred.

Type: Timestamp

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 V2
- AWS SDK for Ruby V3
BotLocaleImportSpecification
Service: Amazon Lex Model Building V2

Provides the bot locale parameters required for importing a bot locale.

Contents

botId

The identifier of the bot to import the locale to.

Type: String

Length Constraints: Fixed length of 10.

Pattern: ^[0-9a-zA-Z]+$

Required: Yes

botVersion

The version of the bot to import the locale to. This can only be the DRAFT version of the bot.

Type: String

Length Constraints: Fixed length of 5.

Pattern: ^DRAFT$

Required: Yes

localeId

The identifier of the language and locale that the bot will be used in. The string must match one of the supported locales. All of the intents, slot types, and slots used in the bot must have the same locale. For more information, see Supported languages.

Type: String

Required: Yes

nluIntentConfidenceThreshold

Determines the threshold where Amazon Lex will insert the AMAZON.FallbackIntent, AMAZON.KendraSearchIntent, or both when returning alternative intents. AMAZON.FallbackIntent and AMAZON.KendraSearchIntent are only inserted if they are configured for the bot.

For example, suppose a bot is configured with the confidence threshold of 0.80 and the AMAZON.FallbackIntent. Amazon Lex returns three alternative intents with the following confidence scores: IntentA (0.70), IntentB (0.60), IntentC (0.50). The response from the PostText operation would be:

• AMAZON.FallbackIntent
• IntentA
• IntentB
• IntentC

Type: Double

Valid Range: Minimum value of 0. Maximum value of 1.

Required: No
voiceSettings

Defines settings for using an Amazon Polly voice to communicate with a user.

Type: VoiceSettings (p. 778) 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 V2
- AWS SDK for Ruby V3
BotLocaleSortBy
Service: Amazon Lex Model Building V2

Specifies attributes for sorting a list of bot locales.

Contents

attribute

The bot locale attribute to sort by.

Type: String

Valid Values: BotLocaleName

Required: Yes

order

Specifies whether to sort the bot locales in ascending or descending order.

Type: String

Valid Values: Ascending | Descending

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 V2
- AWS SDK for Ruby V3
BotLocaleSummary
Service: Amazon Lex Model Building V2

Summary information about bot locales returned by the ListBotLocales operation.

Contents

botLocaleStatus

The current status of the bot locale. When the status is Built the locale is ready for use.

Type: String

Valid Values: Creating | Building | Built | ReadyExpressTesting | Failed | Deleting | NotBuilt | Importing | Processing

Required: No

description

The description of the bot locale.

Type: String

Length Constraints: Minimum length of 0. Maximum length of 200.

Required: No

lastBuildSubmittedDateTime

A timestamp of the date and time that the bot locale was last built.

Type: Timestamp

Required: No

lastUpdatedDateTime

A timestamp of the date and time that the bot locale was last updated.

Type: Timestamp

Required: No

localeId

The language and locale of the bot locale.

Type: String

Required: No

localeName

The name of the bot locale.

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 V2
- AWS SDK for Ruby V3
BotRecommendationResults
Service: Amazon Lex Model Building V2

The object representing the URL of the bot definition, the URL of the associated transcript and a statistical summary of the bot recommendation results.

Contents

associatedTranscriptsUrl
The presigned url link of the associated transcript.
Type: String
Required: No

botLocaleExportUrl
The presigned URL link of the recommended bot definition.
Type: String
Required: No

statistics
The statistical summary of the bot recommendation results.
Type: BotRecommendationResultStatistics (p. 681) 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 V2
- AWS SDK for Ruby V3
BotRecommendationResultStatistics
Service: Amazon Lex Model Building V2

A statistical summary of the bot recommendation results.

Contents

intents

Statistical information about the intents associated with the bot recommendation results.

Type: IntentStatistics (p. 729) object

Required: No

slotTypes

Statistical information about the slot types associated with the bot recommendation results.

Type: SlotTypeStatistics (p. 763) 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 V2
- AWS SDK for Ruby V3
BotRecommendationSummary
Service: Amazon Lex Model Building V2
A summary of the bot recommendation.

Contents

botRecommendationId
The unique identifier of the bot recommendation to be updated.
Type: String
Length Constraints: Fixed length of 10.
Pattern: ^[0-9a-zA-Z]+$
Required: Yes

botRecommendationStatus
The status of the bot recommendation.
If the status is Failed, then the reasons for the failure are listed in the failureReasons field.
Type: String
Valid Values: Processing | Deleting | Deleted | Downloading | Updating | Available | Failed
Required: Yes

creationDateTime
A timestamp of the date and time that the bot recommendation was created.
Type: Timestamp
Required: No

lastUpdatedDateTime
A timestamp of the date and time that the bot recommendation was last updated.
Type: Timestamp
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 V2
- AWS SDK for Ruby V3
BotSortBy
Service: Amazon Lex Model Building V2

Specifies attributes for sorting a list of bots.

Contents

attribute

The attribute to use to sort the list of bots.

Type: String

Valid Values: BotName

Required: Yes

order

The order to sort the list. You can choose ascending or descending.

Type: String

Valid Values: Ascending | Descending

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 V2
- AWS SDK for Ruby V3
**BotSummary**  
Service: Amazon Lex Model Building V2

Summary information about a bot returned by the ListBots operation.

**Contents**

**botId**  
The unique identifier assigned to the bot. Use this ID to get detailed information about the bot with the DescribeBot operation.

Type: String  
Length Constraints: Fixed length of 10.  
Pattern: ^[0-9a-zA-Z]+$  
Required: No

**botName**  
The name of the bot.

Type: String  
Length Constraints: Minimum length of 1. Maximum length of 100.  
Pattern: ^([0-9a-zA-Z][_-])?$  
Required: No

**botStatus**  
The current status of the bot. When the status is Available the bot is ready for use.

Type: String  
Valid Values: Creating | Available | Inactive | Deleting | Failed | Versioning | Importing  
Required: No

**description**  
The description of the bot.

Type: String  
Length Constraints: Minimum length of 0. Maximum length of 200.  
Required: No

**lastUpdatedDateTime**  
The date and time that the bot was last updated.

Type: Timestamp  
Required: No

**latestBotVersion**  
The latest numerical version in use for the bot.
Type: String
Pattern: ^[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 V2
- AWS SDK for Ruby V3
BotVersionLocaleDetails
Service: Amazon Lex Model Building V2

The version of a bot used for a bot locale.

Contents

sourceBotVersion

The version of a bot used for a bot locale.

Type: String


Pattern: ^\(DRAFT|\[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 V2
- AWS SDK for Ruby V3
BotVersionSortBy
Service: Amazon Lex Model Building V2

Specifies attributes for sorting a list of bot versions.

Contents

attribute
The attribute to use to sort the list of versions.
Type: String
Valid Values: BotVersion
Required: Yes

order
The order to sort the list. You can specify ascending or descending order.
Type: String
Valid Values: Ascending | Descending
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 V2
- AWS SDK for Ruby V3
**BotVersionSummary**

Service: Amazon Lex Model Building V2

Summary information about a bot version returned by the `ListBotVersions` operation.

**Contents**

**botName**

The name of the bot associated with the version.

Type: String

Length Constraints: Minimum length of 1. Maximum length of 100.

Pattern: `^([0-9a-zA-Z][_-]?)+$`

Required: No

**botStatus**

The status of the bot. When the status is available, the version of the bot is ready for use.

Type: String

Valid Values: Creating | Available | Inactive | Deleting | Failed | Versioning | Importing

Required: No

**botVersion**

The numeric version of the bot, or DRAFT to indicate that this is the version of the bot that can be updated.

Type: String


Pattern: `^(DRAFT|[0-9]+)$`

Required: No

**creationDateTime**

A timestamp of the date and time that the version was created.

Type: Timestamp

Required: No

**description**

The description of the version.

Type: String

Length Constraints: Minimum length of 0. Maximum length of 200.

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 V2
- AWS SDK for Ruby V3
**BuiltInIntentSortBy**
Service: Amazon Lex Model Building V2

Specifies attributes for sorting a list of built-in intents.

**Contents**

*attribute*

The attribute to use to sort the list of built-in intents.

Type: String

Valid Values: IntentSignature

Required: Yes

*order*

The order to sort the list. You can specify ascending or descending order.

Type: String

Valid Values: Ascending | Descending

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 V2
- AWS SDK for Ruby V3
**BuiltInIntentSummary**
Service: Amazon Lex Model Building V2

Provides summary information about a built-in intent for the **ListBuiltInIntents** operation.

**Contents**

**description**

The description of the intent.

Type: String

Length Constraints: Minimum length of 0. Maximum length of 200.

Required: No

**intentSignature**

The signature of the built-in intent. Use this to specify the parent intent of a derived 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 V2
- AWS SDK for Ruby V3
**BuiltInSlotTypeSortBy**
Service: Amazon Lex Model Building V2

Specifies attributes for sorting a list of built-in slot types.

**Contents**

**attribute**

The attribute to use to sort the list of built-in intents.

Type: String

Valid Values: SlotTypeSignature

Required: Yes

**order**

The order to sort the list. You can choose ascending or descending.

Type: String

Valid Values: Ascending | Descending

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 V2
- AWS SDK for Ruby V3
**BuiltInSlotTypeSummary**  
Service: Amazon Lex Model Building V2

Provides summary information about a built-in slot type for the `ListBuiltInSlotTypes` operation.

**Contents**

**description**

The description of the built-in slot type.

Type: String  
Length Constraints: Minimum length of 0. Maximum length of 200.

Required: No

**slotTypeSignature**

The signature of the built-in slot type. Use this to specify the parent slot type of a derived slot type.

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 V2
- AWS SDK for Ruby V3
Button
Service: Amazon Lex Model Building V2

Describes a button to use on a response card used to gather slot values from a user.

Contents

**text**

The text that appears on the button. Use this to tell the user what value is returned when they choose this button.

Type: String


Required: Yes

**value**

The value returned to Amazon Lex when the user chooses this button. This must be one of the slot values configured for the slot.

Type: String


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 V2
- AWS SDK for Ruby V3
CloudWatchLogGroupLogDestination
Service: Amazon Lex Model Building V2

The Amazon CloudWatch Logs log group where the text and metadata logs are delivered. The log group must exist before you enable logging.

Contents

cloudWatchLogGroupArn

The Amazon Resource Name (ARN) of the log group where text and metadata logs are delivered.

Type: String

Length Constraints: Minimum length of 1. Maximum length of 2048.

Pattern: ^arn:[\w-]+:logs:[\w-]+:[\d]{12}:log-group:[\w\-_/\#A-Za-z0-9] {1,512}(\?\*\*)?$

Required: Yes

logPrefix

The prefix of the log stream name within the log group that you specified

Type: String

Length Constraints: Maximum length of 1024.

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 V2
- AWS SDK for Ruby V3
CodeHookSpecification
Service: Amazon Lex Model Building V2

Contains information about code hooks that Amazon Lex calls during a conversation.

Contents

lambdaCodeHook

Specifies a Lambda function that verifies requests to a bot or fulfills the user's request to a bot.

Type: LambdaCodeHook (p. 733) 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 V2
- AWS SDK for Ruby V3
ConversationLogSettings
Service: Amazon Lex Model Building V2

Configures conversation logging that saves audio, text, and metadata for the conversations with your users.

Contents

audioLogSettings

The Amazon S3 settings for logging audio to an S3 bucket.

Type: Array of AudioLogSetting (p. 663) objects

Array Members: Fixed number of 1 item.

