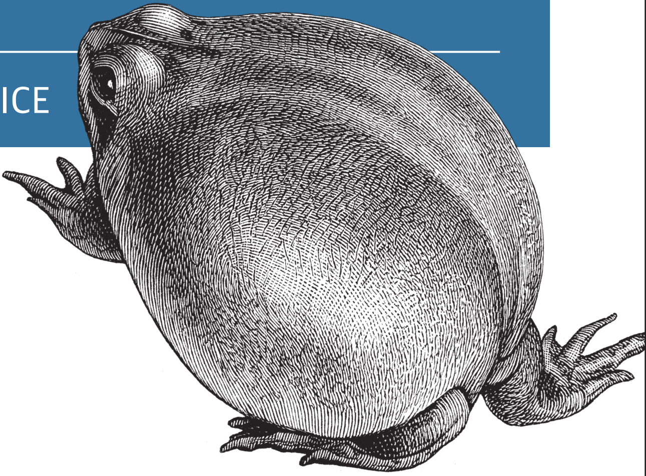
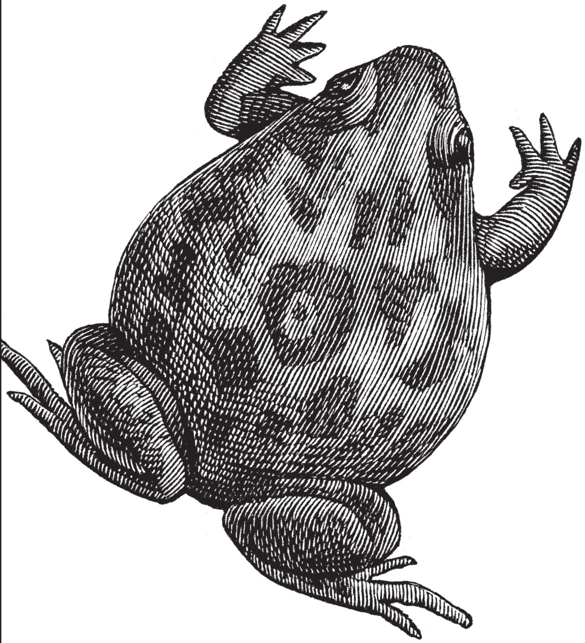
O'REILLY®

Programming Voice Interfaces

GIVING CONNECTED DEVICES A VOICE





Walter Quesada &
Bob Lautenbach

O'REILLY®

Programming Voice Interfaces

We're fast approaching the point where we'll be able to naturally converse with our devices. Advances in voice frameworks and APIs from Amazon, Microsoft, and Google along with inexpensive off-the-shelf components means that you can now quickly and easily get a voice interface prototype up and running on your own.

In this handy reference guide, Walter Quesada and Bob Lautenbach offer an overview of voice interface fundamentals and available integration options before taking you step-by-step through building and iterating two Raspberry Pi-based voice-enabled devices; one using Amazon's Alexa Voice Service (AVS) and the other using Microsoft's Windows IoT Core and Google's API.AI. Whether you're a developer who wants to familiarize yourself with this important new platform or a tinkerer who wants to get started building your own voice-enabled devices, this book teaches you how to bring together common architectural patterns and approaches, components, and frameworks into a voice interface prototype you can continue to build on.

- Use Amazon AVS to build a voice-enabled device that works with a number of Alexa skills out of the box
- Iterate your prototype by creating your own Amazon Alexa skills with Node.js and AWS Lambda
- Create an Alexa-like device with Windows IoT Core using API.AI and build a Universal Windows Platform (UWP) app to handle the voice interactions
- Learn how to extend your device by adding more inputs, such as a motion sensor; secure your device with authentication, cryptography, data validation, and error handling; and create a plan for quality assurance and support

Walter Quesada is a software engineer with over 20 years of experience designing and developing applications for desktop, web, mobile, and devices for the Internet of Things. His primary focus includes integrating Natural User Interfaces such as touch, voice, and gestures into reimagined experiences around retail, hospitality, healthcare, and other sectors. Additionally, Walter speaks at the occasional local code camps, is a Pluralsight instructor, cofounder of the South Florida Emerging Technology Meetup Group, and currently resides in Miami, Florida.

Bob Lautenbach is a veteran of designing software for the hospitality and cruise industries, and he firmly believes voice technology is poised to revolutionize the way the industry operates. His main interests are around leveraging Internet of Things devices to create new and engaging guest experiences. Bob is the cofounder and Chief "Tinker" Officer of Voceio, an Orlando-based firm that creates multichannel conversational platforms. Bob is an Alexa Champion, Pluralsight author, and founder of the Orlando Alexa Meetup group. He currently resides in Orlando, Florida.

US $42.99 CAN $56.99
ISBN: 978-1-491-95600=7
54299
9 781491 956007

Programming Voice Interfaces

Giving Connected Devices a Voice

Walter Quesada and Bob Lautenbach

Beijing · Boston · Farnham · Sebastopol · Tokyo

O'REILLY®

Programming Voice Interfaces
by Walter Quesada and Bob Lautenbach

Published by O’Reilly Media, Inc., 1005 Gravenstein Highway North, Sebastopol, CA 95472.

O’Reilly books may be purchased for educational, business, or sales promotional use. Online editions are also available for most titles (*http://oreilly.com/safari*). For more information, contact our corporate/institutional sales department: 800-998-9938 or *corporate@oreilly.com*.

Editors: Susan Conant and Jeff Bleiel
Production Editor: Shiny Kalapurakkel
Copyeditor: Jasmine Kwityn
Proofreader: Kim Cofer
Indexer: Angela Howard
Interior Designer: David Futato
Cover Designer: Karen Montgomery
Illustrator: Rebecca Demarest

October 2017: First Edition

Revision History for the First Edition
2017-10-04: First Release

978-1-491-95600-7

[LSI]

Table of Contents

Preface

As its title implies, this book focuses on programming voice interfaces for connected devices. We wanted to share what we've learned from experimenting with some of the various platforms currently available, such as Amazon Alexa, API.AI, and others. We also tried to condense the choices to some of the main players and provide step-by-step, how-to examples, allowing you to dig in quickly and start playing. Our goal was to create a reference guide to voice interfaces in under 200 pages. Some of the topics in this book could very well have entire books dedicated to them. We obviously had to make hard choices on what to include and exclude, so we focused on trying to expose you to what's possible; to pique your curiosity to dig deeper and continue experimenting on your own.

We assume you have some experience programming with Node.js and/or C#, but you can still work through all of the examples even if you are a relative "newbie" to coding.

Voice interfaces are here to stay and will only get more intelligent and human-like as time and technology march forward. We hope this book encourages you to be part of this new and exciting platform. We look forward to reading about you in the future and all of the incredible experiences you've created using voice-based technologies.

Who Should Read This Book

There are only two main requirements for readers of this book. The first requirement is some level of experience programming in either Python, C#, or Node.js. Second, simply a desire to learn. We wrote this book for all the "makers" out there—those who live to learn and experiment, and who love to tinker and find technology-based solutions for everyday problems.

Conventions Used in This Book

The following typographical conventions are used in this book:

Italic
: Indicates new terms, URLs, email addresses, filenames, and file extensions.

`Constant width`
: Used for program listings, as well as within paragraphs to refer to program elements such as variable or function names, databases, data types, environment variables, statements, and keywords.

`Constant width bold`
: Shows commands or other text that should be typed literally by the user.

`Constant width italic`
: Shows text that should be replaced with user-supplied values or by values determined by context.

This element signifies a tip or suggestion.

This element signifies a general note.

This element indicates a warning or caution.

O'Reilly Safari

Safari (formerly Safari Books Online) is a membership-based training and reference platform for enterprise, government, educators, and individuals.

Members have access to thousands of books, training videos, Learning Paths, interactive tutorials, and curated playlists from over 250 publishers, including O'Reilly Media, Harvard Business Review, Prentice Hall Professional, Addison-Wesley Professional, Microsoft Press, Sams, Que, Peachpit Press, Adobe, Focal Press, Cisco Press, John Wiley & Sons, Syngress, Morgan Kaufmann, IBM Redbooks, Packt, Adobe Press, FT Press, Apress, Manning, New Riders, McGraw-Hill, Jones & Bartlett, and Course Technology, among others.

For more information, please visit *http://oreilly.com/safari*.

How to Contact Us

Please address comments and questions concerning this book to the publisher:

O'Reilly Media, Inc.
1005 Gravenstein Highway North
Sebastopol, CA 95472
800-998-9938 (in the United States or Canada)
707-829-0515 (international or local)
707-829-0104 (fax)

To comment or ask technical questions about this book, send email to *bookquestions@oreilly.com*.

For more information about our books, courses, conferences, and news, see our website at *http://www.oreilly.com*.

Find us on Facebook: *http://facebook.com/oreilly*

Follow us on Twitter: *http://twitter.com/oreillymedia*

Watch us on YouTube: *http://www.youtube.com/oreillymedia*

Acknowledgments

Bob Lautenbach

Special thanks to Walter for inviting me to participate in this book. It was a fun ride, my friend! I would also like to thank those who have mentored and supported me along the way. Without their support and encouragement I would not be where I am today. I can't possibly name them all, but would like to give a shout out to my friend and coauthor Walter, from whom I have learned so much, my favorite tech evangelist, David Isbitski, and finally, Bart Butler, who has helped me in more ways than I can mention.

Thanks to our editor Jeff Bleiel—his patience and guidance throughout this process was invaluable.

Finally, I want to thank and dedicate this book to my wife, Vivian, and my two incredible boys, Steven and Tyler. Your patience and support energize and motivate me every day.

Walter Quesada

I would like to thank David Noderer for connecting me with Susan Conant at O'Reilly. Thank you both for helping kick off this amazing journey. Thanks to my coauthor in crime, Bob, for saving my ass and helping take the book to the finish line. A special thanks to our editor, Jeff Bleiel, whose patience with my slacker procrastinating ways continues to be extremely appreciated.

I would also like to thank Cathy Pearl and Chris Maury, whose constructive and valuable feedback on the book helped us elevate the content further.

Additionally, I would like to thank those whose advice and mentorship throughout the years were critical in shaping my career and role as a technologist. Some of those awesome folks include David Clarke, Chris Curran, Dan Eckert, and David Isbitski. There are too many to name here so for those unmentioned, you know who you are and know that my thanks goes out to you as well.

Most importantly, I would love to thank and dedicate this book to my family, specifically my amazing mom and sister, my wonderful wife, Monica, and my two all-grown-up sons, Derek and Kyle. My family's love and support is why I do these projects to begin with.

CHAPTER 1

Introduction to Voice Interfaces and the IoT

Bell Labs engineer Homer Dudley invented the first speech synthesis machine, titled "Voder," in the early scientific technical revolution days of 1937. Based on his earlier work in 1928 on *vocoder* (voice encoder) Dudley put together gas tubes, a foot pedal, 10 piano-like keys, and various other components designed to work in harmony to produce human speech from synthesized sound. Then in 1962, at the Seattle World's Fair, IBM introduced the first speech recognition machine called "IBM Shoebox," which understood a whopping 16 spoken English words.

Despite the vast experience of these great institutions, the evolution of synthesized speech and recognition would be slow and steady. In addition to advancements in other technological areas, it would take another 49 years for voice to be widely adopted and accepted. Before Apple's Siri came into the market in 2011, the general population was fed up with voice interfaces, having to deal with spotty voice "options" where we would end up dialing "0" for an operator on calls made to a bank or insurance company.

Both voice interfaces (VI), historically referred to as voice user interfaces (VUI), as well as the Internet of Things (IoT) have been around for many years now in various shapes and sizes. These systems, backed by machine learning (ML), artificial intelligence (AI), and other advances in cognitive computing as well as the rapid decline in hardware costs and the extreme minification of components, have all led to this point. Nowadays, developers can easily integrate voice into their own devices with very little understanding of natural language processing (NLP), natural language understanding (NLU), speech-to-text (STT), text-to-speech (TTS), and all of the other acronyms that make up today's voice-enabled world.

In this book, we bring to light some of the modern implementations of both technological marvels, VI and IoT, as we explore the multitude of integration options out in the wild today. Lastly, we'll attempt to explain how we as makers and developers can bring them together in a single functional product that you can continue to build long after you've finished reading these pages.

We will attempt to limit general theories, design patterns, and high-level overviews of voice interfaces and IoT to Chapters 1 and 2, while focusing on actual hands-on development and implementation in the subsequent chapters. With this in mind, feel free to jump to Chapter 3 to start coding or continue reading here for a primer on voice interfaces and the Internet of Things—we'll look at how they complement each other and most importantly, some design principles to help you create pleasant, non-annoying conversational voice experiences.

Welcome to a NUI World

You've heard the news and read the articles: technology is evolving and expanding at such an exponential rate that it's just a matter of time before machines, humans, and the world around us are completely in sync. Soon, ubiquitous conversational interfaces will be all around us, from appliances and office buildings to vehicles and public spaces. The physical structures themselves will be able to recognize who you are, hold conversations and carry out commands, understand basic human needs, and react autonomously on our behalf—all while we use our natural human senses such as sight, sound, touch, and voice as well as the many additional human traits such as walking, waving, and even thinking to connect with this new, natural user interface (NUI)-enabled smart world.

NUI essentially refers to interfaces that are natural in human-to-human communications versus the artificial means of human-to-machine interfacing such as using a keyboard or mouse. An early example of a NUI is the touchscreen, which allowed users to control graphical user interfaces (GUIs) with the touch of their finger. With the introduction of devices such as the Amazon Echo or Google Home, we've essentially added a natural interface, such as voice, to our everyday lives. Apply Emotiv's Insight EEG headset and you can control your PC with your thoughts. And with a MYO armband, you can control the world around you with a mere wave of your hand. As Arthur C. Clarke declared, "Any sufficiently advanced technology is indistinguishable from magic"—and we are surely headed in that direction. Hopefully one day we will have figured out how the wand chooses the wizard.

As futurists, we dream of all the wonderful technologies that will be available in the years and decades to come. As engineers and makers, we wonder what it will take to get there. While NUIs consist of many interfaces wrapped in a multitude of technologies and sensors, this book will focus primarily on our natural ability to speak, listen, and verbally communicate with one another and machines.

In most cases, our voice is probably one of the most efficient methods of communication; it requires very little effort and energy on our part as compared to hand gestures or typing on a touchscreen. I say in most cases, because as with any technology, you want to make sure you are using the right tools for the right job. For instance, it may be great to walk into your bathroom and say "turn on the lights" but in reality, the lights should just turn on automatically as you walk in the bathroom by way of motion sensors.

Always keep the user experience of your product in mind as you're designing it. You might want to "voice all the things" for the fun of it or because it seems really cool, but if you want to make a viable product, the voice integration will need to make sense. While VI has a long road toward ubiquity, we are getting closer each day to a point where communicating with machines will be as efficient and commonplace as interacting with our friends and family.

Voice All the Things!

Back in 2010 when Allie Brosh published "This Is Why I'll Never Be an Adult" on her blog "Hyperbole and a Half," she had no idea that her meme "Clean all the things" (Figure 1-1) would be a viral sensation whose popularity has continued even seven years later in 2017. For coders, sayings like "JavaScript all the things" and "Hack all the things" are now more relevant than ever in the growing universe of the Internet of Things.

Figure 1-1. Viral meme from Allie Brosh's blog post "This Is Why I'll Never Be an Adult"

While we of course aren't going to actually "Drink all the beers" or "JavaScript all the things," this meme phenomenon does suggest that we should think about how *X* affects all the *Y*. In the case of voice interfaces, let's think about how voice can tie into the everyday objects in our lives. Let's fail fast and rule out scenarios that are too impractical to move forward with: for example, adding voice to a toilet. Wait, is that really all that strange? Maybe for a viable product it is, but if you find it intriguing then by all means, add a voice interface to your toilet, if for nothing else, the entertainment value.

What's a "Wake Word"?

Wake words are the words used to wake up a device (e.g., "Hey Siri," "OK Google," or "Alexa"). These wake words trigger a user's device to start listening and processing the subsequent commands spoken by the user. We'll talk more about wake words later in this chapter.

When adding voice capability to inanimate objects, it is important to consider how many devices within a small proximity will have voice enablement, how many of them share the same wake words, and so on. Think of it this way: imagine being in a room with a dozen voice-enabled devices, then you ask for the time and you get a response from ten different voices. How annoying would that be on a daily basis? In that scenario, ideally there would be one main voice interface that interacts with many different devices.

In thinking about today's consumer voice landscape, the everyday voice experiences include Siri, OK Google, Cortana, and Alexa. Luckily, these all have their respective wake words; therefore, having all these in the same room at the same time shouldn't be an issue unless you say something like "Hey Siri, search OK Google for Alexa." As close to the edge as edge cases can go, this one is pretty damn close but becoming increasingly annoying as more voice interfaces get introduced into the market and, therefore, more potential of clashing instead of communicating between devices is as well. Amazon and Microsoft are trying to thwart this calamity with a newly minted deal in which they will incorporate each of their AI personas on the other's platform.

Additionally, users will sometimes mix up the wake words for their various devices—for example, imagine someone pressing the button on their iPhone and saying "Hey Alexa." One of the biggest issues with voice is discoverability. Alexa has over 15,000 skills and it's very difficult to find out what is available. Even when you do find a relevant skill, it can be difficult to remember the commands. This is why most people tend to go with a single voice interface versus having to deal with remembering multiple wake words and commands.

If you are creating a brand-new device with voice capabilities, there are a ton of things you need to think about to ensure that it offers a pleasant experience. For starters, the wake word. Do you include a wake word, or will you choose not to? If you include it, what should it be? Do you build it from scratch or do you leverage an existing voice framework? These design considerations require significant planning, whether you're building a product for market or even just a new voice interface that you'd like to integrate into your own home or office.

If you want to learn more about the user experience design side of voice interfaces, check out Laura Klein's "Design for Voice Interfaces: Building Products that Talk". In her report she dives into the various dos and don'ts, nuances, and differences between working with VUI versus GUI and multimodal interfaces. Cathy Pearl's book, *Designing Voice User Interfaces*, is another great guide with tons of examples and expert advice to help with all of your design considerations.

This book will cover some existing voice frameworks and APIs that can be leveraged in your own internet-enabled device. We will explore Amazon's Alexa Skills Kit (ASK) and Alexa Voice Service (AVS) as well as Microsoft's native Speech APIs and its cloud-based solution called Cognitive Services (formerly Project Oxford). Additionally, we will quickly touch on a few other voice frameworks, including Web Speech API, Watson, and Google Speech API.

To gain a high-level understanding of how these APIs operate, we must first grasp the underlying concepts and technologies that power them. With this in mind let's quickly dive into some areas specifically around NLP, STT, TTS, and other acronyms that make it possible to program voice interfaces into our IoT devices today.

What Is NLP?

Natural language processing (NLP) uses computational linguistics and machine learning to parse text-based human language (i.e., English, Spanish, etc.) and process the phrases for specific events. In other words, it can be used to initiate commands, teach machines to find specific patterns, and so much more. NLP is primarily used in chatbots (both text-based and voice-enabled versions) as well as other applications that aid in machine learning, artificial intelligence, and human–computer interfaces (HCIs) designed for a wide array of use cases.

The biggest challenge facing NLP and AI engineers today is natural language understanding (NLU)—that is, getting the algorithms to process the differences between a metaphor versus a literal statement. Part of the problem is understanding context, discourse, relationships, pronoun resolution, word-sense disambiguation (WSD), and a whole slew of extremely complex computational challenges that are being tackled each and every day. For instance, in 2016 the Microsoft Research Advanced Speech Recognition (ASR) algorithms matched and in some cases outperformed their human counterparts with an impressive error rate of just 5.9%, according to Microsoft's Chief Speech Scientist Xuedong Huang. Compare that to everyday humans attempting to understand one another, with an error rate of 4% to a whopping 12%, depending on who you ask.

While the concepts behind NLP and NLU are important to comprehend, we will not delve into the details here; however, you will want to make sure that NLP is a key component or feature offered by at least one of the service APIs we will be plugging into our IoT devices. The two top components that complement NLP and allow us as humans to speak and listen to machines are speech-to-text (STT) and text-to-speech (TTS). STT and TTS are crucial components for providing your users with speech recognition and synthesis. We'll take a look at each of these in the following sections.

Speech-to-Text (STT)

While NLP can be used with or without actual voice input, when used with voice you would essentially be bringing in STT to parse audible words to text. While this is a required feature for converting human spoken language to text, some services might not have this available. For example, if you want to plug into one of the many chatbots available today, you might need to leverage an STT API on another service or framework to first translate the spoken words into text-based strings that you can then post to the chatbot API to get your desired end result.

STT is also known as speech recognition (SR) or advanced speech recognition (ASR). [1] Nuance offers a great example of an STT/ASR service; it is used by Siri to convert speech to plain text, which is then processed further by the Siri NLP/NLU layers to understand the user's intents.

Google's Cloud Speech API, which converts an audio stream and recognizes over 80 languages using advanced neural network models, is another excellent example of a straightforward STT service. The API sends back plain text that you can then route to one or many chatbots or NLP-based services. While it may be easy to integrate with a single full-service voice API, one thing you might want to keep in mind is this separation of concerns (SoC) in order to maintain flexibility with your devices' services integration architecture.

Here's a partial example of what a nonstreaming audio HTTP POST request to the Google Cloud Speech API might look like:

```
{
 'config': {
  'encoding': 'LINEAR16',
  'sampleRate': 16000,
  'languageCode': 'en-US'
 },
 'audio': {
  'content': speech_content.decode('UTF-8') //base64 encoded audio
 }
}
```

1 Not to be confused with speaker recognition or voice biometrics, which uses biometric signatures to identify who is speaking versus what they are saying.

Text-to-Speech (TTS)

TTS is the exact opposite direction of STT. With text-to-speech (TTS), text is synthesized into artificial human speech. Similar to STT, TTS might not come bundled with an NLP service or even an STT/ASR service. If you will be using voice for input only (e.g., to speak to a PC and have it display the results rather than verbally respond), then TTS is not necessarily required.

If you simply need TTS there are some standalone services out there today that you can leverage in your projects. For instance, with IVONA (now Amazon Polly), you can post simple HTTP requests with the text you want synthesized and you will receive an HTTP response with an audio stream that you can play back to end users. This is the same TTS service that enables the amazingly smooth voice of Amazon Alexa.

Here's a partial example of what an unsigned HTTP POST request might look like:[2]

```
POST /CreateSpeech HTTP/1.1
Host: tts.eu-west-1.ivonacloud.com
Content-type: application/json
X-Amz-Date: 20130913T092054Z
Content-Length: 32

{"Input":{"Data":"Hello world"}}
```

As you can see in this example, the phrase "Hello world" has been requested to be synthesized. The response would include an HTTP stream of the audible "Hello world" phrase. Since no additional parameters were set, the defaults for encoding, language, voice, and other variables will be used to synthesize the speech—for example, MP3 encoding in US English and so on. With this in mind, you can get up and running quickly with the default settings, then evolve the voice output by adjusting the settings as you continue to develop your voice interface.

PLS, SSML, and Other Acronyms

Aside from NLP, NLU, STT, TTS, ML, and AI, you will hear or read about Pronunciation Lexicon Specification (PLS), Speech Synthesis Markup Language (SSML), Speech Recognition Grammar Specification (SRGS), and a hodgepodge of alphabet soup in this acronym-filled field. Volumes of literature have been written for each of these individually, outlining their various technical approaches and high-level methodologies. Here we will focus on some examples of SSML, SRGS, and other meta and/or markup type of specifications as these are also applicable to developers looking to incorporate existing and flexible voice interfaces into their own IoT devices.

2 This example is taken from IVONA Software's "Signing POST Requests".

In addition to all these wonderful acronyms, we will also learn about related terminology and paradigms such as utterances, slots, domains, context, agents, entities, intents, and conversational interfaces.

To get us started we'll briefly define some of the voice-related terminology you will encounter throughout the following chapters.

Utterances

An utterance is a spoken word or phrase. In some services you will see many references to the word "utterances"—for instance, in Amazon's ASK, you are required to include some "sample utterances" in plain-text form, prefixed with their related intent, as in Example 1-1.

Example 1-1. Sample utterances in the Amazon Alexa Skills Kit

```
ToggleLightIntent turn {OnOff} lights
ToggleLightIntent turn lights {OnOff}

OrderPizzaIntent order a {Product}
OrderPizzaIntent order a {Size} {Product}
...
```

Slots and entities

Slots are the variables and arguments of the NLP world. In Example 1-1, the tokens shown with curly braces (i.e., {OnOff}, {Product}, and {Size}) are all examples of custom slots. In the context of this book, you will specifically read about these Amazon Alexa slot types such as custom slot types, custom slot values, and built-in slot types. In some cases, we will reference other services such as Wit.ai or API.AI, which refer to slot types as "entities." Entities are essentially the type of variable; for instance, the Size slot can be a custom slot type that supports any number of string values such as large, medium, or small. When choosing a service you will want to ensure that you select one that supports both a robust built-in set of slots or entities but also custom slots that you design for your own use cases.

Skills

Skills are somewhat similar to apps—in other words, with Amazon Alexa, you don't necessarily create apps in the traditional graphical sense; instead you create skills that add functionality to the Alexa ecosystem. There are a number of built-in skills like weather, music, and so on. Currently, the Alexa Skills Kit (ASK), the SDK of sorts for Amazon Alexa, includes various types of skills, including Custom Interaction Model, Smart Home Skill, Flash Briefing Skill, and List Skill.

Wake word

The wake word is used to alert the device to begin processing commands. This is an alternative to using a button or other means of activating a voice interface. For example, with the Amazon Tap, you tap a button and Alexa begins to listen. With the Amazon Echo, you say the word "Alexa" and Alexa will begin to listen. A device that uses a wake word is constantly listening, but will ignore any commands it hears before the wake word is spoken. Once the device hears the wake word, it will take all subsequent utterances and process them accordingly. For example, in the case of "Alexa, what's the weather?" the word "Alexa" is the wake word. If a user walks up to an Echo and says "What's the weather?" nothing will happen because the wake word was not invoked.

Wake words are processed on the device itself so there isn't a constant stream from millions of devices being sent to the cloud. From a privacy perspective, understanding how the technology works is important in trusting that your device isn't spying on you.

Wake words are a great way to prevent events firing off randomly. In some cases, however, you might want something that's constantly listening for specific phrases like, "we should meet Monday to discuss this opportunity further," where then a list of available times would appear on a screen or simply a digital assistant responds with "you are both available at 2p.m., would you like me to schedule the meeting for you?" and a user can simply respond yes or no.

With this in mind, it's important to strongly consider the use of wake words, especially if you have decided on an API that requires it.

Intents

The intent is essentially the requests of the user command. For example, when a user asks for the weather, the intent is to obtain weather information; sample utterances such as "What's the weather?" can be mapped to an intent named `GetWeatherIntent`. Then in your code, instead of trying to decipher the myriad ways a person can ask for the weather, you simply take the intent name the NLP service sends back to you and process it accordingly. In other words, it's a lot easier to work with `GetWeatherIntent` than it is to work with "What's the weather?" "What's the temperature?" or "What's the weather in Chicago?"; similarly, phrases such as "Will it rain today?" or "Will it rain tomorrow?" could map to `GetRainIntent` so that you can provide a different response.

Conversational

"Alexa, what's the weather?" This verbal command responds with the weather report, then the conversation is over. This is great for a single command but if you want a greater conversational experience, then you will need to map out your conversational model carefully. We will dive deeper into conversational models throughout this book, but for now keep in mind that conversational models (sometimes referred to as *conversational UI*) help elevate the user experience by providing a deeper level of conversation versus single voice commands. For instance, something as simple as following up the weather response with a "thank you," then having your conversational interface respond with "you're welcome," goes a long way in elevating the user experience.

Experience Design

We touched a bit on user experience earlier in the chapter. Let's dive a bit deeper now on how to ensure your product offers your users a pleasant experience, even if you're the only user. Surely the reason you are interested in building a voice-enabled device is to make life easier right? Or do you think it's just cool or will make you a million bucks? In any case, nobody will use the device if it's not user-friendly, so let's make sure we ask the right questions and think about the right solutions.

With this in mind, and for the purpose of this book, we'll follow a design-thinking approach to the experience design of our device. Design thinking is a human-centered design process used not only for designs but in all areas of innovation. As we go through this exercise, consider the five stages of design thinking: empathize, define, ideate, prototype, and test. We begin with the *empathize* stage, where we try to put ourselves in the shoes of the users and really try to get a feel for their lifestyle, environment, and other aspects of their lives where a voice-enabled device can be beneficial.

We then *define* the core problem discovered during the empathize stage, *ideate* on possible solutions using a variety of brainstorming techniques, *prototype* it, and finally *test* it. This is an extremely high-level explanation, so if you would like to learn more about design thinking, there are a number of great courses available on Lynda.com and Stanford University's d.school as well as numerous online reference materials that will help you get a greater understanding of design thinking.

In addition to the design-thinking process, there are some aspects of voice design that we will need to consider. Starting with Purpose. If your product has no purpose, then what's the point, right? Second, we will need to consider when to use one-off commands versus conversational dialogues. Designing conversational flow diagrams can help you ideate and think through how the conversations will flow with your voice-enabled device. We'll also dive a bit deeper on design concepts for utterances, and inflections, when to use SSML, as well as visual cues and other design considerations.

Purpose

What's the purpose of life? No, don't worry, we're not going down that philosophical rabbit hole! However, we do need to expand on that question and think about the purpose of your device's life. What problem does it solve? This is the empathize stage, during which you get a feel for the user's needs and concerns; even if that user is just you, think about how you would feel if a voice-enabled device was introduced into your environment. Think about that device being in all aspects of your life: sitting in your kitchen, laundry room, car, office, garage, pool side—wherever you are right now!

Instead of simply asking "What can I voice-ify?" another way to approach this is to think about problems people you know are having or suffering from. For example, people with dementia may have memory issues that cause them to forget their grandkids' names. Rather than feeling embarrassed by asking a spouse, they can ask a voice system, with no judgment. Here's another example to consider: because many traffic incidents today are caused by distracted driving, you could perhaps research what drivers want to be able to do the most in a car and see if voice can help.

Write down all the different ways you can think of for using voice no matter how crazy it sounds. Once you have a comprehensive list, score your ideas using whatever rating system you prefer. If you want to score based on frequency of use, for example, go through each idea and give it a score from 1 to 10 with 1 being the lowest use, potentially as low as once a month, and 10 being the highest use, meaning you would use it more, potentially every day. Go ahead and synthesize all those possibilities into one core problem with the highest score, then run with it!

One-Off Versus Conversational

When designing for voice, you can say things like, "Turn off the lights" and the lights turn off, and a voice responds with "OK" or "Done." This is considered a one-off or transactional command. One-offs are great for these types of interactions. Another good example is asking for the weather. In some cases, you would say "What's the weather?" and if contextual awareness or user preferences are available, it will simply respond with the weather in that area. If a location cannot be determined, here's where conversational comes into play. Your voice service can respond with "For which city?" and the conversation can continue from there.

