

Table of Contents

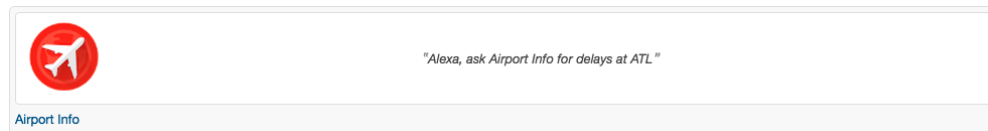
- 1. Certification, Testing, and SSML 1
 - Skill Approval and the Submission Process 1
 - Approval Guidelines and Common Rejection Reasons 2
 - Testing 3
 - SSML 6
 - Conclusion 7
 - Silver Challenge: Testing Airport Info 8
 - Gold Challenge: Playing a Sound using SSML 8

Certification, Testing, and SSML

Skill Approval and the Submission Process


Completing the Skill Submission Process allows your skill to be used on any alexa-enabled device after it has passed Certification.

Figure 1.1 An Approved Skill in the Alexa App Skills Tab



When publishing a skill, the Publishing Information and Privacy & Compliance steps in the skill interface must be fully completed to move forward with the skill approval process. The information configured on the Publishing Information screen will be shown on the installation card for the skill in the alexa app skills tab and used in aiding users when searching for your skill.

Figure 1.2 The Publishing Information Screen

Interaction Model	✓	Short Skill Description * A quick introductory description, which will be shown in the Alexa App in the main list of skills, along with the first example phrase you enter below. Maximum characters: 160	Airport Info allows you to get flight delay information and weather conditions at an Airport you specify by a given Airport code.
Configuration	✓		
Test	✓		
Publishing Information	✓	Full Skill Description * Explanation of the skill's benefits, what it does, how it works, how the user gets started, and any prerequisites, such as an account with your company or particular hardware. Use a conversational tone and correct grammar and punctuation. This description is shown to users in the Alexa App, on the skill's detail card.	Airport Info allows you to get flight delay information and weather conditions at an Airport you specify by it's code. Before leaving for the airport, try asking Airport Info about the conditions to expect before arriving for your flight. Airport Info uses the FAA Airport Status API to provide the information you will be given.
Privacy & Compliance	✓		
		Example Phrases * Important: Many developers fail certification due to this step so please read carefully. Provide three phrases from your Sample Utterances, with any slots filled in with a valid value. These are displayed on the detail card in the Alexa App and should teach users how to interact with the skill. Include the wake word and your invocation name in the first phrase.	<input type="text" value="Alexa, ask airport info for airport status information at ATL"/> <input type="text" value="Alexa, ask airport info for flight delay info at ATL"/> <input type="text" value="Alexa, ask airport info for delay info at ATL"/>
		Category * The general area of functionality of this skill.	<input type="text" value="Travel"/>
		Keywords Search terms used to increase the discoverability of your skill. Use a comma or white space to separate your terms.	<input type="text" value="airport, information, faa, travel, weather, conditions"/>
		Images	
		Small Icon * 108 x 108px PNG(with transparency) or JPG. This is displayed in the Alexa App.	

You should provide an icon, category, keywords, and short and full description for your skill. Also, three example phrases should be provided that show users how to interact with the skill. The example phrases should substitute any slots used in the intent with appropriate values. The example should also include the wake word and invocation name.

It is important that the example phrase be based upon an actual example utterance that can be found in the skill's interaction model. If it is not, Amazon will reject your skill during the approval process.

Next, the privacy and compliance details need to be provided to begin the approval process. Here you indicate whether the skill allows purchases or collects personal data. You also must provide a link to the privacy policy for your skill. A sample privacy policy can be found at <https://www.bbb.org/dallas/for-businesses/bbb-sample-privacy-policy1/> for an idea of how to create one.

Figure 1.3 Privacy and Compliance Screen

The screenshot shows the 'Privacy & Compliance' section of the Amazon Developer console for a skill named 'Airport Info'. On the left, a sidebar lists various configuration sections: Skill Information, Interaction Model, Configuration, Test, Publishing Information, and Privacy & Compliance, each with a green checkmark. The main content area is titled 'Privacy' and contains several fields and checkboxes. It asks 'Does this skill allow users to make purchases or spend real money?' with radio buttons for 'Yes' and 'No'. It also asks 'Does this Alexa skill collect users' personal information?' with radio buttons for 'Yes' and 'No'. Below these are text input fields for 'Privacy Policy URL' and 'Terms of Use URL'. At the bottom, there is a 'Compliance' section with a checkbox for 'Export Compliance' and a text area for a certification statement. A 'Save' button is on the left, and a 'Submit for Certification' button is on the right.

