Extracted from:

Build Talking Apps

Develop Voice-First Applications for Alexa

The Pragmatic Bookshelf

Raleigh, North Carolina

The Pragmatic Programmers

Build Talking Apps

Develop Voice-First Applications for Alexa

Craig Walls

edited by Jacquelyn Carter

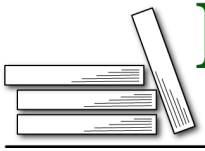
Build Talking Apps

Develop Voice-First Applications for Alexa

Craig Walls

The Pragmatic Bookshelf

Raleigh, North Carolina



For our complete catalog of hands-on, practical, and Pragmatic content for software developers, please visit *https://pragprog.com*.

The team that produced this book includes:

CEO: Dave Rankin
COO: Janet Furlow
Managing Editor: Tammy Coron
Development Editor: Jacquelyn Carter
Copy Editor: Karen Galle
Indexing: Potomac Indexing, LLC
Layout: Gilson Graphics
Founders: Andy Hunt and Dave Thomas

For sales, volume licensing, and support, please contact *support@pragprog.com*.

For international rights, please contact *rights@pragprog.com*.

ISBN-13: 978-1-68050-725-6
Encoded using the finest acid-free high-entropy binary digits.
Book version: P1.0—April 2022

Automated Testing with Alexa Skill Test Framework

The Alexa Skill Test Framework is a JavaScript framework for writing automated tests against Alexa skills. Unlike the semi-automated testing tools described in the previous section, the Alexa Skill Test Framework is able to verify the response from a skill. But like TestFlow, the Alexa Skill Test Framework is able to test a skill without requiring that the skill be deployed first. Consequently, you will get a quick turnaround on feedback from the test during development.

The Alexa Skill Test Framework is based on the Mocha testing tool, so before you start writing tests, you'll need to install the Mocha command line tool (assuming that you have not already done so for some other JavaScript project):

```
$ npm install --global mocha
```

Next, you'll need to install the Alexa Skill Test Framework as a development library in your project, along with the Mocha and Chai testing libraries. The following command line session switches into the lambda directory and installs the necessary libraries:

```
$ cd lambda
$ npm install ask-sdk-test mocha chai --save-dev
```

To enable Mocha to run when npm test is issued at the command line, you'll also need to edit the package.json file, replacing the default value for the test script with "mocha". After this tweak, the scripts section of package.json should look like this:

```
"scripts": {
    "test": "mocha"
},
```

Now we're ready to write some tests. We'll create our first test that will launch the skill and exercise the intent named HelloWorldIntent. Create a new directory named test under the lambda directory, and within the test directory, create a new file named hello-test.js and add the following test code:

test/starport-75/lambda/test/hello-test.js

```
'use strict';

const test = require('ask-sdk-test');
const skillHandler = require('../index.js').handler;

const skillSettings = {
  appId: 'amzn1.ask.skill.00000000-0000-0000-0000-000000000000',
  userId: 'amzn1.ask.account.VOID',
```

```
  deviceId: 'amzn1.ask.device.VOID',
  locale: 'en-US',
};

const alexaTest = new test.AlexaTest(skillHandler, skillSettings);

describe('Star Port 75 Travel - Hello Test', function() {
  describe('LaunchRequest', function() {
    alexaTest.test([
      {
        request: new test.LaunchRequestBuilder(skillSettings).build(),
        saysLike: 'Welcome to Star Port 75 Travel. How can I help you?',
          repromptsNothing: false,
          shouldEndSession: false,
      },
    ]);
  });

  describe('HelloWorldIntent', function() {
    alexaTest.test([
      {
        request: new test.IntentRequestBuilder(
                skillSettings, 'HelloWorldIntent').build(),
        saysLike: 'Have a stellar day!'
      },
    ]);
  });

});
```

The first few lines of hello-test.js import the ask-sdk-test library and the skill's main handler which is exported from index.js.

Then, some essential skill settings are established, including the skill's ID, a test user ID, a test device ID, and the locale. The values given for the skill ID, user ID, and device ID are merely fake test values and do not have to match an actual skill, user, or device. They are only needed when testing more advanced skills that work with data from a user's Amazon account.

The handler and skill settings are then used to create an AlexaTest object that will be used to make assertions against the skill's request handlers in the test.

What follows is the test definition itself. It is structured as a test suite that wraps nested test cases. At the test suite level, the call to describe() names the test suite as Star Port 75 Travel - Hello Test and is given a function that wraps the two tests contained by the test suite.

Within the test suite there are two test cases, named LaunchRequest and HelloWorldIntent by the calls to describe(). Each of these is given a function that contains

the test details. Specifically, each test is described as a call to alexaTest.test(), which is given an array of test descriptors.

The test descriptors specify the kind of request, either a launch request or a named intent request, as well as the expectations of making the request. For example, in the test named HelloWorldIntent, the test descriptor specifies that if the skill is sent an intent request for HelloWorldIntent, then the output speech should be "Have a stellar day!"

There are other properties we could specify in the test descriptor to make assertions against the request and response, but they don't apply to such a simple skill. As we build out our skill and write new tests, we'll see other ways to make assertions in the test descriptor.

With the test written, we are now ready to run it and see if the skill satisfies the test's expectations. You can run the test with npm like this:

```
$ npm test

> hello-world@1.1.0 test
> mocha


  Star Port 75 Travel - Hello Test
    LaunchRequest
      ✓ returns the correct responses
    HelloWorldIntent
      ✓ returns the correct responses

  2 passing (32ms)
```

As you can see, the test completed with two passing tests. But what if the skill didn't meet the test's expectations? In that case, you might see something like this when you run the test:

```
$ npm test

> hello-world@1.1.0 test
> mocha


  Star Port 75 Travel - Hello Test
    LaunchRequest
      ✓ returns the correct responses
    HelloWorldIntent
      1) returns the correct responses

  1 passing (35ms)
  1 failing

  1) Hello World Skill
       HelloWorldIntent
```

```
   returns the correct responses:

Speech did not contain specified text. Expected Have a stellar day
to be like Have a stellar day!
+ expected - actual

-false
+true
```

Here, it shows that one test passed and one test failed. For the failing test, it shows the expectation, following by the actual value. In this case, it seems that the closing exclamation mark is missing from the output speech value.

Recall that one of the benefits of testing with the Alexa Skill Test Framework is that it doesn't require that the skill be deployed before you can verify that it behaves as you expect. And, since the test cases are written in JavaScript, you have the full flexibility of the language to define the test expectations.

Even so, at this point our test only sends a request and performs basic assertions against the response, not really taking advantage of the flexibility of JavaScript. Let's take a look at another testing framework that employs YAML files to declare your test's expectations and get a test coverage as a side-effect.