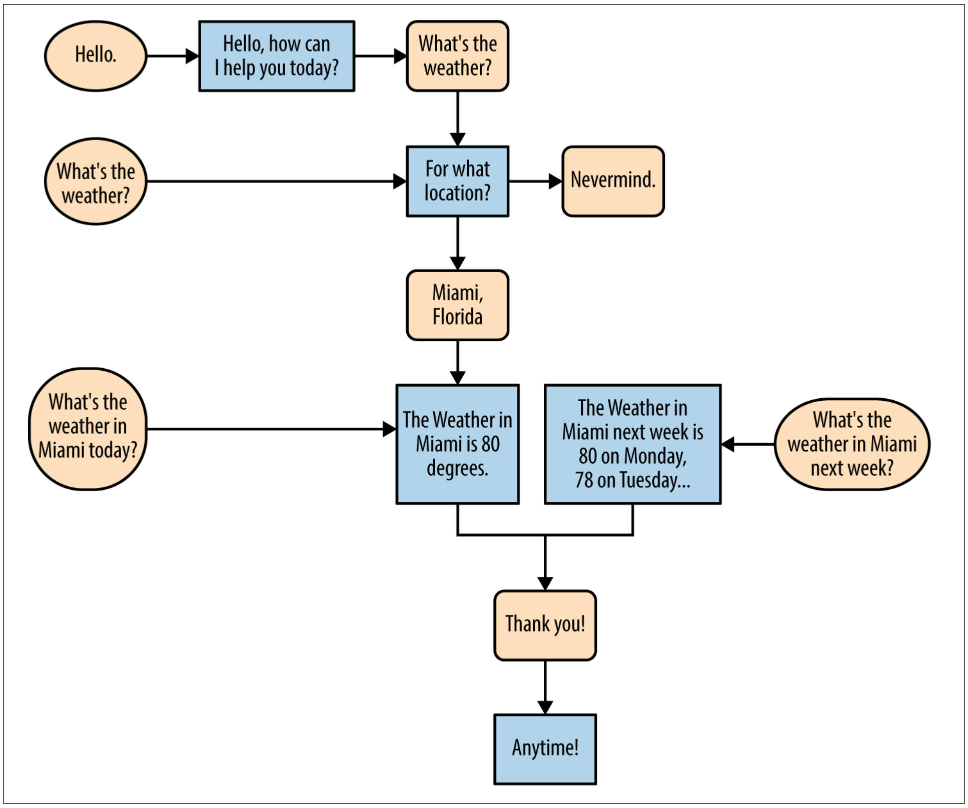
One-off commands are much easier to implement since in a nutshell, you simply receive a request and provide a response. Conversational is a bit more complex in nature, as your skill or bot needs to remember what it just asked the user. We do this by storing the state of the conversation, specifically the previous intent data. For instance, using our weather example, if a user asks for the weather, we would essentially receive a `GetWeatherIntent` payload from the NLP service where we would look for the `Location` slot or entity value. If no location is found, we store the `GetWeatherIntent` payload in a state store.

The conversational state can be stored in any database in memory or disk. We recommend a NoSQL solution where you can simply save the data model in JSON as is. Then you would simply provide the response "For which city?" back to the user. Once the user responds and the subsequent request comes in for that specific user, we would get a payload that could be called `LocationIntent` where we would access the previous request to determine why the user gave us a location to work with.

This is just one example of how conversational works. Alexa Skills Kit and some other services have a `session` object that contains additional information about the session or conversational state that you can also use to store custom parameters such as previous intent name. This helps tremendously when dealing with conversational session state. If all this seems complicated right now, don't worry; we will walk through all the code for this and more in the subsequent chapters.

Conversation Flows

Designing a great voice experience first and foremost involves creating a well-thought-out conversational flow diagram (see Figure 1-2 for an example). The idea is to map out the conversation as best you can starting with all the different ways a person can essentially start a conversation with your skill or bot. For example, a user can say, "Hello," "What's the weather?" "What's the weather in Miami?" "What's the weather in Miami next week?" and so on. Now you don't have to go crazy and list out all the different locations or time periods, just one or two examples per intent and each entity or slot.



Hello.
Hello, how can I help you today?
What's the weather?
What's the weather?
For what location?
Nevermind.
Miami, Florida
What's the weather in Miami today?
The Weather in Miami is 80 degrees.
The Weather in Miami next week is 80 on Monday, 78 on Tuesday...
What's the weather in Miami next week?
Thank you!
Anytime!

Figure 1-2. Conceptual conversational flow

Figure 1-2 shows conceptually how conversations should flow on a general voice-enabled device or service, but it does not represent specific services such as Amazon Alexa. All services are different—for example, with Alexa, in addition to the wake word, invoking the skill name is required. Note the differences between the general conceptual conversational flow diagram (Figure 1-2) and the Alexa-specific conversational flow diagram (Figure 1-3). In the Alexa example, we will call our Alexa skill `WeatherStats` and include that as well as the wake word "Alexa" in our entry points. Additionally, we will modify the response after "Alexa, start WeatherStats" to reflect the standards and best practices of Amazon Alexa for the `LaunchRequest` type.

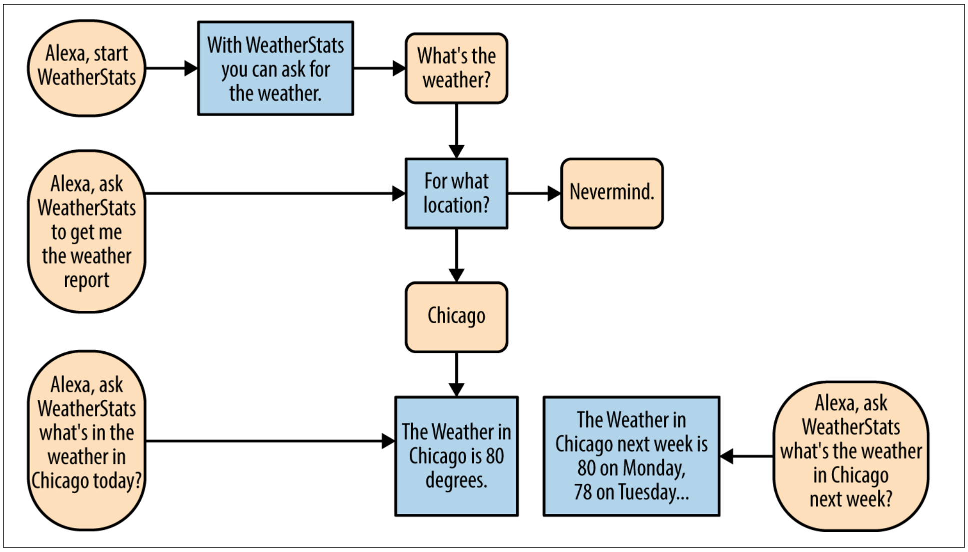

Alexa, start WeatherStats
With WeatherStats you can ask for the weather.
What's the weather?
Alexa, ask WeatherStats to get me the weather report
For what location?
Nevermind.
Chicago
Alexa, ask WeatherStats what's in the weather in Chicago today?
The Weather in Chicago is 80 degrees.
The Weather in Chicago next week is 80 on Monday, 78 on Tuesday...
Alexa, ask WeatherStats what's the weather in Chicago next week?

Figure 1-3. Alexa-specific conversational flow

You will want to create a conversational flow diagram for each skill and intent you create. Before writing any code or intent models, ask coworkers, friends, and family for feedback on each of the flows. This will provide you with great insight and ideas for responses and interactions you might have missed on your own.

Sample Utterances

Creating a list of good sample utterances is key to helping the machine learning in your service infer various ways a person can ask for something. In addition to your code, the sample utterances will also need to be iterated on and tweaked as you evolve your product. As you use your skill or bot, you might discover that certain phrases don't work as well or that users are finding new and interesting ways to invoke your skill and thus will need to be included in your sample utterances list.

Let's take a look at another example: imagine we have a ride hailing skill called `MyRide`. Here are some sample utterances for the `HailRideIntent`:

- I need a ride.
- I need a ride to the supermarket.
- I need a car to take me to the airport.
- Get me a car to the store.
- Get me a ride to the store.

As you can see, there are a multitude of ways one can ask for a ride (there are many possibilities beyond those listed here). In addition, you can add shortened single word commands where appropriate—for instance, we can simply add the word "car" to the preceding list, which would help in the case of "Alexa, ask MyRide for a car." Surely someone is bound to invoke your skill with this phrase. Additionally, you can take the words "ride" and "car," as well as "truck," "van," and "bus," and other "Ride Types" and make those slot type values. Consider the following (nonexhaustive) list, for example:

- I need a car.
- I need a ride.
- I need a bus.
- I need a truck.
- I need a van.
- I need an SUV.

We can simply replace all of this with "I need a {RideType}"—this would take care of all of those sample utterances using one sample utterance and a predefined entity or slot type. Also note how the acronym SUV is in caps. This helps the speech recognition engine match with acronyms better. Alternatively, period separation also helps, such as with "S-U-V" for instance, but using "suv" in your sample utterance could potentially cause problems (and be pronounced as "soov"). Every time you add new sample utterances, you will need to test them thoroughly to ensure they work as expected. It's important to test as you go rather than waiting until the last minute to test your utterances.

Speech Synthesis Markup Language (SSML)

Like HTML and XML, SSML is a markup language that provides information between machines and services. Where HTML provides information for web browsers to display content, SSML provides TTS engines with information on not only what to say, but how to say it. For example, consider the text "54321 go!"; with plain-old text fed to a TTS engine, you would hear something like "Fifty four thousand three hundred twenty one go!" which is most likely not what you were expecting to hear. One thing you can do to get it to actually sound like a countdown is to use the `<say-as>` SSML element:

```
<speak><say-as interpret-as="digits">54321</say-as> go!</speak>
```

Now there are some obvious workarounds here; for instance, you can easily use spaces in between the numbers to force a count down (i.e., "5 4 3 2 1 go!") but for advanced features such as modifying the pronunciation, emphasis, prosody, or sound effects, SSML is your best option. As mentioned before, every service is different and may not support all of SSML (which is a W3C specification), so make sure to read the latest SSML supported tags documentation for each service. At the time of this writing, Amazon Alexa supports the following:

`audio`
: Allows the playing of audio small clips for sound effects.

`break`
: Applies short pauses in the speech.

`emphasis`
: Changes the rate and volume of a tagged phrase.

`p`
: Represents a paragraph, which applies an extra strong break in speech.

`phoneme`
: Modifies the pronunciation of words.

`prosody`
: Allows the modification of volume, pitch, and rate of speech.

`s`
: Represents a sentence, which applies a strong break in speech.

`say-as`
: Changes the interpretation of speech.

`speak`
: The required root element.

`sub`
: Substitutes the tagged word or phrase.

`w`
: Allows you to specify words as a noun, verb, past participle, or sense.

Alexa also supports a custom Amazon-specific tag called `<amazon:effect>` that currently only allows you to add a whispering effect. This tag has a lot of potential so stay on top of the latest updates from Amazon to see if there are any new effects available. Here's an example of whispering:

```
<speak>
    Here's a secret.
    <amazon:effect name="whispered">I'm not human.</amazon:effect>.
    Mind blown?
</speak>
```

The `<audio>` element is pretty straightforward: if you want to play a sound clip, you can provide the URL to the MP3 file using the URL attribute. However, there are some restrictions: it must be an MP3 file, it cannot be longer than 90 seconds, must be hosted and served over SSL (HTTPS), the sample rate must be 16000 Hz, and the bitrate must be a low 48 kbps. If you're not an audiophile and have no idea how to get your sound effects to match these specifications, not to worry: `ffmpeg` to the rescue! Here's the command you'll need to use:

```
ffmpeg -i input-file.mp3 -ac 2 -codec:a libmp3lame
-b:a 48k -ar 16000 output-file.mp3
```

All you need to do is make sure to replace *`input-file.mp3`* with the filename that you are converting from (this can also be other formats so make sure to change the extension if necessary). Also, don't forget to change *`output-file.mp3`* to your desired output filename. Once your output file is ready, simply upload those audio files to a server that can serve those files publicly over HTTPS.

With the `<break>` tag you can add pauses of up to 10 seconds in length. This is great for dramatic effect and can be implemented using the `unit` attribute. With `emphasis`, you can accentuate words or phrases (which will result in them being spoken more slowly) as well as deemphasize them (which speeds them up a bit). This is done using the `level` attribute, which supports `strong`, `moderate`, and `reduced` values. For example, based on our previous SSML example, if you want to really accentuate (i.e., stretch out and make slightly louder) the phrase "Mind blown?" you can simply implement the following:

```
<speak>
    <emphasis level="strong">Mind blown?</emphasis>
</speak>
```

The `<p>` and `<s>` tags are straight to the point. They have no attributes and really do just represent a paragraph and sentence, respectively. As with normal reading, writing, and speech, there's a strong pause on sentences and an extra strong pause on paragraphs. In sentences, this can be accomplished using periods. In paragraphs, you can add the break tag with "`x-strong`" set on the strength attribute, but in that case simply using the `<p>` is cleaner.

`<phoneme>` is extremely helpful for words that might be spelled the same but have different meanings and pronunciations depending on the context. For instance, with the phrase "I have lead" the word "lead," by default is pronounced as [leed] "the *leader* of the country" or "she will *lead* the group on the expedition," but should actually be pronounced [led] as in "I have lead for my pencil." Here's how you would implement this fix in SSML:

```
<speak>
   I have <phoneme alphabet="ipa" ph="led">lead</phoneme>.
</speak>
```

Another one of my favorite examples solves for how folks in different cultures or regions pronounce the same word in different ways, as with the word "tomato" in the famous phrase, "You say *tomayto*, I say *tomahto*!" Here's how to get the correct pronunciations using SSML while maintaining the correct spelling of tomato:

```
<speak>
   You say <phoneme alphabet="ipa" ph="təˈmeɪtəʊ">tomato</phoneme>,
   I say <phoneme alphabet="ipa" ph="təˈmɑːtəʊ">tomato</phoneme>!
</speak>
```

Unless you have either the International Phonetic Alphabet (IPA) or the Extended Speech Assessment Methods Phonetic Alphabet (X-SAMPA) committed to your own memory, the hard part here is finding the correct phonetic transcription (i.e., /tə ˈmeɪtəʊ/ in this example). Luckily you can simply search the web for a phonetic translator and you'll be able to use it to translate just about any word to its phonetic form.

Next we have `<prosody>`, which allows you to control the pitch, rate, and volume of a phrase. For instance, set the pitch attribute to `x-low` and you'll hear a deep voice, but set it to `x-high` and you have what sounds like one of the Chipmunks. With the `rate` attribute set to `x-slow` or `x-high`, you have extremely accentuated and stretched-out words or a voice at an extra high rate of speed that resembles an auctioneer. With the `volume` attribute, you have values from `silent`, `x-soft`, `soft`, `medium`, `loud`, and `x-loud`, where `silent` completely blanks out the phrase (which is great for censoring) and `x-loud` increases the tone of the voice (which is great for expressing excitement or anger).

With `<say-as>`, as with our countdown example earlier where we specified `digits` in the `interpret-as` attribute, we can also specify `cardinal`, `ordinal`, `spell-out`, `fraction`, `unit`, `date`, `time`, `telephone`, `address`, `interjection`, and `expletive`. When working with dates, there's one additional attribute you can specify—the "format" attribute:

```
<speak>
   <say-as interpret-as="date" format="md">3/26/2020</say-as>
</speak>
```

In this last example you will hear "March 26th" without the year verbalized. To also hear the year, you can specify `mdy` as the format and you're all set.

Now, remember to use `<speak>` as your root element on all your SSML requests and if you want to substitute a word while keeping your text intact, use the `<sub>` tag with the `alias` attribute set to the word you want to hear. This is particularly helpful when using symbols or acronyms—for instance, pronouncing the word "magnesium" when using the symbol "Mg" or saying "pound" instead of the word "hash" in the phrase "Press #".

Lastly, if you need to pronounce a word in its past tense or accentuated as a verb versus a noun, then you can use the `<w>` tag, which has one attribute called `role`. With `role`, you can specify "`amazon:VB`" for verbs, "`amazon:VBD`" for past participle, "`amazon:NN`" for nouns, and "`amazon:SENSE_1`" for the nondefault sense of the word. For example, the word “bass” defaults to the low musical range pronunciation and not the fish. If you want to pronounce it as the fish, then you can either use the `<phoneme>` tag or use the `<w>` tag as follows:

```
<speak>
I like <w role="amazon:SENSE_1">bass</w>.
</speak>
```

Other services may support the other SSML elements, which includes `token`, `voice`, `mark`, `desc`, `lexicon`, `lookup`, and other elements as well as possible custom elements as with `<amazon:effect>` so make sure to follow up with those services individually to stay on top of what is and isn’t available.

When to Use Visual Cues

Imagine if the Amazon Echo had no LED light ring: you’d shoot off a command, and seconds might go by with no response. Then, just as you go to ask it what’s taking so long, the device replies. That could be a bit frustrating. To ease that frustration, you really need to use some kind of visual cue like an LED light or a progress indicator in a graphical display to show that something is happening in the background.

The path of least resistance is to use an RGB LED, which has four pins: an anode each for red, green, and blue, and a long cathode pin for your standard ground (Figure 1-4). You can modulate any of the three RGB pins individually to get any one of those three colors or modulate them together to mix the colors and expand your palette to include over 16 million RGB-supported colors. This is done by essentially dimming some colors while brightening others. For instance, the value 0 means off and the value 255 means extremely bright. If we were to set R=80, G=0, and B=80, we would have the color purple.

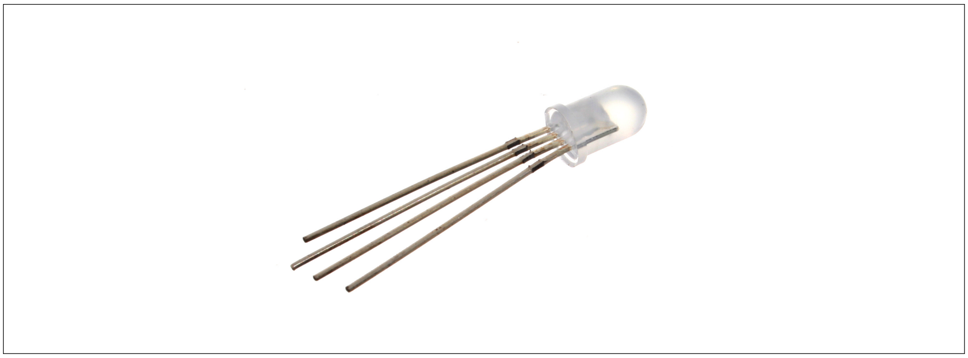


Figure 1-4. RGB LED

If your product has a screen display, the best option is to include some kind of progress indicator (e.g., a status bar or circle, or a glowing light), such as those used in mobile applications or websites when content is being fetched. Avoid making the indicator overbearing or fast—it should be subtle and slow.

In other cases, you might want to show the user an image but the device itself does not have a screen, as with the Amazon Echo. For cases like this, you can create a mobile app that shows images, email or use SMS to send images, or simply display images on a nearby connected device. This isn't necessarily a visual cue for process indication per se, but worthy of mentioning here since it can be used to enhance the experience.

Additional Design Considerations

In addition to the design considerations just outlined, you'll want to put some thought into the design of the first-time setup and configuration process as well as the aesthetics of the product itself, its packaging, and documentation. Although you might not spend a lot of time on these pieces during the early R&D and prototyping stages, they are critical to deploying your product to production and on-boarding new users.

Also, a feedback loop with your early adopters will help you evolve and improve the product. For this you can log data such as most requested intents to unknown intent matches and work on improving those intents and handle unknown intents. In addition to automated logging, receiving verbal feedback directly through the device is a great way to capture what's on your users' minds as they use the product in near real time. You might program the device to respond to unknown intents with something like "Sorry, I didn't understand your command. I have logged it for further analysis. Are there any other ways I can improve this service?" and allow the users to respond with their thoughts.

Additionally, notifications can be sent to users where they will be prompted to start a survey, and if they agree, they will be asked a few questions designed to capture feedback on key aspects of your product. Then you can use this feedback to make improvements as you iterate on your product and voice designs. However, be careful with notifications: you don't want to simply blurt out the message as users may not be around to receive it. Visual cues (e.g., an orange glowing LED or a message icon on a screen) are a great way to capture users' attention so that they can listen to those notifications when they are ready.

This brings us full circle to empathy in design thinking. The main point to remember when designing your product is your audience. How will interacting with this device make your users feel? You'll want to design a product that will improve your users' lives (even if you are designing a product just for yourself) rather than making things more complicated.

Decisions, Decisions...

In the next chapter you will find many references to several currently available services and frameworks. In subsequent chapters, we will narrow our focus on Amazon's Alexa Voice Services (AVS) and provide step-by-step instructions for integrating AVS into your own IoT device, specifically using rapid prototyping hardware such as Raspberry Pi. However, depending on your circumstances, AVS on a Raspberry Pi might not be the right choice for you. For example, your business rules may call for just STT, or require the use of Microsoft products. In some cases, you may want your own backend instead of sending your audio streams to a third party, all while running on your own custom-designed PCB.

While we have tried our best to show how to integrate just about any RESTful HTTP voice service into your own IoT device, ultimately you have the freedom to decide which voice service and hardware to go with. With that in mind, continue on to the next chapter to learn more about some of the great voice services available out in the wild today.

CHAPTER 2

Existing APIs and Libraries

Although digital voice interfaces have been around for several decades, we are really just now at the cusp of advanced voice interfaces, especially in the realm of Voice-as-a-Service (VaaS) and intelligent, near human-like voice-enabled virtual assistants. In this chapter, we will review some of the services currently available, including Amazon Alexa, Microsoft Cognitive Services, Cortana, and Google Cloud Speech API.

While there are plenty of services to choose from these days, they all offer unique functionality that you will want to consider when building out your own service or device. Some are easier than others to implement, but aren't as flexible or customizable (yet may be sufficient for your use case). For example, Amazon Alexa might be a great choice for a virtual assistant with a lot of pre-packaged features, but if you're looking for more straightforward speech to text, Nuance Mix or Amazon Lex may be a better option.

In addition to exploring some of the services available today, we will also touch a bit on some technical architectures, both those you might encounter as well as ones you might build while designing your own project. To start things off, let's take a deeper look at Amazon Alexa.

Amazon Alexa

With the launch of the Amazon Echo in November 2014, Amazon helped propel home-based voice interfaces into the mainstream: using its cloud-based voice assistant "Alexa," anyone within proximity of the device can make a multitude of requests without lifting a finger. The Echo really stands as an IoT hub type of device in which it's connected to the internet and can send commands to other IoT devices. In adding a voice interface, Amazon has successfully accomplished integrating both voice and

IoT together into a simple, non–screen-based home appliance that just about anyone can use (Figure 2-1).

Figure 2-1. Amazon Echo 2014

In developing the Echo and Alexa, Amazon immediately recognized the value in developing a rich, third-party developer community by exposing some extremely simple, HTTP RESTful APIs and developer tools. This allowed Amazon to integrate with services such as Uber, Nest, IFTTT, Insteon, Wemo, SmartThings, and many others. In turn, this gave end users a broader selection of voice-enabled services.

Additionally, Alexa and Echo are actually two separate offerings: Alexa is a cloud-based voice service while Echo is a Bluetooth and WiFi device that connects to Alexa. This also allows Amazon to connect Alexa to additional propietary devices (e.g., the Amazon Fire product line, Echo Dot, and Amazon Tap) as well as third-party hardware makers (e.g., Nucleus, CoWatch, and Triby).

Alexa Skills Kit (ASK)

As a developer, you can also take advantage of the Amazon APIs and make similar products. Currently, there are two main Alexa developer toolkits: the Alexa Skills Kit (ASK) and Alexa Voice Service (AVS). The ASK is what you use to create skills for Alexa. Skills are essentially voice apps; for instance, when you say, "Alexa, ask Uber for a ride," the word "Uber" refers to the Uber skill. Just like a human, the more skills Alexa learns, the smarter she gets.

The ASK supports several request types. The first one we'll look at is the `LaunchRequest`, which can be used to open a dialogue with the user. When a user asks for a skill using the wrong command—for example, "Alexa, start Uber" instead of "Alexa, ask Uber for a ride"—you could program the device to provide information on how to use the skill, suggest possible commands or questions one could ask, then follow up with a question on what the skill should do next in order to keep the conversation going or execute a command. Here's an example of a basic skill request of type `LaunchRequest` posted from Alexa to a third-party endpoint:

```
POST / HTTP/1.1
Content-Type : application/json;charset=UTF-8
Host : your.application.endpoint
Content-Length :
Accept : application/json
Accept-Charset : utf-8
Signature: [ omitted for brevity ]
SignatureCertChainUrl: https://s3.amazonaws.com/echo.api/echo-api-cert.pem

{
  "version": "1.0",
  "session": {
    "new": true,
    "sessionId": "amzn1.echo-api.session.0000000-0000-0000-0000-00000000000",
    "application": {
      "applicationId": "amzn1.echo-sdk-ams.app.000000-d0ed-0000
-ad00-000000d00ebe"
    },
    "attributes": {},
    "user": {
      "userId": "amzn1.account.AM3B00000000000000000000000"
    }
  },
  "context": {
    "System": {
      "application": {
        "applicationId": "amzn1.echo-sdk-ams.app.000000-d0ed-0000
-ad00-000000d00ebe"
      },
      "user": {
        "userId": "amzn1.account.AM3B00000000000000000000000"
      },
      "device": {
        "supportedInterfaces": {
          "AudioPlayer": {}
        }
      }
    },
    "AudioPlayer": {
      "offsetInMilliseconds": 0,
      "playerActivity": "IDLE"
    }
  },
  "request": {
    "type": "LaunchRequest",
    "requestId": "amzn1.echo-api.request.0000000-0000-0000-0000-00000000000",
    "timestamp": "2015-05-13T12:34:56Z",
    "locale": "string"
  }
}
```

As this example demonstrates, Amazon sends over a straightforward JSON-formatted HTTP POST request where the receiving endpoint simply processes it and replies with a JSON-formatted response containing plain-text or SSML-formatted strings for Alexa to read back to the user. Recently, Amazon added the ability to send back a URL to an MP3 file where the calling Alexa-enabled device can then stream back and play for the user to listen to.

Alexa Voice Service (AVS)

The other toolkit, AVS, is what you can use to connect your hardware to Alexa. For instance, you can start prototyping immediately with a Raspberry Pi 3, a microphone, and a speaker. Drop in some AVS code, and there you have it: your own limited-edition Echo. It's "limited edition" not just because there is only one, but because the Echo has some additional hardware such as its seven-microphone array, which greatly improves recognition performance. You will still have the ability to ask for things like the weather and time, and all the other skills Alexa has to offer.

Some of AVS's limitations are geographical. For example, at the time of this writing, iHeartRadio, Kindle, and traffic reports are not available in the UK or Germany. Also, availability on some third-party developer Alexa skills made with the Alexa Skills Kit are also based on geographic location. In addition to geographic limitations, you will want to read the latest version of the AVS Functional Design Guide for the most up-to-date best practices on going to production with your device. For example, you can currently only use "Alexa" as the wake word if you are developing a voice-initiated product.

Another way Amazon and its partners are helping make voice enablement easier is by offering Development Kits for AVS. In one example, Amazon has partnered with Conexant to make available the Conexant DS20924 AudioSmart 4-Mic Development

Kit for AVS (shown in Figure 2-2), which allows developers to rapidly prototype an Alexa voice-enabled device. With a price tag of $349, the kit doesn't come cheap, but it includes some powerful features such as four microphone far-field voice interaction with 360-degree Smart Source Pickup and Smart Source Locator, full duplex Acoustic Echo Cancellation (AEC), as well as Conexant's CX22721 Audio Playback CODEC for optimal audio quality.

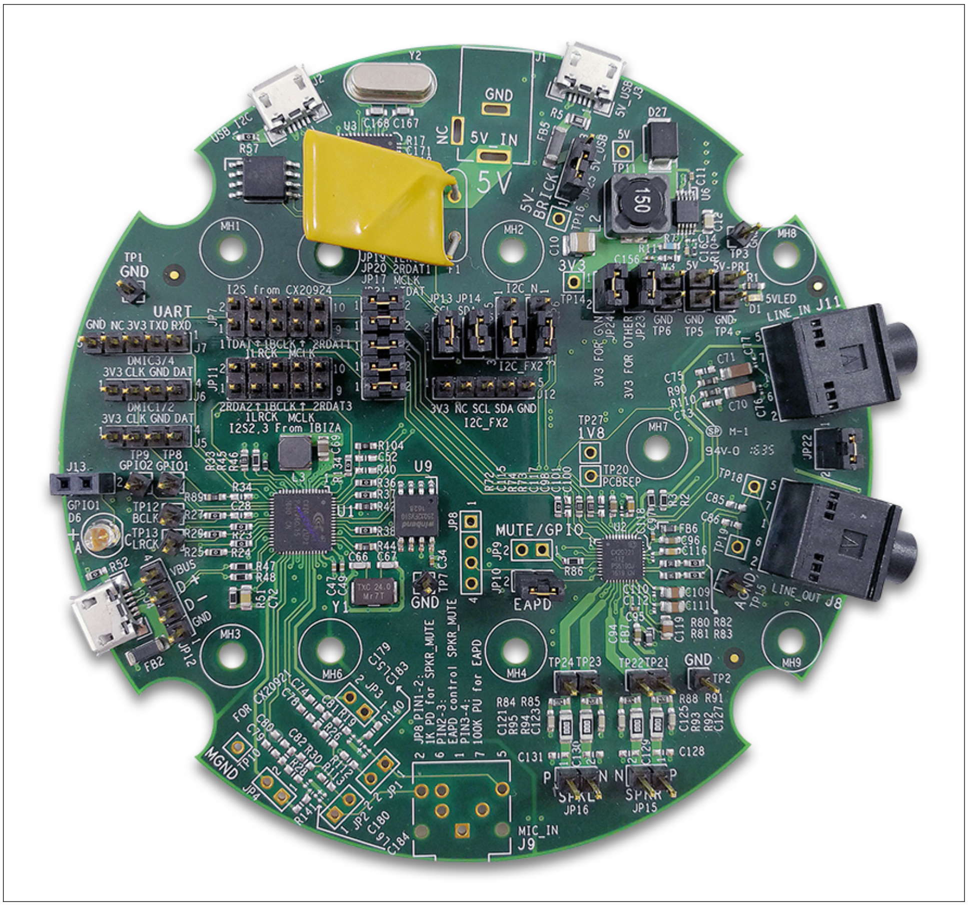


Figure 2-2. Conexant DS20924 AudioSmart 4-Mic Dev Kit for AVS

Conexant also has a 2-mic version for $299 that's price comparable to another AVS kit called Microsemi's ZLK38AVS AcuEdge Development Kit for AVS (shown in Figure 2-3), which also has a 2-mic array. The AcuEdge also supports beam forming, two-way communication, trigger word recognition for hands-free support, as well as Smart Automatic Gain Control. Additionally, it's square versus round and is designed as a Raspberry Pi Hat, which makes for a great Pi topping. In other words, it's designed so the pins plug directly into the Pi for a nice and snug fit.

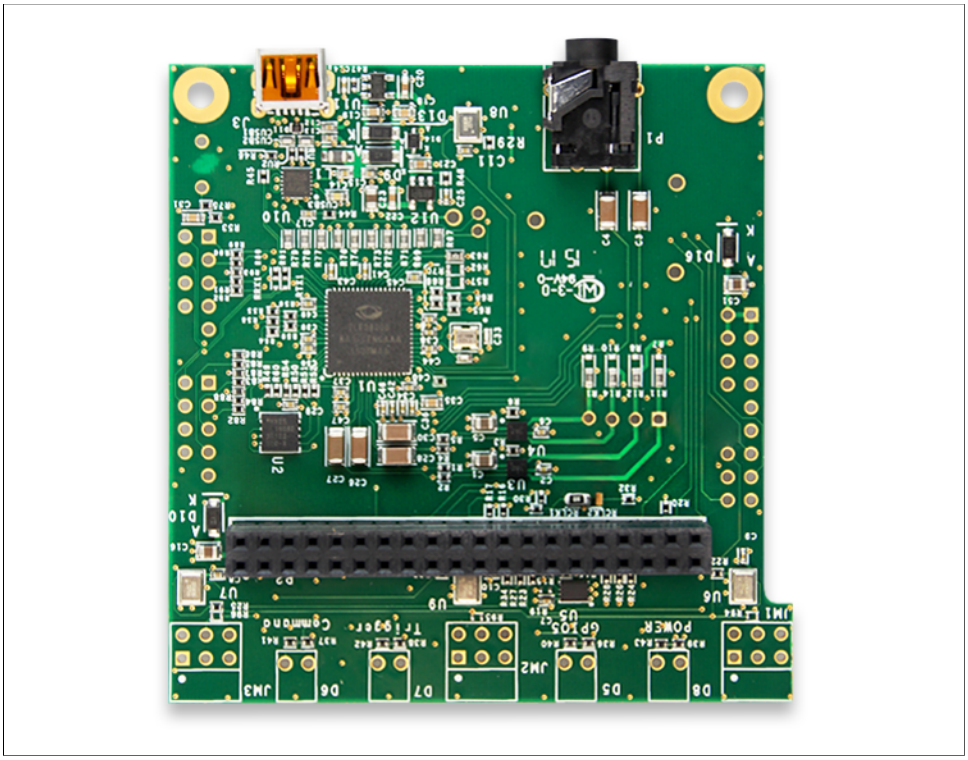


Figure 2-3. Microsemi's ZLK38AVS AcuEdge Development Kit for AVS

If you're on a tight budget, the Matrix Voice (shown in Figure 2-4) is another solid choice. The Matrix Voice is not a listed Amazon AVS–compatible device, but is designed with AVS in mind. It may be cheaper in price, but packs a punch in quality with a 7-microphone array, open source, and an 18 RGB LED ring array. Additionally, it not only supports AVS but also Microsoft Cognitive Services, Google Speech API, Houndify, and others. There's also a standalone version with an embedded ESP32, which is a WiFi- and Bluetooth-enabled 32-bit microcontroller.

