Make your own VUI

Reference Information

NUX Camp 2018

Hello,

First of all, thank you for attending this workshop. I hope you come away feeling a more empowered to play with VUI technology than you arrived. There is a lot of work left for those of us in the User Experience field to transform VUIs into the natural experiences many envision for the future.

This pack is intended to act as a helpful reference during and after the workshop, but it does not follow the structure of the workshop at all. I'd recommend following along with the workshop to get the most understanding out of it.

If you have any feedback about the workshop, want to discuss things further, would like to find out more about Scott Logic or just feel a need to nerd out generally (parenthood, guitar effects pedals and leadership seem to be at the forefront of my mind recently) then grab me once we're done or hit me up on Twitter @g_odds.

Thank you again for your time and attention.

Regards,

Graham Odds

Content

Anatomy of Alexa

Anatomy of an utterance

Wake word – when using Alexa on a device without activiation button, this is the word you say to get Alexa's attention

Invocation name – the name of skill you want to use (can be ommitted for cases where a "default skill" is supported)

Intent – the thing you're wanting the skill to do

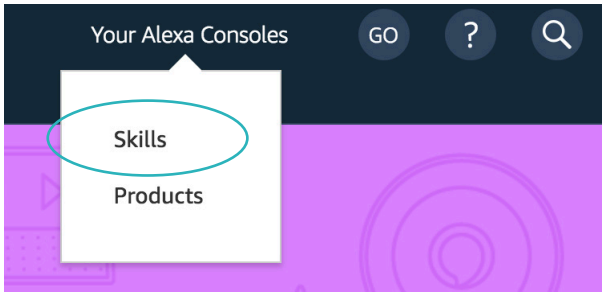
Slot – an additional bits of information you want to pass to the skill that help inform exactly what the intent does

Utterances can be structured in numerous different ways, but typically include at least the invocation name and intent (although the invocation name alone can also be a useful way of "opening an app"). For example, *Alexa, can you book an appointment at 4pm tomorrow for me using my barbershop* should achieve the same outcome as the annotated example above.

For more information go to https://developer.amazon.com/docs/custom-skills/understanding-how-users-invoke-custom-skills.html

Alexa Skill Console

https://developer.amazon.com/alexa



Click here once you've set up your skill to make it testable and test it.

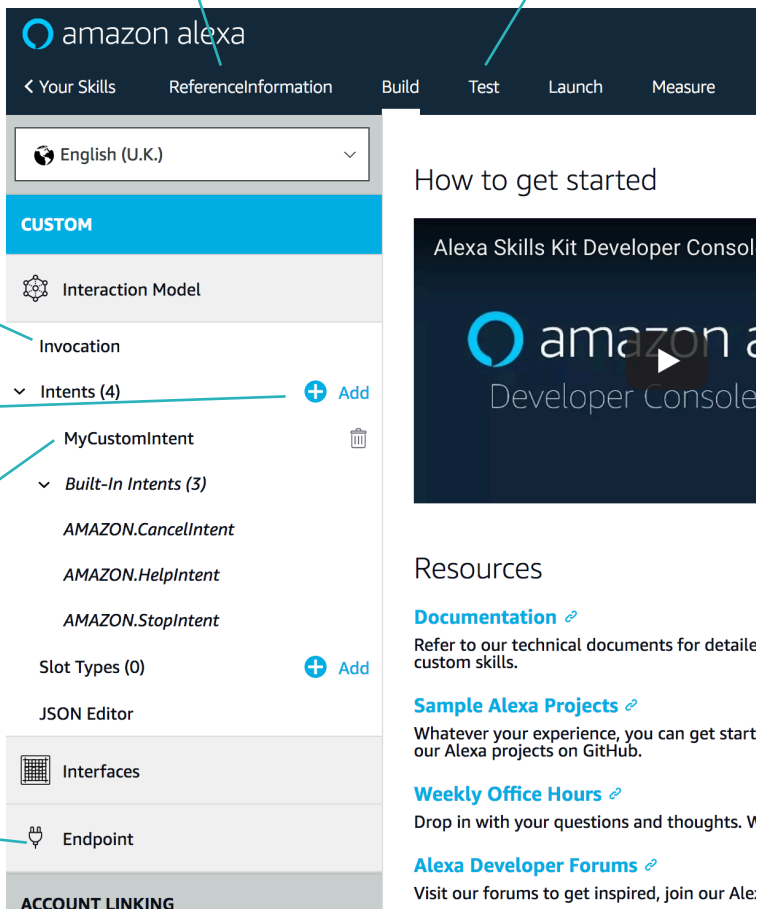
The name of your skill.