Required: No

textLogSettings

The Amazon CloudWatch Logs settings for logging text and metadata.

Type: Array of TextLogSetting (p. 774) objects

Array Members: Fixed number of 1 item.

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 V2
- AWS SDK for Ruby V3
CustomPayload
Service: Amazon Lex Model Building V2

A custom response string that Amazon Lex sends to your application. You define the content and structure the string.

Contents

value

The string that is sent to your application.

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 V2
- AWS SDK for Ruby V3
CustomVocabularyExportSpecification

Service: Amazon Lex Model Building V2

Provides the parameters required for exporting a custom vocabulary.

Contents

botId

The identifier of the bot that contains the custom vocabulary to export.

Type: String

Length Constraints: Fixed length of 10.

Pattern: ^[0-9a-zA-Z]+$

Required: Yes

botVersion

The version of the bot that contains the custom vocabulary to export.

Type: String


Pattern: ^(DRAFT|\d{1})$

Required: Yes

localeId

The locale of the bot that contains the custom vocabulary to export.

Type: String

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 V2
- AWS SDK for Ruby V3
CustomVocabularyImportSpecification
Service: Amazon Lex Model Building V2

Provides the parameters required for importing a custom vocabulary.

Contents

botId

The identifier of the bot to import the custom vocabulary to.

Type: String
Length Constraints: Fixed length of 10.
Pattern: \^[0-9a-zA-Z]+\$

Required: Yes

botVersion

The version of the bot to import the custom vocabulary to.

Type: String
Length Constraints: Fixed length of 5.
Pattern: \^DRAFT\$

Required: Yes

localeId

The identifier of the local to import the custom vocabulary to. The value must be en_GB.

Type: String
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 V2
- AWS SDK for Ruby V3
DataPrivacy
Service: Amazon Lex Model Building V2

By default, data stored by Amazon Lex is encrypted. The DataPrivacy structure provides settings that determine how Amazon Lex handles special cases of securing the data for your bot.

Contents

childDirected

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

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 V2
- AWS SDK for Ruby V3
**DateRangeFilter**  
*Service: Amazon Lex Model Building V2*

The object used for specifying the data range that the customer wants Amazon Lex to read through in the input transcripts.

**Contents**

- **endDate:** A timestamp indicating the end date for the date range filter.  
  Type: Timestamp  
  Required: Yes

- **startDateTime:** A timestamp indicating the start date for the date range filter.  
  Type: Timestamp  
  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 V2
- AWS SDK for Ruby V3
DialogCodeHookSettings
Service: Amazon Lex Model Building V2

Settings that determine the Lambda function that Amazon Lex uses for processing user responses.

Contents

enabled

Enables the dialog code hook so that it processes user requests.

Type: Boolean
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 V2
- AWS SDK for Ruby V3
EncryptionSetting
Service: Amazon Lex Model Building V2

The object representing the passwords that were used to encrypt the data related to the bot recommendation, as well as the KMS key ARN used to encrypt the associated metadata.

Contents

associatedTranscriptsPassword

The password used to encrypt the associated transcript file.

Type: String

Length Constraints: Minimum length of 0. Maximum length of 1024.

Required: No

botLocaleExportPassword

The password used to encrypt the recommended bot recommendation file.

Type: String

Length Constraints: Minimum length of 0. Maximum length of 1024.

Required: No

kmsKeyArn

The KMS key ARN used to encrypt the metadata associated with the bot recommendation.

Type: String


Pattern: ^arn:[\w\-]+:kms:[\w\-]+:[\d]{12}:(?:key\/\[\w\-]+|alias\/\[a-zA-Z0-9: _\-\]{1,256})$

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 V2
- AWS SDK for Ruby V3
ExportFilter
Service: Amazon Lex Model Building V2

Filters the response from the ListExports operation

Contents

name

The name of the field to use for filtering.

Type: String

Valid Values: ExportResourceType

Required: Yes

operator

The operator to use for the filter. Specify EQ when the ListExports operation should return only resource types that equal the specified value. Specify CO when the ListExports operation should return resource types that contain the specified value.

Type: String

Valid Values: CO | EQ

Required: Yes

values

The values to use to filter the response. The values must be Bot, BotLocale, or CustomVocabulary.

Type: Array of strings

Array Members: Fixed number of 1 item.

Length Constraints: Minimum length of 1. Maximum length of 100.

Pattern: ^[0-9a-zA-Z_\-\(\)\s-]*$

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 V2
- AWS SDK for Ruby V3
ExportResourceSpecification
Service: Amazon Lex Model Building V2

Provides information about the bot or bot locale that you want to export. You can specify the botExportSpecification or the botLocaleExportSpecification, but not both.

Contents

botExportSpecification
Parameters for exporting a bot.
Type: BotExportSpecification (p. 668) object
Required: No

botLocaleExportSpecification
Parameters for exporting a bot locale.
Type: BotLocaleExportSpecification (p. 672) object
Required: No

customVocabularyExportSpecification
The parameters required to export a custom vocabulary.
Type: CustomVocabularyExportSpecification (p. 699) 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 V2
- AWS SDK for Ruby V3
ExportSortBy
Service: Amazon Lex Model Building V2

Provides information about sorting a list of exports.

Contents

attribute
The export field to use for sorting.
Type: String
Valid Values: LastUpdatedDateTime
Required: Yes

order
The order to sort the list.
Type: String
Valid Values: Ascending | Descending
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 V2
- AWS SDK for Ruby V3
ExportSummary
Service: Amazon Lex Model Building V2

Provides summary information about an export in an export list.

Contents

creationDateTime
The date and time that the export was created.
Type: Timestamp
Required: No

exportId
The unique identifier that Amazon Lex assigned to the export.
Type: String
Length Constraints: Fixed length of 10.
Pattern: ^[0-9a-zA-Z]+$
Required: No

exportStatus
The status of the export. When the status is Completed the export is ready to download.
Type: String
Valid Values: InProgress | Completed | Failed | Deleting
Required: No

fileFormat
The file format used in the export files.
Type: String
Valid Values: LexJson | TSV
Required: No

lastUpdatedDateTime
The date and time that the export was last updated.
Type: Timestamp
Required: No

resourceSpecification
Information about the bot or bot locale that was exported.
Type: ExportResourceSpecification (p. 706) 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 V2
- AWS SDK for Ruby V3
ExternalSourceSetting
Service: Amazon Lex Model Building V2

Provides information about the external source of the slot type's definition.

Contents

grammarslotTypeSetting
Settings required for a slot type based on a grammar that you provide.

Type: GrammarslotTypeSetting (p. 715) 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 V2
- AWS SDK for Ruby V3
**FulfillmentCodeHookSettings**  
Service: Amazon Lex Model Building V2

Determines if a Lambda function should be invoked for a specific intent.

**Contents**

**enabled**

Indicates whether a Lambda function should be invoked to fulfill a specific intent.

Type: Boolean

Required: Yes

**fulfillmentUpdatesSpecification**

Provides settings for update messages sent to the user for long-running Lambda fulfillment functions. Fulfillment updates can be used only with streaming conversations.

Type: `FulfillmentUpdatesSpecification (p. 714)` object

Required: No

**postFulfillmentStatusSpecification**

Provides settings for messages sent to the user for after the Lambda fulfillment function completes. Post-fulfillment messages can be sent for both streaming and non-streaming conversations.

Type: `PostFulfillmentStatusSpecification (p. 742)` 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 V2
- AWS SDK for Ruby V3
FulfillmentStartResponseSpecification
Service: Amazon Lex Model Building V2

Provides settings for a message that is sent to the user when a fulfillment Lambda function starts running.

Contents

allowInterrupt

Determines whether the user can interrupt the start message while it is playing.

Type: Boolean

Required: No

delayInSeconds

The delay between when the Lambda fulfillment function starts running and the start message is played. If the Lambda function returns before the delay is over, the start message isn’t played.

Type: Integer


Required: Yes

messageGroups

One to 5 message groups that contain start messages. Amazon Lex chooses one of the messages to play to the user.

Type: Array of MessageGroup (p. 736) objects

Array Members: Minimum number of 1 item. Maximum number of 5 items.

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 V2
- AWS SDK for Ruby V3
FulfillmentUpdateResponseSpecification
Service: Amazon Lex Model Building V2

Provides settings for a message that is sent periodically to the user while a fulfillment Lambda function is running.

Contents

allowInterrupt
Determines whether the user can interrupt an update message while it is playing.

Type: Boolean
Required: No

frequencyInSeconds
The frequency that a message is sent to the user. When the period ends, Amazon Lex chooses a message from the message groups and plays it to the user. If the fulfillment Lambda returns before the first period ends, an update message is not played to the user.

Type: Integer
Required: Yes

messageGroups
One to 5 message groups that contain update messages. Amazon Lex chooses one of the messages to play to the user.

Type: Array of MessageGroup (p. 736) objects
Array Members: Minimum number of 1 item. Maximum number of 5 items.
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 V2
* AWS SDK for Ruby V3
**FulfillmentUpdatesSpecification**

Service: Amazon Lex Model Building V2

Provides information for updating the user on the progress of fulfilling an intent.

**Contents**

active

Determines whether fulfillment updates are sent to the user. When this field is true, updates are sent.

If the active field is set to true, the startResponse, updateResponse, and timeoutInSeconds fields are required.

Type: Boolean

Required: Yes

startResponse

Provides configuration information for the message sent to users when the fulfillment Lambda functions starts running.

Type: [FulfillmentStartResponseSpecification](p. 712) object

Required: No

timeoutInSeconds

The length of time that the fulfillment Lambda function should run before it times out.

Type: Integer

Valid Range: Minimum value of 1. Maximum value of 900

Required: No

updateResponse

Provides configuration information for messages sent periodically to the user while the fulfillment Lambda function is running.

Type: [FulfillmentUpdateResponseSpecification](p. 713) 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 V2
- AWS SDK for Ruby V3
GrammarSlotTypeSetting
Service: Amazon Lex Model Building V2

Settings required for a slot type based on a grammar that you provide.

Contents

source

The source of the grammar used to create the slot type.

Type: GrammarSlotTypeSource (p. 716) 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 V2
- AWS SDK for Ruby V3
GrammarSlotTypeSource
Service: Amazon Lex Model Building V2

Describes the Amazon S3 bucket name and location for the grammar that is the source for the slot type.

Contents

kmsKeyArn

The Amazon KMS key required to decrypt the contents of the grammar, if any.

Type: String


Pattern: ^arn:[\\w\-]+:kms:[\\w\-]+:[\\d]{12}:(?::key\|[\\w\-]+|alias\(/[\w\-]{1,256}\))$

Required: No

s3BucketName

The name of the S3 bucket that contains the grammar source.

Type: String


Pattern: ^[\w\-\_\*\'\(\)(a-zA-Z0-9\.\-_\*\'\()]/a-zA-Z0-9]*$

Required: Yes

s3ObjectKey

The path to the grammar in the S3 bucket.

Type: String


Pattern: [\\\-_\!\*\_\'\`\(\()a-zA-Z0-9]/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 V2
- AWS SDK for Ruby V3
ImageResponseCard
Service: Amazon Lex Model Building V2

A card that is shown to the user by a messaging platform. You define the contents of the card, the card is displayed by the platform.

When you use a response card, the response from the user is constrained to the text associated with a button on the card.

Contents

buttons

A list of buttons that should be displayed on the response card. The arrangement of the buttons is determined by the platform that displays the button.

