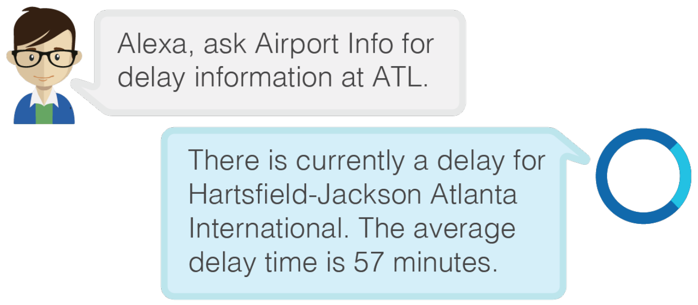
Table of Contents

1

Slots and Slot Types

In this chapter, you will build a skill called Airport Info, that allows you to find out whether there are currently delays at a specified airport. The skill will talk to the Federal Aviation Administration's Airport Information service in order to retrieve the current status of an airport in response to a user's requested airport code. Here is how an interaction with the Airport Info skill will work:

Figure 1.1


Alexa, ask Airport Info for delay information at ATL.
There is currently a delay for Hartsfield-Jackson Atlanta International. The average delay time is 57 minutes.

You will expand your knowledge of the interaction model that you worked with in the previous exercise in this chapter. You will use a feature of the interaction model's utterances definition called *slots*. Slots are an aspect of the interaction model you have not seen yet. They allow you to pass values into an intent handler as a variable in your skill that the user speaks.

Setting up a Local Development Environment

Before you begin building the Airport Info skill you will first set up a local development environment. The local development environment will allow you to author a skill without the need for uploading files to AWS Lambda each time a change is made, enable local debugging, and ease the process of diagnosing any bugs that may occur by allowing you to easily inspect stack traces and logs.

Your service is authored in JavaScript and runs on Node.js. The first step in getting a local development environment working is installing Node.js so that you can run JavaScript on the Node.js platform locally. Use Node Version Manager to install Node.js so that you can easily switch versions of Node.js as new versions become available.

Listing 1.1 installing nvm

```
curl -o- https://raw.githubusercontent.com/creationix/nvm/v0.30.2/install.sh | bash
```

Close and reopen the terminal to ensure it is reloaded and check that nvm installed by running:

Listing 1.2 checking nvm is installed correctly

```
nvm ls
```

This will list the versions of Node.js that have been installed:

Listing 1.3 checking nvm is installed correctly

```
$ nvm ls
       v4.3.2
->     system
```

It is not important if your output from the nvm command looks different. What you are looking for is that the command completed successfully.

Next, install the version of Node.js currently supported on AWS Lambda. This is to ensure that the behavior of your skill locally will closely match the behavior of your skill when it is later deployed to the AWS Lambda environment.

At the time of this writing, the version of Node.js available on AWS Lambda is *v4.3.2*, though this may have changed, since AWS is under active development. You should first check the following url:

```
http://docs.aws.amazon.com/lambda/latest/dg/current-supported-versions.html
```

to find out if the version of Node.js AWS Lambda supports is now a more recent version. Note the most recent version number of Node.js that is available on AWS Lambda and substitute *v4.3.2* for that version if applicable. The text will use *v4.3.2*, since it is currently the most recent version of Node.js available on AWS Lambda.

Install Node.js using the following command:

Listing 1.4 Installing node v4.3.2

```
nvm install v4.3.2
```

After this command completes, set *v4.3.2* as the default version of Node.js the system will use for executing JavaScript using the following command:

Listing 1.5 Setting v4.3.2 to default

```
nvm alias default v4.3.2
```

Adding alexa-app-server

Now that Node.js is set up you will check out an open source project that enables running a skill service locally called *alexa-app-server*. Run the following command:

Listing 1.6 Checking out alexa-app-server

```
git clone https://github.com/matt-kruse/alexa-app-server.git
```

Change to the `alexa-app-server` directory to install the dependencies.

Listing 1.7 Installing Node Package Manager

```
npm install
```

Change to the `examples/apps` directory within the `alexa-app-server` directory. Create a new folder called `airportinfo`. You will be building your skill and its related components within this directory. In the *Testing the Skill Service* section, you'll work more closely with *alexa-app-server* itself when you begin fleshing out the skill service portion of the skill.

Managing Dependencies with Node Package Manager

Your project will require library dependencies to support making the network request. Use the Node Package Manager -- npm -- to add these dependences. npm was installed with Node.js. **`npm`** generates a `package.json` file to manage a list of dependencies your skill service will require. Run the following command within the `airportinfo` directory to create the `package.json` file.

Listing 1.8 Generating the package.json file

```
npm init
```

You will be guided through a series of prompts that will construct a `package.json` file for the project:

```
$ npm init
This utility will walk you through creating a package.json file.
It only covers the most common items, and tries to guess sensible defaults.

See `npm help json` for definitive documentation on these fields
and exactly what they do.

Use `npm install <pkg> --save` afterwards to install a package and
save it as a dependency in the package.json file.

Press ^C at any time to quit.
name: (foobar) airportinfo
version: (1.0.0)
description: an alexa skill for tracking airport information
entry point: (index.js)
test command:
git repository:
keywords:
author: josh skeen
license: (ISC)

About to write to /Users/joshskeen/code/airportinfo/package.json:
{
 "name": "airportinfo",
 "version": "1.0.0",
 "description": "an alexa skill for tracking airport information",
 "main": "index.js",
 "scripts": {
   "test": "echo \"Error: no test specified\" && exit 1"
 },
 "author": "josh skeen",
 "license": "ISC"
}
Is this ok? (yes) yes
```

Now that the `package.json` has been generated, you will add the dependencies the skill will require.

Listing 1.9 Installing dependencies

```
npm install --save alexa-app request-promise lodash
```

The **`--save`** flag tells **`npm`** to add the dependencies to the `package.json` file to keep track of them for installation in different environments.

Here is a brief description of each library you installed. You will learn more about each library as you work with them throughout the chapter.

alexa-app - simplified skill service development with support for running within alexa-app-server

request-promise - simplifies the task of requesting data from a REST endpoint. Handles asynchronous requests as a promise

lodash - a convenience library with many useful utility functions for JavaScript development

Building FAADataHelper

You will begin by creating a helper module for making requests to the FAA Airport Information service,

```
http://services.faa.gov/docs/services/airport/
```

Begin by adding the following to a new file within `airportinfo` called `faa_data_helper.js`.

Listing 1.10 Creating airportinfo/faa_data_helper.js

```
'use strict';
var _ = require('lodash');
var requestPromise = require('request-promise');
var ENDPOINT = 'http://services.faa.gov/airport/status/';

function FAADataHelper() {
}

module.exports = FAADataHelper;
```

You required the libraries needed to build the **`FAADataHelper`** module with. You also defined the service endpoint you will make GET requests to. Last, you defined **`FAADataHelper`**, which you will build upon as you flesh out functionality.