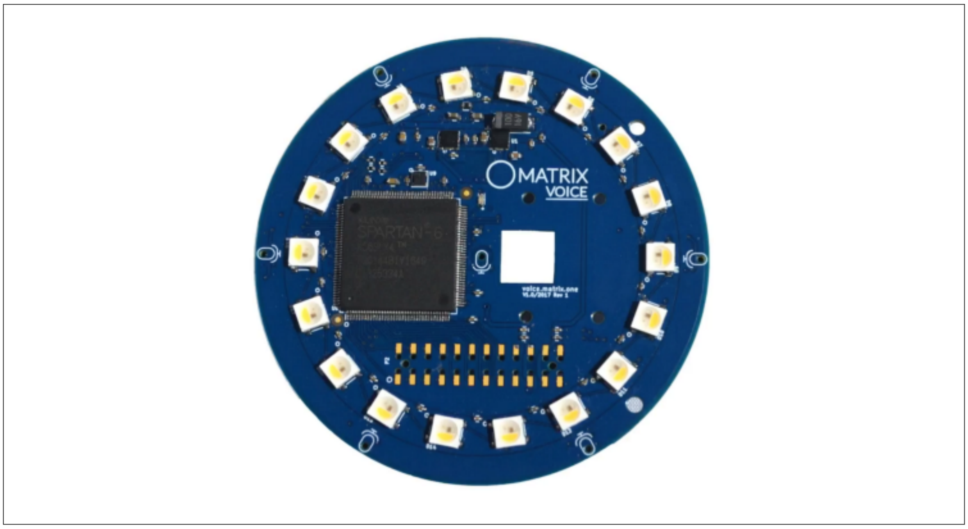


Figure 2-4. Matrix Voice Open Source Dev Board

The three official Amazon AVS dev kits, in addition to the Matrix Voice, all work with Raspberry Pi with undoubtedly more options on the way. While the added cost of the kits along with the existing cost of the Pi and any other components you want to include are exponentially higher, it may be the best option for aspiring hardware engineers just starting out to reduce the learning curve time and get hands-on experience with what the pros are bringing to market. If making your own Alexa-enabled device sounds like something you want to do, you're in luck! We will take a deep dive into the ASK and AVS starting in Chapter 3 so make sure your workbench is ready to rock.

Amazon Lex

At AWS re:Invent 2016, Amazon announced the opening of additional APIs to help drive the future of conversational interfaces. These APIs aren't necessarily part of the Alexa offerings, but are part of the APIs that actually power Amazon Alexa. Among these newly released APIs is Amazon Lex, Alexa's deep learning ASR and NLU engine. Now abstracted and available for developers to integrate into our own applications, we can leverage the power of Lex and get even more flexibility out of the Amazon AI stack.

One of the great features Lex offers is the ability to take both plain-text and audio streams as input, whereas some services out there may only take in plain text while others, such as AVS, only take in an audio stream. However, that doesn't necessarily mean that Lex is the right solution for you; instead, you might want to leverage all the skills available in AVS such as weather, music, calendar, or the many thousands of other third-party skills available today.

Amazon Polly

While Amazon Lex will help you convert natural language from text and voice to intents, Amazon Polly will help you convert plain-text sentences to voice. In other words, Polly is the TTS engine that gives Alexa her voice. By leveraging Polly as your TTS layer, you can take advantage of additional features such as a selection of 47 near natural sounding voices in over 24 languages.

With Polly, you can submit plain text or SSML where you can control the pronunciation, volume, pitch, and speech rate. The output synthesized speech can be encoded in MP3, Ogg Vorbis, or PCM (audio/pcm in a signed 16-bit, 1 channel [mono], little-endian format). This is specified by setting the `"OutputFormat"` parameter when posting a request to the Amazon Polly service.

Here is an example of the request structure:

```
POST /v1/speech HTTP/1.1
Content-type: application/json

{
   "LexiconNames": [ "string" ],
   "OutputFormat": "string",
   "SampleRate": "string",
   "Text": "string",
   "TextType": "string",
   "VoiceId": "string"
}
```

As you can see in the preceding example, in addition to the required `OutputFormat` parameter, there are additional required as well as optional parameters that you can use to get your desired voice response. Take the optional parameter `LexiconNames`, for instance. Here, you can preset lexicons to customize the pronunciation of words. These lexicons must conform to the Pronunciation Lexicon Specification (PLS) by the W3C. This is a great way for transforming l33t speak to voice.

With the optional `SampleRate` parameter, you can optionally set the audio frequency in Hz to `8000`, `16000`, and `22050` for MP3 and ogg (it defaults to `22050`). For PCM, you can specify `8000` and `16000` (the default is `16000`). Another optional parameter is `TextType`, which by default is `text`. This should be set to `ssml` if you are sending in SSML-structured text in the required `Text` parameter, which is where you would include your plain text or SSML to be synthesized.

Finally, with the required `VoiceId` parameter, you will need to specify which voice you would like used for the synthesis. There is an endpoint you can submit a GET request to in order to receive the latest list of voices, their respective IDs, and other information. Consult the documentation for further details, including up-to-date API information.

Microsoft Cognitive Services, Cortana, and More

Another company with a long history of voice-based offerings is Microsoft, which released its Speech Application Programming Interface (SAPI) with the initial rollout of Windows 95. SAPI included a basic form of TTS and STT that allowed developers to add voice into their Windows-based applications. This included features like voice dictation for word processors and screen readers for the visually impaired.

Fast-forward to 2014 and Microsoft has publicly released Cortana, its digital personal assistant. Cortana is the culmination of years of research, development, and acquisitions, giving Microsoft an AI personality that competes with Apple, Google, and Amazon in the voice and AI space. But Microsoft faced the same problem as Apple: its digital assistants were only as smart as its developers made them. In order to make them smarter, they needed to open up.

To expand the ecosystem further, in 2016 Microsoft opened up the APIs that make up Cortana via Cognitive Services. This cloud-based offering includes speech as well as other services around vision, language, knowledge, and search. Under speech and language alone you will find APIs for Translator, Bing Speech, Bing Spell Check, Speaker Recognition, LUIS, Linguistic Analysis, Text Analytics, and Custom Speech Service. Additionally, as an alternative to Alexa, Microsoft announced the release of Cortana Skills Kit, which allows third-party developers to essentially make Cortana smarter.

With the Cortana Skills Kit, developers create bots using the Microsoft Bot Framework, which can plug into LUIS for natural language understanding. The Microsoft Bot Framework has a pretty robust API that you can simply import into your C# or Node.js project. Here's an example of what a basic endpoint looks like in a C# `ApiController` class:

```
public class MyCortanaSkillController : ApiController
{
    public async Task<HttpResponseMessage> Post([FromBody]Activity activity)
    {
        var connector = new ConnectorClient(new Uri(activity.ServiceUrl));
        var reply = activity.CreateReply();

        var qnaService = new QnAMakerService(new QnAMakerAttribute(
            WebConfigurationManager.AppSettings["MyAppKey"],
            WebConfigurationManager.AppSettings["MyAppAId"]));

        var result = await qnaService.QueryServiceAsync(activity.Text);
```

```
            reply.Speak = (result.Answer != null && result.Answers.Count > 0)
                    ? result.Answers[0].Answer
                    : "Sorry, I didn't get that.";

            await connector.Conversations.ReplyToActivityAsync(reply);

            return Request.CreateResponse(HttpStatusCode.OK);
        }
    }
```

In this example, we create a class called `MyCortanaSkillController` that inherits `ApiController` from the Web API framework. Then we create the endpoint method that accepts a `POST` request of type `Activity`. The `activity` object contains a `ServiceUrl`, which we use to create a callback and we generate a reply object using the `activity.CreateReply()` method. Note the `QnAMakerService` object being referenced here. It's not required, but we are using it in lieu of LUIS to show that we can essentially hit any NLP/NLU service we want when the Cortana spoken text or text input from another channel such as Slack or SMS, for example, comes in the `activity.Text` property. In other words, we can use Cortana with API.AI, Wit.ai, Amazon Lex, or even Watson if we wanted to.

Once we get the results back from the NLP/NLU service (in this case, QnA Maker, Microsoft's question-answering, domain-specific NLU service), we can set the `reply.Speak` property to the response and send back to the connector to reply to Cortana or wherever the initial request came in from. While we won't look at Cortana or Bot Framework in great detail, we will explore similar logic in later chapters using Windows IoT Core.

Google Cloud Speech API

In 2016, Google announced the beta release of its Cloud Speech API, available on the Google Cloud Platform, and the results are impressive. With one of the most advanced neural networks and machine learning algorithms, it supports over 80 languages, advanced noise cancellation and signal processing, as well as streaming recognition and word hints for context-aware recognition.

The biggest caveat with the Cloud Speech API is that it's only one side of the coin—it's an STT or ASR API and does not support voice synthesis or TTS at this time. Additionally, while audio under 60 minutes is free, any processing over 60 minutes will incur a small charge. Lastly, it does not have NLP built into it. For this, Google offers a separate Cloud Natural Language Processing API, which takes text input and extracts information that you can use for text analysis as well as understanding sentiment and intents.

Here's an example of what a request to the "recognize" RESTful endpoint on the Google Cloud Speech API looks like:

```
{
  "config": {
      "encoding":"FLAC",
      "sampleRateHertz": 16000,
      "languageCode": "en-US",
      "profanityFilter": true
  },
  "audio": {
      "uri":"gs://cloud-speech-samples/speech/utterance.flac"
  }
}
```

The interesting thing to note here is the FLAC encoding type. Google recommends using FLAC (which stands for Free Lossless Audio Codec) because it does not compromise the voice recognition due to its high quality. However, Google does allow for 16-bit Linear PCM (`LINEAR16`), PCMU/mu-law (`MULAW`), Adaptive Multi-Rate Narrowband at 8,000 hertz (`AMR`), Adaptive Multi-Rate Wideband at 16,000 hertz (`AMR_WB`), Opus-encoded audio frames in Ogg container (`OGG_OPUS`), as well as Speex Wideband at 16,000 hertz (`SPEEX_WITH_HEADER_BYTE`).

Once the request has been posted, the response might look something like this:

```
{
  "results": [
    {
      "alternatives": [
        {
          "transcript": "say hello to Siri and Alexa for me",
          "confidence": 0.98534203
        }
      ]
    }
  ]
}
```

Based on the response, you can then run the "transcript" value through a text-to-speech engine and play the resulting audio through a speaker. In addition to posting audio files, Google also allows for streaming audio via the open source gRPC, which is a high-performance, universal remote procedure call (RPC) framework.

Other Notable Services

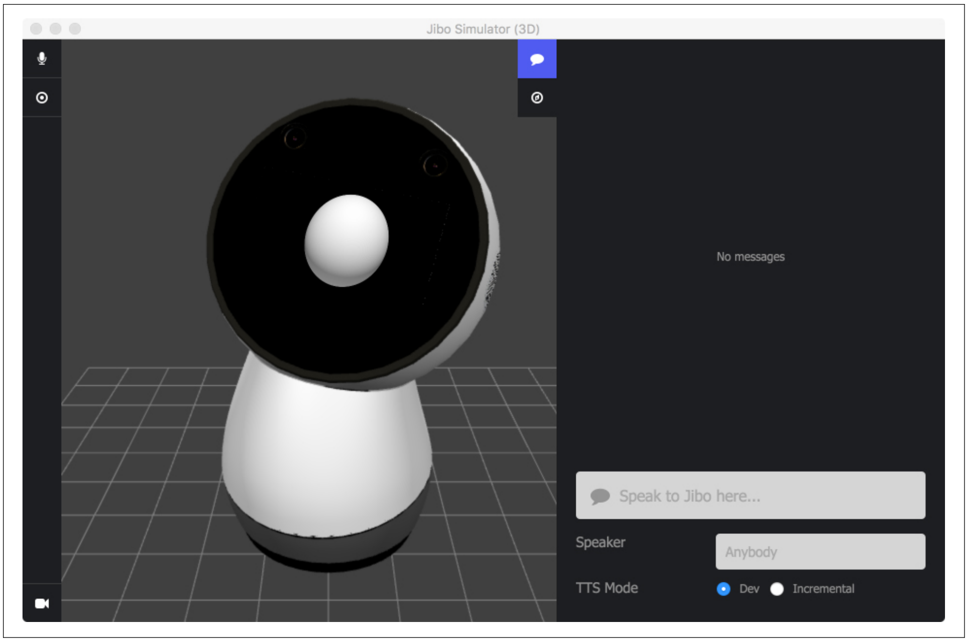
In September 2016, Google acquired API.AI, which was another startup in this growing field of NLP/NLU services. API.AI at the time had a community of over 60,000 developers using its tools and services to create a conversational experience for both voice and text. This gave Google an instant API with a loyal following to create skills (or what Google calls actions) for its Google Home and Google Assistant offerings.

In addition to the powerful NLP/NLU toolset, another key feature that attracted developers to API.AI in the first place was the built-in support for multiple-domain–specific functionality and services such as weather, math, time, calendar, and more. Additionally, built-in support for communications applications such as Facebook Messenger, Slack, Kik, and others made it extremely efficient to deploy chatbots to a broader audience.

Another great service is Wit.ai, which was acquired by Facebook and boasts over 65,000 developers. While there are some differences in built-in capabilities and overall experience (e.g., the number of built-in elements, domains, and whether or not you need an open source private solution), it really comes down to a matter of preference and your requirements. Features change daily so if you decide to go down this route, do some research to see what the latest offerings are and give these services a test run to see what works for your use case.

In addition to Wit.ai and API.AI you will want to check out IBM Watson and Watson Virtual Agent as well as tools such as Jasper, PocketSphinx, Houndify, and Festival. You should also check out the latest offerings from Nuance and be on the lookout for startups such as Jibo, for example. Jibo is an interesting offering in that it's an actual physical robot that moves, blinks, and reacts physically to voice input and output.

While at the time of this writing Jibo isn't publicly available, there are tools developers can download such as the Jibo SDK, which has Atom IDE integration, as well as a Jibo Simulator (shown in Figure 2-5), which is great for visualizing how your code would affect Jibo and how users can engage with the robot.



Speaker: Anybody
TTS Mode: [✓ Dev] Incremental

Figure 2-5. Screen grab of Jibo Simulator

Technical Architecture

Now that we are familiar with the landscape and have a good idea of what to look for in terms of a voice interface service offering that we can plug into, let's quickly highlight some of the architectural aspects of integrating with these services. Because architectures vary from service to service we won't go into great detail here, but we will cover some common conceptual and application architecture approaches to help you with integrating a voice service with your home-brewed IoT device.

Conceptual Architecture

For a high-level understanding of how voice-enabled devices handle voice commands, we'll turn to conceptual architecture. Figure 2-6 illustrates a basic architecture that conceptually visualizes how a command or utterance flows from a user through to the intent handler back out to the user and any additional graphical displays.

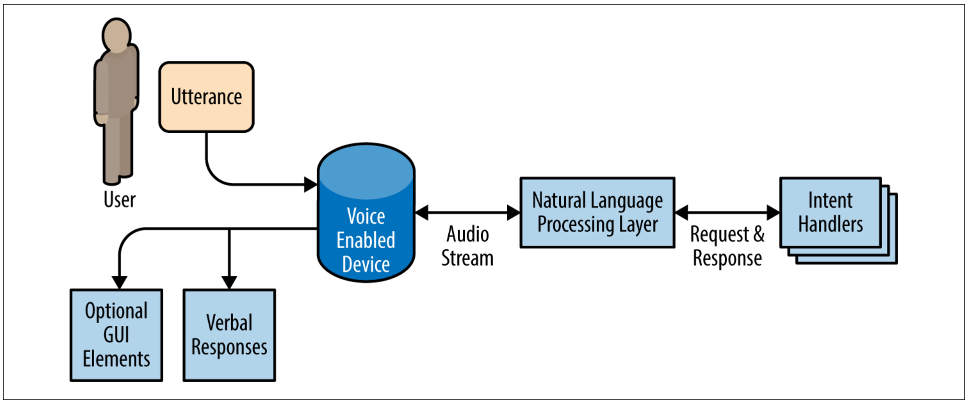


Figure 2-6. High-level conceptual architecture for voice-enabled devices

Let's break this diagram down a bit further. We begin with the user providing an utterance or verbal command that flows to the voice-enabled device by way of a microphone. The microphone sends the audio to the audio processor within the device, then sends an audio stream to the NLP layer (which can either live on the device or in the cloud). The NLP layer parses the audio via speech-to-text (STT), and then maps the text to an intent. As mentioned earlier, in some cases the STT component may be part of a separate service altogether. In either case, it is then routed to its appropriate intent handler for processing.

Inside the intent handlers can live all sorts of business logic for any number of domains. For instance, a weather intent handler can send a request to a weather service for weather data, then return the response in human-readable form where the TTS engine can then convert it to audio form.

To put this all into a bit more context, let's take a look at the Amazon Alexa version of this conceptual architecture diagram (Figure 2-7).

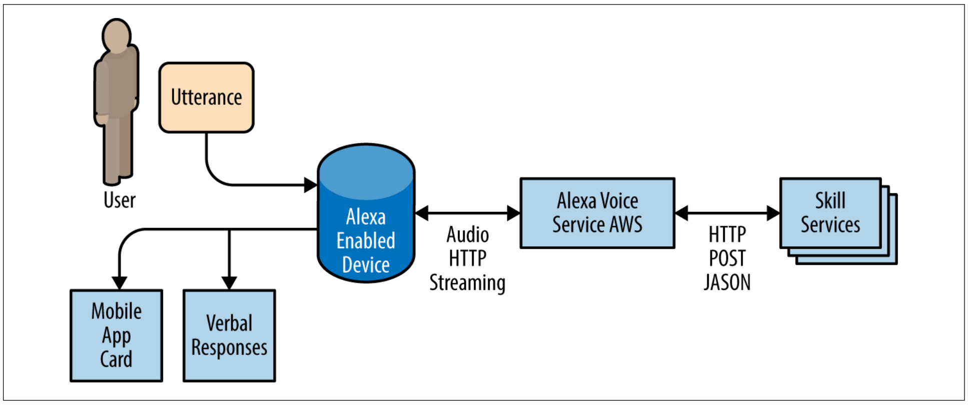


Figure 2-7. Conceptual architecture for Amazon Alexa

Figure 2-7 shows a similar flow as Figure 2-6, with the user utterance going to a voice-enabled device, in this case, an Alexa-enabled device like the Amazon Echo or Dot, for example. The audio stream is then pushed via HTTP streaming to the Alexa Voice Service on AWS for STT and NLP processing. There, the intent is determined and routes the request to the appropriate skill service via a REST HTTP POST request in JSON format. This can be an internal Amazon service or a skill produced by a third-party developer by specifying the endpoint URL in the Amazon Developer Console, the details of which should be outlined in an application architecture.

Keep in mind that the conceptual architecture examples shown here are as basic as it gets. Based on your own product, you will want to add to or modify the architecture as you begin to think about the functions of your device. For example, you'll first want to determine whether your device will use a screen, LEDs, additional protocols or services, and so on. Once you have a solid conceptual architecture in place for how you conceive your own device, then you can move on to designing a solid application architecture.

Application Architecture

You will most likely need multiple application architectures, but those architectures may vary depending on your device. However, at minimum, you will need to focus on what your hardware architecture, embedded software architecture, and web services architecture might look like. Figure 2-8 illustrates an application architecture example for web services that you can use as a foundation.

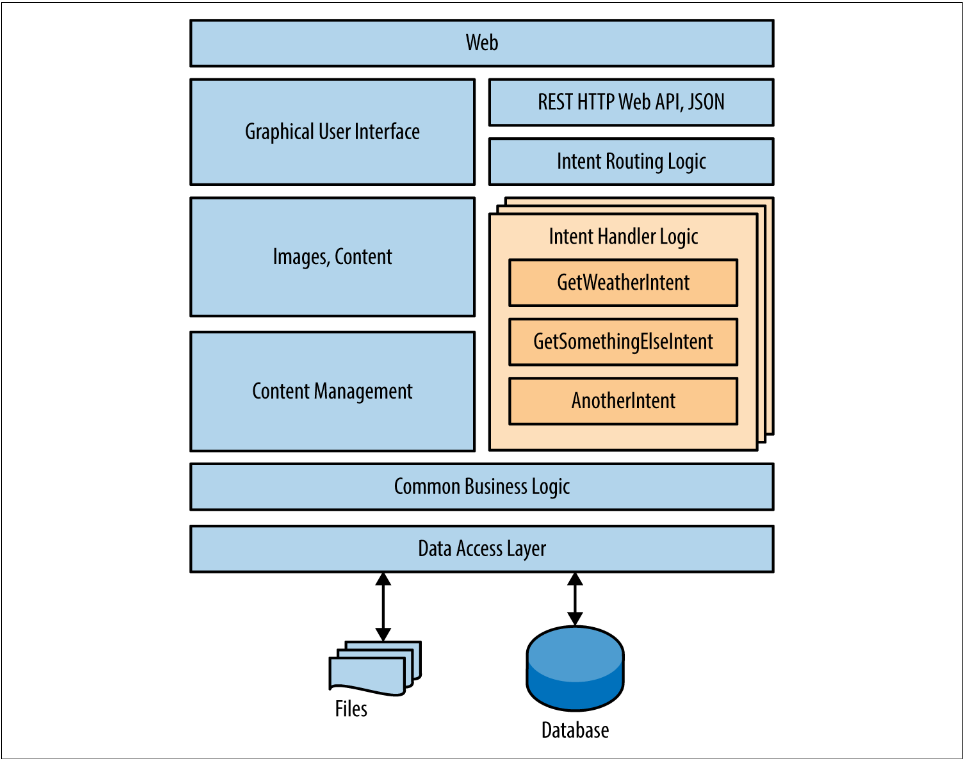


Figure 2-8. Web services architecture

The web services architecture shown here is a bit monolithic with bundled Content Management and a Graphical User Interface for brevity as well as clarity around serving visual content, as with the Alexa Mobile App cards. This might also be overkill if you are creating a simple "Hello, World" application, so think of it as somewhat of a future state architecture. For now, let's break it down a bit further: we start with the web at the top layer, which is where all the traffic comes in and out of. On the two pillars we have REST HTTP Web API component in JSON format, filtering requests to the Intent Routing Logic. Here is where you will determine which intent handler to route to when the requests come in to your endpoint.

There will be multiple intent handlers, one for each of your intents. For example, if someone says, "What's the weather in Miami?" that could be routed to the `GetWeatherIntent` handler. If someone says, "What's the stock price for AMZN?" that would go to something like `GetStockPriceIntent`, and so on. All these components can access the Common Business Logic, which in turn accesses the Data Access Layer for data and files such as images and videos.

The Content Management component provides access to images and content to the Graphical User Interface by way of Pub/Sub event handling or when requested directly via the web. This can be omitted altogether and files can be accessed directly via their URL, but having a content manager or basic handler in place helps with things like security or analytics and if you have a complete Content Management System in place, this can help ease content deployment, as nondevelopers can contribute to uploading content.

Figure 2-9 represents a basic hardware diagram using prototyping hardware such as a Raspberry Pi, a breadboard, and additional components. This is a great way to start rapidly prototyping and testing your product before going full-on production. In the event you want to go full-on production and develop your own board with embedded circuits, sensors, and other components, you will need to design several core and mechanical schematics, drawings, and documentation so that the factory knows exactly how you want the boards produced.

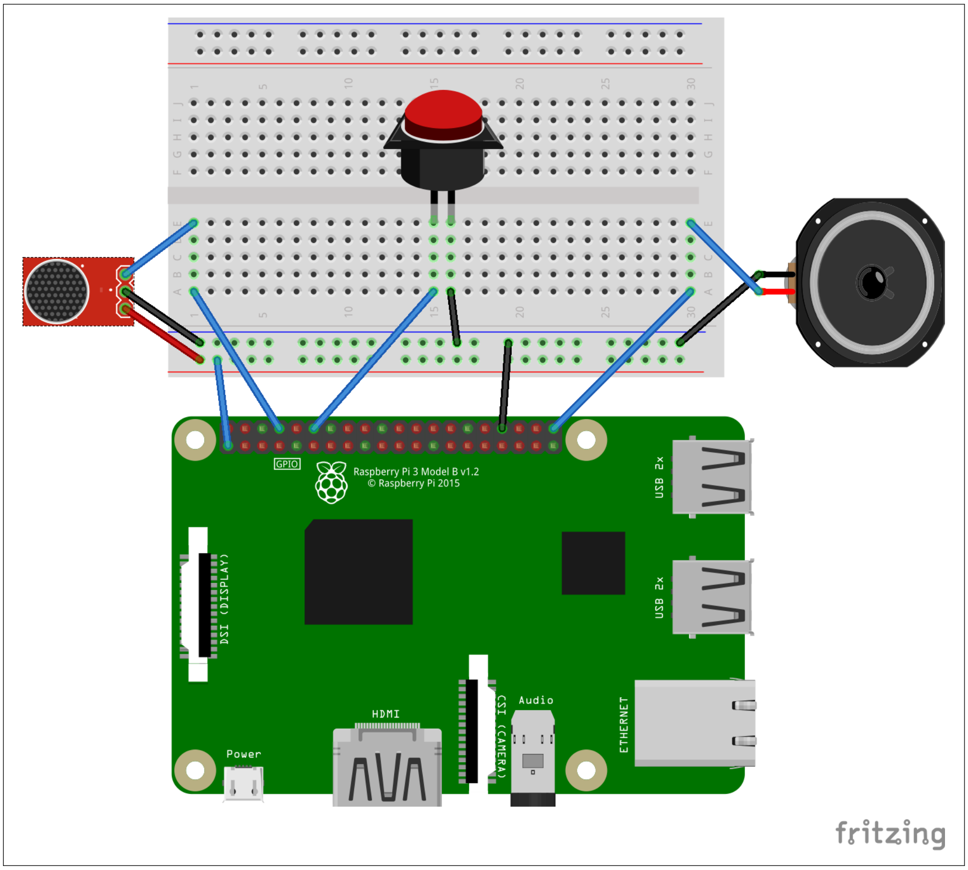


Figure 2-9. Basic hardware architecture diagram for Raspberry Pi with AVS

Figure 2-10 illustrates what one of those advanced schematics could potentially look like.

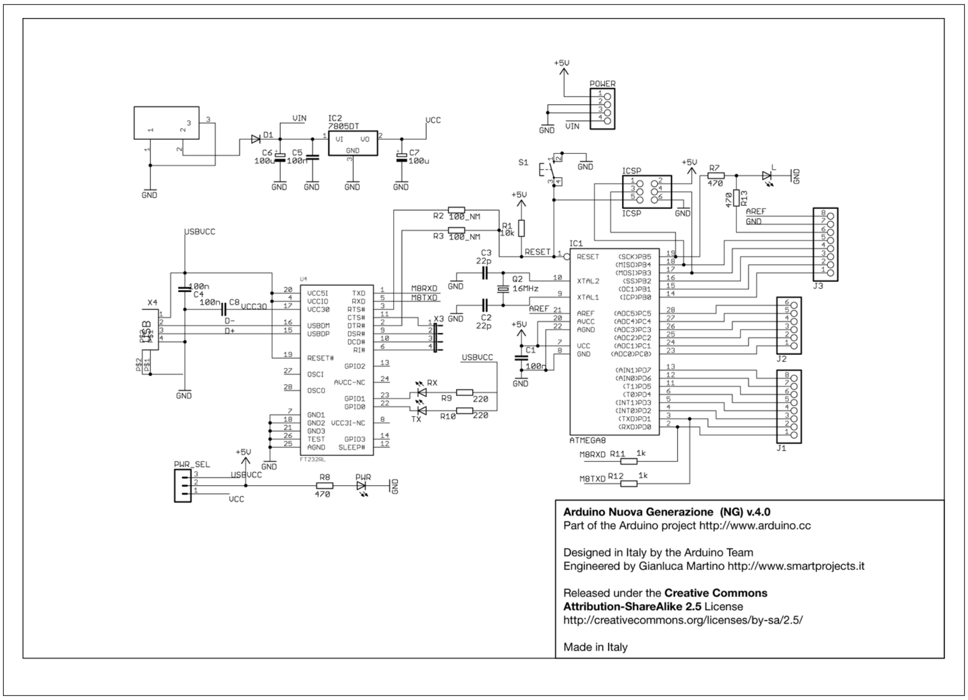

Arduino Nuova Generazione (NG) v.4.0
Part of the Arduino project http://www.arduino.cc
Designed in Italy by the Arduino Team
Engineered by Gianluca Martino http://www.smartprojects.it
Released under the Creative Commons Attribution-ShareAlike 2.5 License
http://creativecommons.org/licenses/by-sa/2.5/
Made in Italy

Figure 2-10. Arduino NG hardware schematic

The schematic just shown is rather complex, and even for seasoned engineers this level of work could take countless hours to fine-tune. As we continue our exploration throughout this book, we'll stick to basic rapid prototyping diagrams and concepts to keep things simple.

Conclusion

This chapter briefly reviewed some of the service offerings out in the wild today, including Amazon Alexa and its ASK and AVS tools. We touched a bit on architecture and what you can expect down the road as things become more and more complicated. It's important to keep things documented and well organized as you evolve your product. Next, we'll start to piece together some rapid prototyping components such as the Raspberry Pi, a speaker, and a microphone. Then we'll download some code, configure some settings, kick some tires, and finally take the Alexa Voice Service out for a spin!

CHAPTER 3

Getting Started with AVS

Now comes the fun part! The preceding chapters presented voice interfaces at a high level. We took a look at some design considerations and covered some of the currently available services and frameworks. Starting with this chapter, during the course of the remainder of the book we will get our hands dirty with hardware tinkering and programming some code as we work on getting our voice interface prototype up and running.

We will begin with creating our very own Alexa-enabled device using a Raspberry Pi 3, a microphone, and a speaker. As we mentioned in Chapter 2, Alexa Voice Service (AVS) enables us to leverage Amazon's intelligent voice platform on our own custom hardware. Essentially, we will be building our own version of an Echo or Dot, just not quite so fancy. There are some limitations with AVS compared to an off-the-shelf Amazon device, such as not being able to make calls or use the device as an intercom. In addition, when a new feature is released on the Alexa platform, it is not always immediately supported on AVS. Nevertheless, AVS enables makers and companies alike to create uniquely branded voice-enabled devices, using the power of Amazon's voice AI. When we're finished, you will be able to ask Alexa for the time or weather, request a song to be played, or even ask her to tell you a joke. A full list of AVS-available features can be found at *http://amzn.to/2xlmO6r*.

