Kit for Node.js

lexa-sdk for Node.js to help you build skills faster and with less complexity. ls Kit, Node.js and AWS Lambda has become one of the most popular ways we non-blocking I/O model of Node.js is well suited for an Alexa skill and Node.js is rce libraries in the world. Plus, with AWS Lambda is free for the first one million sting for most developers. And you don't need to manage any SSL certificates Skills Kit is a trusted trigger).

Lambda, Node.js and the Alexa Skills Kit has been a simple process, the actual amount of code you have had to write has not. We have seen a large amount of time spent in Alexa skills on handling session attributes, skill state persistence, response building and behavior modeling. With that in mind the Alexa team set out to build an Alexa Skills Kit SDK specifically for Node.js that will help you avoid common hang-ups and focus on your skill's logic instead of boiler plate code.

Enabling Faster Alexa Skill Development with the Alexa Skills Kit for Node.js (alexa-sdk)

With the new alexa-sdk, our goal is to help you build skills faster while allowing you to avoid unneeded complexity. Today, we are launching the SDK with the following capabilities:

- Hosted as NPM package allowing simple deployment to any Node.js environment
- Ability to build Alexa responses using built-in events
- Helper events for new sessions and unhandled events that can act as a 'catch-all' events
- Helper functions to build state-machine based Intent handling
  - This makes it possible to define different event handlers based on the current state of the skill
- Simple configuration to enable attribute persistence with DynamoDB
- All speech output is automatically wrapped as SSML
- Lambda event and context objects are fully available via this.event and this.contextAbility to override built-in functions giving you more flexibility on how you manage state or build responses. For example, saving state attributes to AWS S3.

Installing and Working with the Alexa Skills Kit for Node.js (alexa-sdk)

The alexa-sdk is immediately available on github here and can be deployed as a node package using the following command from within your Node.js environment:

```
npm install --save alexa-sdk
```

In order to start using the alexa-sdk you will need to first import the library. To do this within your own project simply create a file named index.js and add the following to it:

```
var Alexa = require('alexa-sdk');

exports.handler = function(event, context, callback){
    var alexa = Alexa.handler(event, context);
};
```

This will import the alexa sdk and set up an alexa object for us to work with. Next, we need to handle when the intents for our skill. Thankfully, the alexa-sdk makes it simple to have a function fire on every Intent we would like. For example, to create a handler for a 'HelloWorldIntent' we simply add the following:

```
var handlers = {
    'HelloWorldIntent': function () {
        this.emit(':tell', 'Hello World!');
    }
```

```
};
```

Notice the new syntax above for ":tell"? The alexa-sdk follows a tell/ask response methodology for generating your outputSpeech response objects. If we wanted to ask the user for information we would replace the above with:

```
this.emit(':ask', 'What would you like to do?', 'Please say that again?');
```

In fact, many of the responses you would generate for your skill follow this same syntax! Here are some additional examples for common skill response generation:

```
var speechOutput = 'Hello world!';
var repromptSpeech = 'Hello again!';

this.emit(':tell', speechOutput);

this.emit(':ask', speechOutput, repromptSpeech);

var cardTitle = 'Hello World Card';
var cardContent = 'This text will be displayed in the companion app card.';

var imageObj = {
   smallImageUrl: 'https://imgs.xkcd.com/comics/standards.png',
   largeImageUrl: 'https://imgs.xkcd.com/comics/standards.png'

};

this.emit(':askWithCard', speechOutput, repromptSpeech, cardTitle, cardContent, imageObj);

this.emit(':tellWithCard', speechOutput, cardTitle, cardContent, imageObj);

this.emit(':tellWithLinkAccountCard', speechOutput);

this.emit(':askWithLinkAccountCard', speechOutput);

this.emit(':responseReady'); // Called after the response is built but before it is returned to the Alexa service. Calls
:saveState.

this.emit(':saveState', false); // Handles saving the contents of this.attributes and the current handler state to
DynamoDB and then sends the previously built response to the Alexa service. Override if you wish to use a different
persistence provider. The second attribute is optional and can be set to 'true' to force saving.

this.emit(':saveStateError'); // Called if there is an error while saving state. Override to handle any errors yourself.
```

Once we have set up our event handlers, in this case for our NewSession, we need to register them using the registerHandlers function of the alexa object we just created.

```
exports.handler = function(event, context, callback){
   var alexa = Alexa.handler(event, context);
   alexa.registerHandlers(handlers);
};
```