Requesting Airport Info

Next, define a method for requesting the status of an airport using an airport code. For example, requesting

```
http://services.faa.gov/airport/status/CLT?format=application/json
```

should return airport information for Charlotte Douglas International Airport similar to the following:

Listing 1.11 Airport information Service Response

```
{
  'delay': 'true',
  'IATA': 'CLT',
  'state': 'North Carolina',
  'name': 'Charlotte Douglas International',
  'weather': {
    'visibility': 8.00,
    'Weather': 'Light Rain',
    'meta': {
      'credit': 'NOAA\'s National Weather Service',
      'updated': '12:52 PM Local',
      'url': 'http://weather.gov/'
    },
    'temp': '63.0 F (17.2 C)',
    'wind': 'Southwest at 18.4mph'
  },
  'ICAO': 'KCLT',
  'city': 'Charlotte',
  'status': {
    'reason': 'WX:Low Ceilings',
    'closureBegin': '',
    'endTime': '',
    'minDelay': '46 minutes',
    'avgDelay': '',
    'maxDelay': '1 hour',
    'closureEnd': '',
    'trend': 'Increasing',
    'type': 'Arrival'
  }
};
```

Define the **`getAirportStatus(airportCode)`** as follows:

Listing 1.12 Defining getAirportStatus(airportCode)

```
...
function FAADataHelper() {
}

FAADataHelper.prototype.getAirportStatus = function(airportCode) {
  var options = {
    method: 'GET',
    uri: ENDPOINT + airportCode,
    json: true
  };
  return requestPromise(options);
};
module.exports = FAADataHelper;
```

This code instructs the *request-promise* library that you imported to make a GET request to the FAA Endpoint and return JSON data as a *promise*. For further reading on the concept of promises, check out:

```
https://www.promisejs.org/
```

A promise allows the asynchronous nature of a network request - making a request and waiting on the results to come back and then performing work - to be easily handled by abstracting the need for having complex callback logic or function nesting, which can become tough to reason about. Instead, the promise object returned by *request-promise* will provide a **`.then()`** method where you will define what happens when the request completes.

Testing the Request

Test that the **`FAADataHelper`** object works as expected from Node.js. First, type **`node`** on the commandline. This will start the interactive Node.js shell.

Listing 1.13 Testing the getAirportStatus method

```
$ node
> var FAADataHelper = require('./faa_data_helper.js'); var faaHelper = new FAADataHelper();
undefined
> faaHelper.getAirportStatus('CLT').then(console.log);
Promise {
  _bitField: 0,
  _fulfillmentHandler0: undefined,
  _rejectionHandler0: undefined,
  _progressHandler0: undefined,
  _promise0: undefined,
  _receiver0: undefined,
  _settledValue: undefined }
> { delay: 'true',
  IATA: 'CLT',
  state: 'North Carolina',
  name: 'Charlotte Douglas International',
  weather:
   { visibility: 10,
     weather: 'Overcast',
     meta:
      { credit: 'NOAA\'s National Weather Service',
        updated: '1:52 PM Local',
        url: 'http://weather.gov/' },
     temp: '70.0 F (21.1 C)',
     wind: 'South at 18.4mph' },
  ICAO: 'KCLT',
  city: 'Charlotte',
  status:
   { reason: 'WX:Low Ceilings',
     closureBegin: '',
     endTime: '',
     minDelay: '15 minutes',
     avgDelay: '',
     maxDelay: '29 minutes',
     closureEnd: '',
     trend: 'Decreasing',
     type: 'Arrival' }
}
```

You should see data logged out from the **`console.log`** method you passed into the **`then()`** method. Because *request-promise* returns a promise object, the **`then()`** method was available.

Formatting the Status Response

Next, you will add a method to **`FAADataHelper`** that formats the response from the webservice to be spoken by Alexa called **`formatAirportStatus(airportStatusObject)`**. This method will accept the `AirportStatusObject` you retrieved from the FAA, and return a string the service will send to Alexa to speak back. The `AirportStatusObject` contains delay information, average delay time, airport name, and more. You will pull the information from the `AirportStatusObject` and build a formatted message for Alexa to speak from it.

Listing 1.14 Adding the formatAirportStatus method

```
FAADataHelper.prototype.getAirportStatus = function(airportCode) {
  var options = {
    method: 'GET',
    uri: ENDPOINT + airportCode,
    json: true
  };
  return requestPromise(options);
};
FAADataHelper.prototype.formatAirportStatus = function(aiportStatusObject) {
  if (aiportStatusObject.delay === 'true') {
    var template = _.template('There is currently a delay for ${airport}. ' +
      'The average delay time is ${delay_time}.');
    return template({
      airport: aiportStatusObject.name,
      delay_time: aiportStatusObject.status.avgDelay
    });
  } else {
    //no delay
    var template = _.template('There is currently no delay at ${airport}.');
    return _.template({ airport: aiportStatusObject.name });
  }
};

module.exports = FAADataHelper;
```

Defining the Skill Service

Now that you have implemented a way to receive the data your skill will share with the user, you can begin defining the skill service. First, create a file called `index.js` in the `airportinfo` directory. The `index.js` file serves as an entry point into the service when requests are made from the skill interface to the skill service. Add the following to `index.js`:

Listing 1.15 Creating airportinfo/index.js

```
'use strict';
module.change_code = 1;
var Alexa = require('alexa-app');
var skill = new Alexa.app('airportinfo');
var FAADataHelper = require('./faa_data_helper');
module.exports = skill;
```

Here, you imported the **`FAADataHelper`** you wrote and the **`alexa-app`** module that you installed earlier.

Notice the *alexa-app* library that was required - this library provides several conveniences for implementing a skill service you will leverage to implement the service. It also makes the skill service runnable locally, which you will see soon. The `module.change_code = 1;` declaration enables live reloading of the skill service when running it locally as changes are made.

Listing 1.16 Responding to the onLaunch Event

Next, you will define an **`onLaunch`** event handler that will allow the service to correctly respond to onLaunch events from the skill interface. The **`onLaunch`** event will be triggered if a user uses the phrase "Alexa, open Airport Info", "Alexa, launch AirportInfo", or "Alexa, start AirportInfo".

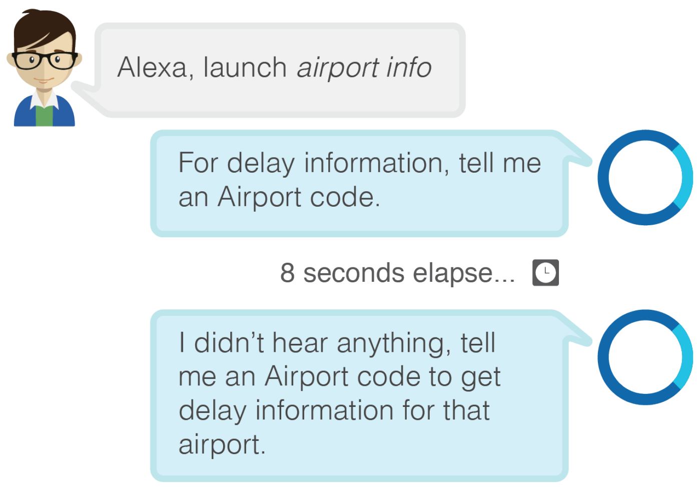
If you have followed the previous exercise for setting up the Greeter skill, this may seem a bit different. This is because of the helper functions provided by the `alexa-app` library, which uses it's own design patterns that slightly differ from Amazon's implementation of `AlexaSkill.js` class.

```
'use strict';
module.change_code = 1;
var _ = require('lodash');
var Alexa = require('alexa-app');
var skill = new Alexa.app('airportinfo');
var FAADataHelper = require('./faa_data_helper');
var reprompt = 'I didn\'t hear an airport code, tell me an Airport code to get delay '
    + 'information for that airport.';
skill.launch(function(request, response) {
  var prompt = 'For delay information, tell me an Airport code.';
  response.say(prompt).reprompt(reprompt).shouldEndSession(false);
});
module.exports = skill;
```

Here you defined the **`onLaunch`** event handler by specifying a function the **`launch()`** function expects as an argument. In this case, if a user launches our skill from the Echo, the launch event will be sent to the skill service and the launch definition will be used to determine how Alexa will behave. As a result of this definition, Alexa will say "For delay information, tell me an airport code" upon launch.