In subsequent chapters, after we have experimented with AVS, we will add in some additional sensors and controls. This will help us to expand further into the IoT world and truly demonstrate how your device can leverage existing voice interfaces to control the world around you.

Tools and Things

To follow along and optimize your learning experience, you will need the components listed in Table 3-1. Additionally, you will want to use your existing workstation, whether it be a Mac, Windows, or Linux machine; this will allow you to simply SSH or remote in from your workstation into your device once it's connected to the internet and has been configured to allow remote connections.

Table 3-1. Bill of Materials (BOM)

Part	Approximate price
Raspberry Pi 3 Model B	$40.00
USB Microphone[a]	$20.00
Mini Audio Speaker w/ 3.5mm Audio Jack	$20.00
Micro SD Card (Minimum 8 GB)	$10.00
USB Keyboard and Mouse bundle	$15.00
Monitor and HDMI cord (you can use your existing PC parts since you will only need a monitor on the Pi during setup)	$90.00
5V 2.4A Switching Power Supply w/ MicroUSB Cord for the Pi 3	$10.00

[a] We highly recommend the Kinobo - Mini "AKIRO" USB Microphone for Desktops because of its cross-platform compatability.

These components are essential to completing the "Hello, World" prototype in this chapter. You may have some or all of these parts already. In some cases, you can substitute parts. For instance, if you already have a Raspberry Pi 2 Model B, you can use that instead, but you will need an Ethernet connection or a WiFi USB adapter since the Pi 2 doesn't have built-in WiFi.

Additionally, if you are starting a Raspberry Pi from scratch, you will need a way to write to a micro SD card. If your workstation has an SD card reader, great! Otherwise, you will need to buy a USB SD card reader and adapter. These are currently available on Amazon for as low as $8.

Once you have all of the necessary components you will need to prepare your Pi. The following section will help you do just that.

Preparing Your Pi

First things first: get everything plugged in. During the development process you will want to hook up your Pi 3 to an HDMI monitor as well as a USB keyboard and mouse. Then hook up your Pi with a power adapter, your microphone, and speaker and you are good to go. With your SD card in hand, follow these steps to set up a new instance of a Raspberry Pi (if you already have a working Pi running Raspbian, go ahead and plug that SD card into your Pi, turn it on, and skip ahead to step 3):

1. To create a new Pi from scratch, you will need to download a copy of Raspbian Jessie Lite onto your workstation. Again, this can be your Mac or PC or other primary machine that you use to code on. The great thing about Jessie Lite is that it's lightweight, as its name suggests. There is no GUI out of the box, so if you prefer using a GUI rather than the command line you can download Raspbian Jessie with Pixel. During our experiments we have found that Jessie Lite required no additional fiddling with audio. If you choose a different version than Jessie Lite, your results may vary based on the type of speaker and microphone you are using.

2. Once you have downloaded the Jessie Lite ZIP file onto your workstation, you will need to unzip it in order to get access to the image file (*.img*). At this point, you can write the Jessie Lite image to your micro SD card. There are various ways you can flash SD cards, depending on your operating system. Here are the steps you'll need to follow:

   Windows

   a. Once you have inserted your SD card into your computer, take note of the drive letter it is assigned in Windows Explorer.
   b. Download and install the Win32 Disk Imager.
   c. When you run Win32 Disk Imager, go ahead and select the Jessie Lite image file you unzipped earlier, select the drive letter of the SD card, and then click the Write button. If you get a permissions error, close the application, then go back and right-click the Win32 Disk Imager icon, and select Run as Administrator.

   Mac

   a. When you insert your SD card, you can locate it by running the following command in Terminal:

      ```
      diskutil list
      ```

      You should be able to tell which disk it is by the disk size. Take note of the device name it gives you for your SD card (it should look something like */dev/disk2*).
   b. Next, you need to unmount the disk. You can do this by running the following command in Terminal:

      ```
      sudo diskutil unmountDisk /dev/diskX
      ```

      Make sure to replace the device name (i.e., `/dev/diskX`) with your own device name discovered in the first step.
   c. Lastly, you can start writing the Jessie Lite image to the SD card by running the following command (this process might take a long time and will not

display any progress indicator, so be patient, grab a cup of coffee or two, and give it a few minutes to run):

```
sudo dd if=~/Desktop/image.img of=/dev/diskX bs=5m
```

Make sure to replace the `if` path to the path of your Jessie Lite image file and change the `of` device name to your own device name that you used to unmount the SD card with.

For additional tips and tricks for imaging an SD card on a Mac, consult the Raspberry Pi Foundation's "Installing Operating System Images on Mac OS".

Linux

a. Similar to Windows and Mac, we need to determine the path of the SD card. We can do this on Linux by running the following command:

```
df -h
```

If you are having a hard time deciphering which drive is which, you can run the preceding command before and after you insert the SD card, and simply observe which is the new listing.

b. Next, let's unmount by running the following command:

```
umount /dev/sddX
```

Make sure to replace the device name (i.e., */dev/sddX*) with your own device name discovered in the first step.

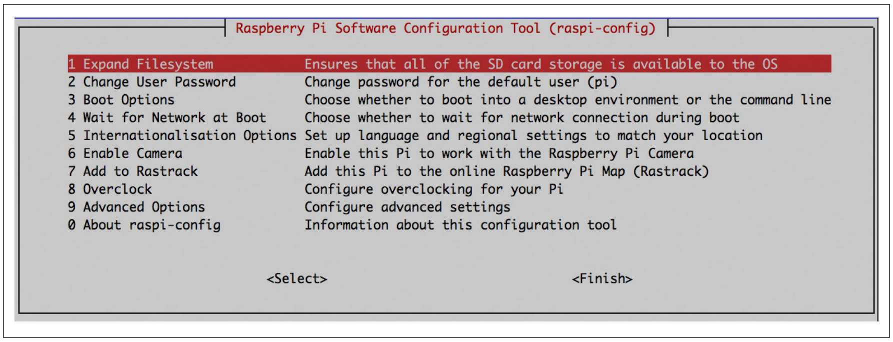
c. Lastly, write Jessie Lite to the SD card by running the following command (as with Mac, there is no progress indicator, so make sure to give it several minutes to run; time varies based on machine specs and SD card size):

```
dd bs=4M if=raspbian-jessie-lite.img of=/dev/sddX
```

For additional tips and tricks for imaging an SD card on a Linux machine, consult the Raspberry Pi Foundation's "Installing Operating System Images on Linux".

3. Once your SD card is ready, you can eject it from your workstation or laptop and plug it into your Pi. Go ahead and power it on, then once your Pi has booted up, you will need to log in to your Pi. If it's a fresh install, you can use the username "pi" and the password "raspberry". Yes, you will need to change this as well as some other house cleaning and configuration items, which are outlined in the following steps:

   - Type the command **`sudo raspi-config`**—you should see a screen similar to the one shown in Figure 3-1.



```
Raspberry Pi Software Configuration Tool (raspi-config)

1 Expand Filesystem            Ensures that all of the SD card storage is available to the OS
2 Change User Password         Change password for the default user (pi)
3 Boot Options                 Choose whether to boot into a desktop environment or the command line
4 Wait for Network at Boot     Choose whether to wait for network connection during boot
5 Internationalisation Options Set up language and regional settings to match your location
6 Enable Camera                Enable this Pi to work with the Raspberry Pi Camera
7 Add to Rastrack              Add this Pi to the online Raspberry Pi Map (Rastrack)
8 Overclock                    Configure overclocking for your Pi
9 Advanced Options             Configure advanced settings
0 About raspi-config           Information about this configuration tool

                  <Select>                    <Finish>
```

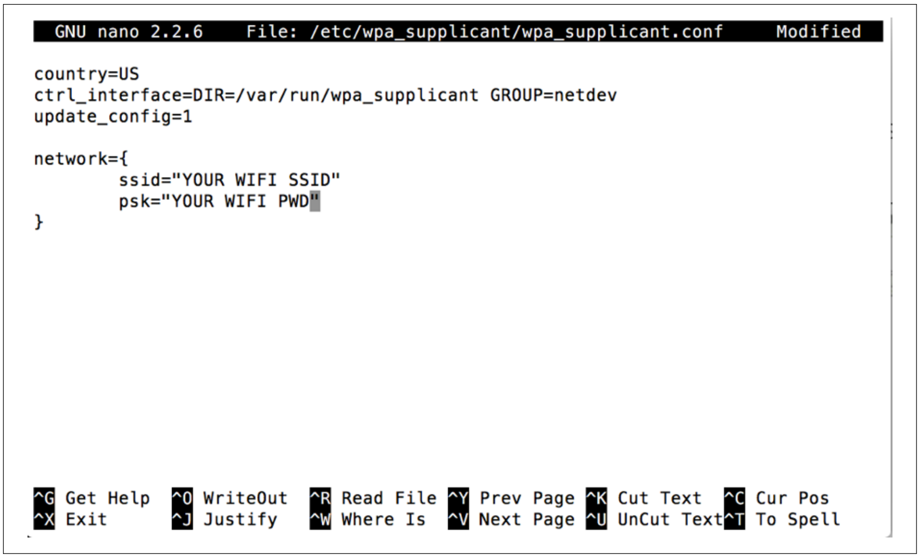
Figure 3-1. Screen capture of raspi-config tool

- Select Expand Filesystem to take advantage of all the space on your SD card. After this is done, select Finish.
- Next, select Advanced Options and then select SSH and enable it. This will allow you to remote into your Pi for easier setup and maintenance.
- If you were not prompted to change your default password when turning on SSH, you should select Change User Password.
- Optionally, if you find that your keyboard is not quite working the way you expected, select Change Keyboard Layout under the Internationalization Options.
- At this point we're done in the raspi-config, so go ahead and click Finish to return to the Pi terminal.

4. Next, let's go ahead and configure the WiFi. We'll do this by entering the following command in the Pi:

   ```
   sudo nano /etc/wpa_supplicant/wpa_supplicant.conf
   ```

   This will open the *wpa_supplicant.conf* file in the nano file editor as shown in Figure 3-2.

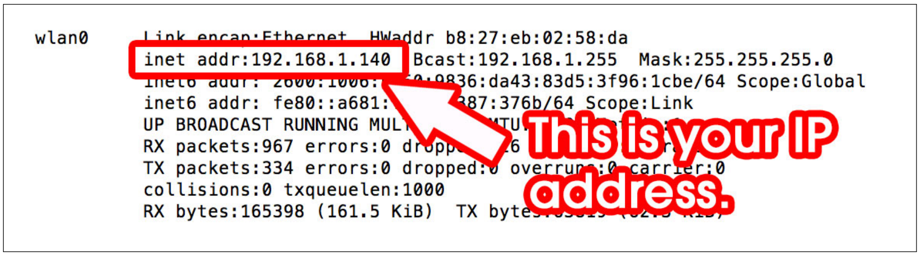


```
 GNU nano 2.2.6    File: /etc/wpa_supplicant/wpa_supplicant.conf     Modified

country=US
ctrl_interface=DIR=/var/run/wpa_supplicant GROUP=netdev
update_config=1

network={
        ssid="YOUR WIFI SSID"
        psk="YOUR WIFI PWD"
}







^G Get Help  ^O WriteOut  ^R Read File ^Y Prev Page ^K Cut Text  ^C Cur Pos
^X Exit      ^J Justify   ^W Where Is  ^V Next Page ^U UnCut Text^T To Spell
```

Figure 3-2. The wpa_supplicant.conf file open in the nano file editor

In the nano editor, if you do not see the `network` entry, go ahead and add it in making sure to replace `YOUR WIFI SSID` with your WiFi's SSID and replace `YOUR WIFI PWD` with your WiFi password. Once you are done with your changes to the file, hit Ctrl-X on your keyboard then follow the nano prompts to save your changes to the disk. Once nano closes, type the command **`sudo reboot`** to reboot the Pi.

5. Once your Pi has rebooted, type the **`ifconfig`** command. As shown in Figure 3-3, the output should include your IP address (look for "inet addr"); make note of this, as you will need it later. If the IP address is not present, this means you are not connected to WiFi, in which case you will want to repeat step 4 and ensure that your SSID and password are both correct.



```
wlan0     Link encap:Ethernet  HWaddr b8:27:eb:02:58:da
          inet addr:192.168.1.140  Bcast:192.168.1.255  Mask:255.255.255.0
          inet6 addr: 2600:1006:    0836:da43:83d5:3f96:1cbe/64 Scope:Global
          inet6 addr: fe80::a681:    87:376b/64 Scope:Link
          UP BROADCAST RUNNING MUL      MTU
          RX packets:967 errors:0 dr           6
          TX packets:334 errors:0 dropped:0 overruns:0 carrier:0
          collisions:0 txqueuelen:1000
          RX bytes:165398 (161.5 KiB)  TX byte
```

Figure 3-3. Screenshot of ifconfig command in a Raspberry Pi

6. Lastly, let's make sure your Pi is up-to-date:
   a. Type the command **`sudo apt-get update`** and follow any prompts.
   b. Type **`sudo apt-get install git`** to install Git. This will allow us to use Git repositories, which will be especially useful for cloning AlexaPi code (we'll explain this in further detail in the following section).
   c. Once you are all set, type in **`sudo reboot`** to make sure everything is all set up as expected.

Congratulations! Your Pi is ready to rock and we're ready for the fun to begin. But before we start, let's register our device with Amazon.

Amazon AVS Configuration

Once you have a working Raspberry Pi ready to go, you will need to register that device with Amazon at its developer console. If you don't have an Amazon account, you will need to go ahead and create one. If you've never done this before, it could get a bit confusing so before we go too deep into the configuration, let's take a quick introductory tour of the dev portal itself and all its amazing Amazon features. If you already have an account and are familiar with the Amazon Developer Console, feel free to skip the following section and jump straight to "AVS Setup" on page 50.

Navigating the Amazon Developer Console

Getting around the Developer Console can be a bit challenging for first-time users, so before we get into setting up AVS, here's a quick primer on what you will find there. Keep in mind that the placement of the links and even their labels may have changed by the time you read these pages. Amazon has a tendency to iterate fast and frequently so don't be surprised if it looks completely different from what you see here. With that in mind, fire up your favorite browser and go to *https://developer.amazon.com/*. There, look for the Alexa tab or link (Figure 3-4 shows what the tab currently looks like when you hover over it).

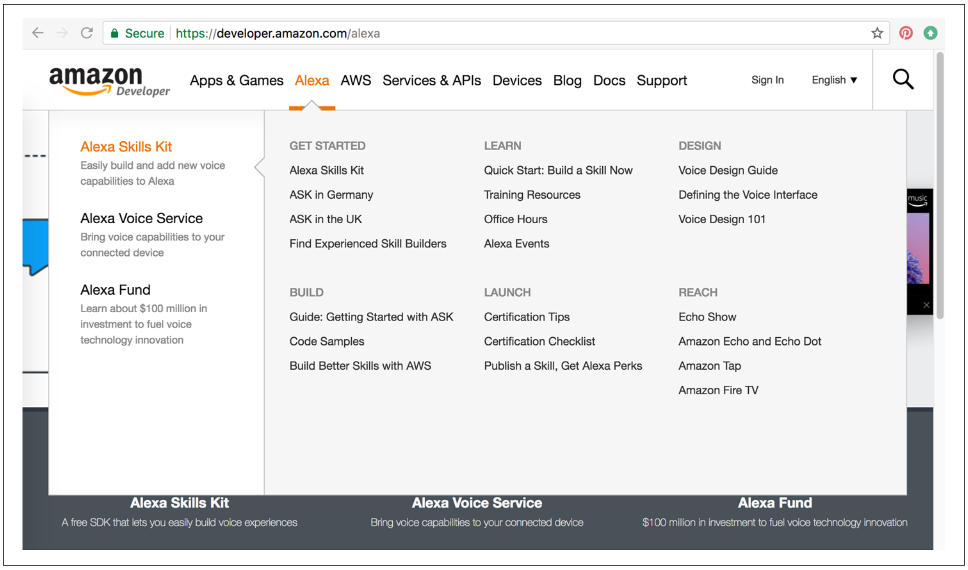
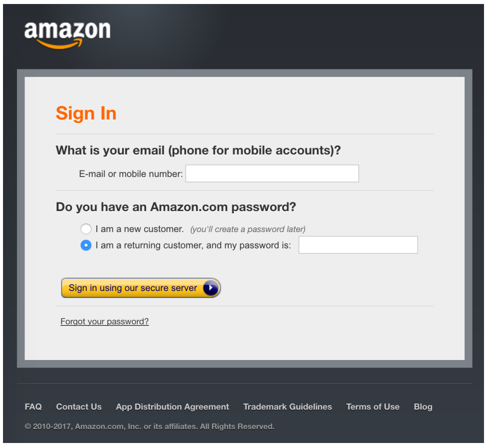

Secure https://developer.amazon.com/alexa
amazon Developer
Apps & Games
Alexa
AWS
Services & APIs
Devices
Blog
Docs
Support
Sign In
English ▼
Alexa Skills Kit
Easily build and add new voice capabilities to Alexa
Alexa Voice Service
Bring voice capabilities to your connected device
Alexa Fund
Learn about $100 million in investment to fuel voice technology innovation
GET STARTED
Alexa Skills Kit
ASK in Germany
ASK in the UK
Find Experienced Skill Builders
LEARN
Quick Start: Build a Skill Now
Training Resources
Office Hours
Alexa Events
DESIGN
Voice Design Guide
Defining the Voice Interface
Voice Design 101
BUILD
Guide: Getting Started with ASK
Code Samples
Build Better Skills with AWS
LAUNCH
Certification Tips
Certification Checklist
Publish a Skill, Get Alexa Perks
REACH
Echo Show
Amazon Echo and Echo Dot
Amazon Tap
Amazon Fire TV
Alexa Skills Kit
A free SDK that lets you easily build voice experiences
Alexa Voice Service
Bring voice capabilities to your connected device
Alexa Fund
$100 million in investment to fuel voice technology innovation

Figure 3-4. Amazon Developer Console, Alexa Menu (June 2017)

Go ahead and click Alexa Voice Service—you'll see a bunch of links for guides on designing and building with AVS. We have synthesized all the important sections into the following pages but if you want to take a deeper dive, feel free to take a few minutes to explore the site further and get a greater understanding of AVS. Once you are ready to get your hands dirty, continue following the instructions here.

Continuing on, the next thing you will want to do is look for the "Sign in" link (see Figure 3-5). There you will need to provide the email address or mobile number associated with your Amazon account. You can use an existing account if you like (perhaps you already have one that's tied to your Echo or Dot) or you can also choose to create a new test account. Just keep in mind whatever account you do end up using, must be tied to your device for testing purposes.



amazon

Sign In

What is your email (phone for mobile accounts)?

E-mail or mobile number:

Do you have an Amazon.com password?

- I am a new customer. *(you'll create a password later)*
- [✓ I am a returning customer, and my password is:]

Sign in using our secure server

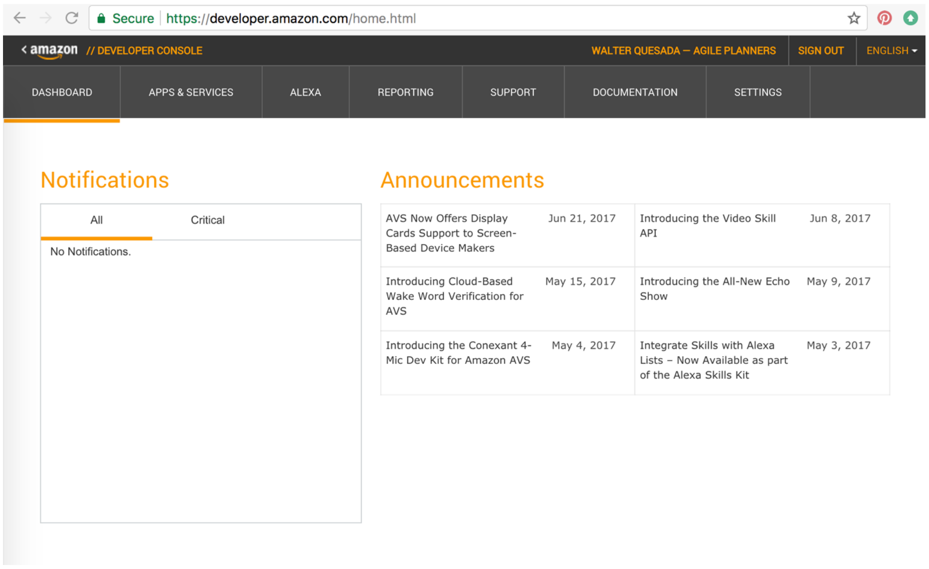
Forgot your password?

FAQ Contact Us App Distribution Agreement Trademark Guidelines Terms of Use Blog

© 2010-2017, Amazon.com, Inc. or its affiliates. All Rights Reserved.

Figure 3-5. Amazon Developer Sign In page

Once you have logged in, you will be at the Developer Console Dashboard (Figure 3-6). Here, you see general announcements, which is a great way to stay apprised of the latest Alexa-related updates. When you're ready, click the "ALEXA" link in the top menu. This will take you to the ASK and AVS options page.



Secure https://developer.amazon.com/home.html

amazon // DEVELOPER CONSOLE WALTER QUESADA — AGILE PLANNERS SIGN OUT ENGLISH

DASHBOARD APPS & SERVICES ALEXA REPORTING SUPPORT DOCUMENTATION SETTINGS

Notifications

All Critical

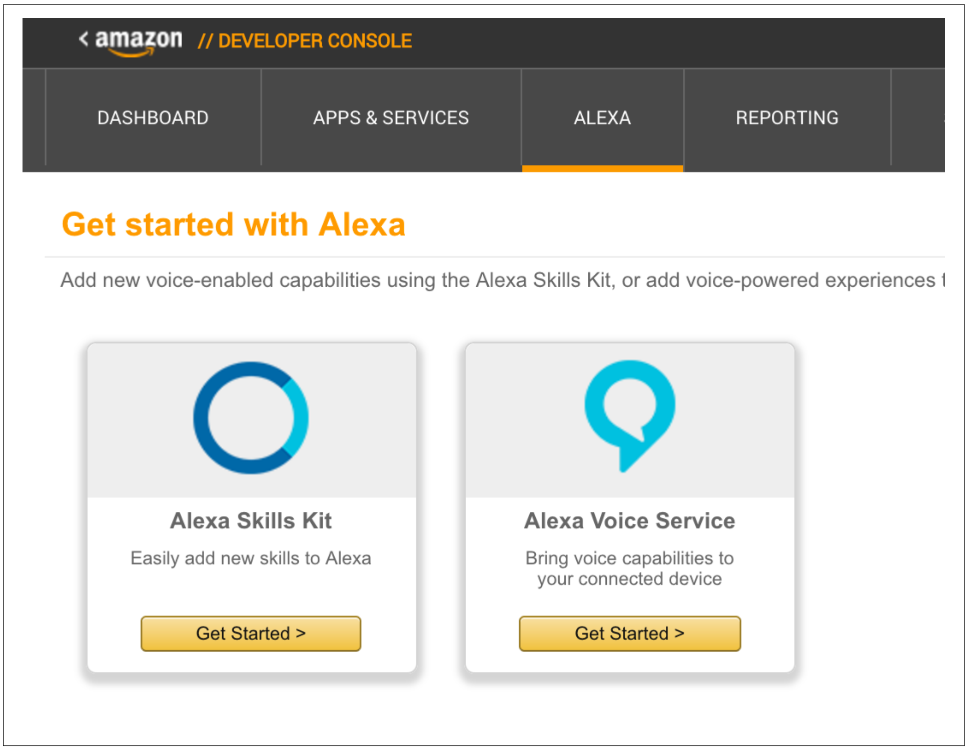
No Notifications.

Announcements

AVS Now Offers Display Cards Support to Screen-Based Device Makers	Jun 21, 2017	Introducing the Video Skill API	Jun 8, 2017
Introducing Cloud-Based Wake Word Verification for AVS	May 15, 2017	Introducing the All-New Echo Show	May 9, 2017
Introducing the Conexant 4-Mic Dev Kit for Amazon AVS	May 4, 2017	Integrate Skills with Alexa Lists – Now Available as part of the Alexa Skills Kit	May 3, 2017

Figure 3-6. Developer Console Dashboard

On the “Get started with Alexa” page you have the option of entering into the ASK or AVS management areas, where you are able to create and edit skills or manage AVS products, respectively (Figure 3-7).



Get started with Alexa

Add new voice-enabled capabilities using the Alexa Skills Kit, or add voice-powered experiences t

Alexa Skills Kit

Easily add new skills to Alexa

Alexa Voice Service

Bring voice capabilities to your connected device

Figure 3-7. Get started with Alexa page—pick your poison!

The following section explains how to register your new product. Once you are done adding the new product to the AVS management area of the Developer Console, make sure to keep it open—you’ll need to come back to this later once you’ve finished adding the code to the Raspberry Pi so that you can copy and paste some of the parameters from the Amazon Developer Console to your Raspberry Pi code.

AVS Setup

Once logged in, go through the following steps:

1. In the Alexa Voice Service section of the Developer Console, click Register a Product Type, then choose Device.
2. You should be on the Device Type Info tab (if not, select it).
    - Enter a value for Device Type ID, (e.g., MyVeryOwnPI).

- Enter a value for Display Name (this can be the same as Device Type ID or any other text value). Then click Next.

3. You should now be on the Security Profile section.

   - From the drop-down menu choose "Create a new profile."
   - Enter names for your Security Profile Name and Security Profile Description (these can be the same as the Device Type ID or Display Name you entered earlier). Then click Next.
   - Click the Web Settings tab, then click the Edit button.
   - Under Allowed Origins click Add Another.
     — Enter **`http://localhost:5050`** and click Add Another.
     — Enter **`http://`*`YOUR_RASPI.IP.ADDRESS`*`:5050`** making sure to replace *`YOUR_RASPI.IP.ADDRESS`* with your IP address that you noted earlier from the ifconfig command you ran on the Pi.
   - Under Allowed Return URLs click Add Another.
     — Enter **`http://localhost:5050/code`** and click Add Another.
     — Enter **`http://`*`YOUR_RASPI.IP.ADDRESS`*`:5050/code`** again making sure to replace *`YOUR_RASPI.IP.ADDRESS`* with your IP address.

4. Click Next to get to the Device Details section, then fill in the Device Details fields with any valid text or drop-down option. You don't need to select an image for your AVS device (but you can if you want).
5. Click Next to get to the Amazon Music tab. For the Apply for Amazon Music option select No, then click the Submit button to finish.

At this point we are all set on the Amazon side. Your AVS device portal configuration is now complete and we can go back to our Pi and finish getting the code ready to connect to AVS. Additionally, make sure to leave the Amazon Developer Console open on your browser, as you will need to copy and paste some of the parameters during the code configuration.

Get the Code!

Now for the fun part! We're going to download some code you can run on your Pi that will connect to AVS. There are multiple options out there, including a Java-based starter kit provided by Amazon, but our favorite is the Python-based AlexaPi project found on GitHub. AlexaPi was originally started by fellow Alexa Champion, Sam Machin out of Bristol, UK. After hundreds of forks, Sam along with Mason Stone and other contributors, created a unified code base that now lives in a GitHub repository.

We will be using the AlexaPi project in this book because Python is the native language of choice for Raspberry Pi. In addition, AlexaPi is simply a smaller code set that will be easier to break down and analyze in this context.

Before we get started, let's quickly cover the two options for working on a Raspberry Pi. Since we enabled SSH earlier, this will allow you to access your Pi from your workstation. However, if you are more comfortable working directly on the Pi, feel free to do so. We have chosen to work on the Pi via SSH, as it makes it easier to simply copy and paste variables from our workstation browser.

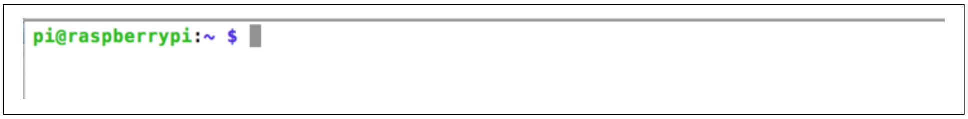
If you are a Windows user, you will need an SSH client for your version of Windows (PuTTY is a great client for this purpose). For Mac and Linux, simply type the following command in Terminal, making sure to replace *`YOUR.RASPI.IP.ADDRESS`*:

```
ssh pi@YOUR.RASPI.IP.ADDRESS
```

If you've forgotten your IP address, simply run the `ifconfig` command again to retrieve it. Windows users can use `ipconfig` in the Command Prompt.

When you enter `ssh pi@`*`YOUR.RASPI.IP.ADDRESS`*, the `pi@` part of the command is the username you use to log in to the Pi itself. If you changed the username during the initial configuration, then you will need to change `pi@` to your username (e.g., `ssh yourusername@`*`YOUR.RASPI.IP.ADDRESS`*).

Once you enter the `ssh` command from your workstation, you might get a warning confirmation; simply enter "y" or "yes" to continue. Next, you will be prompted to enter your password. Once you hit Enter, you should be logged in and ready to go. Your screen should look something like Figure 3-8.



```
pi@raspberrypi:~ $
```

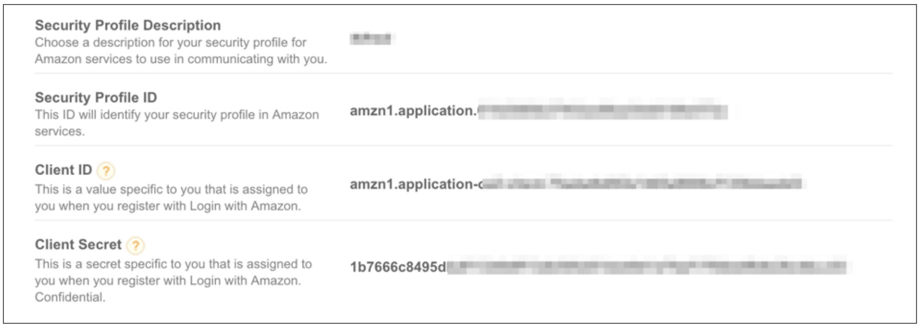
Figure 3-8. Terminal screen in Raspberry Pi

Now that you are logged in to the Pi again, let's go ahead and get started with downloading and configuring the AlexaPi code:

1. The first thing we're going to do is change directory to */opt*. To do so, enter the command **`cd /opt`**. It's important that we download and execute the code from this directory so that AlexaPi runs on boot and you don't have to start it up manually each time you restart the Pi itself. Down the line you might wish to change the path and update the startup service script, but that's not necessary at this point.
2. Type **`sudo git clone https://github.com/alexa-pi/AlexaPi.git`** and then hit Enter. This will grab AlexPi from GitHub.

3. Type **`sudo ./AlexaPi/src/scripts/setup.sh`** and then hit Enter.
4. This is where all the magic happens. You will be prompted to answer questions. Select all the defaults. When you are asked for your Device ID you will need to log back in to the Amazon developer console and navigate back to your AVS Device's Security Profile tab (Figure 3-9). You will need this information to complete the prompts.

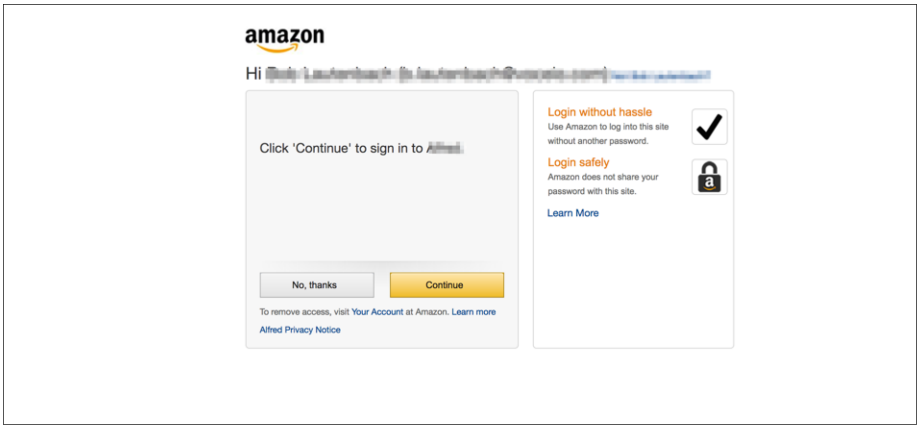
   Note: When prompted for AirPlay, select "n"; we won't need it for this example.