Go here to set up the invocation name.

Click this to add an intent.

Your intents will show up here. You can edit them by clicking their name.

Go here to set up the skill endpoint.

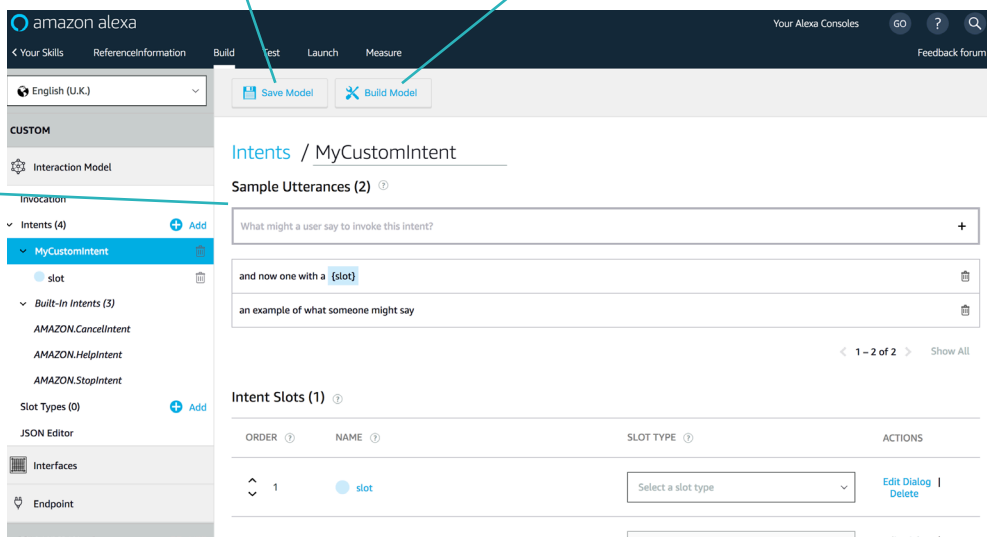


Once your model is good to go, click this to set the machine learning going.

Warning: can take a while!

Click this to save your model.

Type in example phrases you want to map to this intent .

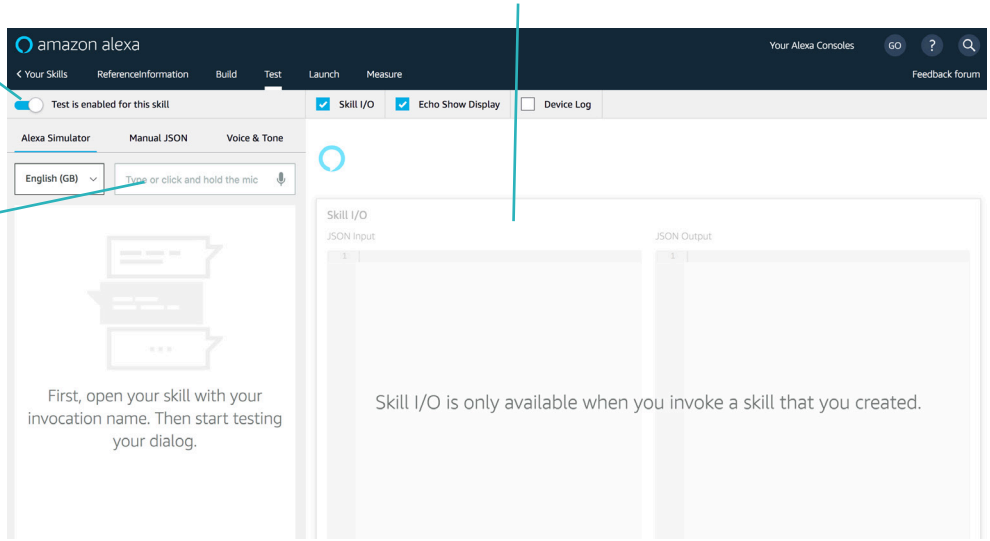


IMPORTANT!