Also, notice you have specified a **`reprompt`** as part of the response. A reprompt defines what happens when nothing is heard in response to the last statement. The interaction would look like this:

Figure 1.2 An interaction with reprompts


Alexa, launch *airport info*
For delay information, tell me an Airport code.
8 seconds elapse...
I didn’t hear anything, tell me an Airport code to get delay information for that airport.

Defining an Intent Handler

Next, you will define the intent handler for a user's requested airport status. As you recall, an intent handler responds to an intent request. An intent request is an indication of a particular task our user would like to perform.

Listing 1.17 Adding an airportInfoIntent handler

```
...
skill.launch(function(request, response) {
  var prompt = 'For delay information, tell me an Airport code.';
  response.say(prompt).reprompt(prompt).shouldEndSession(false);
});
skill.intent('airportInfoIntent', {
    'slots': {
      'AirportCode': 'FAACODES'
    },
    'utterances': ['{|flight|airport} {|delay|status} {|info} {|for} {-|AirportCode}']
  },
  function(request, response) {
        return false;
  }
);

module.exports = skill;
```

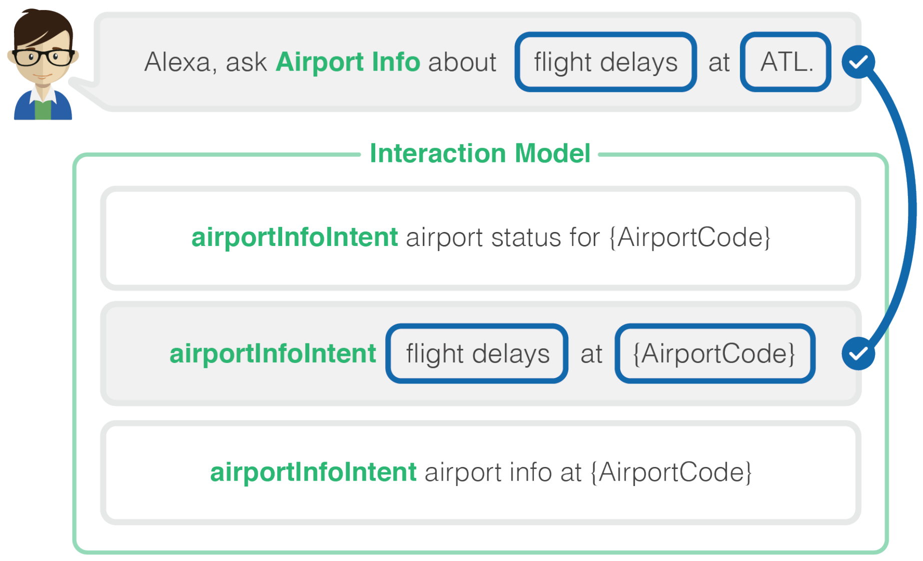
First, you declared that the skill can handle an intent called "airportInfoIntent". This will be triggered on the server by a request from the skill interface every time the utterance that corresponds to it is matched by a phrase the user has spoken.

Next, you provided a *slot type* definition for the intent in the 'slots' definition list. A slot in an Alexa skill is a container for user provided input our program requires. Think of a slot as defining a dictionary that maps a key onto a value. Here, you made a slot that defined an `AirportCode` key that will have values of type `FAACODES`. You will provide the values `AirportCode` can possibly take in the Alexa skill console.

Defining Utterances

An utterance is a phrase a user could say when interacting with your skill, for example "ask airport info about flight delays at ATL". You declared the sample utterances that will be used by the skill interface to match the intent to the user's spoken words.

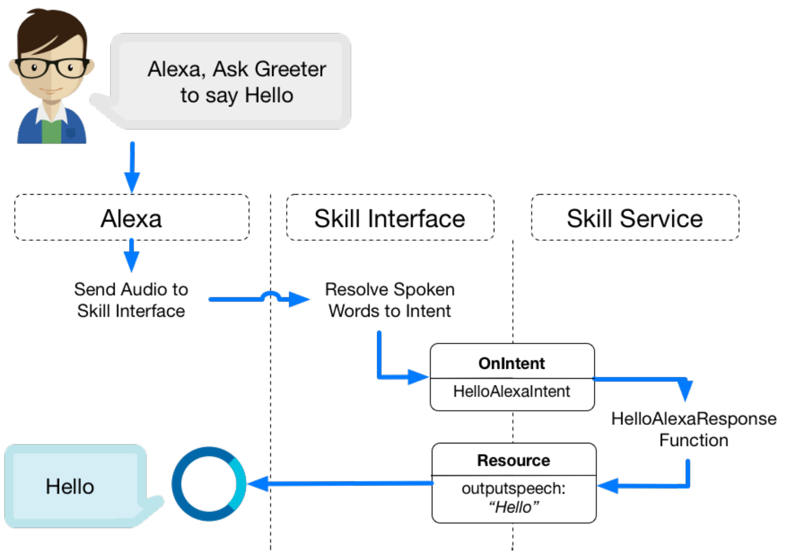
Figure 1.3 Sample Utterances Resolve Spoken Words to Intents


Alexa, ask Airport Info about flight delays at ATL.
Interaction Model
airportInfoIntent airport status for {AirportCode}
airportInfoIntent flight delays at {AirportCode}
airportInfoIntent airport info at {AirportCode}

One clarification about how the skill interface resolves spoken utterances to an intent is that it is not a "one to one" matching operation. The list of sample utterances does not necessarily need to contain all of the possible variations of how the user may phrase his request. Behind the scenes, the list of sample utterances is used to "train" a Natural Language Understanding model that resolves spoken utterances to the Intent and slots.

Once this resolution occurs, the skill interface sends the intent and additional info for the slots to the skill service. The service can then determine what alexa should say, and returns it to the skill interface. Finally, the skill interface forwards this on to the Echo device. To recap, remember the diagram from the following chapter:

Figure 1.4 Recap: Lifecycle of an Alexa Skill Request


Alexa, Ask Greeter to say Hello
Alexa
Skill Interface
Skill Service
Send Audio to Skill Interface
Resolve Spoken Words to Intent
OnIntent
HelloAlexaIntent
HelloAlexaResponse Function
Resource
outputspeech: "Hello"
Hello

The *alexa-app* module allows generating a list of utterances by providing a shorthand notation for creating a list of the expanded form. In the following expression, the pipe character indicates a word or group of words are optional or can be substituted for an alternate within the same set of curly braces: `'{|flight|airport} {|delays|status} {|info} {|for} {-|AirportCode}'` This expression will be evaluated as:

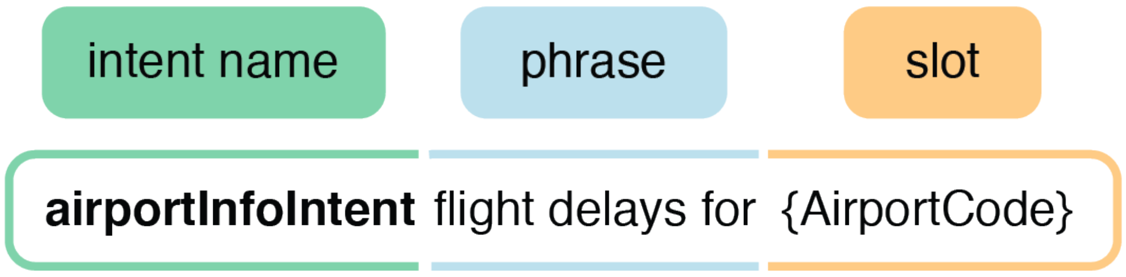
```
airportInfoIntent  {AirportCode}
airportInfoIntent flight {-|AirportCode}
airportInfoIntent airport {-|AirportCode}
airportInfoIntent  delays {-|AirportCode}
airportInfoIntent flight delays {-|AirportCode}
airportInfoIntent airport delays {-|AirportCode}
airportInfoIntent  status {-|AirportCode}
airportInfoIntent flight status {-|AirportCode}
airportInfoIntent airport status {-|AirportCode}
airportInfoIntent  info {-|AirportCode}
airportInfoIntent flight info {-|AirportCode}
airportInfoIntent airport info {-|AirportCode}
airportInfoIntent  delays info {-|AirportCode}
airportInfoIntent flight delays info {-|AirportCode}
airportInfoIntent airport delays info {-|AirportCode}
airportInfoIntent  status info {-|AirportCode}
airportInfoIntent flight status info {-|AirportCode}
airportInfoIntent airport status info {-|AirportCode}
airportInfoIntent  for {-|AirportCode}
airportInfoIntent flight for {-|AirportCode}
airportInfoIntent airport for {-|AirportCode}
airportInfoIntent  delays for {-|AirportCode}
airportInfoIntent flight delays for {-|AirportCode}
airportInfoIntent airport delays for {-|AirportCode}
airportInfoIntent  status for {-|AirportCode}
airportInfoIntent flight status for {-|AirportCode}
airportInfoIntent airport status for {-|AirportCode}
airportInfoIntent  info for {-|AirportCode}
airportInfoIntent flight info for {-|AirportCode}
airportInfoIntent airport info for {-|AirportCode}
airportInfoIntent  delays info for {-|AirportCode}
airportInfoIntent flight delays info for {-|AirportCode}
airportInfoIntent airport delays info for {-|AirportCode}
airportInfoIntent  status info for {-|AirportCode}
airportInfoIntent flight status info for {-|AirportCode}
airportInfoIntent airport status info for {-|AirportCode}
```

Accepted user utterances with this utterance definition would be "Alexa, ask airport info for airport status info for ATL", "Alexa, ask airport info about delays for ATL", "Alexa, ask airport info about flight delays for ATL" for example.

alexa-app will automatically generate all of the permutations for this expression and is a convenient way to manage the list of utterances. You will continue to use the *alexa-app* module throughout the remainder of the course. A note about the `alexa-app` syntax for defining sample utterances: to specify a custom slot type to be emitted in the sample utterances output, the syntax '{-|SlotName}' is used, where 'SlotName' is the slot you would like to use in the sample utterance. The final output to be loaded into the interaction model will equal 'airportInfoIntent airport status info for {SlotName}'.

Notice the "{AirportCode}" portion of each sample utterance. This defines where in the user utterance the slot you defined earlier should be expected. When the skill interface resolves the user's words to a particular intent, if a slot is defined it will also map the words spoken to within the `AirportCode` slot. The following diagram shows the parts of the sample utterance you have specified:

Figure 1.5 Parts of an Example Utterance



Using the Slot Value

Now that you have defined a slot and determined how a user's phrase maps to an intent, you will put the intent and slot to use. The slot will be available on the request object that is passed from the skill interface to the Node.js service. It will be accessed by the name you defined it by in the 'slots' portion of your intent handler definition. In this case, the FAA Code the user would like to query for is available via '`AirportCode`'. The next step is to retrieve the `AirportCode` slot value and pass it in to **`FAADataHelper`**'s **`getAirportStatus(airportcode)`** method. Change the `index.js` file in the following way:

Listing 1.18 Retrieving the Slot Value

```
...
var reprompt = 'I didn\'t hear an airport code, tell me an airport code to get delay'
    + 'information for that airport.';

skill.launch(function(request, response) {
  var prompt = 'For delay information, tell me an airport code.';
  response.say(prompt).reprompt(reprompt).shouldEndSession(false);
});
skill.intent('airportInfoIntent', {
    'slots': {
      'AirportCode': 'FAACODES'
    },
    'utterances': ['{|flight|airport} {|delay|status} {|info} {|for} {-|AirportCode}']
  },
  function(request, response) {
      var airportCode = request.slot('AirportCode');
      var faaDataHelper = new FAADataHelper();
      faaDataHelper.getAirportStatus(airportCode).then(function(airportStatus) {
        console.log(airportStatus);
        response.say(faaDataHelper.formatAirportStatus(airportStatus)).send();
      });
      return false;
  }
);
module.exports = skill;
```

The line `var airportCode = request.slot('AirportCode');` pulls the value the user spoke into the intent so that it can be used to retrieve the appropriate airport code.

Keep in mind that the data the skill service is acting upon is simply a specially formatted JSON payload from the skill interface. The data that was received by your skill service resembles the following JSON structure:

Listing 1.19 It's Just JSON

```
"request": {
    "type": "IntentRequest",
    "requestId": "EdwRequestId.xxxx",
    "intent": {
      "name": "airportInfoIntent",
      "slots": {
        "AirportCode": {
          "name": "AirportCode",
          "value": "ATL"
        }
      }
    },
    "locale": "en-US"
  }
```

As you can see, the slot information and intent information is sent as a JSON formatted payload from the skill interface once it is resolved. The `alexa-app` library makes it more convenient to work with this information in a systematic way.

Handling Edge Cases

Your service will now work in most cases, but there are two additional cases you need to address. The first is if the FAA Airport Service is not working correctly or doesn't understand the code a user provided. In this case the server will return a 404 error. Fortunately, the promise object returned from **`FAADataHelper`** has a built in mechanism for dealing with such a failure called **`catch(function(error){})`** that will be called when these cases occur.

Add the following **`catch(function(error){})`** definition to the method above:

Listing 1.20 Handling a Failed Response

```
...
faaDataHelper.getAirportStatus(airportCode).then(function(airportStatus) {
      console.log(airportStatus);
      response.say(faaDataHelper.formatAirportStatus(airportStatus)).send();
}).catch(function(err) {
    console.log(err.statusCode);
    var prompt = 'I didn\'t have data for an airport code of ' + airportCode;
    response.say(prompt).reprompt(reprompt).shouldEndSession(false).send();
});
...
```

The next case you will address is when the airport code was not heard or submitted by Echo. This can happen if the user misspoke while they were speaking to Echo. Add the following to the skill server code:

Listing 1.21 Gracefully recovering from Bad Input

```
'use strict';
module.change_code = 1;
var Alexa = require('alexa-app');
var skill = new Alexa.app('airportinfo');
var FAADataHelper = require('./faa_data_helper');
var _ = require('lodash');
...

var airportCode = request.slot('AirportCode');
if (_.isEmpty(airportCode)) {
  var prompt = 'I didn\'t hear an airport code. Tell me an airport code.';
  response.say(prompt).reprompt(reprompt).shouldEndSession(false);
  return true;
} else {
    var faaDataHelper = new FAADataHelper();
    faaDataHelper.getAirportStatus(airportCode).then(function(airportStatus) {
          console.log(airportStatus);
          response.say(faaDataHelper.formatAirportStatus(airportStatus)).send();
    }).catch(function(err) {
        console.log(err.statusCode);
        var prompt = 'I didn\'t have data for an airport code of ' + airportCode;
        response.say(prompt).reprompt(reprompt).shouldEndSession(false).send();
    });
    return false;
}
...
```

Here you handle the second case - an empty value for the airport code. The *lodash* module you imported provides a number of JavaScript utilities, including a method called **`_.isEmpty`** , which you use to determine if the AirportCode slot had a value or was empty. If it was empty, you reprompt to instruct the user to try again.