Type: Array of Button (p. 694) objects

Array Members: Minimum number of 0 items. Maximum number of 5 items.

Required: No

imageUrl

The URL of an image to display on the response card. The image URL must be publicly available so that the platform displaying the response card has access to the image.

Type: String


Required: No

subtitle

The subtitle to display on the response card. The format of the subtitle is determined by the platform displaying the response card.

Type: String


Required: No

title

The title to display on the response card. The format of the title is determined by the platform displaying the response card.

Type: String


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 V2
• AWS SDK for Ruby V3
ImportFilter
Service: Amazon Lex Model Building V2

Filters the response from the ListImports operation.

Contents

name

The name of the field to use for filtering.

Type: String

Valid Values: ImportResourceType

Required: Yes

operator

The operator to use for the filter. Specify EQ when the ListImports operation should return only resource types that equal the specified value. Specify CO when the ListImports operation should return resource types that contain the specified value.

Type: String

Valid Values: CO | EQ

Required: Yes

values

The values to use to filter the response. The values must be Bot, BotLocale, or CustomVocabulary.

Type: Array of strings

Array Members: Fixed number of 1 item.

Length Constraints: Minimum length of 1. Maximum length of 100.

Pattern: ^[0-9a-zA-Z_().\-]+$

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 V2
- AWS SDK for Ruby V3
ImportResourceSpecification
Service: Amazon Lex Model Building V2

Provides information about the bot or bot locale that you want to import. You can specify the
botImportSpecification or the botLocaleImportSpecification, but not both.

Contents

botImportSpecification

Parameters for importing a bot.
Type: BotImportSpecification (p. 670) object
Required: No

botLocaleImportSpecification

Parameters for importing a bot locale.
Type: BotLocaleImportSpecification (p. 675) object
Required: No

customVocabularyImportSpecification

Provides the parameters required for importing a custom vocabulary.

Type: CustomVocabularyImportSpecification (p. 700) 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 V2
• AWS SDK for Ruby V3
ImportSortBy
Service: Amazon Lex Model Building V2

Provides information for sorting a list of imports.

Contents

attribute

The export field to use for sorting.

Type: String

Valid Values: LastUpdatedDateTime

Required: Yes

order

The order to sort the list.

Type: String

Valid Values: Ascending | Descending

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 V2
- AWS SDK for Ruby V3
**ImportSummary**
Service: Amazon Lex Model Building V2

Provides summary information about an import in an import list.

**Contents**

- `creationDateTime`
  - The date and time that the import was created.
  - Type: Timestamp
  - Required: No

- `importedResourceId`
  - The unique identifier that Amazon Lex assigned to the imported resource.
  - Type: String
  - Pattern: ^([0-9a-zA-Z_]+)$
  - Required: No

- `importedResourceName`
  - The name that you gave the imported resource.
  - Type: String
  - Pattern: ^([0-9a-zA-Z][-_]?)]+$  
  - Required: No

- `importedResourceType`
  - The type of resource that was imported.
  - Type: String
  - Valid Values: Bot | BotLocale | CustomVocabulary
  - Required: No

- `importId`
  - The unique identifier that Amazon Lex assigned to the import.
  - Type: String
  - Pattern: ^([0-9a-zA-Z]+)$
  - Required: No

- `importStatus`
  - The status of the resource. When the status is `Completed` the resource is ready to build.
Type: String
Valid Values: InProgress | Completed | Failed | Deleting
Required: No

lastUpdatedDateTime
The date and time that the import was last updated.
Type: Timestamp
Required: No

mergeStrategy
The strategy used to merge existing bot or bot locale definitions with the imported definition.
Type: String
Valid Values: Overwrite | FailOnConflict | Append
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 V2
- AWS SDK for Ruby V3
**InputContext**

Service: Amazon Lex Model Building V2

The name of a context that must be active for an intent to be selected by Amazon Lex.

**Contents**

**name**

The name of the context.

Type: String

Length Constraints: Minimum length of 1. Maximum length of 100.

Pattern: ^([0-9a-zA-Z][_-]?)+$

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 V2
- AWS SDK for Ruby V3
IntentClosingSetting
Service: Amazon Lex Model Building V2

Provides a statement the Amazon Lex conveys to the user when the intent is successfully fulfilled.

Contents

active

Specifies whether an intent's closing response is used. When this field is false, the closing response isn't sent to the user. If the active field isn't specified, the default is true.

Type: Boolean
Required: No

closingResponse

The response that Amazon Lex sends to the user when the intent is complete.

Type: ResponseSpecification (p. 747) 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 V2
- AWS SDK for Ruby V3
IntentConfirmationSetting
Service: Amazon Lex Model Building V2

Provides a prompt for making sure that the user is ready for the intent to be fulfilled.

Contents

active

Specifies whether the intent's confirmation is sent to the user. When this field is false, confirmation and declination responses aren't sent. If the active field isn't specified, the default is true.

Type: Boolean
Required: No

declinationResponse

When the user answers "no" to the question defined in promptSpecification, Amazon Lex responds with this response to acknowledge that the intent was canceled.

Type: ResponseSpecification (p. 747) object
Required: Yes

promptSpecification

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.

Type: PromptSpecification (p. 744) 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 V2
- AWS SDK for Ruby V3
**IntentFilter**
Service: Amazon Lex Model Building V2

Filters the response from the ListIntents operation.

**Contents**

**name**

The name of the field to use for the filter.

Type: String

Valid Values: IntentName

Required: Yes

**operator**

The operator to use for the filter. Specify EQ when the ListIntents operation should return only aliases that equal the specified value. Specify CO when the ListIntents operation should return aliases that contain the specified value.

Type: String

Valid Values: CO | EQ

Required: Yes

**values**

The value to use for the filter.

Type: Array of strings

Array Members: Fixed number of 1 item.

Length Constraints: Minimum length of 1. Maximum length of 100.

Pattern: ^[0-9a-zA-Z_\-()+\s-]+$

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 V2
- AWS SDK for Ruby V3
IntentSortBy
Service: Amazon Lex Model Building V2

Specifies attributes for sorting a list of intents.

Contents

attribute

The attribute to use to sort the list of intents.

Type: String

Valid Values: IntentName | LastUpdatedAtDateTime

Required: Yes

order

The order to sort the list. You can choose ascending or descending.

Type: String

Valid Values: Ascending | Descending

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 V2
- AWS SDK for Ruby V3
**IntentStatistics**

Service: Amazon Lex Model Building V2

The object that contains the statistical summary of recommended intents associated with the bot recommendation.

**Contents**

*discoveredIntentCount*

The number of recommended intents associated with the bot recommendation.

Type: Integer

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 V2
- AWS SDK for Ruby V3
**IntentSummary**
Service: Amazon Lex Model Building V2

Summary information about an intent returned by the ListIntents operation.

**Contents**

**description**

The description of the intent.

Type: String
Length Constraints: Minimum length of 0. Maximum length of 200.
Required: No

**inputContexts**

The input contexts that must be active for this intent to be considered for recognition.

Type: Array of InputContext (p. 724) objects
Array Members: Minimum number of 0 items. Maximum number of 5 items.
Required: No

**intentId**

The unique identifier assigned to the intent. Use this ID to get detailed information about the intent with the DescribeIntent operation.

Type: String
Length Constraints: Fixed length of 10.
Pattern: ^[0-9a-zA-Z]+$
Required: No

**intentName**

The name of the intent.

Type: String
Length Constraints: Minimum length of 1. Maximum length of 100.
Pattern: ^([0-9a-zA-Z][-_]?)+$
Required: No

**lastUpdatedDateTime**

The timestamp of the date and time that the intent was last updated.

Type: Timestamp
Required: No

**outputContexts**

The output contexts that are activated when this intent is fulfilled.

Type: Array of OutputContext (p. 739) objects
Array Members: Minimum number of 0 items. Maximum number of 10 items.

Required: No

parentIntentSignature

If this intent is derived from a built-in intent, the name of the parent 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 V2
- AWS SDK for Ruby V3
KendraConfiguration
Service: Amazon Lex Model Building V2

Provides configuration information for the AMAZON.KendraSearchIntent intent. When you use this intent, Amazon Lex searches the specified Amazon Kendra index and returns documents from the index that match the user's utterance.

Contents

kendraIndex

The Amazon Resource Name (ARN) of the Amazon Kendra index that you want the AMAZON.KendraSearchIntent intent to search. The index must be in the same account and Region as the Amazon Lex bot.

Type: String

Length Constraints: Minimum length of 32. Maximum length of 2048.

Pattern: ^arn:aws:kendra:\[a-z]+-[a-z]+-[0-9]:[0-9]{12}:index\/[a-zA-Z0-9][a-zA-Z0-9-0-9_]*$*

Required: Yes

queryFilterString

A query filter that Amazon Lex sends to Amazon Kendra to filter the response from a query. The filter is in the format defined by Amazon Kendra. For more information, see Filtering queries.

Type: String

Length Constraints: Minimum length of 1. Maximum length of 5000.

Required: No

queryFilterStringEnabled

Determines whether the AMAZON.KendraSearchIntent intent uses a custom query string to query the Amazon Kendra index.

Type: Boolean

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 V2
- AWS SDK for Ruby V3
LambdaCodeHook
Service: Amazon Lex Model Building V2

Specifies a Lambda function that verifies requests to a bot or fulfills the user's request to a bot.

Contents

codeHookInterfaceVersion
The version of the request-response that you want Amazon Lex to use to invoke your Lambda function.

Type: String


Required: Yes

lambdaARN
The Amazon Resource Name (ARN) of the Lambda function.

Type: String


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 V2
- AWS SDK for Ruby V3
LexTranscriptFilter
Service: Amazon Lex Model Building V2

The object that contains transcript filter details that are associated with a bot recommendation.

Contents

dateRangeFilter

The object that contains a date range filter that will be applied to the transcript. Specify this object if you want Amazon Lex to only read the files that are within the date range.

Type: DateRangeFilter (p. 702) 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 V2
- AWS SDK for Ruby V3
**Message**

Service: Amazon Lex Model Building V2

The object that provides message text and its type.

**Contents**

- **customPayload**
  A message in a custom format defined by the client application.
  
  Type: CustomPayload (p. 698) object
  
  Required: No

- **imageResponseCard**
  A message that defines a response card that the client application can show to the user.
  
  Type: ImageResponseCard (p. 717) object
  
  Required: No

- **plainTextMessage**
  A message in plain text format.
  
  Type: PlainTextMessage (p. 741) object
  
  Required: No

- **ssmlMessage**
  A message in Speech Synthesis Markup Language (SSML).
  
  Type: SSMLMessage (p. 771) 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 V2
- AWS SDK for Ruby V3
MessageGroup

Service: Amazon Lex Model Building V2

Provides one or more messages that Amazon Lex should send to the user.

Contents

message

The primary message that Amazon Lex should send to the user.

Type: Message (p. 735) object

Required: Yes

variations

Message variations to send to the user. When variations are defined, Amazon Lex chooses the primary message or one of the variations to send to the user.

Type: Array of Message (p. 735) objects

Array Members: Minimum number of 0 items. Maximum number of 2 items.

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 V2
- AWS SDK for Ruby V3
MultipleValuesSetting
Service: Amazon Lex Model Building V2

Indicates whether a slot can return multiple values.

Contents

allowMultipleValues

Indicates whether a slot can return multiple values. When true, the slot may return more than one value in a response. When false, the slot returns only a single value.

Multi-value slots are only available in the en-US locale. If you set this value to true in any other locale, Amazon Lex throws a ValidationException.