You can also register multiple handler objects at once. So instead of just the handlers object, we create for NewSession we could have many different types of handles for other events and register them all at once like this:

```
alexa.registerHandlers(handlers, handlers2, handlers3, ...);
```

The handlers you define can call each other, making it possible to ensure your responses are uniform. Here is an example where our LaunchRequest and IntentRequest (of HelloWorldIntent) both return the same 'Hello World' message.

where our LaunchRequest and IntentRequest (for HelloWorldIntent) both return the same "Hello World" message.

```
var handlers = {
    'LaunchRequest': function () {
        this.emit('HelloWorldIntent');
    },
    'HelloWorldIntent': function () {
        this.emit(':tell', 'Hello World!');
};
```

Once you are done registering all of your intent handler functions, you simply use the execute function from the alexa object to run your skill's logic. The final line would look like this:

```
exports.handler = function(event, context, callback){
    var alexa = Alexa.handler(event, context);
    alexa.registerHandlers(handlers);
    alexa.execute();
};
```

You can download a full working sample off github. We have also updated the following Node.js sample skills to work with the alexa-sdk: Fact, HelloWorld, HighLow, HowTo and Trivia.

Making Skill State Management Simpler

The alexa-sdk will route incoming intents to the correct function handler based on state. It is simply a string stored in your session attributes indicating the current state of the skill. You can emulate the built-in intent routing by appending the state string to the intent name when defining your intent handlers, but the alexa-sdk helps do that for you.

For example, let's create a simple number-guessing game with "start" and "guess" states based on our previous example of handling a NewSession event.

```
var states = {
    GUESSMODE: '_GUESSMODE', // User is trying to guess the number.
    STARTMODE: '_STARTMODE'  // Prompt the user to start or restart the game.
};

var newSessionHandlers = {

 // This will short-cut any incoming intent or launch requests and route them to this handler.
  'NewSession': function() {
    this.handler.state = states.STARTMODE;
    this.emit(':ask', 'Welcome to The Number Game. Would you like to play?.');
  }
 };
```

Notice that when a new session is created we simply set the state of our skill into STARTMODE using this.handler.state. The skills state will automatically be persisted in your skill's session attributes, and will be optionally persisted across sessions if you set a DynamoDB table.

It is also important point out that NewSession is a great catch-all behavior and a good entry point but it is not required. NewSession will only be invoked if a handler with that name is defined. Each state you define can have its own NewSession handler which will be invoked if you are using the built-in persistence. In the above example we could define different NewSession behavior for both states.STARTMODE and states.GUESSMODE giving us added flexibility.

In order to define intents that will respond to the different states of our skill we need to use the Alexa.CreateStateHandler function. Any intent handlers defined here will only work when the skill is in a specific state, giving us even greater flexibility!

For example, if we are in the GUESSMODE state we defined above we want to handle a user responding to a question. This can be done using StateHandlers like this:

```
var guessModeHandlers = Alexa.CreateStateHandler(states.GUESSMODE, {
    'NewSession': function () {
        this.handler.state = '';
        this.emitWithState('NewSession'); // Equivalent to the Start Mode NewSession handler
    },
    'NumberGuessIntent': function() {
        var guessNum = parseInt(this.event.request.intent.slots.number.value);
```

```
        var guessNum = parseInt(this.event.request.intent.slots.number.value);
        var targetNum = this.attributes["guessNumber"];
        console.log('user guessed: ' + guessNum);

        if(guessNum > targetNum){
            this.emit('TooHigh', guessNum);
        } else if( guessNum < targetNum){
            this.emit('TooLow', guessNum);
        } else if (guessNum === targetNum){
            // With a callback, use the arrow function to preserve the correct 'this' context
            this.emit('JustRight', () => {
                this.emit(':ask', guessNum.toString() + 'is correct! Would you like to play a new game?',
                'Say yes to start a new game, or no to end the game.');
            })
        } else {
            this.emit('NotANum');
        }
    },
    'AMAZON.HelpIntent': function() {
        this.emit(':ask', 'I am thinking of a number between zero and one hundred, try to guess and I will tell you' +
            ' if it is higher or lower.', 'Try saying a number.');
    },
    'SessionEndedRequest': function () {
        console.log('session ended!');
        this.attributes['endedSessionCount'] += 1;
        this.emit(':saveState', true);
    },
    'Unhandled': function() {
        this.emit(':ask', 'Sorry, I didn\'t get that. Try saying a number.', 'Try saying a number.');
    }
});
```