Once you have completed these fields, click the Submit for Certification button. Amazon responds within 5 - 7 days about the status of the approval process. Amazon will indicate whether the skill is approved or rejected. If rejected, Amazon will explain for what reasons so that you can correct the issues and resubmit.

Approval Guidelines and Common Rejection Reasons

With the Amazon skill approval process, there are guidelines that should be kept in mind when developing a skill. A skill should contain no profanity or obscene content. A skill cannot target children under the age of 13 as an audience. A skill must also adhere to the voice user experience guidelines that Amazon requires a skill implement.

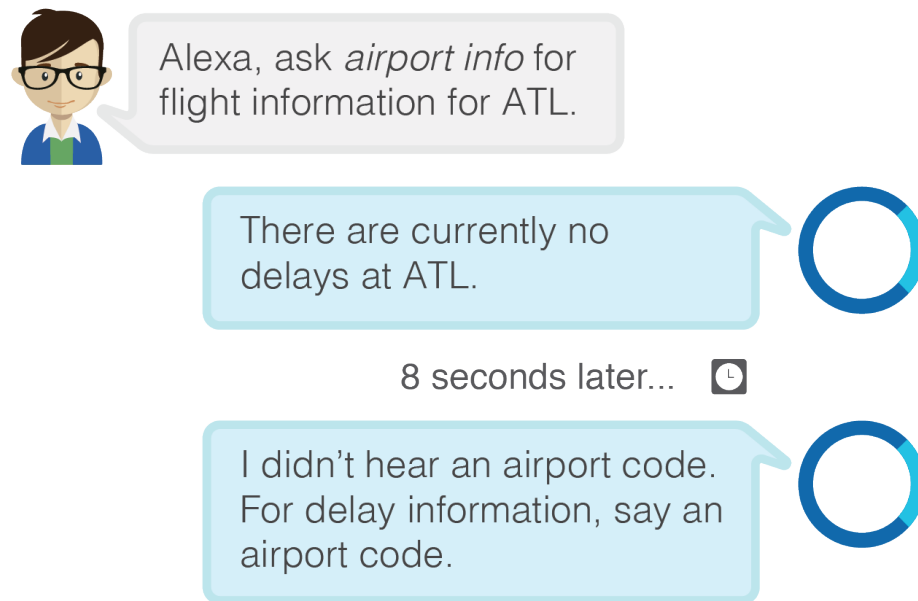
Figure 1.4 A Skill Must Follow Compliance Guidelines to be Approved



A common mistake is to not handle the required built-in intents. These required built-in intents are the `AMAZON.HelpIntent`, `AMAZON.CancelIntent`, and `AMAZON.StopIntent`. If you do not handle these in the skill service, the skill will be rejected because the skill will be difficult to use intuitively.

Recall that a built-in intent must be specified in the intent schema, as with any intent. The term built-in refers to the fact that the example utterances that resolve the user's spoken words to the intent and name of intent are pre-defined by Amazon. Built-in does not mean that you automatically gain certain intent handlers or behavior - you must implement handlers for these intents as with any other intent.

Figure 1.5 An Example of Incorrect Stream Management



Another reason for a skill being rejected is due to incorrect "stream management". Stream management is the act of closing or continuing the interaction with a user. The stream management for a skill must be done in a way that is considered intuitive from a user's perspective. In the example above, re-prompting after answering a user's question would be considered incorrect stream management and would result in a rejected submission.

A skill could also be rejected if it is hosted on a server other than AWS Lambda and the platform does not meet security requirements. The platform must implement SSL, respond on port 443, and the certificate used to sign requests must be from an Amazon-approved authority. For the list of Amazon-approved authorities, see <https://wiki.mozilla.org/CA:IncludedCAs>.

Testing

Listing 1.1 Mocha and Chai Tests for the FAADDataHelper

```
→ faa-info x moc
  FAADDataHelper
    #getAirportStatus
      with an invalid airport code
        ✓ returns invalid airport code (314ms)
      with a valid airport code
        ✓ returns airport code (276ms)
    success - received airport info for SF0
      ✓ returns airport code (276ms)
    #formatAirportStatus
      with a status containing no delay
        ✓ formats the status as expected
      with a status containing a delay
        ✓ formats the status as expected
```

You may have wondered how to go about building unit tests to ensure your skill service works correctly as your skill grows more complex. For larger skills, a test suite will be an invaluable aid to producing code that behaves predictably and is maintainable. On Node.js, there are many solutions for building such tests. For the following examples you will be seeing the mocha and chai test framework and assertion libraries. Mocha and chai are popularly used for writing unit tests for Node.js. If you have ever done Ruby on Rails or Java development, mocha and chai are very similar to RSpec or jUnit and AssertJ. If you haven't done Ruby on Rails or Java development, don't worry - the mocha and chai style are very "plain english" and easy to adopt.