If the allowMultipleValues is not set, the default value is false.

Type: Boolean

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 V2
- AWS SDK for Ruby V3
ObfuscationSetting
Service: Amazon Lex Model Building V2

Determines whether Amazon Lex obscures slot values in conversation logs.

Contents

obfuscationSettingType

Value that determines whether Amazon Lex obscures slot values in conversation logs. The default is to obscure the values.

Type: String

Valid Values: None | DefaultObfuscation

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 V2
- AWS SDK for Ruby V3
OutputContext
Service: Amazon Lex Model Building V2

Describes a session context that is activated when an intent is fulfilled.

Contents

name

The name of the output context.

Type: String

Length Constraints: Minimum length of 1. Maximum length of 100.

Pattern: ^([0-9a-zA-Z][_\-]?)+$

Required: Yes

timeToLiveInSeconds

The amount of time, in seconds, that the output context should remain active. The time is figured from the first time the context is sent to the user.

Type: Integer


Required: Yes

turnsToLive

The number of conversation turns that the output context should remain active. The number of turns is counted from the first time that the context is sent to the user.

Type: Integer


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 V2
- AWS SDK for Ruby V3
PathFormat
Service: Amazon Lex Model Building V2

The object that contains a path format that will be applied when Amazon Lex reads the transcript file in the bucket you provide. Specify this object if you only want Lex to read a subset of files in your Amazon S3 bucket.

Contents

objectPrefixes

A list of Amazon S3 prefixes that points to sub-folders in the Amazon S3 bucket. Specify this list if you only want Lex to read the files under this set of sub-folders.

Type: Array of strings

Array Members: Minimum number of 1 item. Maximum number of 2 items.

Length Constraints: Minimum length of 1.

Pattern: ^[\/]?[a-zA-Z0-9!_.*'()-]+(/[a-zA-Z0-9!_.*'()-]+)*$

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 V2
- AWS SDK for Ruby V3
PlainTextMessage
Service: Amazon Lex Model Building V2

Defines an ASCII text message to send to the user.

Contents

value

The message to send to the user.

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 V2
- AWS SDK for Ruby V3
PostFulfillmentStatusSpecification
Service: Amazon Lex Model Building V2

Provides a setting that determines whether the post-fulfillment response is sent to the user. For more information, see https://docs.aws.amazon.com/lexv2/latest/dg/streaming-progress.html#progress-complete

Contents

failureResponse
  Specifies a list of message groups that Amazon Lex uses to respond the user input.
  Type: ResponseSpecification (p. 747) object
  Required: No

successResponse
  Specifies a list of message groups that Amazon Lex uses to respond the user input.
  Type: ResponseSpecification (p. 747) object
  Required: No

timeoutResponse
  Specifies a list of message groups that Amazon Lex uses to respond the user input.
  Type: ResponseSpecification (p. 747) 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 V2
- AWS SDK for Ruby V3
Principal
Service: Amazon Lex Model Building V2

The IAM principal that you allowing or denying access to an Amazon Lex action. You must provide a service or an arn, but not both in the same statement. For more information, see AWS JSON policy elements: Principal.

Contents

arn

The Amazon Resource Name (ARN) of the principal.

Type: String


Pattern: ^arn:aws:iam::[0-9]{12}:(root|(user|role)/.*)$

Required: No

service

The name of the AWS service that should allowed or denied access to an Amazon Lex action.

Type: String


Pattern: ^[0-9a-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 V2
- AWS SDK for Ruby V3
**PromptSpecification**
Service: Amazon Lex Model Building V2

Specifies a list of message groups that Amazon Lex sends to a user to elicit a response.

**Contents**

**allowInterrupt**
Indicates whether the user can interrupt a speech prompt from the bot.

Type: Boolean

Required: No

**maxRetries**
The maximum number of times the bot tries to elicit a response from the user using this prompt.

Type: Integer

Valid Range: Minimum value of 0. Maximum value of 5.

Required: Yes

**messageGroups**
A collection of messages that Amazon Lex can send to the user. Amazon Lex chooses the actual message to send at runtime.

Type: Array of MessageGroup (p. 736) objects

Array Members: Minimum number of 1 item. Maximum number of 5 items.

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 V2
- AWS SDK for Ruby V3
RecommendedIntentSummary
Service: Amazon Lex Model Building V2

An object that contains a summary of a recommended intent.

Contents

intentId

The unique identifier of a recommended intent associated with the bot recommendation.

Type: String

Length Constraints: Fixed length of 10.

Pattern: ^[0-9a-zA-Z]+$

Required: No

intentName

The name of a recommended intent associated with the bot recommendation.

Type: String

Length Constraints: Minimum length of 1. Maximum length of 100.

Pattern: ^([0-9a-zA-Z][_-]?)++$

Required: No

sampleUtterancesCount

The count of sample utterances of a recommended intent that is associated with a bot recommendation.

Type: Integer

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 V2
- AWS SDK for Ruby V3
RelativeAggregationDuration
Service: Amazon Lex Model Building V2

Specifies the time window that utterance statistics are returned for. The time window is always relative to the last time that the utterances were aggregated. For example, if the ListAggregatedUtterances operation is called at 1600, the time window is set to 1 hour, and the last refresh time was 1530, only utterances made between 1430 and 1530 are returned.

You can choose the time window that statistics should be returned for.

- **Hours** - You can request utterance statistics for 1, 3, 6, 12, or 24 hour time windows. Statistics are refreshed every half hour for 1 hour time windows, and hourly for the other time windows.
- **Days** - You can request utterance statistics for 3 days. Statistics are refreshed every 6 hours.
- **Weeks** - You can see statistics for one or two weeks. Statistics are refreshed every 12 hours for one week time windows, and once per day for two week time windows.

**Contents**

?-Dimension

The type of time period that the timeValue field represents.

Type: String

Valid Values: Hours | Days | Weeks

Required: Yes

timeValue

The period of the time window to gather statistics for. The valid value depends on the setting of the timeDimension field.

- **Hours** - 1/3/6/12/24
- **Days** - 3
- **Weeks** - 1/2

Type: Integer


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 V2
- AWS SDK for Ruby V3
ResponseSpecification
Service: Amazon Lex Model Building V2

Specifies a list of message groups that Amazon Lex uses to respond the user input.

Contents

allowInterrupt

Indicates whether the user can interrupt a speech response from Amazon Lex.

Type: Boolean
Required: No

messageGroups

A collection of responses that Amazon Lex can send to the user. Amazon Lex chooses the actual response to send at runtime.

Type: Array of MessageGroup (p. 736) objects
Array Members: Minimum number of 1 item. Maximum number of 5 items.
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 V2
- AWS SDK for Ruby V3
**S3BucketLogDestination**
Service: Amazon Lex Model Building V2

Specifies an Amazon S3 bucket for logging audio conversations

**Contents**

**kmsKeyArn**

The Amazon Resource Name (ARN) of an AWS Key Management Service (KMS) key for encrypting audio log files stored in an S3 bucket.

Type: String


Pattern: `^arn:[:\w\-]+:kms[:\w\-]+:[\d]{12}:(?:key/[:\w\-]+|alias/[:a-zA-Z0-9\-_]{1,256})$`

Required: No

**logPrefix**

The S3 prefix to assign to audio log files.

Type: String

Length Constraints: Maximum length of 1024.

Required: Yes

**s3BucketArn**

The Amazon Resource Name (ARN) of an Amazon S3 bucket where audio log files are stored.

Type: String

Length Constraints: Minimum length of 1. Maximum length of 2048.

Pattern: `^arn:[:\w\-]+:s3::[:a-z0-9\-][.\-a-z0-9]{1,61}[a-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 V2
- AWS SDK for Ruby V3
S3BucketTranscriptSource
Service: Amazon Lex Model Building V2

The object representing the Amazon S3 bucket containing the transcript, as well as the associated metadata.

Contents

**kmsKeyArn**

The ARN of the KMS key that customers use to encrypt their Amazon S3 bucket. Only use this field if your bucket is encrypted using a customer-managed KMS key.

Type: String


Pattern: ^arn:[^\w\-]+:kms:[^\w\-]+:[^d]{12}:(?::key/[^\w\-]+|alias/[a-zA-Z0-9: \_/\-]{1,256})$

Required: No

**pathFormat**

The object that contains a path format that will be applied when Amazon Lex reads the transcript file in the bucket you provide. Specify this object if you only want Lex to read a subset of files in your Amazon S3 bucket.

Type: PathFormat (p. 740) object

Required: No

**s3BucketName**

The name of the bucket containing the transcript and the associated metadata.

Type: String


Pattern: ^[a-zA-Z0-9][\._a-zA-Z0-9]{1,61}[a-zA-Z0-9]$

Required: Yes

**transcriptFilter**

The object that contains the filter which will be applied when Amazon Lex reads through the Amazon S3 bucket. Specify this object if you want Amazon Lex to read only a subset of the Amazon S3 bucket based on the filter you provide.

Type: TranscriptFilter (p. 775) object

Required: No

**transcriptFormat**

The format of the transcript content. Currently, Genie only supports the Amazon Lex transcript format.

Type: String

Valid Values: Lex

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 V2
- AWS SDK for Ruby V3
SampleUtterance
Service: Amazon Lex Model Building V2

A sample utterance that invokes an intent or respond to a slot elicitation prompt.

Contents

utterance

The sample utterance that Amazon Lex uses to build its machine-learning model to recognize intents.

Type: String
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 V2
- AWS SDK for Ruby V3
SampleValue
Service: Amazon Lex Model Building V2

Defines one of the values for a slot type.

Contents

value

The value that can be used for a 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 V2
- AWS SDK for Ruby V3
SentimentAnalysisSettings
Service: Amazon Lex Model Building V2

Determines whether Amazon Lex will use Amazon Comprehend to detect the sentiment of user utterances.

Contents

detectSentiment

Sets whether Amazon Lex uses Amazon Comprehend to detect the sentiment of user utterances.

Type: Boolean

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 V2
- AWS SDK for Ruby V3
SlotDefaultValue
Service: Amazon Lex Model Building V2

Specifies the default value to use when a user doesn't provide a value for a slot.

Contents

defaultValue

The default value to use when a user doesn't provide a value for a slot.

Type: String


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 V2
- AWS SDK for Ruby V3
SlotDefaultValueSpecification
Service: Amazon Lex Model Building V2

Defines a list of values that Amazon Lex should use as the default value for a slot.

Contents

defaultValueList

A list of default values. Amazon Lex chooses the default value to use in the order that they are presented in the list.

Type: Array of SlotDefaultValue (p. 754) objects

Array Members: Minimum number of 0 items. Maximum number of 10 items.

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 V2
- AWS SDK for Ruby V3
SlotFilter
Service: Amazon Lex Model Building V2
Filters the response from the ListSlots operation.

Contents

name
The name of the field to use for filtering.
Type: String
Valid Values: SlotName
Required: Yes

operator
The operator to use for the filter. Specify EQ when the ListSlots operation should return only aliases that equal the specified value. Specify CO when the ListSlots operation should return aliases that contain the specified value.
Type: String
Valid Values: CO | EQ
Required: Yes

values
The value to use to filter the response.
Type: Array of strings
Array Members: Fixed number of 1 item.
Length Constraints: Minimum length of 1. Maximum length of 100.
Pattern: /^[\d\-a-zA-Z_()-\s]+$/
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 V2
- AWS SDK for Ruby V3
SlotPriority
Service: Amazon Lex Model Building V2

Sets the priority that Amazon Lex should use when eliciting slot values from a user.