Switch this on, or you can't use your skill.

This panel shows the JSON gunk that is sent to your skill endpoint, and what your endpoint returns.

Type in (or click and hold the mic to record) what you'd say to Alexa and see what it does.



Skill Endpoint

https://glitch.com/edit/#!/nuxcamp

Glitch is a free (and very clever) code editor and runner. It let's you "remix" (copy) projects and then play around with them. The code is JavaScript and it is being executed in a NodeJS runtime environment. Or blah blah blah JavaScript blah blah Node blah blah – if you just want the useful keywords.

The project I have set up for the workshop contains some boilerplate code that runs a server (using a library called `express`) to act as your skill endpoint. I have also loaded it up with a library called `alexa-app` that takes away a lot of the complexity of writing a skill endpoint for you. The `Speech` library makes it easier to control what Alexa says in response to Intents and the light library is some nasty code I hacked together so you can play with the lamp in front of you at the workshop.

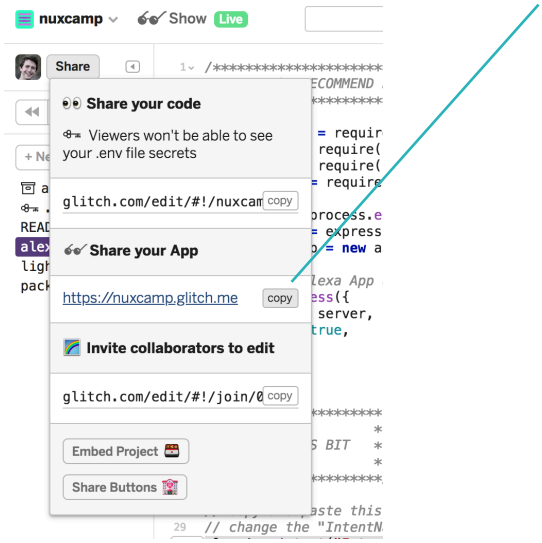
Add intent logic

The alexa-app library means we can handle any intent with the following code structure

```
alexaApp.intent("IntentName", function(request, response) {
  // put any code you want to add here, for example:
  response.say("Wehay, it's working");
});
```

Endpoint URL (for Alexa Skill Console)

To connect the skill you set up in the Alexa Skill Console to your Skill Endpoint, you need the URL of your endpoint. You can get the URL of your Glitch project here



Small debugging tip

If you get stuck, you can use `console.log(<text>)` to write a string to the logs. To watch the logs in realtime click Logs

JavaScript Basics

Literals

Literals are the lowest level building blocks for your code: values. There are a few different types, notably:

Numbers – e.g. `1`, `5`, `100`

Booleans – `true` or `false`

Strings – text, denoted by surrounding `""`, e.g. `"hello"`, `"I'm a longer string"`

Variables

You can think of variables as a container for a literal, and you can change what is in that container! You declare a variable using the var keyword and then whatever name you want to give it (as long as it doesn't have spaces). You assign a value to the variable with =. For example

```
var myVariable = 5;
myVariable = "hello";
```

At the end of this code, the variable myVariable contains the value "hello";

Operators

Operators let you manipulate one or more literals or variables to start doing more interesting things with them.

Boolean Operators

Boolean literals or variables can be logically combined using `||` (OR) or `&&` (AND).

Boolean literals or variables can be inverted using `!` (NOT).

Literals and variables can be evaluated for equality using `==` (equals) or `!=` (not equals).

Mathematical Operators

You can use the `+`, `-`, `*` and `/` operators to do the arithmetic operations plus, minus, times and divide respectively on numeric literals or variables.

