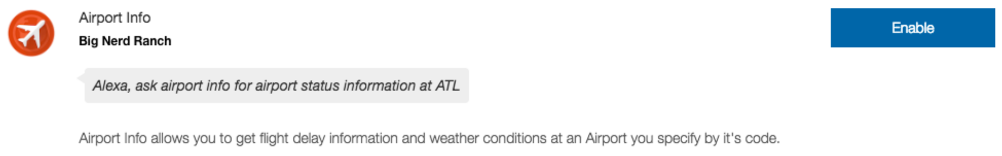
Table of Contents

1
Certification and Testing

Skill Approval and the Submission Process

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

Figure 1.1 An Approved Skill in the Alexa App Skills Tab

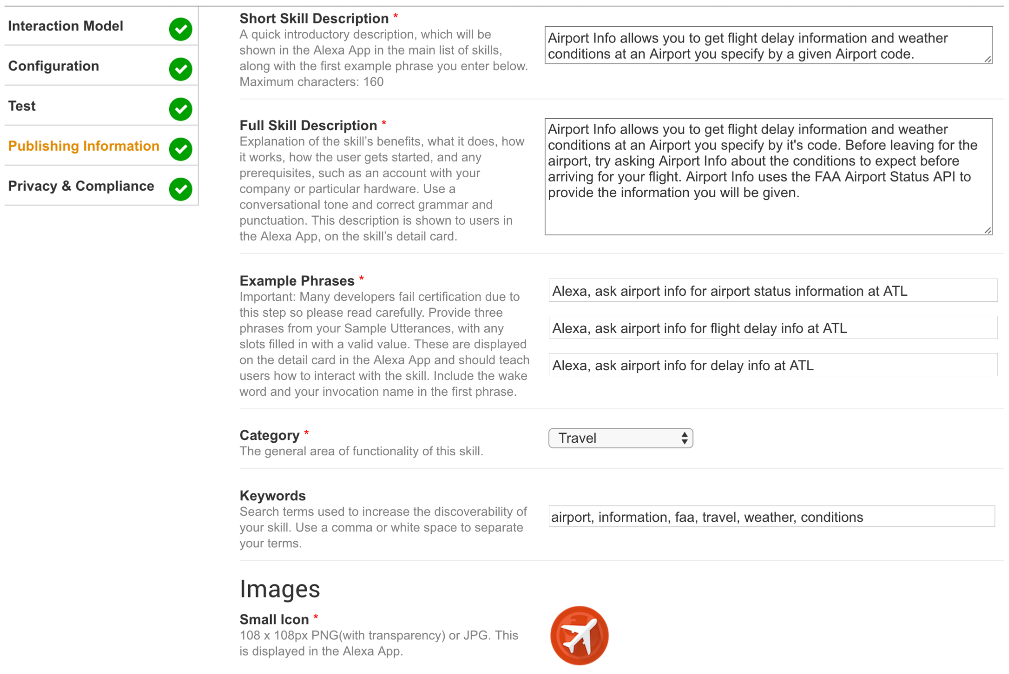


Alexa, ask airport info for airport status information at ATL

Airport Info allows you to get flight delay information and weather conditions at an Airport you specify by it's code.

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



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.

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.

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.

Alexa, ask airport info for airport status information at ATL

Alexa, ask airport info for flight delay info at ATL

Alexa, ask airport info for delay info at ATL

Category *
The general area of functionality of this skill.

Travel

Keywords
Search terms used to increase the discoverability of your skill. Use a comma or white space to separate your terms.

airport, information, faa, travel, weather, conditions

Images

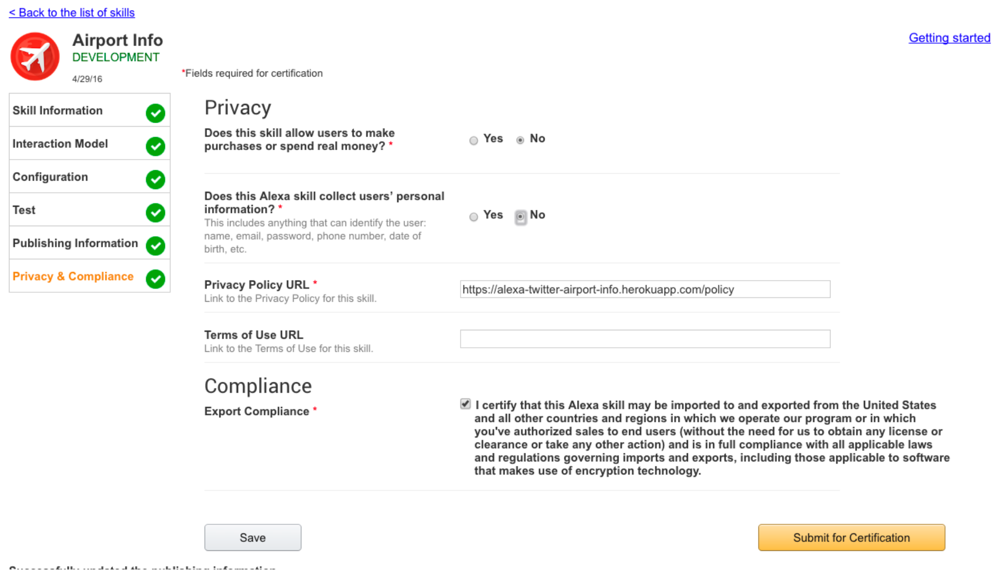
Small Icon *
108 x 108px PNG(with transparency) or JPG. This is displayed in the Alexa App.