Contents

priority

The priority that a slot should be elicited.

Type: Integer

Valid Range: Minimum value of 0. Maximum value of 100.

Required: Yes

slotId

The unique identifier of the slot.

Type: String

Length Constraints: Fixed length of 10.

Pattern: ^[0-9a-zA-Z]+$

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 V2
- AWS SDK for Ruby V3
SlotSortBy
Service: Amazon Lex Model Building V2

Specifies attributes for sorting a list of bots.

Contents

attribute

The attribute to use to sort the list.

Type: String

Valid Values: SlotName | LastUpdatedDateTime

Required: Yes

order

The order to sort the list. You can choose ascending or descending.

Type: String

Valid Values: Ascending | Descending

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 V2
- AWS SDK for Ruby V3
SlotSummary
Service: Amazon Lex Model Building V2

Summary information about a slot, a value that the bot elicits from the user.

Contents

description
The description of the slot.
Type: String
Length Constraints: Minimum length of 0. Maximum length of 200.
Required: No

lastUpdatedDateTime
The timestamp of the last date and time that the slot was updated.
Type: Timestamp
Required: No

slotConstraint
Whether the slot is required or optional. An intent is complete when all required slots are filled.
Type: String
Valid Values: Required | Optional
Required: No

slotId
The unique identifier of the slot.
Type: String
Length Constraints: Fixed length of 10.
Pattern: ^[0-9a-zA-Z]+$
Required: No

slotName
The name given to the slot.
Type: String
Length Constraints: Minimum length of 1. Maximum length of 100.
Pattern: ^([0-9a-zA-Z][_-]?)+$
Required: No

slotTypeId
The unique identifier for the slot type that defines the values for the slot.
Type: String

Pattern: ^((AMAZON\.)[a-zA-Z_]+|\d[a-zA-Z]+)$

Required: No

valueElicitationPromptSpecification

Prompts that are sent to the user to elicit a value for the slot.

Type: PromptSpecification (p. 744) 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 V2
- AWS SDK for Ruby V3
SlotTypeFilter
Service: Amazon Lex Model Building V2

Filters the response from the ListSlotTypes operation.

Contents

name

The name of the field to use for filtering.

Type: String

Valid Values: SlotTypeName | ExternalSourceType

Required: Yes

operator

The operator to use for the filter. Specify EQ when the ListSlotTypes operation should return only aliases that equal the specified value. Specify CO when the ListSlotTypes operation should return aliases that contain the specified value.

Type: String

Valid Values: CO | EQ

Required: Yes

values

The value to use to filter the response.

Type: Array of strings

Array Members: Fixed number of 1 item.

Length Constraints: Minimum length of 1. Maximum length of 100.

Pattern: ^[0-9a-zA-Z_\(\)\s-]+$

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 V2
- AWS SDK for Ruby V3
SlotTypeSortBy
Service: Amazon Lex Model Building V2

Specifies attributes for sorting a list of slot types.

Contents

attribute
The attribute to use to sort the list of slot types.

Type: String

Valid Values: SlotTypeName | LastUpdatedDateTime

Required: Yes

order
The order to sort the list. You can say ascending or descending.

Type: String

Valid Values: Ascending | Descending

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 V2
- AWS SDK for Ruby V3
SlotTypeStatistics
Service: Amazon Lex Model Building V2

The object that contains the statistical summary of the recommended slot type associated with the bot recommendation.

Contents

discoveredSlotTypeCount

The number of recommended slot types associated with the bot recommendation.

Type: Integer
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 V2
- AWS SDK for Ruby V3
SlotTypeSummary
Service: Amazon Lex Model Building V2

Provides summary information about a slot type.

Contents

description
The description of the slot type.
Type: String
Length Constraints: Minimum length of 0. Maximum length of 200.
Required: No

lastUpdatedDateTime
A timestamp of the date and time that the slot type was last updated.
Type: Timestamp
Required: No

parentSlotTypeSignature
If the slot type is derived from a built-on slot type, the name of the parent slot type.
Type: String
Required: No

slotTypeCategory
Indicates the type of the slot type.
- Custom - A slot type that you created using custom values. For more information, see Creating custom slot types.
- Extended - A slot type created by extending the AMAZON.AlphaNumeric built-in slot type. For more information, see AMAZON.AlphaNumeric.
- ExternalGrammar - A slot type using a custom GRXML grammar to define values. For more information, see Using a custom grammar slot type.
Type: String
Valid Values: Custom | Extended | ExternalGrammar
Required: No

slotTypeId
The unique identifier assigned to the slot type.
Type: String
Length Constraints: Fixed length of 10.
Pattern: ^[0-9a-zA-Z]+$
Required: No

slotTypeName
The name of the slot type.
Type: String

Length Constraints: Minimum length of 1. Maximum length of 100.

Pattern: ^([0-9a-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 V2
- AWS SDK for Ruby V3
SlotTypeValue
Service: Amazon Lex Model Building V2

Each slot type can have a set of values. Each SlotTypeValue represents a value that the slot type can take.

Contents

sampleValue

The value of the slot type entry.

Type: SampleValue (p. 752) object

Required: No

synonyms

Additional values related to the slot type entry.

Type: Array of SampleValue (p. 752) objects

Array Members: Minimum number of 1 item. Maximum number of 10000 items.

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 V2
- AWS SDK for Ruby V3
SlotValueElicitationSetting

Service: Amazon Lex Model Building V2

Settings that you can use for eliciting a slot value.

Contents

defaultValueSpecification

A list of default values for a slot. Default values are used when Amazon Lex hasn’t determined a value for a slot. You can specify default values from context variables, session attributes, and defined values.

Type: SlotDefaultValueSpecification (p. 755) object

Required: No

promptSpecification

The prompt that Amazon Lex uses to elicit the slot value from the user.

Type: PromptSpecification (p. 744) object

Required: No

sampleUtterances

If you know a specific pattern that 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 SampleUtterance (p. 751) objects

Required: No

slotConstraint

Specifies whether the slot is required or optional.

Type: String

Valid Values: Required | Optional

Required: Yes

waitAndContinueSpecification

Specifies the prompts that Amazon Lex uses while a bot is waiting for customer input.

Type: WaitAndContinueSpecification (p. 779) 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 V2
- AWS SDK for Ruby V3
SlotValueRegexFilter
Service: Amazon Lex Model Building V2

Provides a regular expression used to validate the value of a slot.

Contents

pattern

A regular expression used to validate the value of a slot.

Use a standard regular expression. Amazon Lex supports the following characters in the regular expression:
- A-Z, a-z
- 0-9
- Unicode characters (\ u<Unicode>)

Represent Unicode characters with four digits, for example \u0041 or \u005A.

The following regular expression operators are not supported:
- Infinite repeaters: *, +, or {x,} with no upper bound.
- Wild card (.)

Type: String

Length Constraints: Minimum length of 1. Maximum length of 300.

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 V2
- AWS SDK for Ruby V3
SlotValueSelectionSetting
Service: Amazon Lex Model Building V2

Contains settings used by Amazon Lex to select a slot value.

Contents

advancedRecognitionSetting

Provides settings that enable advanced recognition settings for slot values.

Type: AdvancedRecognitionSetting (p. 655) object

Required: No

regexFilter

A regular expression used to validate the value of a slot.

Type: SlotValueRegexFilter (p. 769) object

Required: No

resolutionStrategy

Determines the slot resolution strategy that Amazon Lex uses to return slot type values. The field can be set to one of the following values:

- OriginalValue - Returns the value entered by the user, if the user value is similar to the slot value.
- TopResolution - If there is a resolution list for the slot, return the first value in the resolution list as the slot type value. If there is no resolution list, null is returned.

If you don't specify the valueSelectionStrategy, the default is OriginalValue.

Type: String

Valid Values: OriginalValue | TopResolution

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 V2
- AWS SDK for Ruby V3
SSMLMessage
Service: Amazon Lex Model Building V2

Defines a Speech Synthesis Markup Language (SSML) prompt.

Contents

value

The SSML text that defines the prompt.

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 V2
- AWS SDK for Ruby V3
StillWaitingResponseSpecification
Service: Amazon Lex Model Building V2

Defines the messages that Amazon Lex sends to a user to remind them that the bot is waiting for a response.

Contents

allowInterrupt
Indicates that the user can interrupt the response by speaking while the message is being played.

Type: Boolean
Required: No

frequencyInSeconds
How often a message should be sent to the user. Minimum of 1 second, maximum of 5 minutes.

Type: Integer
Valid Range: Minimum value of 1. Maximum value of 300.

Required: Yes

messageGroups
One or more message groups, each containing one or more messages, that define the prompts that Amazon Lex sends to the user.

Type: Array of MessageGroup (p. 736) objects
Array Members: Minimum number of 1 item. Maximum number of 5 items.

Required: Yes

timeoutInSeconds
If Amazon Lex waits longer than this length of time for a response, it will stop sending messages.

Type: Integer

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 V2
- AWS SDK for Ruby V3
TextLogDestination
Service: Amazon Lex Model Building V2

Defines the Amazon CloudWatch Logs destination log group for conversation text logs.

Contents

cloudWatch

Defines the Amazon CloudWatch Logs log group where text and metadata logs are delivered.

Type: CloudWatchLogGroupLogDestination (p. 695) 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 V2
- AWS SDK for Ruby V3
TextLogSetting
Service: Amazon Lex Model Building V2

Defines settings to enable text conversation logs.

Contents

destination

Defines the Amazon CloudWatch Logs destination log group for conversation text logs.

Type: TextLogDestination (p. 773) object

Required: Yes

destination

enabled

Determines whether conversation logs should be stored for an alias.

Type: Boolean

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 V2
- AWS SDK for Ruby V3
TranscriptFilter
Service: Amazon Lex Model Building V2

The object representing the filter that Amazon Lex will use to select the appropriate transcript.

Contents

lexTranscriptFilter

The object representing the filter that Amazon Lex will use to select the appropriate transcript when the transcript format is the Amazon Lex format.

Type: LexTranscriptFilter (p. 734) 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 V2
- AWS SDK for Ruby V3
TranscriptSourceSetting
Service: Amazon Lex Model Building V2

Indicates the setting of the location where the transcript is stored.

Contents

s3BucketTranscriptSource

Indicates the setting of the Amazon S3 bucket where the transcript is stored.

Type: S3BucketTranscriptSource (p. 749) 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 V2
- AWS SDK for Ruby V3
UtteranceAggregationDuration
Service: Amazon Lex Model Building V2

Provides parameters for setting the time window and duration for aggregating utterance data.

Contents

relativeAggregationDuration

The desired time window for aggregating utterances.

Type: RelativeAggregationDuration (p. 746) 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 V2
- AWS SDK for Ruby V3
VoiceSettings
Service: Amazon Lex Model Building V2

Defines settings for using an Amazon Polly voice to communicate with a user.

Contents

engine

Indicates the type of Amazon Polly voice that Amazon Lex should use for voice interaction with the user. For more information, see the engine parameter of the SynthesizeSpeech operation in the Amazon Polly developer guide.

If you do not specify a value, the default is standard.

Type: String

Valid Values: standard | neural

Required: No

voiceId

The identifier of the Amazon Polly voice to use.

Type: String

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 V2
- AWS SDK for Ruby V3
**WaitAndContinueSpecification**  
Service: Amazon Lex Model Building V2

Specifies the prompts that Amazon Lex uses while a bot is waiting for customer input.