Testing the Skill Service

Now that you have implemented the skill service, you will test it by starting the **`alexa-app-server`**, which will run the skill service you have built locally. Change to the `examples/` directory where you should see a `server.js` file, and run the command:

Listing 1.22 Starting alexa-app-server

```
node server
```

You should see output similar to the following:

```
$ node server
Serving static content from: public_html
Loading server-side modules from: server
```

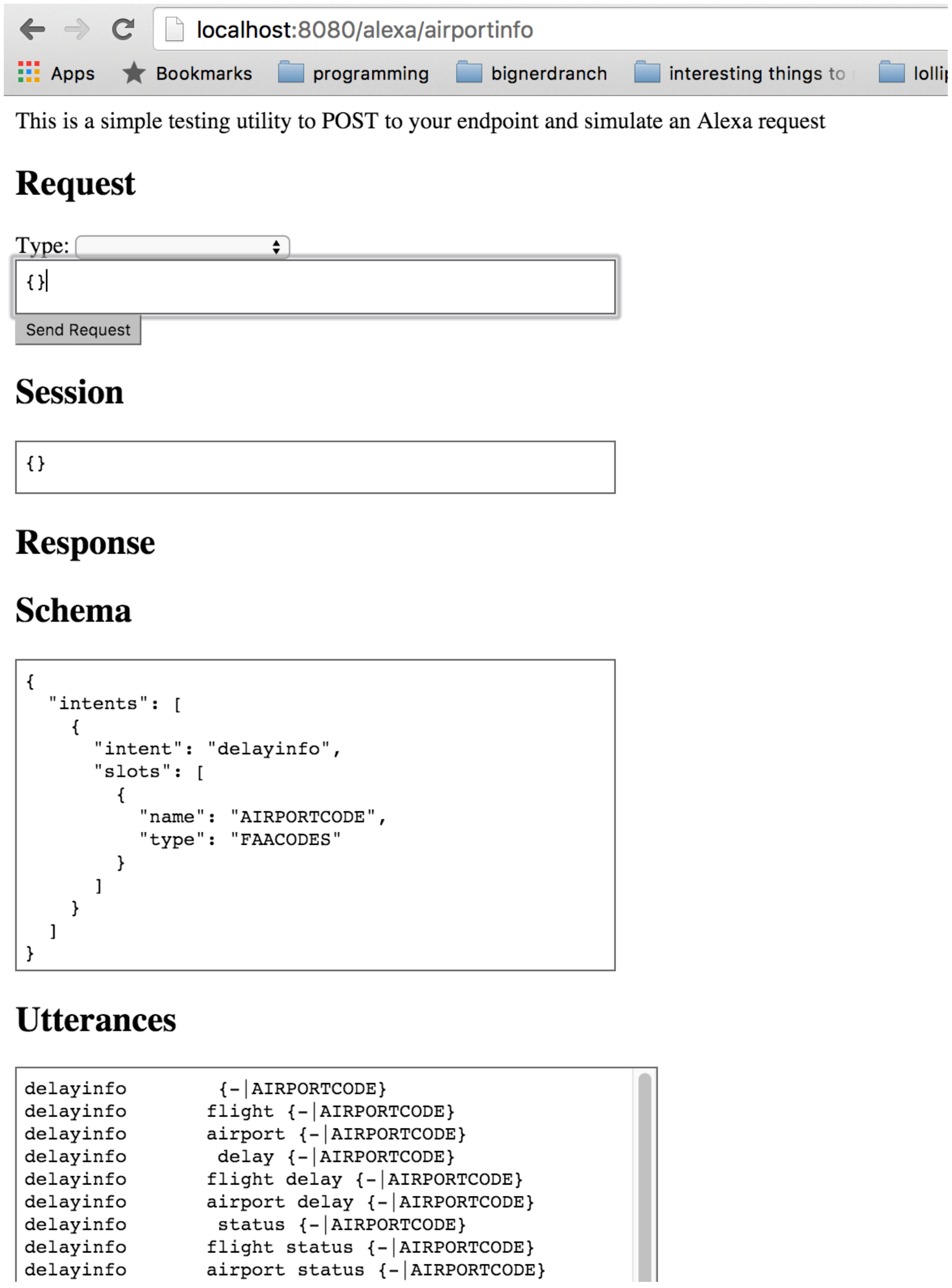
```
   Loaded /Users/joshskeen/code/alexa-app-server/examples/server/login.js
Loading apps from: apps
   Loaded app [airportinfo] at endpoint: /alexa/airportinfo
Listening on HTTP port 8080
```

Now that you have started the server, visit the following url in your web browser:

```
  http://localhost:8080/alexa/airportinfo
```

You should see a web page that includes the intent schema, slot type, and utterances you defined earlier.

Figure 1.6 Alexa App Server Web Interface



This is a simple testing utility to POST to your endpoint and simulate an Alexa request

Request

Type:

```
{}
```

Session

```
{}
```

Response

Schema

```
{
  "intents": [
    {
      "intent": "delayinfo",
      "slots": [
        {
          "name": "AIRPORTCODE",
          "type": "FAACODES"
        }
      ]
    }
  ]
}
```

Utterances

```
delayinfo        {-|AIRPORTCODE}
delayinfo       flight {-|AIRPORTCODE}
delayinfo       airport {-|AIRPORTCODE}
delayinfo        delay {-|AIRPORTCODE}
delayinfo       flight delay {-|AIRPORTCODE}
delayinfo       airport delay {-|AIRPORTCODE}
delayinfo        status {-|AIRPORTCODE}
delayinfo       flight status {-|AIRPORTCODE}
delayinfo       airport status {-|AIRPORTCODE}
```

First, test that the launch intent handler behaves as expected. In the "Type" dropdown, select "Launch Request" and click Send Request. Check that the "Response" window contains text similar to the following:

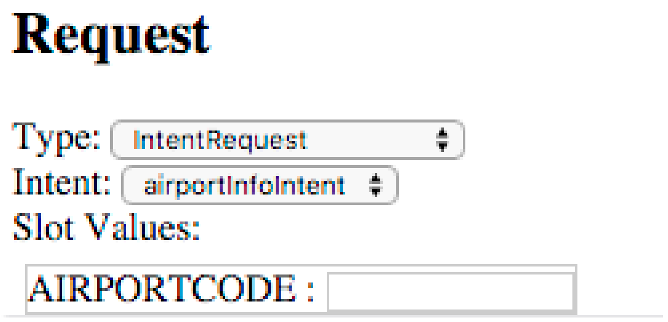
```
{
  "version": "1.0",
  "sessionAttributes": {},
  "response": {
    "shouldEndSession": false,
    "outputSpeech": {
      "type": "SSML",
      "ssml": "<speak>For delay information, tell me an airport code.</speak>"
    },
    "reprompt": {
      "outputSpeech": {
        "type": "SSML",
        "ssml": "<speak>I didn't hear an airport code, tell me an Airport code to get delay information for t
      }
    }
  },
  "dummy": "text"
}
```

Intent Requests

Next, you will test the behavior of the skill service when it is sent an intent request from the skill interface.