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 example phrases for how to interact with the skill. For example, with Airport Info, the example utterance "delay status at {AirportCode}" resolves an intent to the `delayInfo` intent handler For the example phrases to provide should substitute any slots used in the intent with appropriate values, in this case "delay status at ATL". The example should also include the Wake word and Invocation name, as well. A valid example phrase to provide would therefore be "Alexa, ask Airport Info about delay status at ATL".

It is important that the example phrase is 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



*Fields required for certification

Privacy

Does this skill allow users to make purchases or spend real money? * Yes [✓ No]

Does this Alexa skill collect users' personal information? * Yes [✓ No]
This includes anything that can identify the user: name, email, password, phone number, date of birth, etc.

Privacy Policy URL * https://alexa-twitter-airport-info.herokuapp.com/policy
Link to the Privacy Policy for this skill.

Terms of Use URL
Link to the Terms of Use for this skill.

Compliance

Export Compliance * [✓] I certify that this Alexa skill may be imported to and exported from the United States and all other countries and regions in which we operate our program or in which you've authorized sales to end users (without the need for us to obtain any license or clearance or take any other action) and is in full compliance with all applicable laws and regulations governing imports and exports, including those applicable to software that makes use of encryption technology.

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.

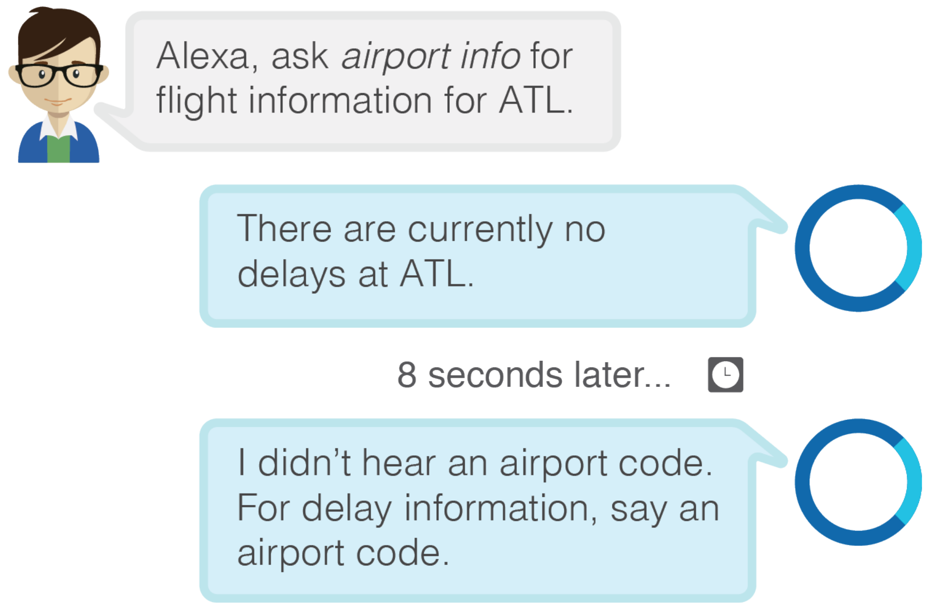
Approval Guidelines and Common Rejection Reasons

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 as an audience. There are also a number of common but more subtle mistakes that you can avoid for ensuring your skill is approved.

A first common mistake made 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 1.4 An Example of Incorrect Stream Management


Alexa, ask *airport info* for flight information for ATL.
There are currently no delays at ATL.
8 seconds later...
I didn't hear an airport code. For delay information, say an airport code.

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.

Another reason a skill could be rejected is if you require hosting your skill on a platform other than AWS Lambda. Amazon enforces additional requirements for approval in this case. 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 FAADataHelper

```
➜  faa–info ✗ moc
  FAADataHelper
    #getAirportStatus
      with an invalid airport code
        ✓ returns invalid airport code (314ms)
      with a valid airport code
success – received airport info for SFO
        ✓ 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 expecte
```

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 `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 1.5 Mocha and Chai are Popular Test Libraries for Node.js Development

For the example you will see how to go about writing unit tests for **`FAADataHelper`**. 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 **`FAADataHelper`**, 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 FAADataHelper = 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 FAADataHelper = require('../faa_data_helper');
describe('FAADataHelper', function() {
    var subject = new FAADataHelper();
    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 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() {
            //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
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/`