**Contents**

**active**

Specifies whether the bot will wait for a user to respond. When this field is false, wait and continue responses for a slot aren't used. If the active field isn't specified, the default is true.

Type: Boolean  
Required: No

**continueResponse**

The response that Amazon Lex sends to indicate that the bot is ready to continue the conversation.

Type: ResponseSpecification (p. 747) object  
Required: Yes

**stillWaitingResponse**

A response that Amazon Lex sends periodically to the user to indicate that the bot is still waiting for input from the user.

Type: StillWaitingResponseSpecification (p. 772) object  
Required: No

**waitingResponse**

The response that Amazon Lex sends to indicate that the bot is waiting for the conversation to continue.

Type: ResponseSpecification (p. 747) 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 V2
- AWS SDK for Ruby V3

**Amazon Lex Runtime V2**

The following data types are supported by Amazon Lex Runtime V2:

- ActiveContext (p. 781)
- ActiveContextTimeToLive (p. 783)
- AudioInputEvent (p. 784)
- AudioResponseEvent (p. 786)
- Button (p. 787)
- ConfidenceScore (p. 788)
- ConfigurationEvent (p. 789)
- DialogAction (p. 791)
- DisconnectionEvent (p. 793)
- DTMFInputEvent (p. 794)
- HeartbeatEvent (p. 795)
- ImageResponseCard (p. 796)
- Intent (p. 798)
- IntentResultEvent (p. 799)
- Interpretation (p. 801)
- Message (p. 802)
- PlaybackCompletionEvent (p. 803)
- PlaybackInterruptionEvent (p. 804)
- RuntimeHintDetails (p. 805)
- RuntimeHints (p. 806)
- RuntimeHintValue (p. 807)
- SentimentResponse (p. 808)
- SentimentScore (p. 809)
- SessionState (p. 810)
- Slot (p. 812)
- StartConversationRequestEventStream (p. 813)
- StartConversationResponseEventStream (p. 815)
- TextInputEvent (p. 818)
- TextResponseEvent (p. 819)
- TranscriptEvent (p. 820)
- Value (p. 821)
ActiveContext
Service: Amazon Lex Runtime V2

Contains information about the contexts that a user is using in a session. You can configure Amazon Lex V2 to set a context when an intent is fulfilled, or you can set a context using the PutSession (p. 623), RecognizeText (p. 628), or RecognizeUtterance (p. 635) operations.

Use a context to indicate to Amazon Lex V2 intents that should be used as follow-up intents. For example, if the active context is order-fulfilled, only intents that have order-fulfilled configured as a trigger are considered for follow up.

Contents

contextAttributes

A list of contexts active for the request. A context can be activated when a previous intent is fulfilled, or by including the context in the request.

If you don’t specify a list of contexts, Amazon Lex V2 will use the current list of contexts for the session. If you specify an empty list, all contexts for the session are cleared.

Type: String to string map

Map Entries: Minimum number of 0 items. Maximum number of 10 items.

Key Length Constraints: Minimum length of 1. Maximum length of 100.


Required: Yes

name

The name of the context.

Type: String

Length Constraints: Minimum length of 1. Maximum length of 100.

Pattern: ^([A-Za-z_\-]+)$

Required: Yes

timeToLive

Indicates the number of turns or seconds that the context is active. Once the time to live expires, the context is no longer returned in a response.

Type: ActiveContextTimeToLive (p. 783) 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 V2
- AWS SDK for Ruby V3
**ActiveContextTimeToLive**

Service: Amazon Lex Runtime V2

The time that a context is active. You can specify the time to live in seconds or in conversation turns.

**Contents**

**timeToLiveInSeconds**

The number of seconds that the context is active. You can specify between 5 and 86400 seconds (24 hours).

Type: Integer


Required: Yes

**turnsToLive**

The number of turns that the context is active. You can specify up to 20 turns. Each request and response from the bot is a turn.

Type: Integer


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 V2
- AWS SDK for Ruby V3
AudioInputEvent
Service: Amazon Lex Runtime V2

Represents a chunk of audio sent from the client application to Amazon Lex V2. The audio is all or part of an utterance from the user.

Amazon Lex V2 accumulates audio chunks until it recognizes a natural pause in speech before processing the input.

Contents

audioChunk

An encoded stream of audio.

Type: Base64-encoded binary data object

Required: No

clientTimestampMillis

A timestamp set by the client of the date and time that the event was sent to Amazon Lex V2.

Type: Long

Required: No

contentType

The encoding used for the audio chunk. You must use 8 KHz PCM 16-bit mono-channel little-endian format. The value of the field should be:

audio/lpcm; sample-rate=8000; sample-size-bits=16; channel-count=1; is-big-endian=false

Type: String

Length Constraints: Minimum length of 1.

Required: Yes

eventId

A unique identifier that your application assigns to the event. You can use this to identify events in logs.

Type: String


Pattern: [0-9a-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 V2
- AWS SDK for Ruby V3
**AudioResponseEvent**
Service: Amazon Lex Runtime V2

An event sent from Amazon Lex V2 to your client application containing audio to play to the user.

**Contents**

**audioChunk**
A chunk of the audio to play.
Type: Base64-encoded binary data object
Required: No

**contentType**
The encoding of the audio chunk. This is the same as the encoding configure in the contentType field of the ConfigurationEvent.
Type: String
Length Constraints: Minimum length of 1.
Required: No

**eventId**
A unique identifier of the event sent by Amazon Lex V2. The identifier is in the form RESPONSE-N, where N is a number starting with one and incremented for each event sent by Amazon Lex V2 in the current session.
Type: String
Pattern: [0-9a-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 V2
- AWS SDK for Ruby V3
Button
Service: Amazon Lex Runtime V2

A button that appears on a response card show to the user.

Contents

text

The text that is displayed on the button.

Type: String


Required: Yes

value

The value returned to Amazon Lex V2 when a user chooses the button.

Type: String


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 V2
- AWS SDK for Ruby V3
ConfidenceScore
Service: Amazon Lex Runtime V2

Provides a score that indicates the confidence that Amazon Lex V2 has that an intent is the one that satisfies the user's intent.

Contents

score

A score that indicates how confident Amazon Lex V2 is that an intent satisfies the user's intent. Ranges between 0.00 and 1.00. Higher scores indicate higher confidence.

Type: Double
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 V2
- AWS SDK for Ruby V3
**ConfigurationEvent**

*Service: Amazon Lex Runtime V2*

The initial event sent from the application to Amazon Lex V2 to configure the conversation, including session and request attributes and the response content type.

**Contents**

- **clientTimestampMillis**
  
  A timestamp set by the client of the date and time that the event was sent to Amazon Lex V2.

  Type: Long

  Required: No

- **disablePlayback**
  
  Determines whether Amazon Lex V2 should send audio responses to the client application.

  Set this field to false when the client is operating in a playback mode where audio responses are played to the user. If the client isn't operating in playback mode, such as a text chat application, set this to true so that Amazon Lex V2 doesn't wait for the prompt to finish playing on the client.

  Type: Boolean

  Required: No

- **eventId**
  
  A unique identifier that your application assigns to the event. You can use this to identify events in logs.

  Type: String


  Pattern: \[0-9a-zA-Z._:-]+

  Required: No

- **requestAttributes**
  
  Request-specific information passed between the client application and Amazon Lex V2.

  The namespace `x-amz-lex:` is reserved for special attributes. Don't create any request attributes for prefix `x-amz-lex:`.

  Type: String to string map

  Key Length Constraints: Minimum length of 1.

  Required: No

- **responseContentType**
  
  The message that Amazon Lex V2 returns in the response can be either text or speech based on the `responseContentType` value.

  - If the value is `text/plain;charset=utf-8`, Amazon Lex V2 returns text in the response.
  - If the value begins with `audio/`, Amazon Lex V2 returns speech in the response. Amazon Lex V2 uses Amazon Polly to generate the speech using the configuration that you specified in the `requestContentType` parameter. For example, if you specify `audio/mpeg` as the value, Amazon Lex V2 returns speech in the MPEG format.
• If the value is `audio/pcm`, the speech returned is `audio/pcm` in 16-bit, little-endian format.
• The following are the accepted values:
  • `audio/mpeg`
  • `audio/ogg`
  • `audio/pcm`
  • `audio/*` (defaults to mpeg)
  • `text/plain; charset=utf-8`

Type: String
Length Constraints: Minimum length of 1.
Required: Yes

**sessionState**

The state of the user's session with Amazon Lex V2.

Type: `SessionState (p. 810)` object

Required: No

**welcomeMessages**

A list of messages to send to the user.

If you set the `welcomeMessage` field, you must also set the `DialogAction` structure's `type` field.

Type: Array of `Message (p. 802)` objects

Array Members: Maximum number of 10 items.

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 V2**
• **AWS SDK for Ruby V3**
DialogAction
Service: Amazon Lex Runtime V2

The next action that Amazon Lex V2 should take.

Contents

slotElicitationStyle

Configures the slot to use spell-by-letter or spell-by-word style. When you use a style on a slot, users can spell out their input to make it clear to your bot.

- Spell by letter - "b" "o" "b"
- Spell by word - "b as in boy" "o as in oscar" "b as in boy"

For more information, see Using spelling to enter slot values.

Type: String

Valid Values: Default | SpellByLetter | SpellByWord

Required: No

slotToElicit

The name of the slot that should be elicited from the user.

Type: String

Length Constraints: Minimum length of 1.

Required: No

type

The next action that the bot should take in its interaction with the user. The possible values are:

- Close - Indicates that there will not be a response from the user. For example, the statement "Your order has been placed" does not require a response.
- ConfirmIntent - The next action is asking the user if the intent is complete and ready to be fulfilled. This is a yes/no question such as "Place the order?"
- Delegate - The next action is determined by Amazon Lex V2.
- ElicitIntent - The next action is to elicit an intent from the user.
- ElicitSlot - The next action is to elicit a slot value from the user.

Type: String

Valid Values: Close | ConfirmIntent | Delegate | ElicitIntent | ElicitSlot | None

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 V2
• AWS SDK for Ruby V3
DisconnectionEvent
Service: Amazon Lex Runtime V2

A notification from the client that it is disconnecting from Amazon Lex V2. Sending a DisconnectionEvent event is optional, but can help identify a conversation in logs.

Contents

clientTimestampMillis

A timestamp set by the client of the date and time that the event was sent to Amazon Lex V2.

Type: Long

Required: No

eventId

A unique identifier that your application assigns to the event. You can use this to identify events in logs.

Type: String


Pattern: [0-9a-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 V2
- AWS SDK for Ruby V3
**DTMFInputEvent**

Service: Amazon Lex Runtime V2

A DTMF character sent from the client application. DTMF characters are typically sent from a phone keypad to represent numbers. For example, you can have Amazon Lex V2 process a credit card number input from a phone.

**Contents**

**clientTimestampMillis**

A timestamp set by the client of the date and time that the event was sent to Amazon Lex V2.

Type: Long

Required: No

**eventId**

A unique identifier that your application assigns to the event. You can use this to identify events in logs.

Type: String


Pattern: `[0-9a-zA-Z._:-]+`

Required: No

**inputCharacter**

The DTMF character that the user pressed. The allowed characters are A - D, 0 - 9, # and *.

Type: String

Length Constraints: Fixed length of 1.