The first case you will address is when no `AirportCode` is submitted to the intent handler. Select IntentRequest from the "Type" dropdown and airportInfoIntent from the "Intent" dropdown. leave the AirportCode field under Slot Values empty, and press "Send Request".

Figure 1.7 Testing an Empty AirportCode Slot



Request

Type: IntentRequest

Intent: airportInfoIntent

Slot Values:

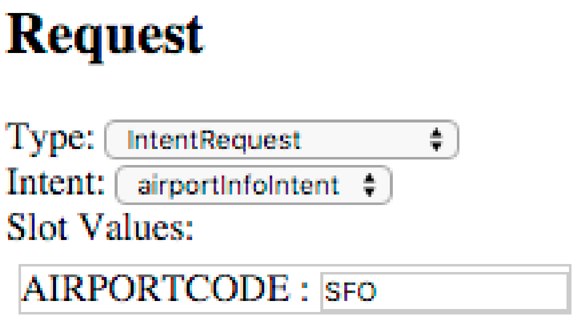
AIRPORTCODE :

In the response section of the web page, you should see the following output:

```
{
  "version": "1.0",
  "sessionAttributes": {},
  "response": {
    "shouldEndSession": false,
    "outputSpeech": {
      "type": "SSML",
      "ssml": "<speak>I didn't hear an airport code. Tell me an airport code.</speak>"
    },
    "reprompt": {
      "outputSpeech": {
        "type": "SSML",
        "ssml": "<speak>I didn't hear an airport code, tell me an Airport code to get delay information for t
      }
    }
  },
  "dummy": "text"
}
```

Test an `airportInfoIntent` with a valid `AirportCode` responds with the airport status information. Enter 'SFO' for the AirportCode field.

Figure 1.8 Testing a valid AirportCode



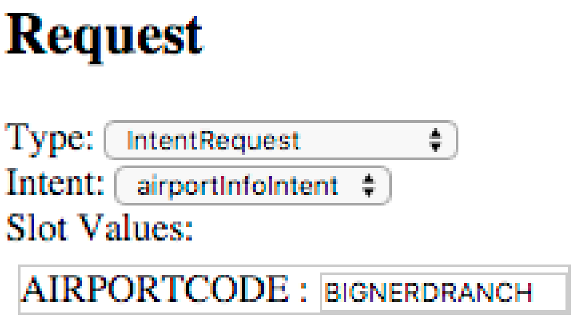
Type: IntentRequest
Intent: airportInfoIntent
Slot Values:
AIRPORTCODE : SFO

You should get a response from the skill service indicating the airport status for San Francisco International Airport:

```
{
  "version": "1.0",
  "sessionAttributes": {},
  "response": {
    "shouldEndSession": true,
    "outputSpeech": {
      "type": "SSML",
      "ssml": "<speak>There is currently no delay at San Francisco International.</speak>"
    }
  },
  "dummy": "text"
}
```

Test an `airportInfoIntent` with an invalid `AirportCode` responds with the airport status information. Enter 'BIGNERDRANCH' for the `AirportCode` field.

Figure 1.9 Testing an invalid AirportCode



Type: IntentRequest
Intent: airportInfoIntent
Slot Values:
AIRPORTCODE : BIGNERDRANCH

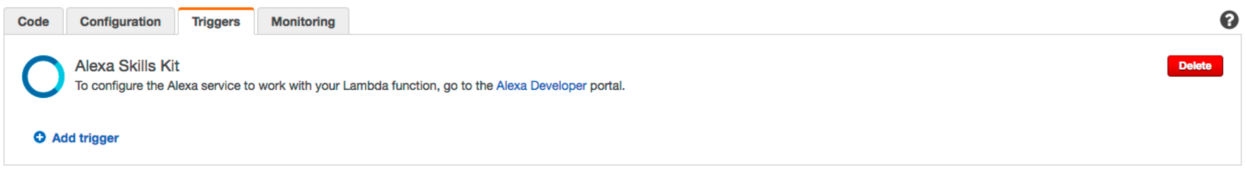
You should get a response from the skill service indicating there was no data for an airport code of BIGNERDRANCH:

```
{
  "version": "1.0",
  "sessionAttributes": {},
  "response": {
    "shouldEndSession": false,
    "outputSpeech": {
      "type": "SSML",
      "ssml": "<speak>I didn't have data for an airport code of BIGNERDRANCH</speak>"
    },
    "reprompt": {
      "outputSpeech": {
        "type": "SSML",
        "ssml": "<speak>I didn't hear an airport code, tell me an Airport code to get delay information for
      }
    }
  },
  "dummy": "text"
```

Deploying the Skill

Now that you have tested the skill service locally, it is time to deploy it and set up a skill configuration so that it can be used on a real Echo. Compress all files in the `airportinfo` directory including the `node_modules` directory and create a new AWS Lambda instance using the same steps from the previous exercise. Ensure that you enable Alexa Skills Kit in the Trigger section of the AWS Lambda page.

Figure 1.10 Enabling the Trigger on AWS Lambda



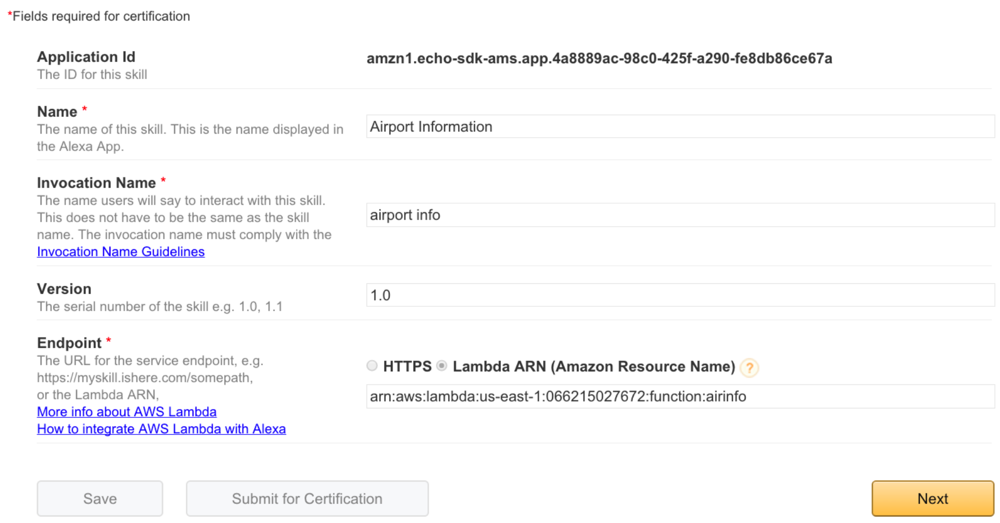
Alexa Skills Kit
To configure the Alexa service to work with your Lambda function, go to the Alexa Developer portal.