On the flip side, if I am in STARTMODE I can define my StateHandlers to be the following:

```
var startGameHandlers = Alexa.CreateStateHandler(states.STARTMODE, {
    'NewSession': function () {
        this.emit('NewSession'); // Uses the handler in newSessionHandlers
    },
    'AMAZON.HelpIntent': function() {
        var message = 'I will think of a number between zero and one hundred, try to guess and I will tell you if it' +
            ' is higher or lower. Do you want to start the game?';
        this.emit(':ask', message, message);
    },
    'AMAZON.YesIntent': function() {
        this.attributes["guessNumber"] = Math.floor(Math.random() * 100);
        this.handler.state = states.GUESSMODE;
        this.emit(':ask', 'Great! ' + 'Try saying a number to start the game.', 'Try saying a number.');
    },
    'AMAZON.NoIntent': function() {
```

```
    'AMAZON.NoIntent': function() {
        this.emit(':tell', 'Ok, see you next time!');
    },
    'SessionEndedRequest': function () {
        console.log('session ended!');
        this.attributes['endedSessionCount'] += 1;
        this.emit(':saveState', true);
    },
    'Unhandled': function() {
        var message = 'Say yes to continue, or no to end the game.';
        this.emit(':ask', message, message);
    }
```

Take a look at how AMAZON.YesIntent and AMAZON.NoIntent are not defined in the guessModeHandlers object, since it doesn't make sense for a "yes" or "no" response in this state. Those intents will be caught by the 'Unhandled' handler.

Also, notice the different behavior for NewSession and Unhandled across both states? In this game, we 'reset' the state by calling a NewSession handler defined in the newSessionHandlers object. You can also skip defining it and alexa-sdk will call the intent handler for the current state. Just remember to register your State Handlers before you call alexa.execute() or they will not be found.

Your attributes will be automatically saved when you end the session, but if the user ends the session you have to emit the ':saveState' event (this.emit(':saveState', true) to force a save. You should do this in your SessionEndedRequest handler which is called when the user ends the session by saying 'quit' or timing out. Take a look at the example above.

We have wrapped up the above example into a high/low number guessing game skill you can download here.

Persisting Skill Attributes through DynamoDB

Many of you would like to persist your session attribute values into storage for further use. The alexa-sdk integrates directly with Amazon DynamoDB (a NoSQL database service) to enable you to do this with a single line of code.

Simply set the name of the DynamoDB table on your alexa object before you call alexa.execute.

```
exports.handler = function(event, context, callback) {
    var alexa = Alexa.handler(event, context);
    alexa.appId = appId;
    alexa.dynamoDBTableName = 'YourTableName'; // That's it!
    alexa.registerHandlers(State1Handlers, State2Handlers);
    alexa.execute();
};
```

Then later on to set a value you simply need to call into the attributes property of the alexa object. No more put and get separate functions!

```
this.attributes["yourAttribute"] = 'value';
```

You can create the table manually beforehand or simply give your Lambda function DynamoDBcreate table permissions and it will happen automatically. Just remember it can take a minute or so for the table to be created on the first invocation.

Try extending the HighLow game:

- Have it store your average number of guesses per game
- Add sound effects
- Give the player a limited amount of guesses

For more information about getting started with the Alexa Skills Kit, check out the following additional assets:

Alexa Skills Kit for Node.js
Alexa Dev Chat Podcast
Alexa Training with Big Nerd Ranch
Intro to Alexa Skills On Demand
Voice Design 101 On Demand
Alexa Skills Kit (ASK)
Alexa Developer Forums

-Dave (@TheDaveDev)

Want The Latest?

Subscribe to Alexa Announcements

First Name

Last Name

Email Address

Zip Code

Select Country...

Submit

Subscribe via RSS

Alexa Topics

Alexa Skills Kit

Alexa Voice Service

Alexa Science

Announcements

Developer Spotlight

Tips and Tutorials

Recent Posts

Why Voice Design Matters: We Don’t Speak the Way We Write

Der Spiele-Klassiker Mau-Mau begeistert auch auf Alexa

Developer Tools to Help You Build Alexa Skills

Request Permission and Access Contact Information with the New Customer Profile API

Accelerate Skill Development Using the ASK Toolkit for Visual Studio Code (Beta)

Archive

Year Month GO

Language English

Follow Us:

amazon alexa

amazon appstore

aws

amazon software + games

amazon dash services