Pattern: `^[A-D0-9#\*]{1}$`

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 V2
- AWS SDK for Ruby V3
HeartbeatEvent
Service: Amazon Lex Runtime V2

Event that Amazon Lex V2 sends to indicate that the stream is still open between the client application and Amazon Lex V2

Contents

eventId

A unique identifier of the event sent by Amazon Lex V2. The identifier is in the form RESPONSE-N, where N is a number starting with one and incremented for each event sent by Amazon Lex V2 in the current session.

Type: String


Pattern: [0-9a-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 V2
- AWS SDK for Ruby V3
ImageResponseCard
Service: Amazon Lex Runtime V2

A card that is shown to the user by a messaging platform. You define the contents of the card, the card is displayed by the platform.

When you use a response card, the response from the user is constrained to the text associated with a button on the card.

Contents

buttons

A list of buttons that should be displayed on the response card. The arrangement of the buttons is determined by the platform that displays the button.

Type: Array of Button (p. 787) objects

Array Members: Minimum number of 0 items. Maximum number of 5 items.

Required: No

imageUrl

The URL of an image to display on the response card. The image URL must be publicly available so that the platform displaying the response card has access to the image.

Type: String


Required: No

subtitle

The subtitle to display on the response card. The format of the subtitle is determined by the platform displaying the response card.

Type: String


Required: No

title

The title to display on the response card. The format of the title is determined by the platform displaying the response card.

Type: String


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 V2
• AWS SDK for Ruby V3
**Intent**
Service: Amazon Lex Runtime V2

The current intent that Amazon Lex V2 is attempting to fulfill.

**Contents**

*confirmationState*

Contains information about whether fulfillment of the intent has been confirmed.

Type: String

Valid Values: Confirmed | Denied | None

Required: No

*name*

The name of the intent.

Type: String

Length Constraints: Minimum length of 1.

Required: Yes

*slots*

A map of all of the slots for the intent. The name of the slot maps to the value of the slot. If a slot has not been filled, the value is null.

Type: String to Slot (p. 812) object map

Key Length Constraints: Minimum length of 1.

Required: No

*state*

Contains fulfillment information for the intent.

Type: String

Valid Values: Failed | Fulfilled | InProgress | ReadyForFulfillment | Waiting | FulfillmentInProgress

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 V2
- AWS SDK for Ruby V3
**IntentResultEvent**
Service: Amazon Lex Runtime V2

Contains the current state of the conversation between the client application and Amazon Lex V2.

**Contents**

**eventId**
A unique identifier of the event sent by Amazon Lex V2. The identifier is in the form `RESPONSE-N`, where N is a number starting with one and incremented for each event sent by Amazon Lex V2 in the current session.

Type: String


Pattern: `[0-9a-zA-Z._:-]+`

Required: No

**inputMode**
Indicates whether the input to the operation was text or speech.

Type: String

Valid Values: Text | Speech | DTMF

Required: No

**interpretations**
A list of intents that Amazon Lex V2 determined might satisfy the user's utterance.

Each interpretation includes the intent, a score that indicates how confident Amazon Lex V2 is that the interpretation is the correct one, and an optional sentiment response that indicates the sentiment expressed in the utterance.

Type: Array of [Interpretation (p. 801)] objects

Array Members: Maximum number of 5 items.

Required: No

**requestAttributes**
The attributes sent in the request.

Type: String to string map

Key Length Constraints: Minimum length of 1.

Required: No

**sessionId**
The identifier of the session in use.

Type: String


Pattern: `[0-9a-zA-Z._:-]+`
Required: No

**sessionState**

The state of the user's session with Amazon Lex V2.

Type: `SessionState (p. 810)` 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 V2
- AWS SDK for Ruby V3
Interpretation
Service: Amazon Lex Runtime V2

An intent that Amazon Lex V2 determined might satisfy the user's utterance. The intents are ordered by the confidence score.

Contents

intent

A list of intents that might satisfy the user's utterance. The intents are ordered by the confidence score.

Type: Intent (p. 798) object

Required: No

nluConfidence

Determines the threshold where Amazon Lex V2 will insert the AMAZON.FallbackIntent, AMAZON.KendraSearchIntent, or both when returning alternative intents in a response. AMAZON.FallbackIntent and AMAZON.KendraSearchIntent are only inserted if they are configured for the bot.

Type: ConfidenceScore (p. 788) object

Required: No

sentimentResponse

The sentiment expressed in an utterance.

When the bot is configured to send utterances to Amazon Comprehend for sentiment analysis, this field contains the result of the analysis.

Type: SentimentResponse (p. 808) 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 V2
- AWS SDK for Ruby V3
Message
Service: Amazon Lex Runtime V2

Container for text that is returned to the customer.

Contents

content

The text of the message.

Type: String


Required: No

contentType

Indicates the type of response.

Type: String

Valid Values: CustomPayload | ImageResponseCard | PlainText | SSML

Required: Yes

imageResponseCard

A card that is shown to the user by a messaging platform. You define the contents of the card, the card is displayed by the platform.

When you use a response card, the response from the user is constrained to the text associated with a button on the card.

Type: ImageResponseCard (p. 796) 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 V2
- AWS SDK for Ruby V3
PlaybackCompletionEvent
Service: Amazon Lex Runtime V2

Event sent from the client application to Amazon Lex V2 to indicate that playback of audio is complete and that Amazon Lex V2 should start processing the user's input.

Contents

clientTimestampMillis

A timestamp set by the client of the date and time that the event was sent to Amazon Lex V2.

Type: Long

Required: No

eventId

A unique identifier that your application assigns to the event. You can use this to identify events in logs.

Type: String


Pattern: [0-9a-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 V2
- AWS SDK for Ruby V3
Playback Interruption Event
Service: Amazon Lex Runtime V2

Event sent from Amazon Lex V2 to indicate to the client application should stop playback of audio. For example, if the client is playing a prompt that asks for the user's telephone number, the user might start to say the phone number before the prompt is complete. Amazon Lex V2 sends this event to the client application to indicate that the user is responding and that Amazon Lex V2 is processing their input.

Contents

causedeByEventId

The identifier of the event that contained the audio, DTMF, or text that caused the interruption.

Type: String


Pattern: [0-9a-zA-Z._:-]+

Required: No

eventId

A unique identifier of the event sent by Amazon Lex V2. The identifier is in the form RESPONSE-N, where N is a number starting with one and incremented for each event sent by Amazon Lex V2 in the current session.

Type: String


Pattern: [0-9a-zA-Z._:-]+

Required: No

eventReason

Indicates the type of user input that Amazon Lex V2 detected.

Type: String

Valid Values: DTMF_START_DETECTED | TEXT_DETECTED | VOICE_START_DETECTED

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 V2
- AWS SDK for Ruby V3
RuntimeHintDetails
Service: Amazon Lex Runtime V2

Provides an array of phrases that should be given preference when resolving values for a slot.

Contents

runtimeHintValues

One or more strings that Amazon Lex V2 should look for in the input to the bot. Each phrase is given preference when deciding on slot values.

Type: Array of RuntimeHintValue (p. 807) objects

Array Members: Minimum number of 1 item. Maximum number of 100 items.

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 V2
- AWS SDK for Ruby V3
RuntimeHints
Service: Amazon Lex Runtime V2

You can provide Amazon Lex V2 with hints to the phrases that a customer is likely to use for a slot. When a slot with hints is resolved, the phrases in the runtime hints are preferred in the resolution. You can provide hints for a maximum of 100 intents. You can provide a maximum of 100 slots.

Before you can use runtime hints with an existing bot, you must first rebuild the bot.

For more information, see Using runtime hints to improve recognition of slot values.

Contents

slotHints

A list of the slots in the intent that should have runtime hints added, and the phrases that should be added for each slot.

The first level of the slotHints map is the name of the intent. The second level is the name of the slot within the intent. For more information, see Using hints to improve accuracy.

The intent name and slot name must exist.

Type: String to string to runtimeHintDetails (p. 805) object map map

Key Length Constraints: Minimum length of 1. Maximum length of 100.

Key Pattern: ^([0-9a-zA-Z][_\-]?)+$  
Key Length Constraints: Minimum length of 1. Maximum length of 100.

Key Pattern: ^([0-9a-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 V2
- AWS SDK for Ruby V3
**RuntimeHintValue**
Service: Amazon Lex Runtime V2

Provides the phrase that Amazon Lex V2 should look for in the user's input to the bot.

**Contents**

**phrase**

The phrase that Amazon Lex V2 should look for in the user's input to the bot.

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 V2
- AWS SDK for Ruby V3
SentimentResponse
Service: Amazon Lex Runtime V2

Provides information about the sentiment expressed in a user's response in a conversation. Sentiments are determined using Amazon Comprehend. Sentiments are only returned if they are enabled for the bot.

For more information, see Determine Sentiment in the Amazon Comprehend developer guide.

Contents

sentiment

The overall sentiment expressed in the user's response. This is the sentiment most likely expressed by the user based on the analysis by Amazon Comprehend.

Type: String

Valid Values: MIXED | NEGATIVE | NEUTRAL | POSITIVE

Required: No

sentimentScore

The individual sentiment responses for the utterance.

Type: SentimentScore (p. 809) 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 V2
- AWS SDK for Ruby V3
SentimentScore
Service: Amazon Lex Runtime V2

The individual sentiment responses for the utterance.

**Contents**

**mixed**

The level of confidence that Amazon Comprehend has in the accuracy of its detection of the **MIXED** sentiment.

Type: Double

Required: No

**negative**

The level of confidence that Amazon Comprehend has in the accuracy of its detection of the **NEGATIVE** sentiment.

Type: Double

Required: No

**neutral**

The level of confidence that Amazon Comprehend has in the accuracy of its detection of the **NEUTRAL** sentiment.

Type: Double

Required: No

**positive**

The level of confidence that Amazon Comprehend has in the accuracy of its detection of the **POSITIVE** sentiment.

Type: Double

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 V2
- AWS SDK for Ruby V3
SessionState
Service: Amazon Lex Runtime V2

The state of the user's session with Amazon Lex V2.

Contents

activeContexts

One or more contexts that indicate to Amazon Lex V2 the context of a request. When a context is active, Amazon Lex V2 considers intents with the matching context as a trigger as the next intent in a session.

Type: Array of ActiveContext (p. 781) objects

Array Members: Minimum number of 0 items. Maximum number of 20 items.

Required: No

dialogAction

The next step that Amazon Lex V2 should take in the conversation with a user.

Type: DialogAction (p. 791) object

Required: No

intent

The active intent that Amazon Lex V2 is processing.

Type: Intent (p. 798) object

Required: No

originatingRequestId

A unique identifier for a specific request.

Type: String

Length Constraints: Minimum length of 1.

Required: No

runtimeHints

Hints for phrases that a customer is likely to use for a slot. Amazon Lex V2 uses the hints to help determine the correct value of a slot.

Type: RuntimeHints (p. 806) object

Required: No

sessionAttributes

Map of key/value pairs representing session-specific context information. It contains application information passed between Amazon Lex V2 and a client application.

Type: String to string map

Key Length Constraints: Minimum length of 1.

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 V2
- AWS SDK for Ruby V3
Slot
Service: Amazon Lex Runtime V2

A value that Amazon Lex V2 uses to fulfill an intent.

Contents

shape

When the shape value is List, it indicates that the values field contains a list of slot values. When the value is Scalar, it indicates that the value field contains a single value.

Type: String

Valid Values: Scalar | List

Required: No

value

The current value of the slot.

Type: Value (p. 821) object

Required: No

values

A list of one or more values that the user provided for the slot. For example, if a for a slot that elicits pizza toppings, the values might be "pepperoni" and "pineapple."