Figure 1.6 Mocha and Chai are Popular Test Libraries for Node.js Development



As an example, you will see how to write unit tests for **FAADDataHelper**. First, you install mocha, chai, and an extension to chai that lets you easily test asynchronous methods. You install the test libraries in your project directory, in this case, /faa-info.

Listing 1.2 Installing the Test Libraries

```
$ npm install --save mocha chai chai-as-promised
```

All of the tests should be added within a new folder in your project called /test, which you create. Within /test you typically will add a new file to match each of the classes you would like to test. The example will show testing **FAADDataHelper**, so you will add a new file called test_faa_data_helper.js.

Next, you import the libraries that are needed to write the tests - the class under test, and the chai assertion library. You also declare the name of the test, using the method **describe**. Typically, you will add a new description for each method you want.

Listing 1.3 Setting up the Test

```
'use strict';
var chai = require('chai');
var expect = chai.expect;
var FAADDataHelper = require('../faa_data_helper');
describe('#getAirportStatus', function() {
  //test goes here!
});
```

Listing 1.4 Setting up the Test

```
'use strict';
var chai = require('chai');
var chaiAsPromised = require('chai-as-promised');
chai.use(chaiAsPromised);

var expect = chai.expect;
var FAADDataHelper = require('../faa_data_helper');
describe('FAADDataHelper', function() {
  var subject = new FAADDataHelper();
  var airport_code;
  //tests go here!
});
```

Now that you have described the name of the test (the class you want to test), you next describe the methods that should be tested. These should be all of the publicly visible methods a class offers. The first method, **getAirportStatus(airportCode)** should be tested. There are two situations - an invalid `airportCode` is passed to the method, or a valid one is passed to the method. These are referred to as "contexts", because they define the situation or context a method or group of methods have in common. Add a description for the `getAirportStatus` method test and the two contexts:

Listing 1.5 Testing the getAirportStatus Method

```
'use strict';
var chai = require('chai');
var chaiAsPromised = require('chai-as-promised');
chai.use(chaiAsPromised);

var expect = chai.expect;
var FAADDataHelper = require('../faa_data_helper');

describe('FAADDataHelper', function() {
  var subject = new FAADDataHelper();
  var airport_code;
  describe('#getAirportStatus', function() {
    context('with an invalid airport code', function() {
      //assertions go here
    });
    context('with a valid airport code', function() {
      //assertions go here
    });
  });
});
```

Now that you have defined the contexts for the **getAirportStatus(airportCode)** method, you can make assertions about what you can expect the method to do. With an invalid `airportCode`, the method should raise an error. With a valid airport code, the result from the FAA server should be returned, including a matching IATA code. You can assert that it raises an error and returns the expected code using the following `chai` syntax:

Listing 1.6 Testing the getAirportStatus Method

```
'use strict';
var chai = require('chai');
var chaiAsPromised = require('chai-as-promised');
chai.use(chaiAsPromised);

var expect = chai.expect;
var FAADataHelper = require('../faa_data_helper');

describe('FAADataHelper', function() {
  var subject = new FAADataHelper();
  var airport_code;
  describe('#getAirportStatus', function() {
    context('with an invalid airport code', function() {
      it('returns invalid airport code', function() {
        airport_code = 'PUNKYBREWSTER';
        return expect(subject.requestAirportStatus(airport_code)).to.be.rejectedWith(Error);
      });
    });
    context('with a valid airport code', function() {
      it('returns airport code', function() {
        airport_code = 'SFO';
        var value = subject.requestAirportStatus(airport_code).then(function(obj) {
          return obj.IATA;
        });
        return expect(value).to.eventually.eq(airport_code);
      });
    });
  });
});
```

Here you check that calling the method with particular arguments results in the behavior you expected. You can now run the test by calling mocha from the command line within the `faa-info` directory:

Listing 1.7 Running the Test

```
$ mocha

  FaaDataHelper
    #getAirportStatus
      with an invalid airport code
        ✓ returns invalid airport code (1276ms)
      with a valid airport code
        ✓ returns airport code (252ms)
  success - received airport info for SFO
    ✓ returns airport code (252ms)

  2 passing (2s)
```

Notice the test output indicates that the assertions match the behavior of the class. To see the rest of the **FAADataHelper** test, check out the test directory in

<https://github.com/bignerdranch/alexa-airportinfo>

To learn more about testing a skill with mocha and chai, visit

<https://www.bignerdranch.com/blog/developing-alexa-skills-locally-with-nodejs-setting-up-your-local-environment/>.