Security Profile Description Choose a description for your security profile for Amazon services to use in communicating with you.	[redacted]
Security Profile ID This ID will identify your security profile in Amazon services.	amzn1.application.[redacted]
Client ID This is a value specific to you that is assigned to you when you register with Login with Amazon.	amzn1.application-o[redacted]
Client Secret This is a secret specific to you that is assigned to you when you register with Login with Amazon. Confidential.	1b7666c8495d[redacted]

Figure 3-9. AVS Security Profile screen

5. When setup is finished you will be prompted to press Ctrl-C after the authentication has completed; don't press Ctrl-C just yet—there are still a few more steps we need to complete before we're done.
6. Open a web browser on your Mac/PC and go to the following URL: *http://YOU_PIS_IP_ADDRESS:5050*.
7. You should see a page similar to the one shown in Figure 3-10. To complete the authorization, click Continue.



amazon

Hi [redacted]

Click 'Continue' to sign in to [redacted]

No, thanks | Continue

To remove access, visit Your Account at Amazon. Learn more

Alfred Privacy Notice

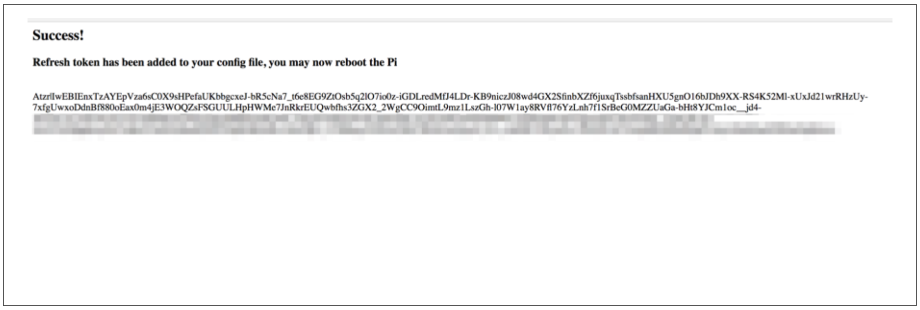
Login without hassle
Use Amazon to log into this site without another password.

Login safely
Amazon does not share your password with this site.

Learn More

Figure 3-10. AVS OAuth sign in screen

8. The completed authorization screen should look something like Figure 3-11.



Success!

Refresh token has been added to your config file, you may now reboot the Pi

```
AtzrlIwEBIEnxTzAYEpVza6sC0X9sHPefaUKbbgcxeJ-bR5cNa7_t6e8EG9ZtOsb5q2lO7io0z-iGDLredMfJ4LDr-KB9niczJ08wd4GX2SfinbXZf6juxqTssbfsanHXU5gnO16bJDh9XX-RS4K52Ml-xUxJd21wrRHzUy-
7xfgUwxoDdnBf880oEax0m4jE3WOQZsFSGUULHpHWMe7JnRkrEUQwbfhs3ZGX2_2WgCC9OimtL9mz1LszGh-l07W1ay8RVfl76YzLnh7f1SrBeG0MZZUaGa-bHt8YJCm1oc__jd4-
[redacted]
```

Figure 3-11. AVS OAuth successful authentication screen

9. Reboot your Pi (you can use the Ctrl-C command in your SSH session and then type **`sudo reboot`** (and then hit Enter) or you can simply unplug and replug it in.

Congratulations! You now have your very own Alexa using AVS on a Raspberry Pi device. When you have rebooted, you should be able to ask your Pi, "Alexa, what time is it?" or "Alexa, what's the weather in Miami?" or any number of the other commands you can ask Alexa out of the box. There are a few caveats, however: at this time, you can't play music or control your Nest, among other out-of-the box Amazon Echo features that one would expect. This may change in the future but for now, we'll just focus on how we can make our own Pi work with our own skills, which we will start doing in the next chapter.

But How Does It All Work?

Before we jump into creating custom skills, let's quickly cover some of the key components. There are a handful of important files that you should really get familiar with if you want to start customizing the embedded code on your device. The files in the root directory are mostly Git, Travis, and documentation-related files as well as a Fritzing diagram that shows you how to wire up and test with some LEDs, resistors, and a button. This setup is optional but if you have some spare parts laying around, you can use the diagram in Figure 3-12 to set that up as well.

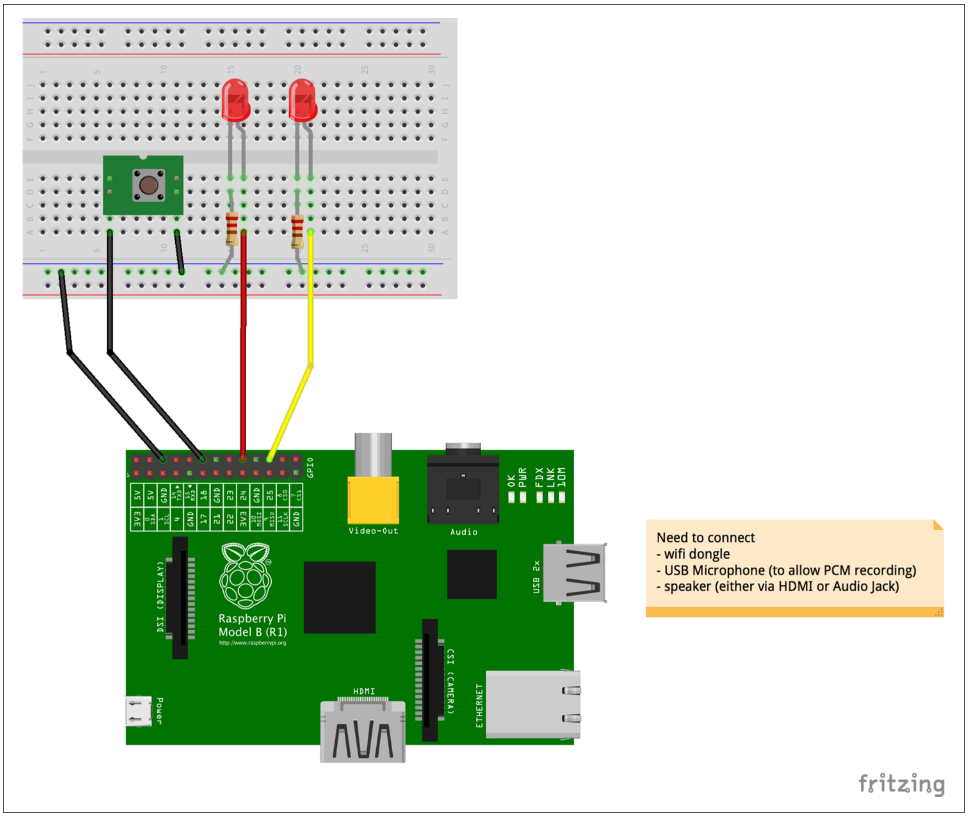


Figure 3-12. AlexaPi Button and LED circuit diagram

The following is breakdown of the current file structure. These files are all located under the */src* directory:

- */alexapi*
  — */resources*
  — */scripts*
  — *auth_web.py*
  — *config.template.yaml*
  — *dev-requirements.txt*
  — *main.py*
  — *requirements.txt*

The important files to remember here are *auth_web.py* and *main.py*, as well as the *config.template.yaml* file (which gets converted to *config.yaml* during the setup process). Your Amazon AVS credentials are stored in the autogenerated *config.yaml* file, which is then used by *auth_web.py* and *main.py*. Should you need to update your credentials later, simply update the *config.yaml* file.

The *auth_web.py* is essentially a web server that helps generate the OAuth refresh tokens for the Amazon user account. The token is also stored in the *config.yaml* file and is updated automatically when new tokens are generated. The *main.py* serves as the main Alexa client application that runs on a loop listening for either the trigger word ("Alexa," for instance) or for the button press if you choose to add a button.

Working with a Button and LEDs

The optional button and LEDs setup is a great way to test and ensure that your device is indeed working as it should. There might be times where you don't hear a response, in which case the LEDs can provide a hint to help you determine at which point it failed. For example, if the red LED lights up, that means it's recording audio. When both are lit, the audio is being posted and we are now awaiting a response from AVS. If there's a delay in the response, only the green LED is lit and if there's an error, the red LED will flash three times consecutively.

The wake word is recognized by a library called PocketSphinx, which is an open source lightweight continuous speech recognition engine initially developed by Carnegie Mellon University circa 2006. In the config file, you can also modify the wake word by modifying the `phrase` value under `pocketsphinx`.

If you are going to add the button and LEDs, you can also change the GPIO pins, if you choose to change them, in the configuration file as well. Under the *raspberrypi* section, you will see the following key values:

- button: 18
- plb_light: 24
- rec_light: 25

This current configuration means that the button is set to GPIO pin 18, the green playback/activity LED light is set to pin 24, and the red LED light is set to pin 25. Technically you can choose whichever color LEDs you have—just remember to note which color indicates which status. Alternatively, you can also use a single dual color LED or a RGB LED. However, for the RGB LED option, you will need to modify the code in */src/alexapi/device_platforms/rpilikeplatform.py* to accommodate the blue lead. For example, you would apply the red lead to pin 25, the green lead to pin 24, and the blue lead to pin 23. Then in the *rpilikeplatform.py* file, any references made to those pins would need to be updated to support RGB LEDs.

If you want to really get adventurous, you can alternatively use an Adafruit NeoPixel RGB LED Ring, which is an individually addressable ring of LEDs that range from 16 LEDs to 60 LEDs (Figure 3-13).

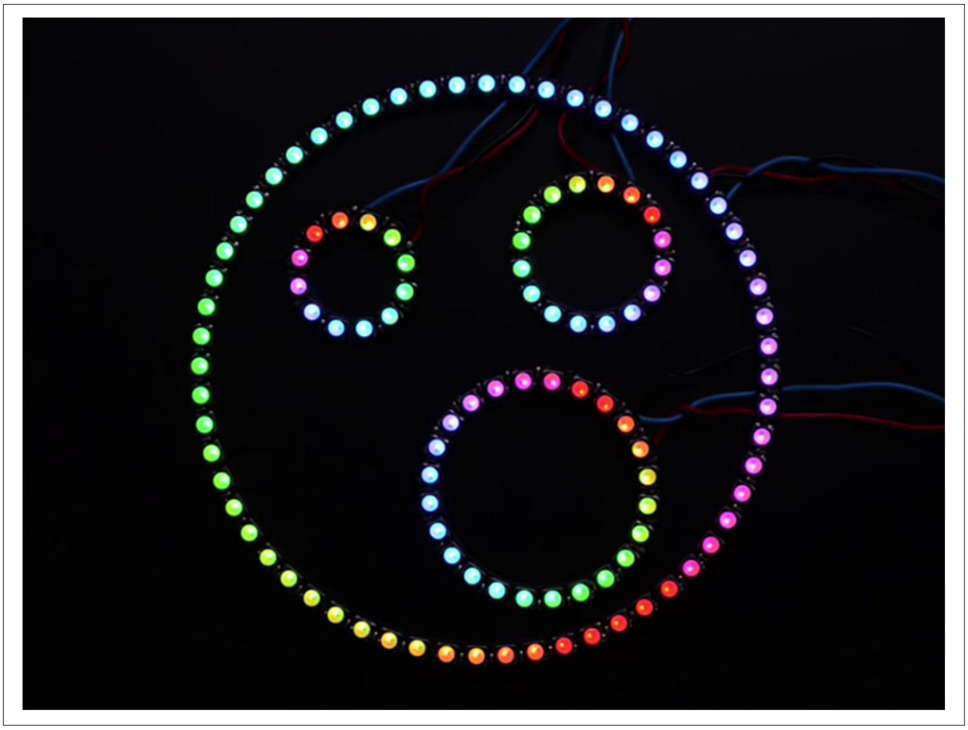


Figure 3-13. NeoPixel Rings (Creative Commons photo by Phillip Burgess)

What's in the Resources Directory?

Let's take another look at the files. The *src/sources/* directory contains some audio files you might want to consider changing, such as that lovely "Hello" from Alexa when your device boots up and connects to Alexa. You can change it to your own voice, or Yoda's voice, or some random sound like a clucking chicken. Totally up to you. Simply keep the same filenames and you won't need to modify any existing code. It should just work so long as they are encoded correctly.

Customizing the Success Page

Remember that Success page we saw earlier when we authenticated our Amazon Alexa account? Well to edit that template, open up *auth_web.py* and scroll down to lines 65 through 75 and you will find something that looks like this:

```
return "<h2>Success!</h2>" \
  "<p>The refresh token has been added to your config file.</p>" \
  "<p>Now:</p>" \
  "<ul>" \
  "<li>close your this browser window,</li>"
  "<li>exit the setup script as indicated,</li>" \
  "<li>and follow the Post-installation steps.</li>" \
  "</ul>"
```

Deeper Look at the AVS Requests

Now let's digest some of the meat and potatoes here—in other words, let's take a deep dive into some of the actual AVS request functions in the AlexaPi. In *main.py*, we see a function definition called `internet_on`. This function simply checks to see if the AVS token endpoint is accessible. If it is, a token request will be made via the `Token` class. Here, we see how the token is requested from the AVS in the `renew` function. This is what the code should look like there:

```
def renew(self):

  logger.info("AVS token: Requesting a new one")

  payload = {
    "client_id": self._aconfig['Client_ID'],
    "client_secret": self._aconfig['Client_Secret'],
    "refresh_token": self._aconfig['refresh_token'],
    "grant_type": "refresh_token"
  }

  url = "https://api.amazon.com/auth/o2/token"
  try:
    response = requests.post(url, data=payload)
    resp = json.loads(response.text)

    self._token = resp['access_token']
    self._timestamp = time.time()

    logger.info("AVS token: Obtained successfully")
  except requests.exceptions.RequestException as exp:
    logger.critical("AVS token: Failed to obtain a token: "
      + str(exp))
```

Here we see some basic logging with the message `AVS token: Requesting a new one`. Then setting up the payload with the `client_id`, `client_secret`, `refresh_token`, and `grant_type` all being populated from the values in the configuration file with the exception of `grant_type`, which is hardcoded to `refresh_token`. Next, the `url` variable is set to the AVS OAuth token endpoint currently set to `https://api.amazon.com/auth/o2/token`. Lastly, the call is made via `requests.post`, JSON is returned and parsed into the `resp` variable, where then the `self._token` field is set to the new `access_token` available in the `resp` variable.

So far, all this code is essentially executed at runtime in the Python `__main__` routine, which means AlexaPi is running directly and not imported into another Python module. You will find this at the bottom of the *main.py* file starting with the `if` `__name__ == "__main__":` line. Under here, you will find all the setup and configuration processes as well as the connection check and token refresh routines we just talked about. Additionally, you will find trigger callback setup as well as playing the "Hello" MP3 file when AlexaPi loads on reboot. The `while True:` loop then keeps the code running indefinitely while the trigger callback functions listen on a separate thread for either the push button down event or the wake word.

The `trigger_process` function handles the trigger events. Let's take a look at what's inside there:

```
def trigger_process(trigger):
  if player.is_playing():
    player.stop()

  # clean up the temp directory
  if not debug:
    for some_file in os.listdir(tmp_path):
      file_path = os.path.join(tmp_path, some_file)
      try:
        if os.path.isfile(file_path):
          os.remove(file_path)
      except Exception as exp:
        logger.warning(exp)

  if event_commands['pre_interaction']:
    subprocess.Popen(event_commands['pre_interaction'],
      shell=True, stdout=subprocess.PIPE)

  force_record = None
  if trigger.event_type in triggers.types_continuous:
    force_record = (trigger.continuous_callback,
      trigger.event_type in triggers.types_vad)

  if trigger.voice_confirm:
    player.play_speech(resources_path + 'alexayes.mp3')

  audio_stream = capture.silence_listener(force_record=force_record)
  alexa_speech_recognizer(audio_stream)

  triggers.enable()

  if event_commands['post_interaction']:
    subprocess.Popen(event_commands['post_interaction'],
      shell=True, stdout=subprocess.PIPE)
```

The `trigger_process` function starts off with stopping the audio player then goes into cleaning up the */temp* directory, which includes files like previously recorded audio, for example. Interestingly, the subsequent line is a conditional statement to see if there's a `pre_interaction` event command specified. Additionally, at the bottom of the function, there's a condition to check for `post_interaction`. If there is, it will execute it on a subprocess, which is a cool way to set up your own routine to run before and/or after the callback trigger executes.

The lines in between the `pre_interaction` and `post_interaction` conditions execute the trigger events, run a `voice_confirm` audio clip if the trigger is a voice confirmation, then listen for the audio from the microphone routing that same audio stream to the `alexa_speech_recognizer` function definition. This function essentially takes in an audio stream and prepares it for departure. Let's break down the speech recognizer code for a bit:

```
def alexa_speech_recognizer(audio_stream):
  url = 'https://access-alexa-na.amazon.com/v1/avs/speechrecognizer/recognize'
  boundary = 'this-is-a-boundary'
  headers = {
    'Authorization': 'Bearer %s' % token,
    'Content-Type': 'multipart/form-data; boundary=%s' % boundary,
    'Transfer-Encoding': 'chunked',
  }

  data = alexa_speech_recognizer_generate_data(audio_stream, boundary)
  resp = requests.post(url, headers=headers, data=data)

  platform.indicate_processing(False)
  process_response(resp)
```

Here we see a request being prepared for the AVS `recognize` endpoint. The headers suggest it's a multipart chunked message with the `Authorization` header set to our refreshed OAuth token. Before the call is made to the endpoint, however, the `audio_stream` is converted to the chunked data in the `alexa_speech_recognizer_generate_data` function. Once the data is ready, it's set to the `data` parameter, along with URL and headers in the `requests.post` function. Once that returns and populates `resp` with the AVS response, the `platform.indicate_processing(False)` call will turn off the processing LED and then the `process_response(resp)` call will prepare the response for audio playback on your device.

Conclusion

This chapter explained how to get Alexa to work in a Raspberry Pi. Keep in mind, however, that this is just one of several ways to do so. There are even more ways to accomplish this, particularly once we begin incorporating Cortana, Nuance, or any one of the other voice services out there. No matter which service you use, most likely it will use your own web service where your core business logic will live. While this could be added to your IoT device on the edge, ideally it would be in the cloud, which helps to facilitate smooth updates; otherwise, you would have to continuously push updates to your device.

In the next chapter we will focus on creating a custom web service that you can access from your newly created voice-enabled IoT device. We'll use a combination of technologies to accomplish this, but will focus primarily on the Alexa Skills Kit, which is how we as developers extend and enhance Alexa's skills.

CHAPTER 4

Iterate: Evolve the Prototype

So now that we have a working voice-enabled piece of hardware for us to work with, we'll want to start evolving it and customizing it to fit our use cases. First, let's get our feet wet and start with a basic "Hello, World" Amazon Alexa skill. Once we have that in place, you will have a better understanding of how to create skills, how to use intents and slots, as well as how to handle sessions, exceptions, and the various request types (e.g., `LaunchRequest`, `IntentRequest`, and `SessionEndedRequest`).

Why Iterate?

The evolution of a product isn't a single cycle occurrence. This is something that requires constant attention and fine-tuning. Development cycles should be kept short and nimble. This allows us to rule out issues fast, which is where the common mantra of "fail fast" comes from. Issues can arise from various points along the product lifecycle, from technical issues to user experience.

At this point, you're simply experimenting with the prototype we're building in these pages and learning as you go. But in the long run, you will want to plan for maintenance even if you're just building a device for yourself rather than something for market. We will touch on basic over-the-air firmware updates to cloud-based services updates in Chapter 6. This will help you set up efficient methods for updating your voice-enabled IoT device long into the future.

Intents, Utterances, Slots, and Invocation Names

Before we dive into creating our API, we need to learn some common natural language processing (NLP) terms. These terms are fairly universal across the current spectrum of NLP systems and form the foundational guide rails for natural language understanding (NLU) and the engines that drive them.

There are really four core terms you must be aware of: intents, utterances, slots (also known as entities), and invocation names. We touched upon some of them in Chapter 1, but they bear repeating. Let's break them down one by one:

Intents
: These are the actions users/bots can take within your skill. For example, if there is a skill for handling spa reservations, you would have intents that take care of date requests or filtering services. The naming convention for intents typically includes the word "Intent" in the name. For example, `ReservationDateIntent` or `ServiceTypeIntent`. It's not a hard-and-fast rule, but it does help to self-document code, as you will see in our examples. A skill's intents are created using an intent schema. An intent schema outlines all of the intents and slots (defined momentarily) that a given skill will handle. We will see more about intent schemas when we create our skill in the Amazon Developer Portal later in this chapter.

Utterances
: The words or phrases that will invoke a given intent. Think of utterances as the words users will say while using a particular skill. For instance, consider our reservation skill example. We would expect users to interact with our skill by saying phrases like "I'd like to book a spa appointment." That entire phrase is considered a single utterance, which we would then map to a single intent—we might call it the `InitiateReservationIntent`, for example. It's important to note that not every user will interact with your skill in the same way. For example, when scheduling an appointment, some users might say "I'd like to book a spa appointment," while others might say "Book me a spa appointment," or "I'd like to reserve a spa appointment." Thus, as developers, we need to anticipate and map all of the possible utterances (the "domain" of phrases) we believe users might speak to initiate a single action. In other words, we need to map multiple utterances to a given intent. This will help to ensure we trigger the correct intent, regardless of how users decide to phrase their request (note: this is not always easy and will take time to evolve as you learn how users interact with your skill. Local dialects, intonations, and accents can also affect how your skill will react to user utterances).

Slots (entities)
: These are the set (domain) of allowable terms for a given word within an utterance. For example, in our `InitiateReservationIntent`, we could create multiple utterances similar to the following:

- I'd like to book a spa appointment
- I'd like to reserve a spa appointment
- Book me a spa appointment
- Reserve me a spa appointment

While this is perfectly acceptable, slots simplify the utterance creation process. Let's revise the preceding list using slots:

- I'd like to `{verbType}` a spa appointment
- `{verbType}` me a spa appointment

As you can see, using slots trimmed our utterance list down to just two phrases. While that may not seem significant, it is extremely beneficial in a more practical, real-world skill example where you might have 15–30 utterances for a given intent. In this example, the `{verbType}` is a named slot containing all of the possible values we expect to be spoken by our users. For example, in the case of `InitiateReservationIntent`, the slot values for `{verbType}` are `book` and `reserve`. The key takeaway here is that slots can greatly simplify your utterance development and maintenance and should be used whenever possible.

Invocation name

An invocation name is the phrase used to invoke your skill (Amazon requires invocation names to be at least two words). For our spa reservation skill, we might use "My Spa" as the invocation name. Users of our skill would invoke it by saying, "Alexa, ask My Spa to book me a spa appointment."

Because we are working with AVS and ASK in this chapter, it's important to understand that an Alexa skill is simply a set of intents (or actions) around an idea or concept. Think of an Alexa skill as the voice equivalent of a smartphone app. The equivalent to a skill on the Google Home platform is referred to as an action, which can obviously create some confusion. Just remember to learn the terms and definitions of the platform you are working with before diving in—it will save you some headaches later on.

Amazon Alexa Requests and Responses

Users interact with your Alexa Skill via requests and responses. These objects are sent and received in the well-known JSON format. Amazon updates these objects periodically, as new features are added to the Alexa platform. For that reason, we won't break down each element of the request and response objects, but we want to outline the top-level request types before we move forward. The primary request types are as follows:

`LaunchRequest`

Sent when the user invokes a skill without providing a specific intent. For example, "Alexa, open my skill."

`IntentRequest`
: Sent when a user makes a request matching one of the intents within your skill intent schema.

`SessionEndedRequest`
: Sent when the skill session ends for any reason other than your code closing the session.

There is only one type of response object that can be returned from a skill, but the elements contained within the response may vary depending upon your particular skill. Both the request and response objects have mandatory and optional elements. Amazon has documented each element on the Amazon Developer Portal and we recommend reviewing the Custom Skill reference to familiarize yourself with JSON format.

Create Your Own API

First things first: you will need to pick your poison as it relates to web stacks. Do you love Node.js? Java? Python? C#? Have no idea what we're talking about here? No problem, we'll pick one for you and you can go on to experiment from there in case your use case is better suited to a different language. We will demonstrate how you can create a quick web API that's capable of handling Alexa requests in Node.js. While you can technically host these almost anywhere, we will focus our attention on Amazon Web Services (AWS), specifically Amazon's lambda functions using the `alexa-sdk` node module.

Alexa "Hello, World" in Node.js

OK, it's time to dive in. Amazon makes it very simple to get a skill up and running. We'll follow a step-by-step process, similar to the one we used to create our Raspberry Pi AVS solution. First, we'll configure and create our node skill within the AWS console. Next, we will set up a new skill in the Developer Portal (just like we did in Chapter 3). Finally, we'll test our skill using both the Developer Portal and an Amazon Dot, Echo, or Show. Let's get started!

Creating your Node.js Skill in AWS

1. Open a browser and go to the Amazon AWS console.
2. From the main navigation menu, click the My Account menu and select AWS Management Console (see Figure 4-1).

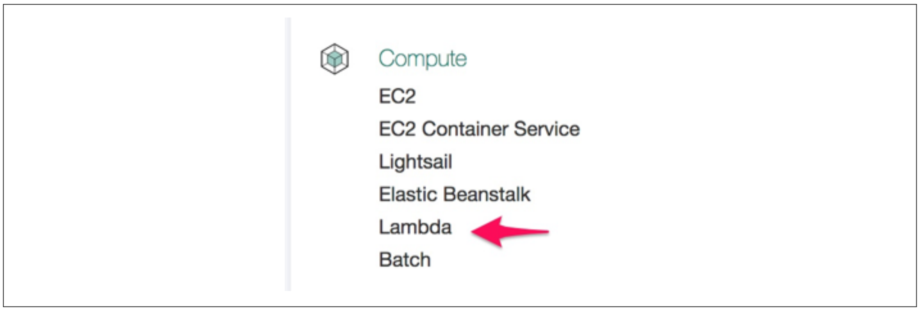


Figure 4-1. Accessing the AWS Management Console

3. Using the Amazon Developer Portal login you created in Chapter 3, log in to AWS. If successful, you will be redirected to the AWS Services page. There are a bunch of items listed on this page but we only need to focus on one. Under the Compute category, select Lambda (Figure 4-2).

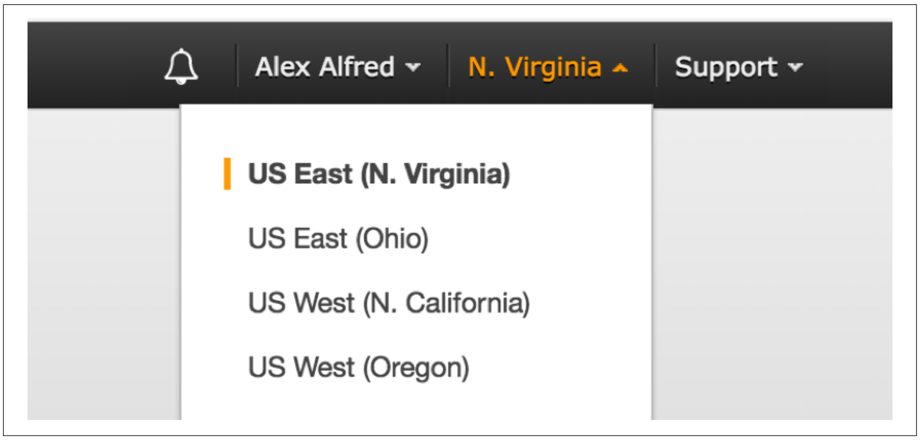
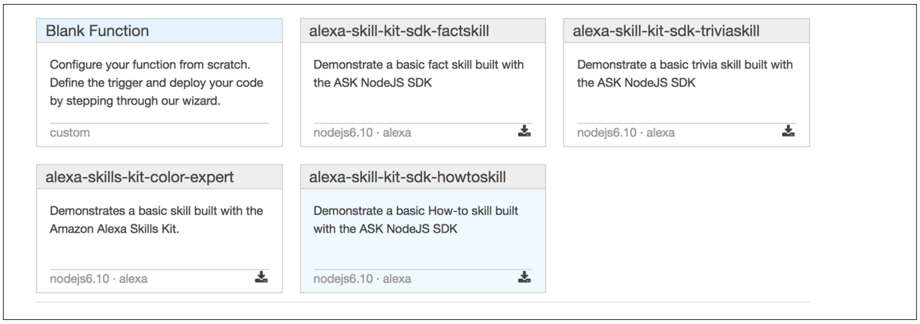


Figure 4-2. Selecting the Lambda service

You will be redirected to the AWS Lambda page. Click the region drop-down in the upper-right corner of the console and select US East (N. Virginia); see Figure 4-3. This is currently the only region that supports hosting of Alexa Skills.


Blank Function

Configure your function from scratch. Define the trigger and deploy your code by stepping through our wizard.

custom

alexa-skill-kit-sdk-factskill

Demonstrate a basic fact skill built with the ASK NodeJS SDK

nodejs6.10 · alexa

alexa-skill-kit-sdk-triviaskill

Demonstrate a basic trivia skill built with the ASK NodeJS SDK

nodejs6.10 · alexa

alexa-skills-kit-color-expert

Demonstrates a basic skill built with the Amazon Alexa Skills Kit.

nodejs6.10 · alexa

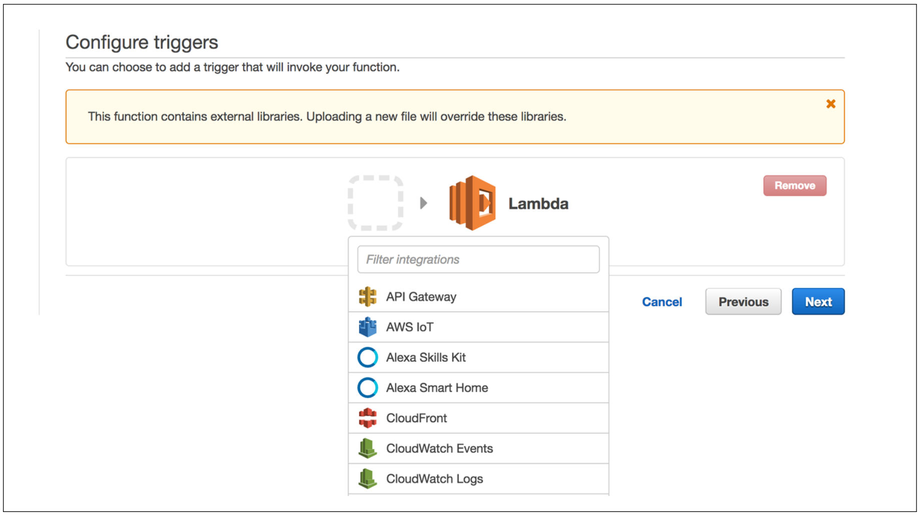
alexa-skill-kit-sdk-howtoskill

Demonstrate a basic How-to skill built with the ASK NodeJS SDK

nodejs6.10 · alexa

Figure 4-3. Selecting your region

4. Next, click the Get Started button. This will direct you to the Lambda blueprint page. You will see Select Runtime and Filter options. If you click the Select Runtime option, you will see that Amazon supports multiple flavors of Node.js as well as Python (and ASP.NET core, although it's not listed in the runtime dropdown). For our example, select Node.js 6.x.
5. Next, in the Filter text box, enter "alexa" (without the quotes). This will filter the skill templates to just Alexa-based options.
6. Amazon provides a great list of Alexa Skill templates and we encourage you to take each of them for a test drive. They provide great examples of how you can create a variety of Alexa skills. For our example, we will start with the "alexa-skill-kit-sdk-howtoskill" template (Figure 4-4). Using this template ensures that the Alexa-SDK library is installed as part of the template. Having the SDK preinstalled simplifies the remaining steps.