Type: Array of Slot (p. 812) 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 V2
- AWS SDK for Ruby V3
StartConversationRequestEventStream

Service: Amazon Lex Runtime V2

Represents a stream of events between your application and Amazon Lex V2.

Contents

AudioInputEvent

Speech audio sent from your client application to Amazon Lex V2. Audio starts accumulating when Amazon Lex V2 identifies a voice and continues until a natural pause in the speech is found before processing.

Type: AudioInputEvent (p. 784) object

Required: No

ConfigurationEvent

Configuration information sent from your client application to Amazon Lex V2

Type: ConfigurationEvent (p. 789) object

Required: No

DisconnectionEvent

Event sent from the client application to indicate to Amazon Lex V2 that the conversation is over.

Type: DisconnectionEvent (p. 793) object

Required: No

DTMFInputEvent

DTMF information sent to Amazon Lex V2 by your application. Amazon Lex V2 accumulates the DTMF information from when the user sends the first character and ends
• when there's a pause longer that the value configured for the end timeout.
• when there's a digit that is the configured end character.
• when Amazon Lex V2 accumulates characters equal to the maximum DTMF character configuration.

Type: DTMFInputEvent (p. 794) object

Required: No

PlaybackCompletionEvent

Event sent from the client application to Amazon Lex V2 to indicate that it has finished playing audio and that Amazon Lex V2 should start listening for user input.

Type: PlaybackCompletionEvent (p. 803) object

Required: No

TextInputEvent

Text sent from your client application to Amazon Lex V2. Each TextInputEvent is processed individually.

Type: TextInputEvent (p. 818) 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 V2
- AWS SDK for Ruby V3
StartConversationResponseEventStream

Service: Amazon Lex Runtime V2

Represents a stream of events between Amazon Lex V2 and your application.

Contents

AccessDeniedException

Exception thrown when the credentials passed with the request are invalid or expired. Also thrown when the credentials in the request do not have permission to access the StartConversation operation.

Type: Exception
HTTP Status Code: 403
Required: No

AudioResponseEvent

An event sent from Amazon Lex V2 to your client application containing audio to play to the user.

Type: AudioResponseEvent (p. 786) object
Required: No

BadGatewayException

Type: Exception
HTTP Status Code: 502
Required: No

ConflictException

Exception thrown when two clients are using the same AWS account, Amazon Lex V2 bot, and session ID.

Type: Exception
HTTP Status Code: 409
Required: No

DependencyFailedException

Type: Exception
HTTP Status Code: 424
Required: No

HeartbeatEvent

Event that Amazon Lex V2 sends to indicate that the stream is still open between the client application and Amazon Lex V2

Type: HeartbeatEvent (p. 795) object
Required: No

IntentResultEvent

Event sent from Amazon Lex V2 to the client application containing the current state of the conversation between the user and Amazon Lex V2.
Type: `IntentResultEvent (p. 799)` object

Required: No

**InternalServerException**

An error occurred with Amazon Lex V2.

Type: Exception
HTTP Status Code: 500

Required: No

**PlaybackInterruptionEvent**

Event sent from Amazon Lex V2 to indicate to the client application should stop playback of audio. For example, if the client is playing a prompt that asks for the user's telephone number, the user might start to say the phone number before the prompt is complete. Amazon Lex V2 sends this event to the client application to indicate that the user is responding and that Amazon Lex V2 is processing their input.

Type: `PlaybackInterruptionEvent (p. 804)` object

Required: No

**ResourceNotFoundException**

Exception thrown if one of the input parameters points to a resource that does not exist. For example, if the bot ID specified does not exist.

Type: Exception
HTTP Status Code: 404

Required: No

**TextResponseEvent**

The event sent from Amazon Lex V2 to your application with text to present to the user.

Type: `TextResponseEvent (p. 819)` object

Required: No

**ThrottlingException**

Exception thrown when your application exceeds the maximum number of concurrent requests.

Type: Exception
HTTP Status Code: 429

Required: No

**TranscriptEvent**

Event sent from Amazon Lex V2 to your client application that contains a transcript of voice audio.

Type: `TranscriptEvent (p. 820)` object

Required: No

**ValidationException**

Exception thrown when one or more parameters could not be validated. The message contains the name of the field that isn't valid.

Type: Exception
HTTP Status Code: 400

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 V2
- AWS SDK for Ruby V3
**TextInputEvent**

Service: Amazon Lex Runtime V2

The event sent from your client application to Amazon Lex V2 with text input from the user.

**Contents**

**clientTimestampMillis**

A timestamp set by the client of the date and time that the event was sent to Amazon Lex V2.

Type: Long  
Required: No

**eventId**

A unique identifier that your application assigns to the event. You can use this to identify events in logs.

Type: String  
Pattern: [0-9a-zA-Z._:-]+  
Required: No

**text**

The text from the user. Amazon Lex V2 processes this as a complete statement.

Type: String  
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 V2
- AWS SDK for Ruby V3
TextResponseEvent
Service: Amazon Lex Runtime V2

The event sent from Amazon Lex V2 to your application with text to present to the user.

Contents

eventId

A unique identifier of the event sent by Amazon Lex V2. The identifier is in the form `RESPONSE-N`, where N is a number starting with one and incremented for each event sent by Amazon Lex V2 in the current session.

Type: String


Pattern: `[0-9a-zA-Z._:-]+`

Required: No

messages

A list of messages to send to the user. Messages are ordered based on the order that you returned the messages from your Lambda function or the order that the messages are defined in the bot.

Type: Array of Message (p. 802) objects

Array Members: Maximum number of 10 items.

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 V2
- AWS SDK for Ruby V3
TranscriptEvent
Service: Amazon Lex Runtime V2

Event sent from Amazon Lex V2 to your client application that contains a transcript of voice audio.

Contents

eventId
A unique identifier of the event sent by Amazon Lex V2. The identifier is in the form RESPONSE-N, where N is a number starting with one and incremented for each event sent by Amazon Lex V2 in the current session.

Type: String
Pattern: [0-9a-zA-Z._:-]+
Required: No

transcript
The transcript of the voice audio from the user.

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 V2
- AWS SDK for Ruby V3
Value
Service: Amazon Lex Runtime V2

The value of a slot.

Contents

interpretedValue

The value that Amazon Lex V2 determines for the slot. The actual value depends on the setting of the value selection strategy for the bot. You can choose to use the value entered by the user, or you can have Amazon Lex V2 choose the first value in the resolvedValues list.

Type: String
Length Constraints: Minimum length of 1.
Required: Yes

originalValue

The text of the utterance from the user that was entered for the slot.

Type: String
Length Constraints: Minimum length of 1.
Required: No

resolvedValues

A list of additional values that have been recognized for the slot.

Type: Array of strings
Length Constraints: Minimum length of 1.
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 V2
- AWS SDK for Ruby V3

Common Errors

This section lists the errors common to the API actions of all AWS services. For errors specific to an API action for this service, see the topic for that API action.

AccessDeniedException

You do not have sufficient access to perform this action.

HTTP Status Code: 400
IncompleteSignature
The request signature does not conform to AWS standards.
HTTP Status Code: 400

InternalFailure
The request processing has failed because of an unknown error, exception or failure.
HTTP Status Code: 500

InvalidAction
The action or operation requested is invalid. Verify that the action is typed correctly.
HTTP Status Code: 400

InvalidClientTokenId
The X.509 certificate or AWS access key ID provided does not exist in our records.
HTTP Status Code: 403

InvalidParameterCombination
Parameters that must not be used together were used together.
HTTP Status Code: 400

InvalidParameterValue
An invalid or out-of-range value was supplied for the input parameter.
HTTP Status Code: 400

InvalidQueryParameter
The AWS query string is malformed or does not adhere to AWS standards.
HTTP Status Code: 400

MalformedQueryString
The query string contains a syntax error.
HTTP Status Code: 400

MissingAction
The request is missing an action or a required parameter.
HTTP Status Code: 400

MissingAuthenticationToken
The request must contain either a valid (registered) AWS access key ID or X.509 certificate.
HTTP Status Code: 403

MissingParameter
A required parameter for the specified action is not supplied.
HTTP Status Code: 400

NotAuthorized
You do not have permission to perform this action.
OptInRequired

The AWS access key ID needs a subscription for the service.

HTTP Status Code: 403

RequestExpired

The request reached the service more than 15 minutes after the date stamp on the request or more than 15 minutes after the request expiration date (such as for pre-signed URLs), or the date stamp on the request is more than 15 minutes in the future.

HTTP Status Code: 400

ServiceUnavailable

The request has failed due to a temporary failure of the server.

HTTP Status Code: 503

ThrottlingException

The request was denied due to request throttling.

HTTP Status Code: 400

ValidationError

The input fails to satisfy the constraints specified by an AWS service.

HTTP Status Code: 400

Common Parameters

The following list contains the parameters that all actions use for signing Signature Version 4 requests with a query string. Any action-specific parameters are listed in the topic for that action. For more information about Signature Version 4, see Signature Version 4 Signing Process in the Amazon Web Services General Reference.

Action

The action to be performed.

Type: string

Required: Yes

Version

The API version that the request is written for, expressed in the format YYYY-MM-DD.

Type: string

Required: Yes

X-Amz-Algorithm

The hash algorithm that you used to create the request signature.

Condition: Specify this parameter when you include authentication information in a query string instead of in the HTTP authorization header.
Type: string

Valid Values: AWS4-HMAC-SHA256

Required: Conditional

**X-Amz-Credential**

The credential scope value, which is a string that includes your access key, the date, the region you are targeting, the service you are requesting, and a termination string ("aws4_request"). The value is expressed in the following format: access_key/YYYYMMDD/region/service/aws4_request.

For more information, see Task 2: Create a String to Sign for Signature Version 4 in the Amazon Web Services General Reference.

Condition: Specify this parameter when you include authentication information in a query string instead of in the HTTP authorization header.

Type: string

Required: Conditional

**X-Amz-Date**

The date that is used to create the signature. The format must be ISO 8601 basic format (YYYYMMDD'T'HHMMSS'Z'). For example, the following date time is a valid X-Amz-Date value: 20120325T120000Z.

Condition: X-Amz-Date is optional for all requests; it can be used to override the date used for signing requests. If the Date header is specified in the ISO 8601 basic format, X-Amz-Date is not required. When X-Amz-Date is used, it always overrides the value of the Date header. For more information, see Handling Dates in Signature Version 4 in the Amazon Web Services General Reference.

Type: string

Required: Conditional

**X-Amz-Security-Token**

The temporary security token that was obtained through a call to AWS Security Token Service (AWS STS). For a list of services that support temporary security credentials from AWS Security Token Service, go to AWS Services That Work with IAM in the IAM User Guide.

Condition: If you're using temporary security credentials from the AWS Security Token Service, you must include the security token.

Type: string

Required: Conditional

**X-Amz-Signature**

Specifies the hex-encoded signature that was calculated from the string to sign and the derived signing key.

Condition: Specify this parameter when you include authentication information in a query string instead of in the HTTP authorization header.

Type: string

Required: Conditional
X-Amz-SignedHeaders

Specifies all the HTTP headers that were included as part of the canonical request. For more information about specifying signed headers, see Task 1: Create a Canonical Request For Signature Version 4 in the Amazon Web Services General Reference.

Condition: Specify this parameter when you include authentication information in a query string instead of in the HTTP authorization header.

Type: string

Required: Conditional
AWS glossary

For the latest AWS terminology, see the AWS glossary in the AWS General Reference.