Copy the ARN from the Lambda instance, and create a new skill configuration in the Skill Configuration console at

```
https://developer.amazon.com/edw/home.html#/skills/list
```

For the invocation name, enter "airport info".

Figure 1.11 Registering the Skill Information



*Fields required for certification

Application Id The ID for this skill	amzn1.echo-sdk-ams.app.4a8889ac-98c0-425f-a290-fe8db86ce67a
Name * The name of this skill. This is the name displayed in the Alexa App.	Airport Information
Invocation Name * The name users will say to interact with this skill. This does not have to be the same as the skill name. The invocation name must comply with the Invocation Name Guidelines	airport info
Version The serial number of the skill e.g. 1.0, 1.1	1.0
Endpoint * The URL for the service endpoint, e.g. https://myskill.ishere.com/somepath, or the Lambda ARN, More info about AWS Lambda How to integrate AWS Lambda with Alexa	HTTPS [✓ Lambda ARN (Amazon Resource Name)] arn:aws:lambda:us-east-1:066215027672:function:airinfo

Advance to the "interaction model" page. You will now copy the Intent Schema and Sample Utterances information from your local skill server to the relevant fields. Refer to

```
http://localhost:8080/alexa/airportinfo
```

and copy the Intent Schema field into the Amazon Skill Configuration Console's Intent Schema field. Copy the Utterances field into the Amazon Skill Configuration Console's Utterances field.

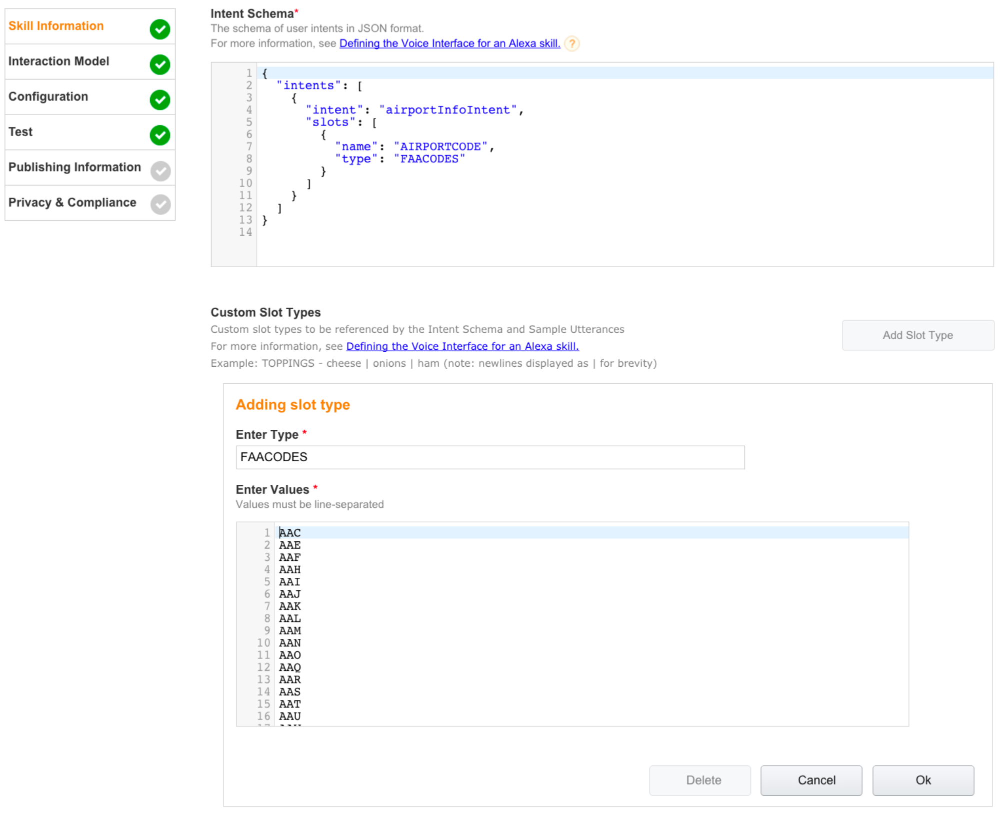
Defining the Custom Slot Type

Your skill also makes use of a custom slot type called `FAACODES`, which must be registered in the skill interface to work correctly. You must provide a set of example data the Slot Type will use to improve the chances of matching a phrase to the set of data. The example data you will use is a list list of FAA Airport Codes. Copy the list here at

```
https://raw.githubusercontent.com/bignerdranch/alexa-airportinfo/master/resources/FAACODES.txt
```

Under the Custom Slot Types area, click Add Slot Type. Enter `FAACODES` for "Enter Type", and for "Enter Values", paste the list of `FAACODES` you downloaded.

Figure 1.12 Adding the Custom Slot Type



Intent Schema*
The schema of user intents in JSON format.
For more information, see Defining the Voice Interface for an Alexa skill.

```
{
  "intents": [
    {
      "intent": "airportInfoIntent",
      "slots": [
        {
          "name": "AIRPORTCODE",
          "type": "FAACODES"
        }
      ]
    }
  ]
}
```

Custom Slot Types
Custom slot types to be referenced by the Intent Schema and Sample Utterances
For more information, see Defining the Voice Interface for an Alexa skill.
Example: TOPPINGS - cheese | onions | ham (note: newlines displayed as | for brevity)

Adding slot type

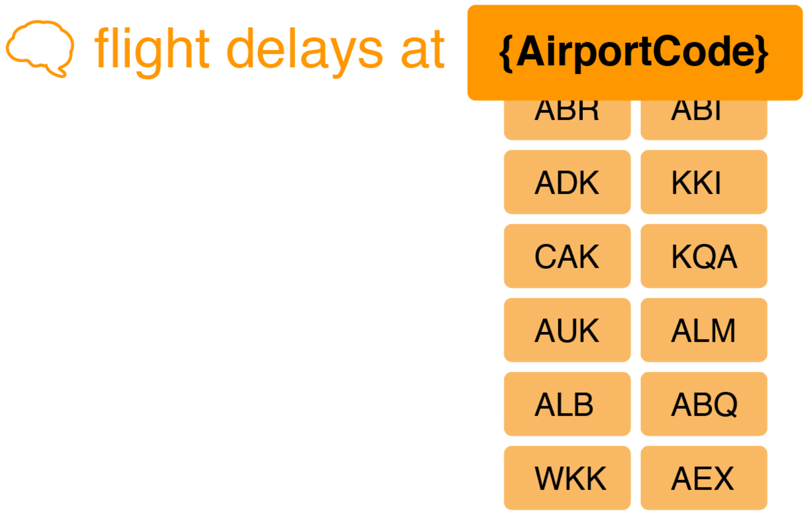
Enter Type *
FAACODES

Enter Values *
Values must be line-separated

```
AAC
AAE
AAF
AAH
AAI
AAJ
AAK
AAL
AAM
AAN
AAO
AAQ
AAR
AAS
AAT
AAU
```

The custom slot type defines the list of acceptable three letter FAACODES the skill interface can resolve a user's spoken words to. You added the AirportCode slot type to the intent schema and used the slot in the sample utterances, so the values from the custom slot type could be used to resolve the user's utterance to an AirportCode.