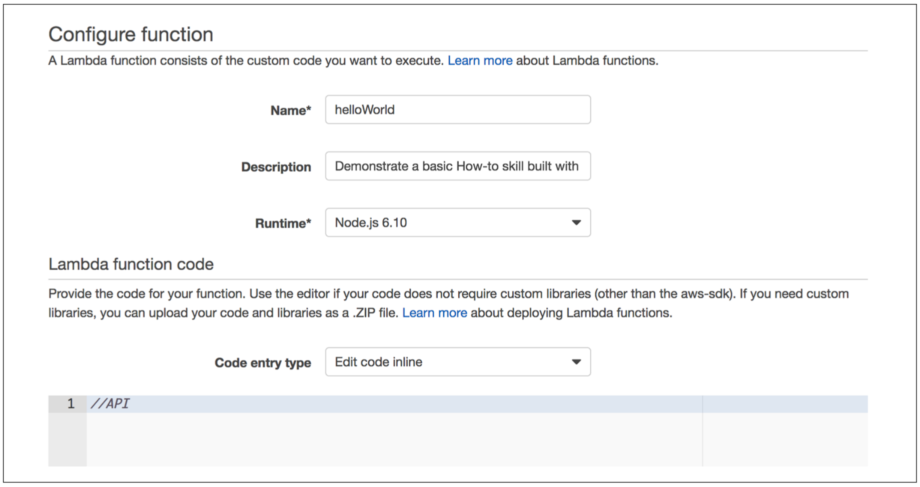

Configure triggers

You can choose to add a trigger that will invoke your function.

This function contains external libraries. Uploading a new file will override these libraries.

Figure 4-4. Selecting an Alexa SDK template

7. Adding the template will direct you to the "Configure triggers" page. From the pop-up window, click the dash-enclosed box to reveal a trigger list. Select Alexa Skills Kit and then click Next (Figure 4-5).



Configure function

A Lambda function consists of the custom code you want to execute. Learn more about Lambda functions.

Name*	helloWorld
Description	Demonstrate a basic How-to skill built with
Runtime*	Node.js 6.10

Lambda function code

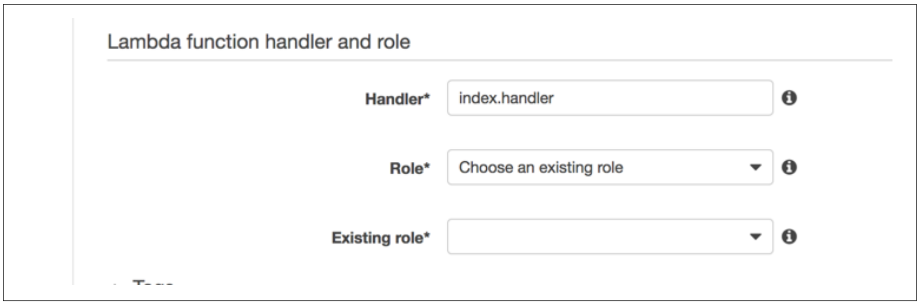
Provide the code for your function. Use the editor if your code does not require custom libraries (other than the aws-sdk). If you need custom libraries, you can upload your code and libraries as a .ZIP file. Learn more about deploying Lambda functions.

Code entry type	Edit code inline

```
//API
```

Figure 4-5. Selecting a trigger for our API

8. You should now be at the "Configure function" page. This is where we will name and code our hello world skill function. Let's call our skill "Hello, World." The description is optional, but make sure the runtime is set to Node.js 6.xx (Figure 4-6). Finally, select all of the Node.js code in the editor window and delete it. In the editor window, add a simple remark as a placeholder for our future API.



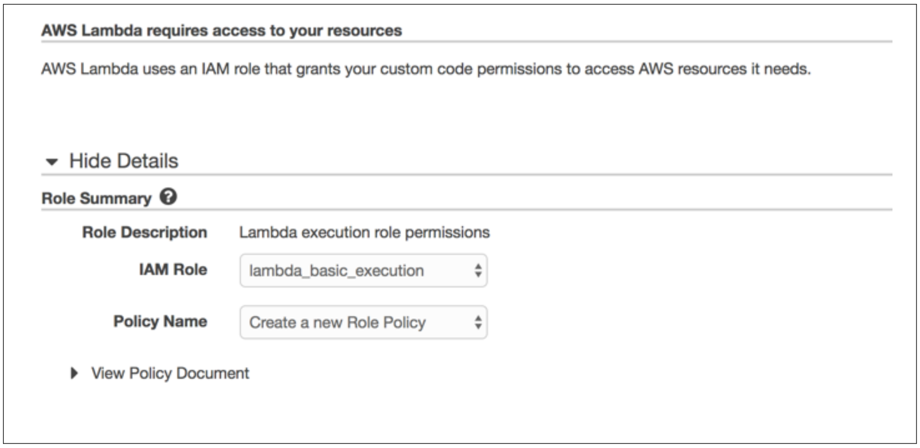
Lambda function handler and role

Handler*	index.handler
Role*	Choose an existing role
Existing role*	

Figure 4-6. AWS Lambda function configuration form

You might be wondering why we deleted all that code. We used this template to ensure the Alexa-SDK is pre-installed, however, we want to create our own custom skill. Consequently, we need to remove templated code and replace it with our own, which we will do shortly. If you're feeling more adventurous and would like to start with the "Blank Template," you will need to upload a ZIP file with your source and the alexa-sdk node modules rather than simply starting in the code editor. For further details, visit the latest documentation on the topic.

9. Scroll down to the "Lambda function handler and role" section. The default value for Handler does not need to be changed for our example. However, we do need to update the Role value. Using the Role drop-down box, select "Create custom role" (Figure 4-7).



AWS Lambda requires access to your resources

AWS Lambda uses an IAM role that grants your custom code permissions to access AWS resources it needs.

▾ Hide Details

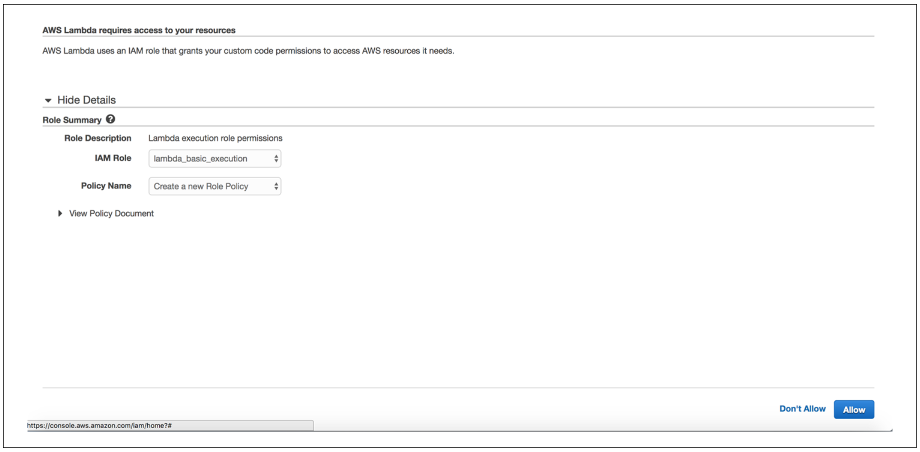
Role Summary

Role Description	Lambda execution role permissions
IAM Role	lambda_basic_execution
Policy Name	Create a new Role Policy

▸ View Policy Document

Figure 4-7. Specifying function handler and role

10. Creating a custom role will open IAM Role page (Figure 4-8). You can accept all the default settings and just select Allow. This will close the IAM page and update the Role on your function page.

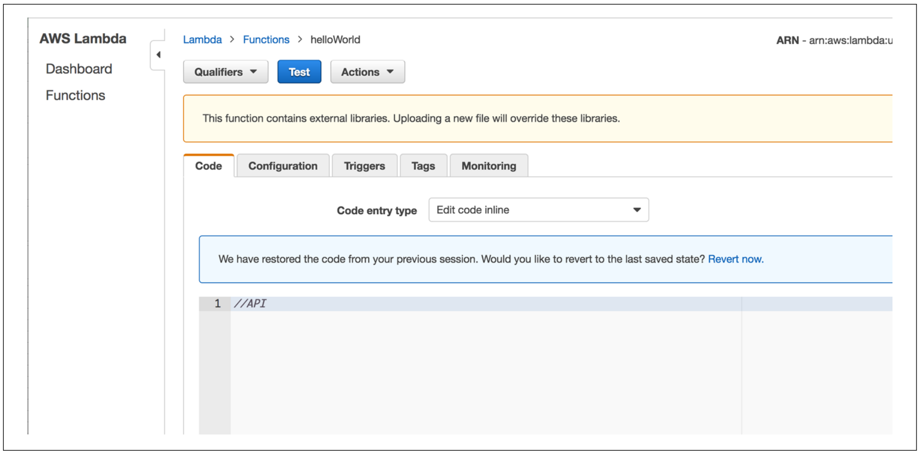

AWS Lambda requires access to your resources

AWS Lambda uses an IAM role that grants your custom code permissions to access AWS resources it needs.

Role Description: Lambda execution role permissions
IAM Role: lambda_basic_execution
Policy Name: Create a new Role Policy

Figure 4-8. Setting up a new IAM role

11. Now that we have the existing role updated, our function will be able to execute. You can leave all of the remaining defaults and select Next (Figure 4-9).



This function contains external libraries. Uploading a new file will override these libraries.

Code entry type: Edit code inline

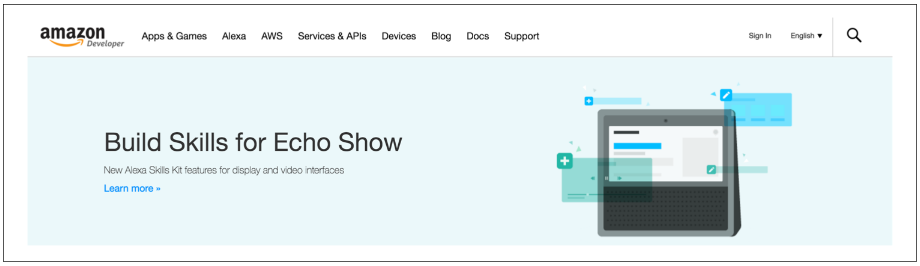
We have restored the code from your previous session. Would you like to revert to the last saved state? Revert now.

```
//API
```

Figure 4-9. Selecting an existing role

12. You will then be taken to the Review page, where you are given an opportunity to edit your settings before creating your function. If you followed the preceding steps you can just select Create Function.

13. So far so good! Now we can start coding up our `helloWorld` function. When you clicked Create Function, you should have been directed to the page shown in Figure 4-10. From there, you can create the intent handlers and wire them up to the main request handler. When we create our skill in the Amazon Developer Portal we will need the ARN, located at the top right of the function page. Let's get started.



Build Skills for Echo Show

New Alexa Skills Kit features for display and video interfaces

Learn more »

Figure 4-10. Lambda code editor

Writing our lambda function in Node.js

This section assumes a basic familiarity with Node.js, JavaScript, and node packages. With that little caveat out of the way, let's get started on our "Hello, World" API.

In the code editor window of your lambda function, remove the "//API" remark we added when we created the functions, and replace it with the code shown here (we have included inline notes to explain the purpose of each major section within the code):

```
'use strict';

//Wire-up the Alexa SDK for Node.js
var Alexa = require('alexa-sdk');

//we will used this later to validate
//requests to our lambda function
var APP_ID = undefined;

exports.handler = (event, context, callback) => {
    var alexa = Alexa.handler(event, context);
    //When we wire up our APP_ID, we ensure that our
    //API can only process requests from our registered
    //skill
    alexa.APP_ID = APP_ID;
    //Here is where we register our intent handlers
    //more than one handler can be registered
    alexa.registerHandlers(handlers);
    alexa.execute();
};

//This is our intent handler
//All requests to our skill will be routed to
//this handler for resolution.
var handlers = {
    //This handles all LaunchRequests
    'LaunchRequest': function () {
        this.emit('SayHelloWorld');
    },
    //This handles all SessionEndedRequests
    'SessionEndedRequest': function () {
        //used for any housekeeping
    },
    //This handles our custom HelloWorldIntent
    'HelloWorldIntent': function () {
        this.emit('SayHelloWorld')
    },
    //This is our custom HelloWorld function
    //It calls the Alexa SDK via the emit event emitter
    //to build a HelloWorld response for our users.
    'SayHelloWorld': function () {
        this.emit(':tell', 'Hello World!');
    }
};
```

For more information on the Alexa SDK for Node.js, visit the full documentation at *https://github.com/alexa/alexa-skills-kit-sdk-for-nodejs*.

After you have made the code updates, click the Save button located just above the coding window.

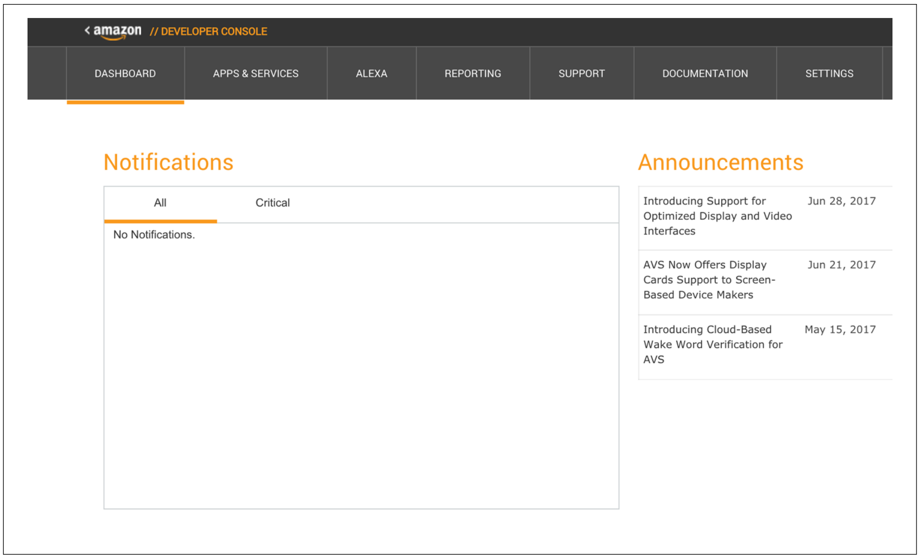
Congratulations! You just created your first Node.js Alexa API.

Now that we have our API in place, we need to create and register our custom skill within the Amazon Developer Portal. There we will create our intent schema and utterances as well as connect our skill to our new API. Let's head over to the Developer Portal and get started.

Amazon Developer Portal

This is a great place for a quick recap of what we've accomplished thus far. In Chapter 3 we created our Raspberry Pi AVS, which we will use as our test Alexa device. And in this chapter, we've created our our lambda function, which we will use as our skill's API. Now we need to create and register our custom skill in the Developer Portal. Let's get started:

1. Visit the Amazon Developer Portal and click Sign In (top right of the menu bar, as shown in Figure 4-11).

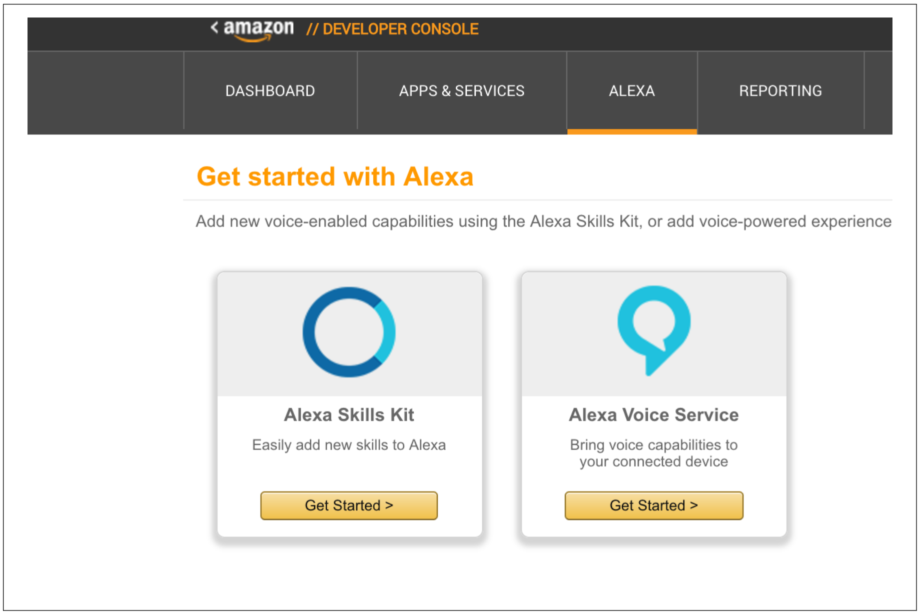


No Notifications.

Introducing Support for Optimized Display and Video Interfaces	Jun 28, 2017
AVS Now Offers Display Cards Support to Screen-Based Device Makers	Jun 21, 2017
Introducing Cloud-Based Wake Word Verification for AVS	May 15, 2017

Figure 4-11. Amazon Developer Portal

2. Using the Amazon developer username you created in Chapter 3, log in to the portal. If successful, you will be redirected to the Developer Dashboard page (Figure 4-12). There are many options to choose from on this page but we only need to focus on one. Click the Alexa option on the menu bar.



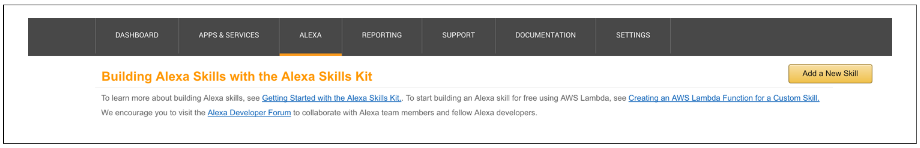
Add new voice-enabled capabilities using the Alexa Skills Kit, or add voice-powered experience

Easily add new skills to Alexa

Bring voice capabilities to your connected device

Figure 4-12. Developer dashboard

3. Once you are directed to the Get Started with Alexa page, click Get Started within the Alexa Skills Kit option (Figure 4-13).

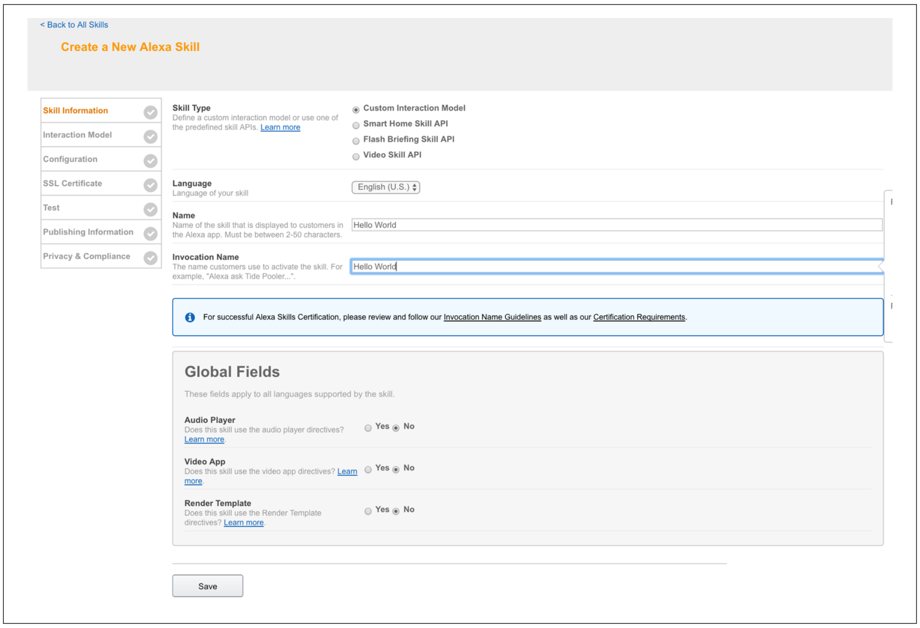


Building Alexa Skills with the Alexa Skills Kit

To learn more about building Alexa skills, see Getting Started with the Alexa Skills Kit.. To start building an Alexa skill for free using AWS Lambda, see Creating an AWS Lambda Function for a Custom Skill.
We encourage you to visit the Alexa Developer Forum to collaborate with Alexa team members and fellow Alexa developers.

Figure 4-13. Getting started with Alexa

4. You should now see an Add a New Skill button. This page is also your custom skill dashboard, where all of the custom skills you develop will be listed. We haven't created any skills yet, so go ahead and click Add a New Skill (Figure 4-14).



Create a New Alexa Skill

Skill Type
Define a custom interaction model or use one of the predefined skill APIs. Learn more
[✓ Custom Interaction Model]
Smart Home Skill API
Flash Briefing Skill API
Video Skill API

Language
Language of your skill
English (U.S.)

Name
Name of the skill that is displayed to customers in the Alexa app. Must be between 2-50 characters.
Hello World

Invocation Name
The name customers use to activate the skill. For example, "Alexa ask Tide Pooler...".
Hello World

For successful Alexa Skills Certification, please review and follow our Invocation Name Guidelines as well as our Certification Requirements.

Global Fields

These fields apply to all languages supported by the skill.

Audio Player
Does this skill use the audio player directives? Learn more.
Yes [✓ No]

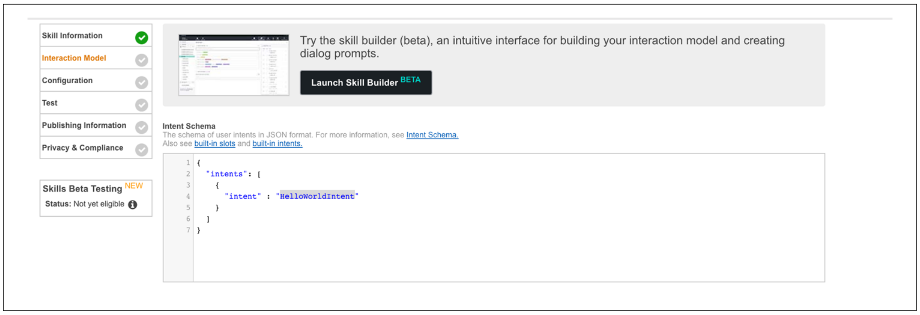
Video App
Does this skill use the video app directives? Learn more.
Yes [✓ No]

Render Template
Does this skill use the Render Template directives? Learn more.
Yes [✓ No]

Figure 4-14. Your custom skills list

5. Now we can start creating our skill details for registration in the Developer Portal. Notice there are seven tabs running across the lefthand side of the page. Each tab holds configuration data for your custom skill. The Skill Information tab elements are shown in Figure 4-15. Figure 4-15 shows the completed tab for our "Hello, World" skill. Fill in the Skill Information tab with the settings shown here, and click Save and then Next.

The options available on your custom skill tabs may be slightly different from the ones shown, but the core items should be identical.



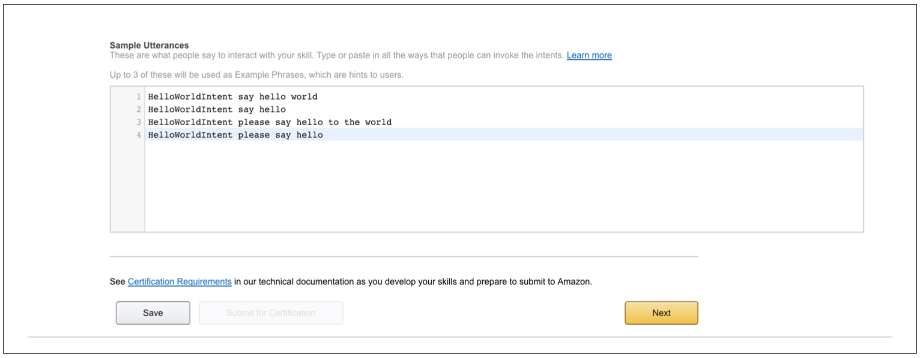
Try the skill builder (beta), an intuitive interface for building your interaction model and creating dialog prompts.

Intent Schema
The schema of user intents in JSON format. For more information, see Intent Schema.
Also see built-in slots and built-in intents.

```
{
  "intents": [
    {
      "intent" : "HelloWorldIntent"
    }
  ]
}
```

Figure 4-15. Skill Information tab

6. The Interaction Model is where we set up up our intent schema, slots, and utterances. We won't need slots for this "Hello, World" example but our intent schema and utterances are shown in Figures 4-16 and 4-17. You'll notice that we created four utterances for our skill. This allows our users to ask our "Hello, World" skill to "Say, 'Hello'" using one of the four utterances shown. For example, our users can say "Alexa, ask Hello World to say 'hello'" or "Alexa, ask 'Hello, World' to please say 'hello' to the world." Skills typically have multiple intents and each of those intents will have multiple utterances. We kept it rather simple for our particular sample, but it's important to remember that the consistency and accuracy of your skill is greatly dependent upon the diversity of your sample intent utterances. With that said, let's update your Interaction Model tab as shown in Figure 4-16 and click Save and then Next. (Save can take some time with more complex models, so please be patient as it builds your model.)



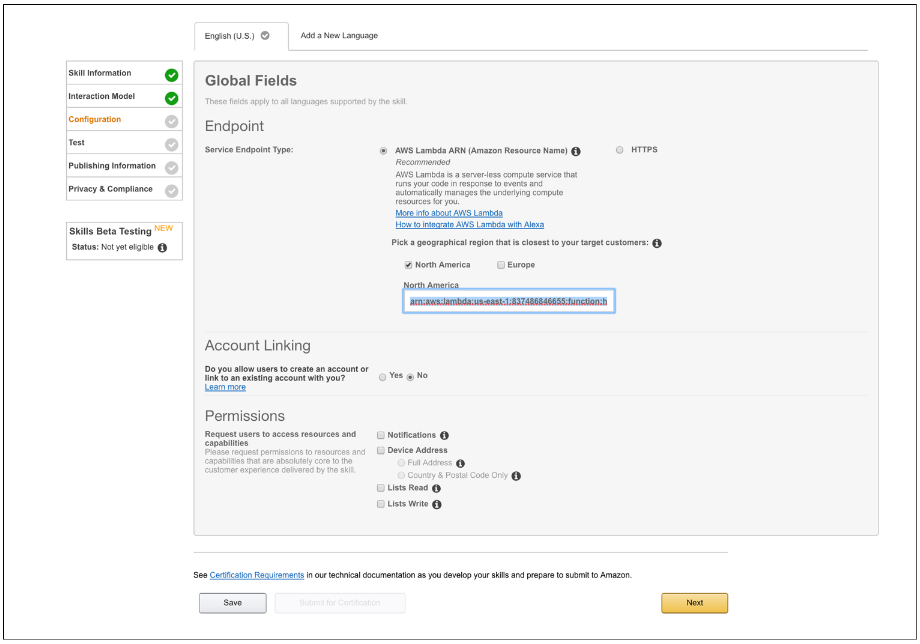
Sample Utterances
These are what people say to interact with your skill. Type or paste in all the ways that people can invoke the intents. Learn more

Up to 3 of these will be used as Example Phrases, which are hints to users.

```
HelloWorldIntent say hello world
HelloWorldIntent say hello
HelloWorldIntent please say hello to the world
HelloWorldIntent please say hello
```

See Certification Requirements in our technical documentation as you develop your skills and prepare to submit to Amazon.

Figure 4-16. "Hello, World" intent schema


English (U.S.) | Add a New Language

Skill Information ✓
Interaction Model ✓
Configuration
Test
Publishing Information
Privacy & Compliance

Skills Beta Testing NEW
Status: Not yet eligible

Global Fields

These fields apply to all languages supported by the skill.

Endpoint

Service Endpoint Type: [✓ AWS Lambda ARN (Amazon Resource Name)] HTTPS

Recommended

AWS Lambda is a server-less compute service that runs your code in response to events and automatically manages the underlying compute resources for you.

More info about AWS Lambda

How to integrate AWS Lambda with Alexa

Pick a geographical region that is closest to your target customers:

[✓ North America] Europe

North America: arn:aws:lambda:us-east-1:837486846655:function:h

Account Linking

Do you allow users to create an account or link to an existing account with you? Learn more

Yes [✓ No]

Permissions

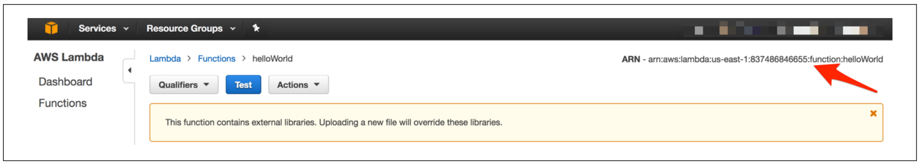
Request users to access resources and capabilities
Please request permissions to resources and capabilities that are absolutely core to the customer experience delivered by the skill.

- ☐ Notifications
- ☐ Device Address
  - ○ Full Address
  - ○ Country & Postal Code Only
- ☐ Lists Read
- ☐ Lists Write

See Certification Requirements in our technical documentation as you develop your skills and prepare to submit to Amazon.

Figure 4-17. Sample utterances

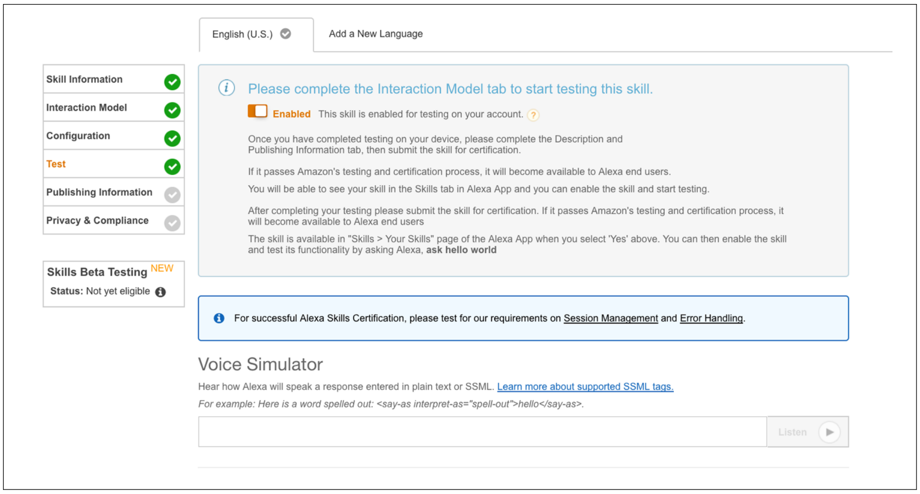
7. The Configuration tab is where we connect our portal skill to our lamdba function. The link to your lambda function can be found in the AWS portal (shown in Figure 4-18). Copy the entire ARN value and paste as indicated in Figure 4-19. You'll notice that we could also use a standard HTTPS endpoint as our skill API endpoint, so you have many options when it comes to hosting your Alexa API.



ARN - arn:aws:lambda:us-east-1:837486846655:function:helloWorld

This function contains external libraries. Uploading a new file will override these libraries.

Figure 4-18. AWS Portal ARN ID


English (U.S.)

Add a New Language

Skill Information ✓

Interaction Model ✓

Configuration ✓

Test ✓

Publishing Information

Privacy & Compliance

Skills Beta Testing NEW

Status: Not yet eligible

Please complete the Interaction Model tab to start testing this skill.

Enabled This skill is enabled for testing on your account.

Once you have completed testing on your device, please complete the Description and Publishing Information tab, then submit the skill for certification.

If it passes Amazon's testing and certification process, it will become available to Alexa end users.

You will be able to see your skill in the Skills tab in Alexa App and you can enable the skill and start testing.