String Concatenation

You can concatenate two strings together using +. For example

```
var name = "Peter";
var greeting = "Hello " + name + ", nice to meet you."
```

will result in greeting containg the string "Hello Peter, nice to meet you".

If... Then... Else...

If you fancy adding a bit of logic to your code then this is the simplest and most powerful way. The basic structure is

```
if (<condition>) {
    some code that runs if <condition> is true
}
else if (<condition 2>) {
    some code that runs if <condition> is true
}
else {
    some code that runs if none of the previous conditions are true
}
```

You can add as many `else if`s as you want (or none!) and the `else` bit is optional.

`<condition>` must evaluate to a Boolean literal, i.e. it must be `true` or `false`, a variable with the value `true` or `false`, or a Boolean Operator.

Speech

Introduction

Alexa does a pretty decent automatic job of intonation, all things considered. However, sometimes she can benefit from you providing some guidance around pauses and the like. To do this, rather than passing Alexa a simple string in response, it can handle something called SSML (Speech Synthesis Markup Language) that includes this extra guidance (although its capabilities are fairly limited at the moment). It's a bit of a pig to construct, but fortunately there is an open source library called `ssml-builder` (that I have renamed `Speech` in the template) to make our life easier.

You can find out more about SSML at https://developer.amazon.com/docs/custom-skills/speech-synthesis-markup-language-ssml-reference.html

Example usage

```
var speech = new Speech();
speech.say("Hello")
      .pause("1s")
      .whisper("I'm going to speak quietly, ")
      .emphasis("strong", "unless I don't want to.")
      .pause("500ms")
      .say("Or I could keep it normal);
response.say(speech.ssml(true));
```

Functions

`.say(<text>)`
Something witty (or not) you want Alexa to say.
`<text>` should be a string.

`.pause(<length>)`
Instruct Alexa to take a break between saying things.
`<length>` should be a string representing the length of the pause, including units, e.g. `"1s"` or `"500ms"`

`.whisper(<text>)`
Get Alexa to say something in hushed tones.
`<text>` should be a string.

`.emphasis("strong"` or `"moderate"` or `"reduced", <text>)`
Emphasise or de-emphasise something that is being said.
`<text>` should be a string.

For more, go to https://github.com/mandnyc/ssml-builder

Light

Background

The smart lights we are using in the workshop are a ramshackle homebrew solution: an Ikea Tradfri bulb and Ikea Tvars lamp, driven by a Phillips Hue Bridge along with some code I have running to get us around some (very sensible!) security constraints. I hacked the `light.js` code together specifically to play with this setup; it is not pretty or clever. As a result, this information about using the light will serve little to no use away from the workshop... sorry.

Usage Notes

Unfortunately the lights hardware is not the most reliable. At times they simply won't respond to commands, or they respond with a frustrating delay. This happens even with the official app/tooling.

Please do not hit the API with lots of requests in a short space of time. This is all but guaranteed to make the whole setup sulk and wil spoil the fun for everyone!

Functions

`light.on()`
Turns the light on (cryptic!).

`light.off()`
Turns the light off (who could have guessed?!).

`light.bright()`
Makes the light as bright as it can go without popping.

`light.dim()`
Makes the light as dim as it can go without being off.

`light.dim(<brightness>)`
Sets the light's brightness to `<brightness>`%.
`<brightness>` should be a number between 0 and 100.

`light.warm()`
Gives the light a yellow glow to warm the cockles of your heart.

`light.cold()`
Gives the light a white/blue glow for that edgy industrial feel.

`light.pulse(<rate>)`
Makes the light pulse at a regular rhythm every `<rate>` milliseconds.
`<rate>` should be a value between 200 and 10,000 (any slower and you'll fall asleep trying to watch the pulsing)

more on next page

`light.morseCode(<phrase>)`

Instructs the light flash out <phrase> in Morse Code.

`<phrase>` should be a string (preferably a short one!).

Warning: this can be pretty intense and headache inducing.

`light.stop()`

Asks the light to stop whatever pulse or Morse Code nonsense it is up to.