Figure 1.13 Making Use of the FAACODE Custom Slot Type

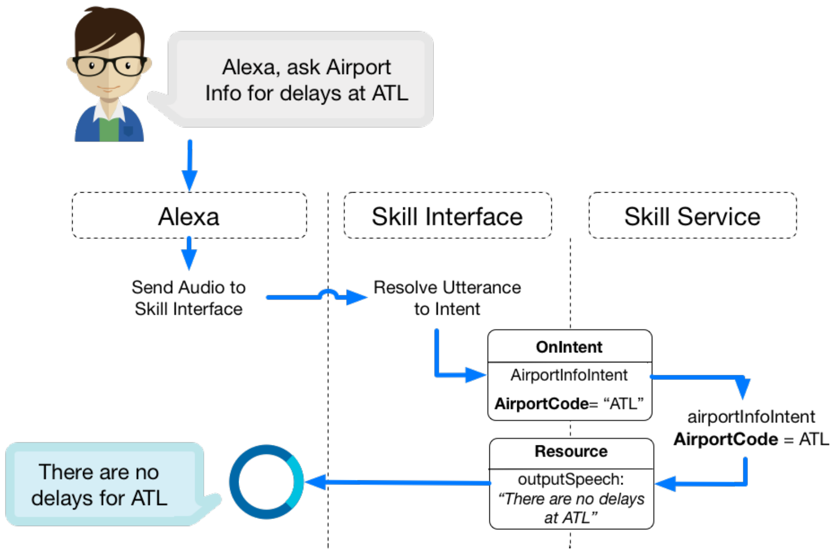


Congratulations, you have successfully developed and deployed the Airport Info Skill to the Developer Portal and AWS Lambda. You may now test the skill in the Developer Portal Test page or on an Alexa-enabled device.

Understanding Airport Info

Examine the following interaction diagram for the Airport Info skill.

Figure 1.14 Airport Info Diagram

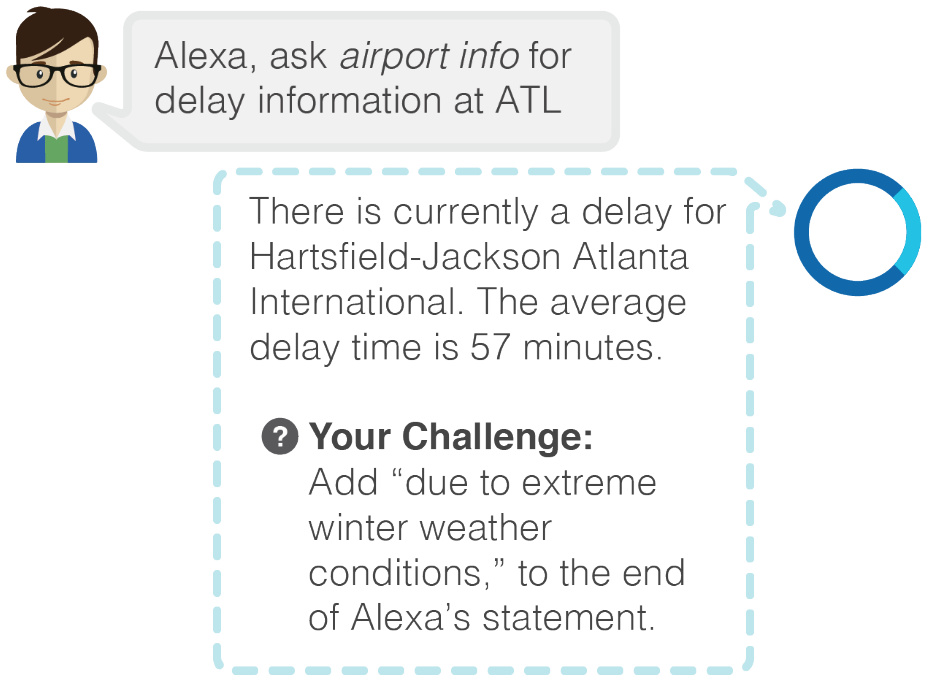

Alexa, ask Airport Info for delays at ATL
Alexa
Skill Interface
Skill Service
Send Audio to Skill Interface
Resolve Utterance to Intent
OnIntent
AirportInfoIntent
AirportCode= "ATL"
airportInfoIntent
AirportCode = ATL
Resource
outputSpeech: "There are no delays at ATL"
There are no delays for ATL

This diagram will appear similar to the Greeter skill, with a key difference. On the skill interface, the variable "AirportCode" was defined and sent to the skill service as part of the intent resolution the skill interface provided. Because you defined the custom slot type "FAACODES" you were able to update the skill interface's interaction model with an intent schema that defined a new slot. The slot definition you created on the intent schema mapped "FAACODES" to "AirportCode", which you then made use of in your sample utterances. The skill interface resolved the user's words to an airport code because of the custom slot type definition, intent schema, and sample utterances in the interaction model. As you can see in the diagram, the skill interface passes the resolved AirportCode to the skill service as part of the intent, so that the skill service can make use of it in it's business logic.

Silver Challenge: Adding a Delay Reason

Users would like to know why the delay at the airport they specified occured. Add the delay reason information to Alexa's response, which should have been included as part of the response from the service.

Figure 1.15


Alexa, ask *airport info* for delay information at ATL
There is currently a delay for Hartsfield-Jackson Atlanta International. The average delay time is 57 minutes.
Your Challenge:
Add "due to extreme winter weather conditions," to the end of Alexa's statement.

Gold Challenge: Weather Information

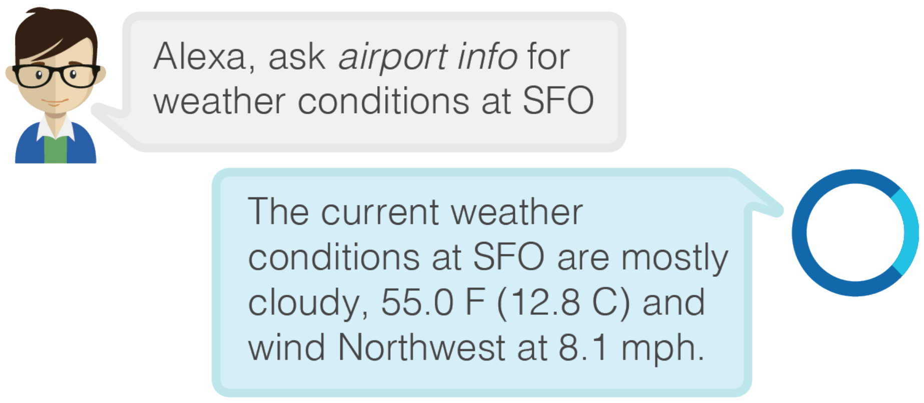
Users would like to know what the weather is like at the airport they have specified. If you inspect the FAA server response this information is included as part of the data:

Listing 1.23 Weather Information

```
"weather": {
   "visibility": 10,
   "weather": "A Few Clouds",
   "meta": {
      "credit": "NOAA's National Weather Service",
      "updated": "6:56 AM Local",
      "url": "http://weather.gov/"
   },
   "temp": "58.0 F (14.4 C)",
   "wind": "West at 4.6mph"
}
```

Use the "weather" portion of the response from the FAA service to create a new intent response for your skill. Weather information should include the weather conditions, temperature, and wind speed. Add a new intent called "airportWeatherIntent" to handle this request. The intent should respond to utterances matching "weather conditions at {AirportCode}".

Figure 1.16


Alexa, ask *airport info* for weather conditions at SFO
The current weather conditions at SFO are mostly cloudy, 55.0 F (12.8 C) and wind Northwest at 8.1 mph.