After completing your testing please submit the skill for certification. If it passes Amazon's testing and certification process, it will become available to Alexa end users

The skill is available in "Skills > Your Skills" page of the Alexa App when you select 'Yes' above. You can then enable the skill and test its functionality by asking Alexa, **ask hello world**

For successful Alexa Skills Certification, please test for our requirements on Session Management and Error Handling.

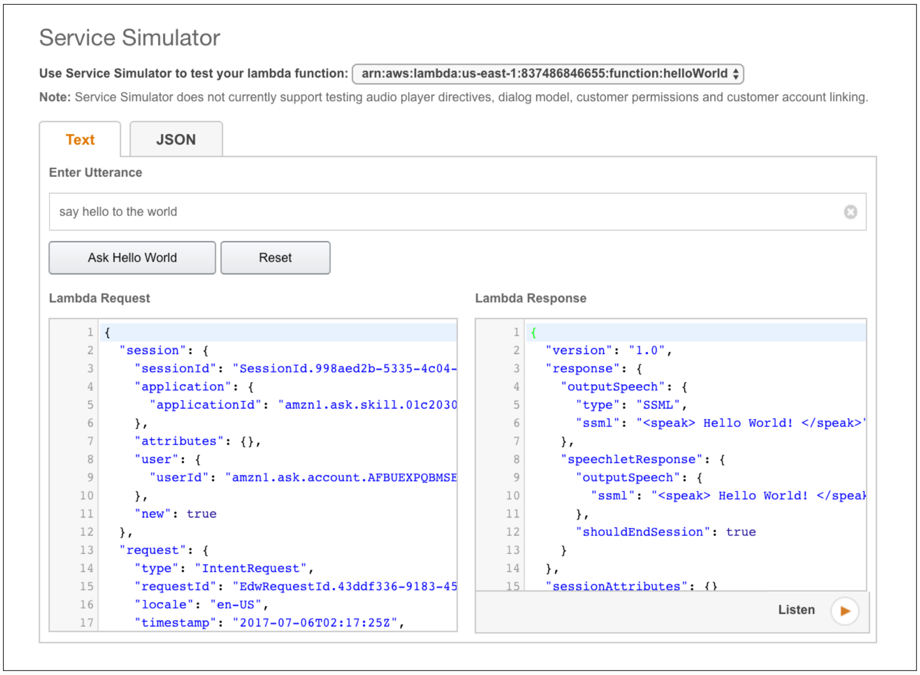
Voice Simulator

Hear how Alexa will speak a response entered in plain text or SSML. Learn more about supported SSML tags.

For example: Here is a word spelled out: <say-as interpret-as="spell-out">hello</say-as>.

Figure 4-19. Configuration tab

8. The final tab we will set up for our “Hello, World” skill is the Test tab. The Test tab allows us to test our skill by passing in typed sample utterances. The first thing we need to do is turn on testing for this skill (Figure 4-20). Make sure the testing option is set to Enabled. Once enabled, scroll down to the Service Simulator and type the sample utterance, **`say hello world`**, then click “Ask Hello World” (Figure 4-21).


Service Simulator

Use Service Simulator to test your lambda function: arn:aws:lambda:us-east-1:837486846655:function:helloWorld

Note: Service Simulator does not currently support testing audio player directives, dialog model, customer permissions and customer account linking.

Enter Utterance

say hello to the world

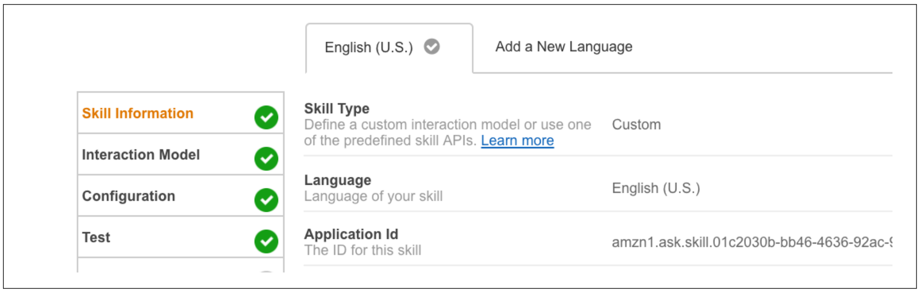
Lambda Request

```
{
  "session": {
    "sessionId": "SessionId.998aed2b-5335-4c04-
    "application": {
      "applicationId": "amzn1.ask.skill.01c2030
    },
    "attributes": {},
    "user": {
      "userId": "amzn1.ask.account.AFBUEXPQBMSE
    },
    "new": true
  },
  "request": {
    "type": "IntentRequest",
    "requestId": "EdwRequestId.43ddf336-9183-45
    "locale": "en-US",
    "timestamp": "2017-07-06T02:17:25Z",
```

Lambda Response

```
{
  "version": "1.0",
  "response": {
    "outputSpeech": {
      "type": "SSML",
      "ssml": "<speak> Hello World! </speak>"
    },
    "speechletResponse": {
      "outputSpeech": {
        "ssml": "<speak> Hello World! </speak
      },
      "shouldEndSession": true
    }
  },
  "sessionAttributes": {}
```

Figure 4-20. Enabling testing of your skill



Skill Type Define a custom interaction model or use one of the predefined skill APIs. Learn more — Custom

Language Language of your skill — English (U.S.)

Application Id The ID for this skill — amzn1.ask.skill.01c2030b-bb46-4636-92ac-

Figure 4-21. Testing your skill

Notice that you don't need to type the invocation name of "Hello, World" to run this test. Invocation names are only needed for voice-based tests. The simulator lists the exact request object sent to your lambda function as well as the response object returned by your skill. As you can see, there are quite a few JSON elements in each object and we encourage you to read through the Amazon documentation referenced earlier. The Alexa SDK node package handles most of the heavy lifting of building response objects for your skill, but it also provides finer control of each element should your skill require it. There are a couple things to point out, however: first, the Alexa SDK always returns your `outputSpeech` as Speech

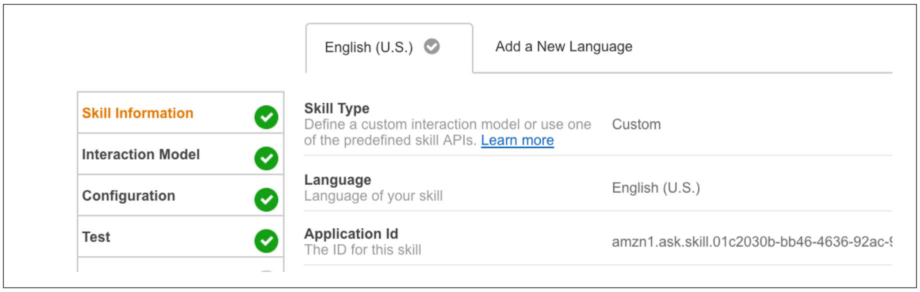
Synthesis Markup Language (SSML), so if your response output contains special characters like an ampersand (&) or brackets (< or >), you will need to escape those characters before they are returned. Second, by default the Alexa SDK sets `shouldEndSession` to true, so if you need to keep the session open and wait for user input, you will need to build your response object using syntax similar to the following:

```
...
//This is our intent handler
//All requests to our skill will be routed to
//this handler for resolution.
var handlers = {
    ...
    //This is our custom HelloWorld function
    //It calls the Alexa SDK via the emit event emitter
    //to build a HelloWorld response for our users.
    'SayHelloWorld': function () {
        //sets the outputSpeech element
        this.response.speak('Hello World!');
        //sets shouldEndSesssion to false and
        //and sets the repromptSpeech element
        this.response.listen('I said, Hello World!');
        //returns the response object
        this.emit(':reponseReady');
    }
};
```

Finally, if you click the Listen icon, you will hear the response as it would be played through an Alexa device.

Amazon provides a rich set of SSML offerings that you can use as part of your `outputSpeech` response element. SSML allows you to emphasize words, and can generally make your interactions feel more realistic and human-like. For more information on SSML with Alexa, check out the documentation.

9. The remaining tabs, Publishing Information and Privacy and Compliance, are required whenever you wish to submit your skill for certification. The field elements are self-explanatory and Amazon provides comprehensive documentation detailing how to complete the required items. For our example, we can skip those tabs.
10. Now that we have the skill registered, we need to add the skill App ID to our lambda function. Go back to the Skill Information tab and select and copy the entire value for your skill's Application ID; it starts with "amzn1.ask.skill" (Figure 4-22).



Skill Type
Define a custom interaction model or use one of the predefined skill APIs. Learn more
Custom

Language
Language of your skill
English (U.S.)

Application Id
The ID for this skill
amzn1.ask.skill.01c2030b-bb46-4636-92ac-

Figure 4-22. Application ID of skill

Next, log back in to the AWS portal so we can add the App ID to your lambda function. Open your lambda function in the portal code editor and replace the text "undefined" with your skill's Application ID (enclosed in single quotes) and save your lambda function. Now your lambda function will only accept requests from the skill matching your Application ID:

```
'use strict';

//Wire-up the Alexa Node.js
var Alexa = require('alexa-sdk');

//we will use this later to validate
//requests to our lambda function
var APP_ID = undefined;
...
```

Congratulations, you have completed your very first Amazon skill! While this first skill may be very simplistic in function, you now have the foundational understanding to create skills of varying size and complexity. Be proud of yourself!

Testing Custom Skills on Your Device

We created our lambda function and our skill, and we even tested a sample utterance in the Developer Portal. All that is great but let's face it: we want our skill to work using our voice requests. Luckily, testing a skill is really simple.

In this chapter, we created and registered our custom Alexa skill with the same Amazon developer login used to create our Raspberry Pi AVS. Since we used the same login ID for both, our AVS Raspberry Pi (and any other Amazon device registered with the same Amazon login) automatically has access to every custom skill created under that login. The caveat of course is that the custom skill must be "enabled" for testing under the Testing tab in the Developer Portal. Since we already enabled our skill for testing, your AVS device created in Chapter 3 should be ready to access your new "Hello, World" skill. Let's give it a try, shall we?

If your AVS device isn't already running, plug it in or if need be, follow the steps for creating the AVS-enabled device in Chapter 3. Once your device is up and running, just ask Alexa to say "hello" by saying "Alexa, ask 'Hello, World' to say 'hello.'" If you followed the instructions outlined in Chapters 3 and 4, Alexa should respond with "Hello, World." Welcome to the world of voice IoT!

What's Next?

We covered a lot in this chapter. However, there are a few more items we wanted to share as you start down your voice-based path.

As with any system, user experience (UX) is paramount. If the interaction performs inconsistently or feels too robotic or clunky, users will not enjoy the experience. A main goal of AI engineers and UX designers is to have the experience feel as natural as possible, like a conversation with another human. That is not a simple feat. We are still in the early stages of truly intelligent voice interfaces. We have only scratched the surface of where this technology will eventually lead. And while there are challenges with creating rich, interactive voice interfaces, there are things we can do to make the experience more natural. Here are a few principles we keep in mind when designing your interactions:

Design for conversation
: When creating your skill's interaction flow, design it as if you were writing a movie script rather than a Q&A with a computer. Amazon has recently added new interfaces to help with this process, specifically the Dialog Interface. This new interface simplifies the multiturn (conversational) interactions between the user and your skill. Official details and examples can be found at *https://github.com/alexa/alexa-skills-kit-sdk-for-nodejs*.

Prepare for the unexpected
: If you have experience with developing software, you are keenly aware that some users do the unexpected—they want to do things differently or in a flow that diverges from the original design. When it comes to voice-based interactions, this is magnified by 1,000. It can be very challenging to predict all variations of intent utterances, not to mention account for various local vernacular differences (*soda pop* versus *cola*, *hoagie* versus *sub*). Plan for the unexpected in your design. Experiment with utterance wording, research alternate terms for similar words within your slots, and test, test, test.

Vary the dialogue
: This principle is closely tied to designing for conversation. When people interact with one another, the personalities of those involved affects the dialogue. For example, a simple question like "How are you doing?" will elicit a number of responses depending upon who is being asked. Some people may respond with,

"I'm great! How about you?" while others might simply say, "I'm fine." The point is, variation is human, variation is natural. When you design your responses, create natural variations, especially when providing listed responses. Varying the phrasing with which you respond provides a more human feel to your AI, offering a richer user experience. For instance, consider our "Hello, World" example. To make it more interesting and human-like, we could create an array of possible responses for `HelloWorldIntent`. Instead of just saying "Hello, World" we could say "Well, hello there, World" or "Hello, cold, cruel World" or perhaps even, "Hello, you big, beautiful, blue marble." These simple variations make interacting with skills more fun and even provide a form of anticipation, as our users aren't quite sure what to expect (in a good way).

In the not-too-distant future, the AIs we interact with will be more intelligent and have a more natural feel to them (think Star Trek computer). They will "learn" in real time and be able alternate dialogue and even adapt based on conversational context. Until then, however, it's up to us to design AI experiences to be as human-like and natural as possible.

Account Linking

We also wanted to address the topic of account linking. Account linking enables you to tie your skill's users to a new or existing backend system. Banking-related skills leverage account linking extensively, but any skill can use the feature. Account linking is a more advanced topic and multiple chapters could be written about it, but if you do plan on using it, we recommend a thorough read (and perhaps a second read as well) of the Amazon Account linking documentation.

State Management

Another topic of interest is session or state management. Very often, a skill will need to maintain state to track where a user is within a given process—a reservation system, for instance. There are several ways this can be achieved, but Amazon provides detailed documentation on maintaining state using the Alexa SDK and AWS. We highly encourage reading through this documentation once you begin building more complex skills.

Bigger Picture

Finally, while Node.js and AWS are incredible tools for building custom skills, you can also build skills using other tools and platforms. If you love C# (and let's face it, who doesn't), we encourage you to check out Walter's Pluralsight course: Developing Alexa Skills for Amazon Echo. The course walks you through creating an Alexa skill hosted on Microsoft's Azure Cloud platform, and even touches on account linking and session management.

That's a wrap on this chapter! In Chapter 5, we will take a look at using Windows IoT Core on a Raspberry Pi with Google's API.AI engine.

CHAPTER 5

A Different Approach Using IoT Core and API.AI

So far we have built a Raspberry Pi–based AVS and created our very first lambda function to support a custom Alexa skill. It's safe to say, we have accomplished quite a bit in the past few chapters. In this chapter, we are going even further, but with a twist. Rather than further extending our custom Alexa skill, we will bring another platform into the mix to keep things fun and interesting. We are going to build an Alexa-like device using Windows IoT Core running on a Raspberry Pi. On top of all that, we will leverage Google's API.AI as our cloud-based NLU/NLP. Pretty exciting, eh? Why are we doing this, you might ask? Well, in addition to having fun, we want to expose you to another platform and demonstrate the benefits of experimenting with different AI systems. Kick the tires, so to speak. We already know that Alexa is an amazing intelligent voice platform, but API.AI is powerful in its own right and we want to introduce you to some of its core features.

We'll be building another "Hello, World" project using API.AI as the NLU/NLP. In addition, we'll build a Universal Windows Platform (UWP) app that will handle the voice interactions (record and playback) between you and API.AI. While our app will not have the power and sophistication that Alexa offers out of the box, we will be able to create robust, highly customized voice interactions using our UWP and API.AI. Lastly, the most important takeaway from this effort is that you will have an understanding of how to build intelligent interactions using API.AI, not to mention you will have learned how to create a cool little voice-activated app on IoT Core.

Chapter 2 briefly introduced API.AI, but here we will dive a little deeper and demonstrate how to create an API.AI custom agent, intents, and responses. Then we will outline how to integrate your API.AI agent into a custom UWP app running on IoT Core. Figure 5-1 illustrates what we will build in this chapter.

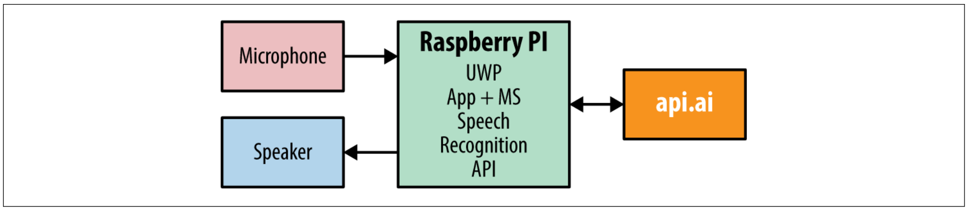
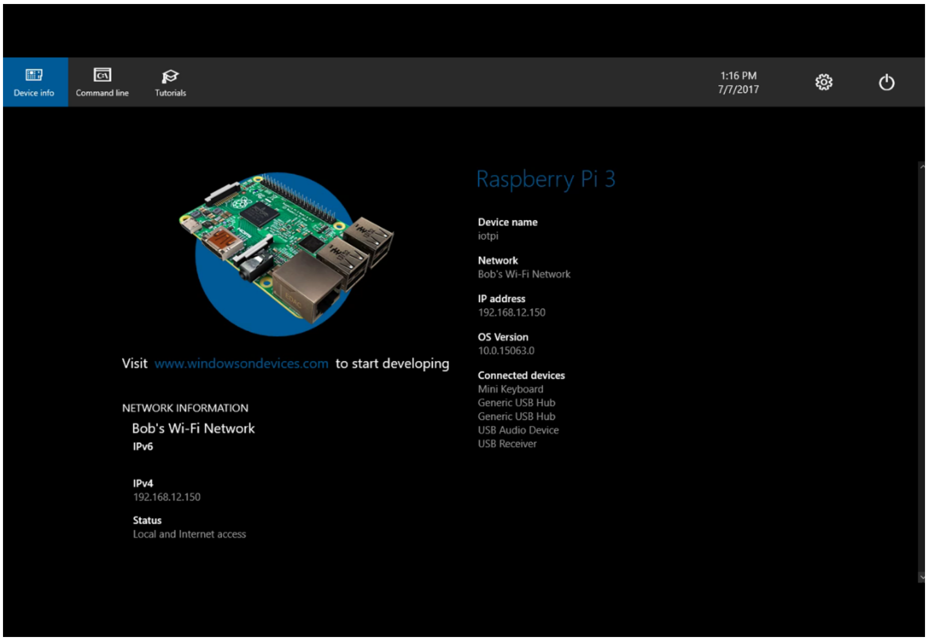


Figure 5-1. Voice interface on IoT Core + API.AI

We will be moving away from Node.js at this point, and instead will create a UWP using C#. We'll touch more on API.AI as we move through the build process.

IoT Core

Before we get started we should probably talk a little about Windows IoT Core. IoT Core is Microsoft's OS for IoT. Consider it a scaled-down version of Windows 10 that will run on x86/x64 and ARM devices, like a Raspberry Pi. It doesn't come with all the bells and whistles of Windows 10 for the desktop; there isn't a task bar or personalized desktop, for example. In fact, there is just a main dashboard, shown in Figure 5-2.



Raspberry Pi 3

Device name
iotpi

Network
Bob's Wi-Fi Network

IP address
192.168.12.150

OS Version
10.0.15063.0

Connected devices
Mini Keyboard
Generic USB Hub
Generic USB Hub
USB Audio Device
USB Receiver

Visit www.windowsondevices.com to start developing

NETWORK INFORMATION
Bob's Wi-Fi Network
IPv6

IPv4
192.168.12.150

Status
Local and Internet access

Figure 5-2. Windows IoT Core main dashboard

Microsoft is very serious about IoT Core and is committed to expanding its reach into the world of IoT devices. It even offers a version of Windows IoT for the Enterprise. As you will learn in this chapter, IoT Core is a very capable OS for IoT, especially for those of you who are well versed in C# or other MS platform languages.

Tools and Things

The good news is that you can reuse all the same hardware from Chapter 3 (refer back to "Tools and Things" on page 42). However, we recommend using a separate SD card so you don't overwrite the AVS project we created in Chapter 4. For this exercise, you will also need a workstation/laptop running Windows 10, as the tools provided by MS to provision IoT Core are currently only available for Windows. Of course, once you have installed IoT Core onto your Raspberry Pi, you will be able to control most of its features from any web browser.

As mentioned in Chapter 3, you will need a way to write to a micro SD card. If your workstation has an SD card reader, great; otherwise, you will need to buy a USB SD card reader and adapter. These are currently available on Amazon for as low as $8.

Once you have all of the necessary components, you will need to prepare your Pi. The following section will help you do just that.

Preparing Your Pi

Let's get connection logistics out of the way first. We recommend connecting your Pi 3 to an HDMI monitor as well as a USB keyboard and mouse. Then hook up your Pi with a power adapter, a microphone, and a speaker. With your SD card in hand, follow these steps to set up a new instance of a IoT Core on Raspberry Pi:

1. To create a new IoT Core Pi from scratch, you will need to download a copy of IoT Core onto your Windows workstation.
2. Once you have downloaded the IoT Core *Setup.exe* file, double-click it to launch the IoT Dashboard Tools. If you have any existing IoT Core devices running on your network, you will see them listed here. We are creating a new IoT Core device, so go ahead and click the "Set up a new device" option on the left menu (Figure 5-3).

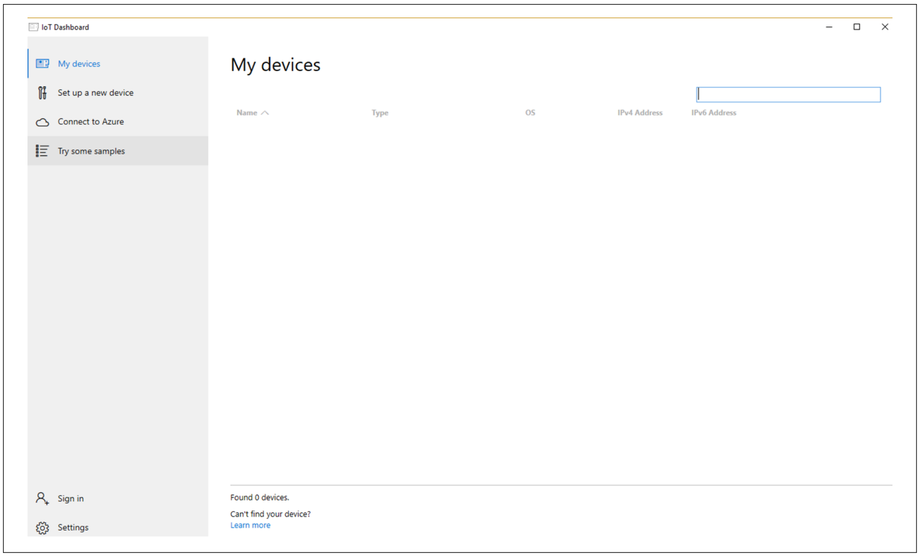
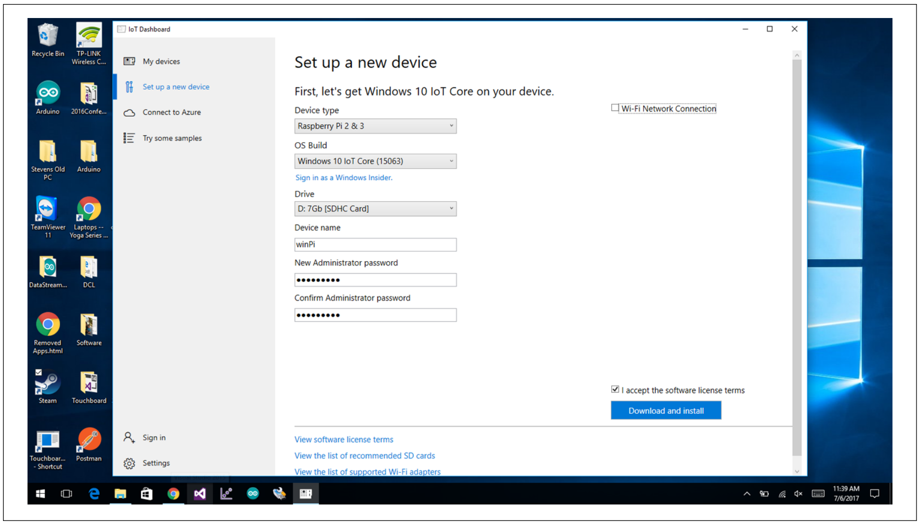


Figure 5-3. IoT Core dashboard

3. Make sure you have a formatted SD card inserted into your workstation (your SD drive letter may be different than the one shown in our screenshots). If you need help formatting the SD card, refer to "Preparing Your Pi" on page 42. We've named our device "winPI," but you can choose any name you prefer. Next, create a password for the administrative user, which you will need later in the process for authenticating and connecting to your device via a browser. You can leave the Wi-Fi Network Connection unchecked; we will configure that at a later point. At this point, you are ready to flash IoT Core to your SD card. Accept the license by checking the box and click the "Download and install" button; progress bars will appear to indicate that the download and flash have commenced (see Figure 5-4).



Set up a new device

First, let's get Windows 10 IoT Core on your device.

Device type: Raspberry Pi 2 & 3

OS Build: Windows 10 IoT Core (15063)

Sign in as a Windows Insider.

Drive: D: 7Gb [SDHC Card]

Device name: winPi

New Administrator password: •••••••••

Confirm Administrator password: •••••••••

☐ Wi-Fi Network Connection

☑ I accept the software license terms

Download and install

View software license terms

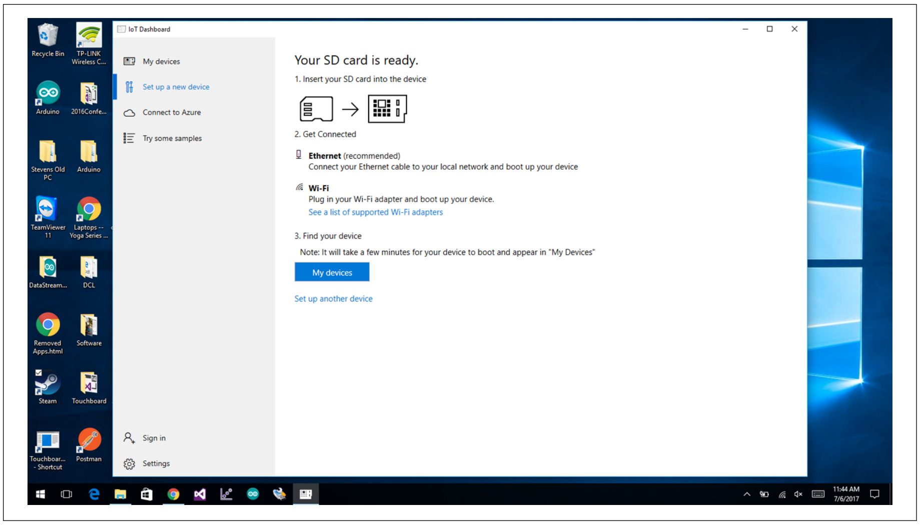
View the list of recommended SD cards

View the list of supported Wi-Fi adapters

11:39 AM 7/6/2017

Figure 5-4. *Setting up a new IoT Core device*

4. Once the SD flashing process is finished, you will be presented with the screen shown in Figure 5-5. Take your newly minted IoT Core SD card, and insert it into your Raspberry Pi (be sure to power down your Raspberry Pi beforehand). Next, connect your HDMI cable and USB keyboard and mouse to your Pi and power the device back up. Finally, click the "My devices" button. Your device might not appear in the list at first, but you should see it once it is powered on and configured.



Your SD card is ready.

1. Insert your SD card into the device

2. Get Connected

Ethernet (recommended)
Connect your Ethernet cable to your local network and boot up your device

Wi-Fi
Plug in your Wi-Fi adapter and boot up your device.
See a list of supported Wi-Fi adapters

3. Find your device

Note: It will take a few minutes for your device to boot and appear in "My Devices"

My devices

Set up another device

11:44 AM 7/6/2017

Figure 5-5. *New device SD card ready screen in IoT dashboard*

5. As your IoT Core Pi boots for the first time, you will see the screen shown in Figure 5-6. It may reboot a few times and this screen may appear for longer than you'd expect. Don't panic; it's just IoT Core running some installation housekeeping and configuration processes during the initial boot.

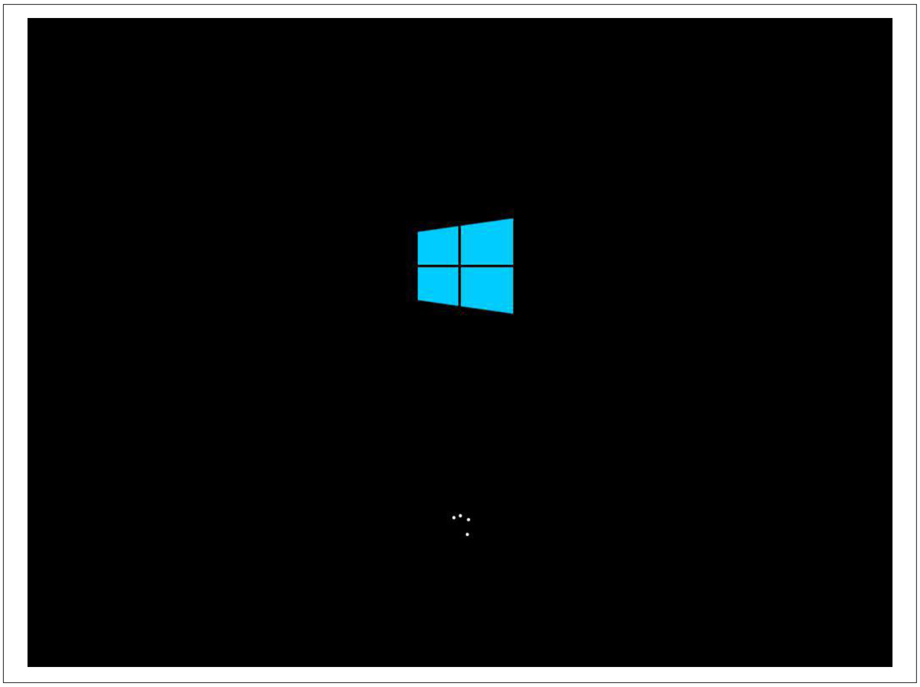
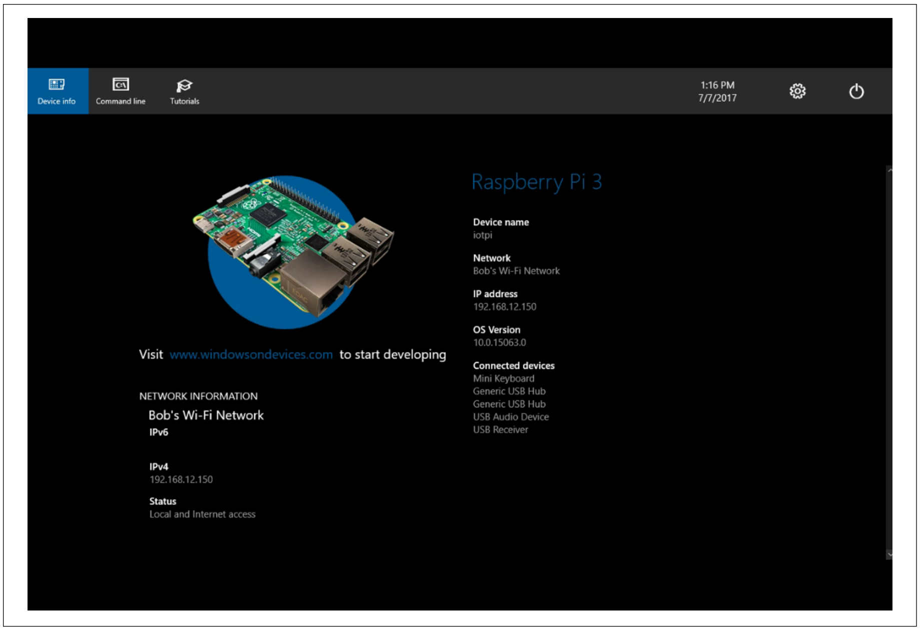


Figure 5-6. IoT Core boot screen

6. Once the initial boot sequence has completed, you will be prompted to select your language and choose a WiFi network. You will then be directed to the main IoT Core dashboard, which should like something like the screen shown in Figure 5-7.