SSML

SSML, or *Speech Synthesis Markup Language*, is a markup syntax that can be used to customize how Alexa will pronounce sounds. SSML also supports requesting mp3 audio files to be played by Alexa. When you build an SSML-formatted `outputSpeech` command for Alexa, you configure the response from the skill service to include the following:

Listing 1.8 An SSML Formatted Speech Command

```
"outputSpeech": {
  "type": "ssml",
  "ssml": "<speak> you say <phoneme alphabet='ipa' ph='tə'meɪtəʊ'>tomato</phoneme>,
  i say <phoneme alphabet='ipa' ph='tə'məʊtəʊ'>tomato</phoneme>! </speak>"
}
```

The outputSpeech element should include a type attribute of SSML, and an ssml element with the additional SSML tags you would like to embed. The first step in using the SSML features.

SSML extends Alexa to support a number of additional capabilities. First, an SSML outputSpeech element's ssml attribute can contain <audio> elements that point to an mp3 file. Mp3 is supported only, and must be encoded to 16k bitrate resolution.

Listing 1.9 Playing an audio file with SSML

```
"outputSpeech": {
  "type": "SSML",
  "ssml": "<speak>
  <audio src='https://s3.amazonaws.com/ask-storage/tidePooler/0ceanWaves.mp3' />
  </speak>"
}
```

Additionally, the mp3 file must be hosted on an https endpoint. To encode a file to the correct format, you can use ffmpeg to correctly transcode the file. If you have **brew**, you may install ffmpeg via the following command :

Listing 1.10 Installing ffmpeg

```
brew install ffmpeg
```

Use the following invocation to encode a file to an mp3 file Alexa Skills Kit will support the playback of:

Listing 1.11 Encoding an audio file to an ASK supported format

```
ffmpeg -y -i name_of_file.wav -ar 16000 -ab 48k -codec:a libmp3lame -ac 1 name_of_file.mp3
```

Another useful element when writing an SSML-flavored outputSpeech is the <phoneme> element. The phoneme element accepts a ph element which can be a series of either ipa (international phonetic alphabet) or x-sampa (The Extended Speech Assessment Methods Phonetic Alphabet) notation characters. For example:

Listing 1.12 An SSML Formatted Speech Command

```
"outputSpeech": {
  "type": "SSML",
  "ssml": "<speak> you say <phoneme alphabet='ipa' ph='tə'meɪtəʊ'>tomato</phoneme>,
  i say <phoneme alphabet='ipa' ph='tə'məʊtəʊ'>tomato</phoneme>! </speak>"
}
```

The above listing would pronounce tomato in subtly different ways. Using phonemes allows you to fine tune pronunciation of words. For further reading on IPA and X-Sampa please visit :

<https://developer.amazon.com/public/solutions/alexa/alexa-skills-kit/docs/speech-synthesis-markup-language-ssml-reference>

Conclusion

Now that you have completed the course you've gained a solid understanding of the capabilities of the Alexa Skills Kit platform. You're well on your way to building your own great skills for Alexa-enabled devices everywhere!

Should you ship a skill to the Alexa app skills tab, the authors of this course would love to hear about it! Contact them at developingalexaskills@gmail.com. We can't wait to see what you build!

Silver Challenge: Testing Airport Info

To take the next step in understanding unit testing an alexa skill, explore the `test/` directory within the Chapter 6 solutions repository found here:

<https://github.com/bignerdranch/amazon-alexa-course/tree/master/exercises/faa-info>

Using what you have learned in the chapter, run the test suite for the project. Are there additional tests you could write to expand the test coverage for Airport Info?

Gold Challenge: Playing a Sound using SSML

Encode, upload, and play back an mp3 as a SSML `<audio>` message to Alexa. This "1-up" sound can be used if you are looking for a free sound to use:

http://themushroomkingdom.net/sounds/wav/smb/smb_1-up.wav.

It will need to be encoded with `ffmpeg` using the encoding settings from the chapter.

Hints: For https storage for the file, use an AWS S3 bucket. To format an SSML message from alexa-app, you can use the SSML node library:

<https://www.npmjs.com/package/ssml>.

For an extra hint, you can take a look at the SSML example in the chapter 6 solutions repository:

https://github.com/bignerdranch/developing-alexa-skills-solutions/tree/master/6_certificationTesting/beatmaster