1:16 PM
7/7/2017

Raspberry Pi 3

Device name
iotpi

Network
Bob's Wi-Fi Network

IP address
192.168.12.150

OS Version
10.0.15063.0

Connected devices
Mini Keyboard
Generic USB Hub
Generic USB Hub
USB Audio Device
USB Receiver

Visit www.windowsondevices.com to start developing

NETWORK INFORMATION
Bob's Wi-Fi Network
IPv6

IPv4
192.168.12.150

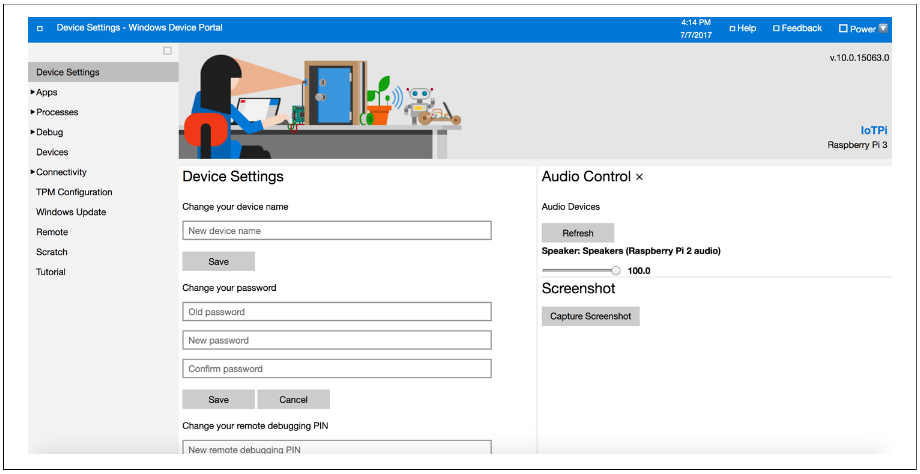
Status
Local and Internet access

Figure 5-7. IoT Core dashboard

7. Congratulations! You've completed the IoT Core setup. Before we move on to coding API.AI and our UWP, we need to update our device's time zone. The following steps also demonstrate how to connect to your IoT Core device via a web browser, and show you where you can control the apps that run on your device.

8. Open up a web browser of your choosing and in the URL box, type **`http://`** ***`YOUR_DEVICE_IP`*****`:8080`** and then hit Enter (make sure to replace *`YOUR_DEVICE_IP`* with the actual IP address of your IoT Core device). You will be prompted to authenticate before you can access the device. Type **`Administrator`** for the user and then enter the password you created for the administrative user in step 3. Once you are authenticated you will be directed to the IoT Core Device Portal for your Pi, shown in Figure 5-8.

You can also access your device via a browser using the IoT Dashboard tool. In the IoT Dashboard tool, select My Devices from the menu, right-click your IoT device, and select Open in Device Portal.



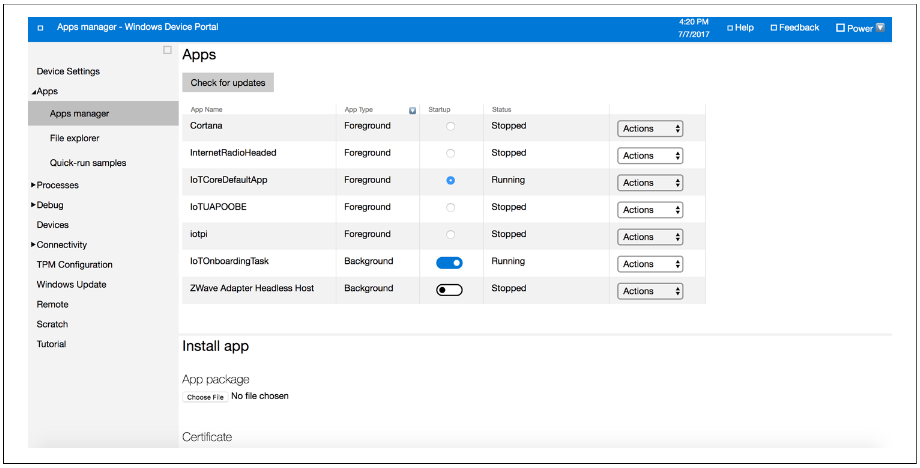
Speaker: Speakers (Raspberry Pi 2 audio)
100.0

Figure 5-8. IoT Core web-based device portal

9. Under the Device Settings section, scroll down until you see the time zone option. Update the setting to your local time zone and click Save. There are many other menu options and settings available to view and update but they are outside the scope of this chapter. IoT Core is a great operating system for IoT, so we encourage you to play around and learn all you can about it.

> To see performance metrics for your device, select the Processes→Performance menu options from the left menu.

10. At this point, we need to familiarize ourselves with one other feature of IoT Core. Using the menu on the left, navigate to Apps→Apps Manager. This brings up a list of all the apps that are installed and/or running on your device (Figure 5-9). From here you can change which apps start at boot; you can restart, start, or stop an app; and you can even uninstall an app using the Uninstall option from the drop-down box. There are actually two ways to install a UWP app onto an IoT Core device. The first is to simply deploy the app from Visual Studio, which is the method we will be using. The second option is to build an installation package in Visual Studio and then upload that package using the tools available under the "Install app" section, just below the Apps list.



Apps

App Name	App Type	Startup	Status	
Cortana	Foreground	○	Stopped	Actions
InternetRadioHeaded	Foreground	○	Stopped	Actions
IoTCoreDefaultApp	Foreground	●	Running	Actions
IoTUAPOOBE	Foreground	○	Stopped	Actions
iotpi	Foreground	○	Stopped	Actions
IoTOnboardingTask	Background	●	Running	Actions
ZWave Adapter Headless Host	Background	○	Stopped	Actions

Install app

App package

No file chosen

Certificate

Figure 5-9. IoT Core Apps list

11. Finally, if you need to power down or restart the device, you can do so through the portal by simply selecting the Power menu option.

That wraps up the installation and configuration of IoT Core on Raspberry Pi. Now we can get back to doing some of the more fun stuff.

Welcome to API.AI

As we mentioned in Chapter 2, API.AI is combination NLP and NLU platform that allows developers to create rich, conversational human–computer interactions. At the core of API.AI are agents. Agents are discrete NLU modules containing a set of intents, slots (entities), and utterances. Agents can be connected to apps such as Facebook Messenger or Twilio to act as a bot. They can also be connected to devices like our IoT Core Raspberry Pi to provide natural conversational experiences via a custom UWP app. API.AI also provides numerous pre-built integrations that make it extremely easy to connect agents to popular external platforms like Twitter, Skype, and Slack. Figure 5-10 illustrates agents' place in the API.AI model.

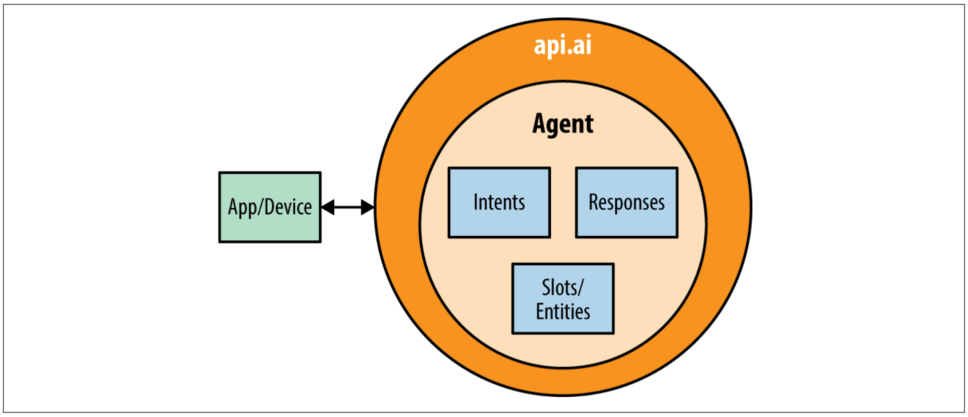


Figure 5-10. API.AI Agent

Figure 5-10 is, of course, a high-level view of the agent's role within API.AI. There are other features available as well, but for our "Hello, World" example we need only focus on intents and responses.

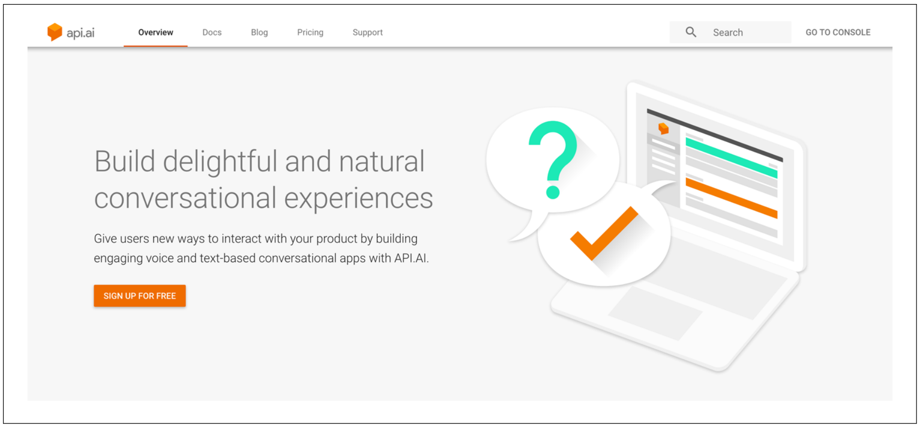
Before we begin creating our agent on API.AI, I'd like to mention one other powerful API.AI feature: Smalltalk. Smalltalk is an out-of-the-box feature that allows you to create responses for commonly encountered phrases (e.g., "Who are you?"; "Are you a person?"; "Thank you"; "You're welcome"; etc.). API.AI organizes common phrases into Smalltalk topics, making it easy for you to create responses to those questions. The folks at API.AI update the Smalltalk topics based on observations from conversational interactions that occur within their platform. This makes Smalltalk a very convenient, time-saving feature that can free you from having to create intents for every mundane question your agents might encounter. We'll include a small example of Smalltalk in our "Hello, World" agent, just because we love it!

Now, let's go build an agent for our UWP app!

Building an API.AI Agent

For our agent build, we will follow a simple step-by-step process (you'll need a Google account to access API.AI, as the company was recently acquired by Google):

1. Open a web browser and visit *http://api.ai*. You should land on the page shown in Figure 5-11. Click GO TO CONSOLE at the upper-right section of the window.

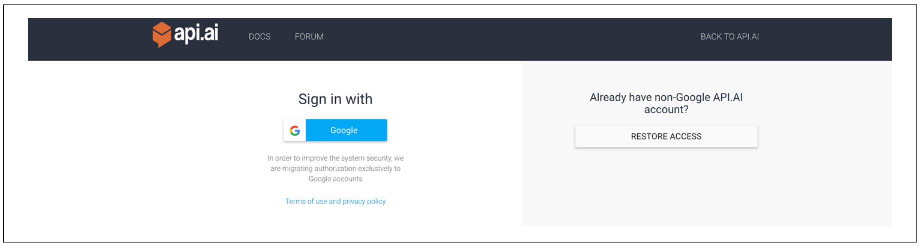


Build delightful and natural conversational experiences

Give users new ways to interact with your product by building engaging voice and text-based conversational apps with API.AI.

Figure 5-11. API.AI landing page

You will then be directed to the login page, shown in Figure 5-12.



In order to improve the system security, we are migrating authorization exclusively to Google accounts

Already have non-Google API.AI account?

Figure 5-12. API.AI login

2. Go ahead and log in with your Google account credentials. You should then be redirected to the page shown in Figure 5-13. From there, click the CREATE AGENT button.

> If you are prompted to authorize API.AI due to insufficient permissions, simply grant the required access to continue the process.

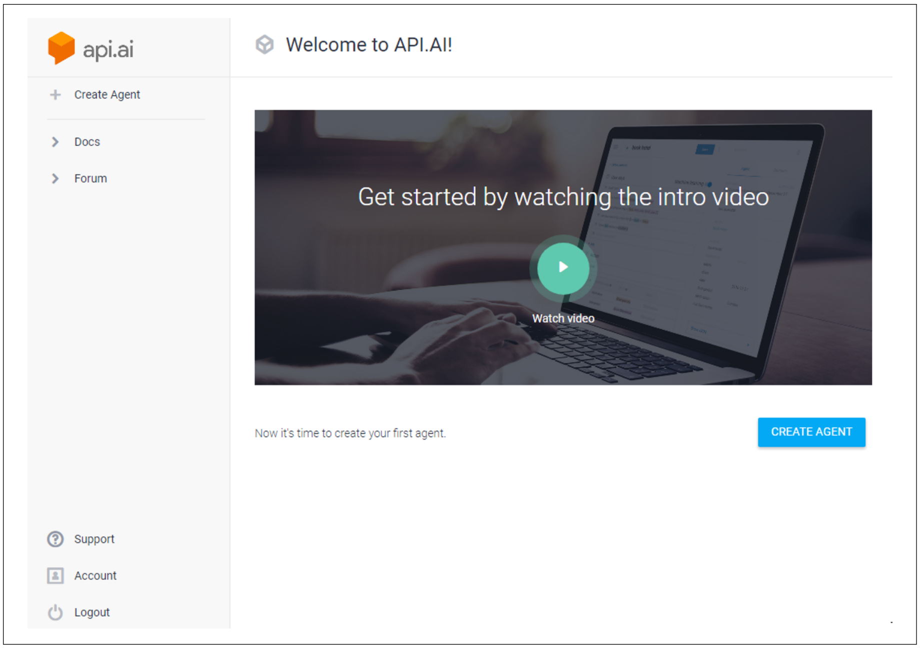
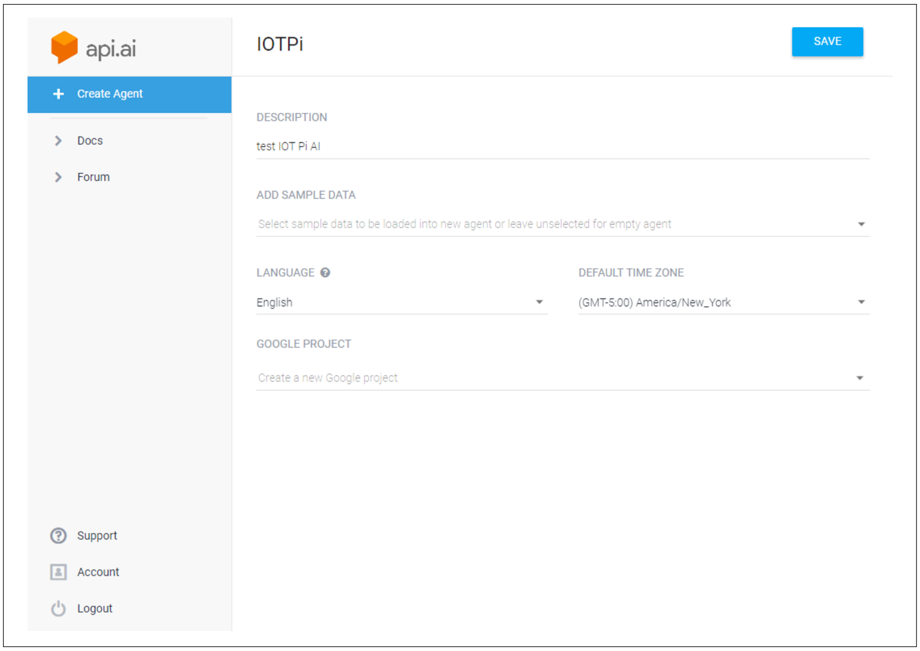


Figure 5-13. API.AI landing page

3. You should now be presented with the Create Agent form (Figure 5-14). Give your agent a name and a brief description, and set your default time zone. We don't need to provide sample data or create a Google project for our "Hello, World" example, so you can simply click Save to initialize our agent.



IOTPi

DESCRIPTION
test IOT Pi AI

ADD SAMPLE DATA
Select sample data to be loaded into new agent or leave unselected for empty agent

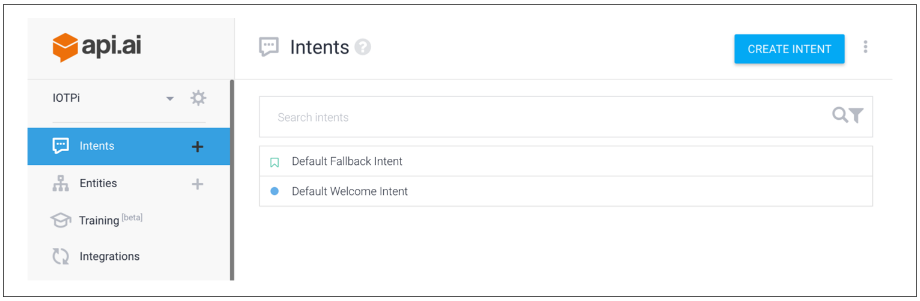
LANGUAGE
English

DEFAULT TIME ZONE
(GMT-5:00) America/New_York

GOOGLE PROJECT
Create a new Google project

Figure 5-14. API.AI's Create Agent form

4. Once your agent is successfully created, you will be directed to the page shown in Figure 5-15. This is where we can create the intents and entities for our agent. Let's build our `HelloWorldIntent` now. Click CREATE INTENT.

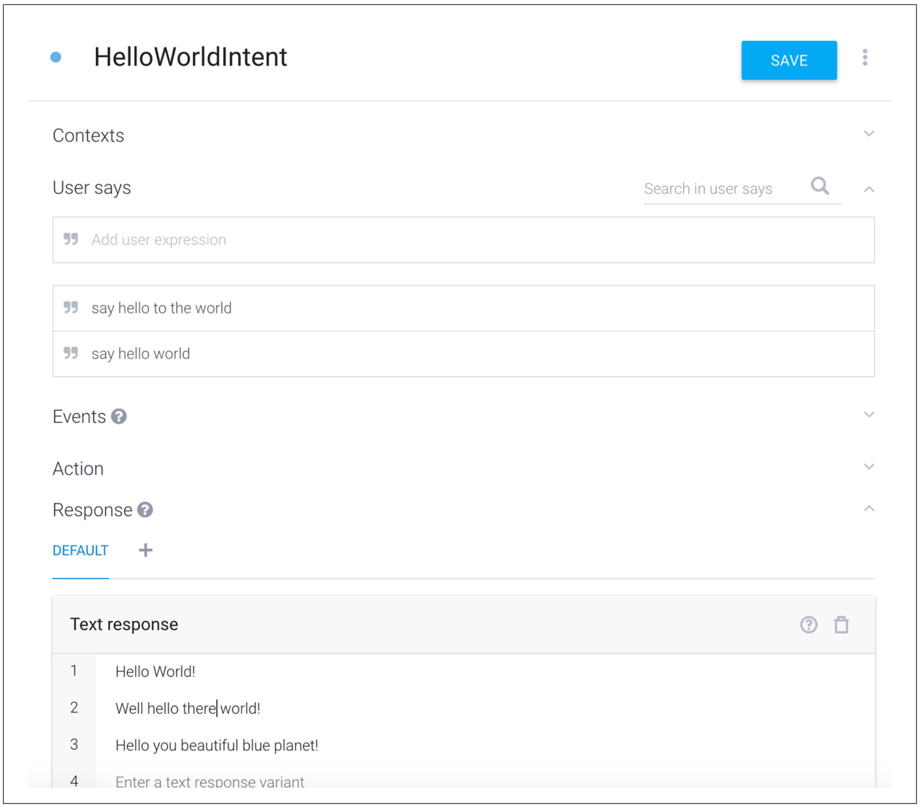


Default Fallback Intent
Default Welcome Intent

Figure 5-15. API.AI Agent console

5. There are a lot of options on the "Create Intent" form (Figure 5-16), but we only need to focus on three tasks: naming our intent, defining "User says" utterances, and building a list of responses. First things first, let's name our intent `HelloWorldIntent`. Next, enter the utterances we expect our users to say for our "Hello, World" example. For example, "Say 'hello' to the world" or "Say 'Hello, World'". As you can see, this is virtually identical to how we create utterances for Alexa in the Amazon Developer Portal. Next, let's create a list of possible responses that

can be returned whenever our `HelloWorldIntent` is triggered. This is one of the more powerful features of API.AI. As you'll recall from Chapter 4, we had to create a lambda function to handle and build the responses for our Alexa skill. All of our skill logic was managed in code. However, with API.AI, we now have a code-free way to not only return a response but also to randomize the phrases returned. This makes our interactions more interesting and conversational, all without writing one line of code. As already mentioned, there are many other items available on the Create Intent form, but we won't cover these here; however, we encourage you to experiment with each of the available options. For now though, just go ahead and save the work we did for the `HelloWorldIntent`.


HelloWorldIntent

Contexts

User says

Add user expression

say hello to the world

say hello world

Events

Action

Response

DEFAULT

Text response

1 Hello World!

2 Well hello there world!

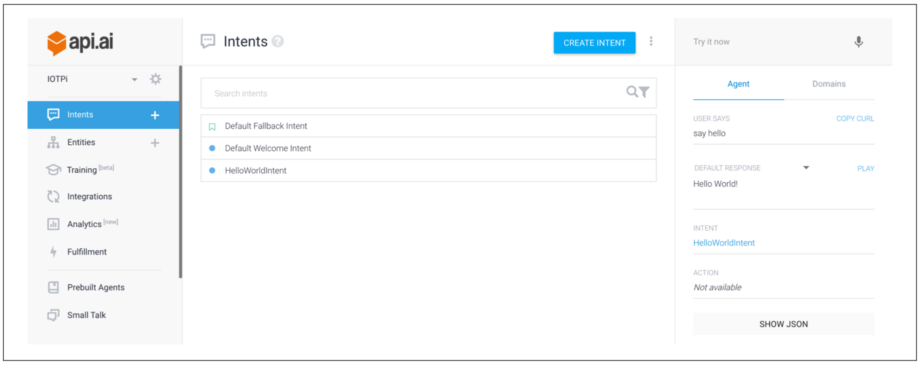
3 Hello you beautiful blue planet!

4 Enter a text response variant

Figure 5-16. New Intent form

6. After saving your `HelloWorldIntent`, you will be brought back to the main agent console where you should now see your `HelloWorldIntent` listed (Figure 5-17). Before we begin building our UWP app, let's take our agent for a test run. Type **`Say 'Hello'`** in the "Try it now" box and hit Enter. This will trigger your agent. You should then see one of your `HelloWorldIntent` responses listed under Default Response. You should also see the name of the intent your agent trig-

gered based on the Say 'Hello' utterance. If everything worked as expected, congratulations! You successfully created a custom API.AI agent. A quick note on the two default intents, which are included in every agent. The Default Fallback Intent is triggered whenever your agent can't find an intent for a given user utterance. API.AI provides a nice list of canned responses as part of the "fallback" intent, but you can modify the list as needed. The Default Welcome Intent is triggered by Welcome Events, typically fired by social media platform bots, although it can also be triggered via the API.AI API. Our "Hello, World" example won't utilize the Welcome Intent, but we wanted to explain its usage.



Default Fallback Intent
Default Welcome Intent
HelloWorldIntent

USER SAYS
say hello

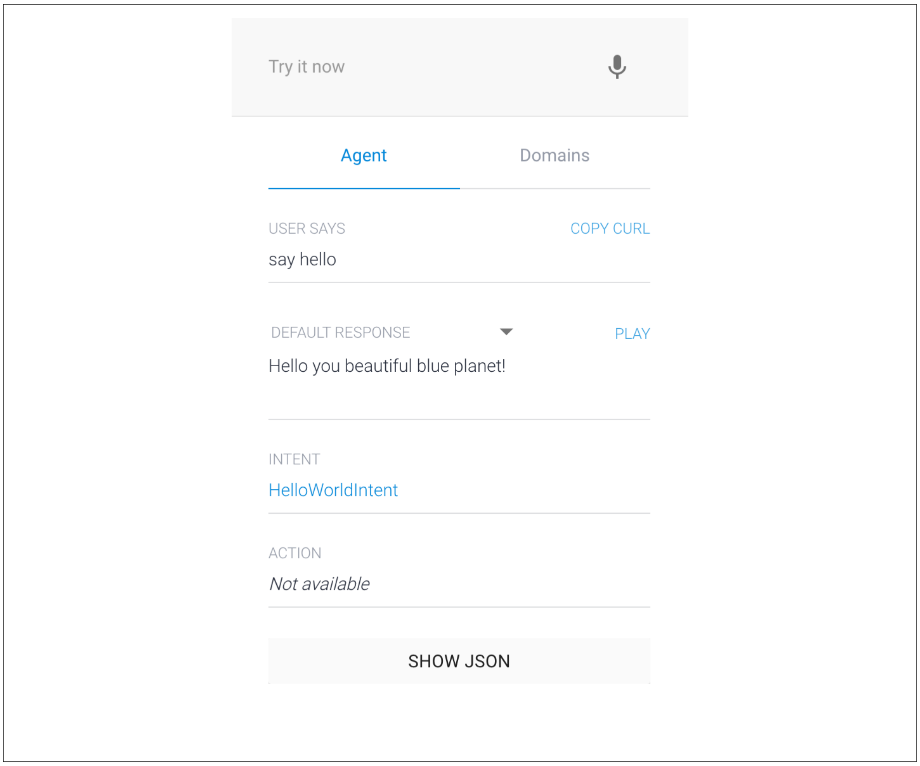
DEFAULT RESPONSE
Hello World!

INTENT
HelloWorldIntent

ACTION
Not available

Figure 5-17. Agent test

7. There are just a few more things we need to do in preparation for our UWP app integration. First, click the COPY CURL link (Figure 5-18). This will provide a sample HTTPS call we can use to connect our UWP app to our API.AI agent.


Try it now

Agent

Domains

USER SAYS

COPY CURL

say hello

DEFAULT RESPONSE

PLAY

Hello you beautiful blue planet!

INTENT

HelloWorldIntent

ACTION

Not available

SHOW JSON

Figure 5-18. Test output

The copied `curl` statement looks similar to this:

```
curl 'https://api.api.ai/api/query?v
=20150910&query=say%20hello&lang=
en&sessionId=4f1b46ac-1ef7-
43ad-a502-3310d6e1f094&timezone=2017-07-08T12:51:48-0400'
-H 'Authorization:Bearer
   092e4a9c2334fed9e4f50e7ebf56d2d'
```

We've highlighted the important bits. API.AI requires certain parameters for query calls. Specifically, we will need to provide the following query string parameters:

`Query`
: The actual utterance spoken by our user

`SessionId`
: A random string (we will use a GUID)

`Lang`
: The language of the response (because we are working with one language in our example, we'll pass "en" for English)

`v`
: The version of API.AI API we wish to use

Time zone is not required, and while Bearer token (Client Access Token) is mandatory for each request, it will be passed as part of the HTTP request header. We've reformatted the request to give us a base URL, which we will use in our UWP app (take note of this URL, you will need it later in the chapter):

```
//our modified base URL for API.AI
https://api.ai.ai/api/query?v=20150910&lang=en
```

You will need your Client Access Token for our UWP app. To retrieve it, simply click the gear icon next to your agent name at the top-left of the main agent console (see Figure 5-19). Your Client Access Token will be listed under the API Keys section.

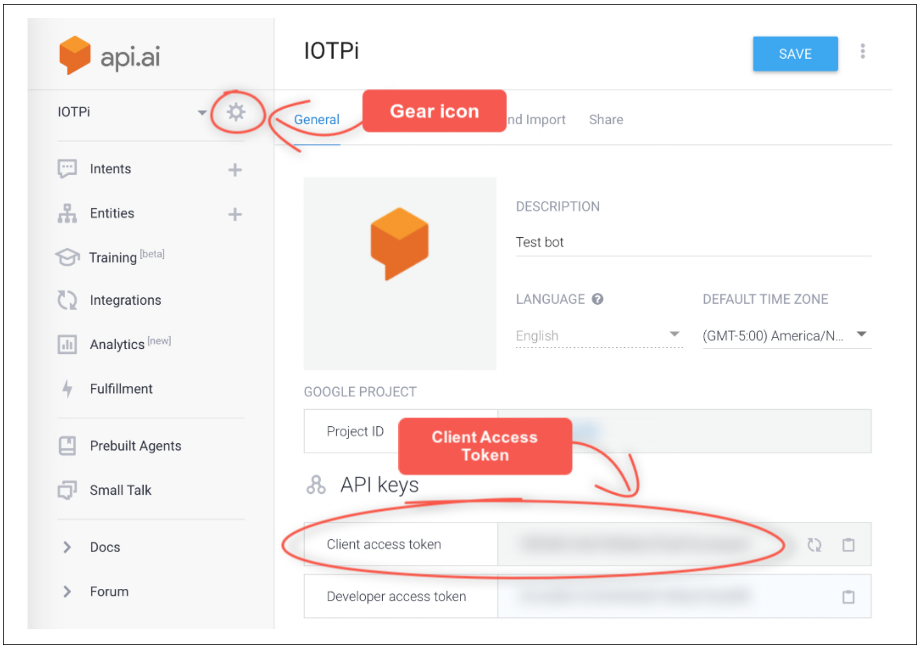
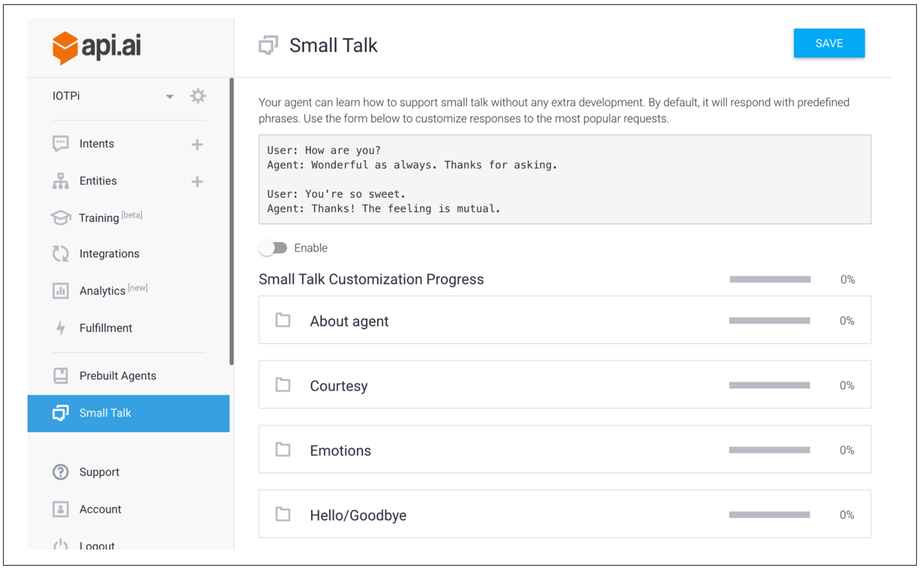


Figure 5-19. Click on the gear icon to retrieve your access token

8. The second item we need to cover is the response object returned from API.AI. If you click the SHOW JSON button shown in Figure 5-18 it will reveal the response JSON returned from our `HelloWorldIntent` (Figure 5-20). The element

we are most interested in is contained within the `fulfillment` object. The `speech` element contains one of the response phrases we entered in the `HelloWorldIntent`. This is the value our UWP app will need to parse from the JSON object. (There is also a speech element within the messages array. This is a text value supporting line breaks to support Facebook Messenger, Kik, Slack, and other messaging services. You can certainly use this value, but it's not required.) You don't need to take any action at the moment; this image is provided simply as a reference to the API.AI response object. Full details regarding the response object can be found at *https://api.ai/docs/reference/agent/query#response*.



Your agent can learn how to support small talk without any extra development. By default, it will respond with predefined phrases. Use the form below to customize responses to the most popular requests.

```
User: How are you?
Agent: Wonderful as always. Thanks for asking.

User: You're so sweet.
Agent: Thanks! The feeling is mutual.
```

Enable

Small Talk Customization Progress 0%

About agent 0%

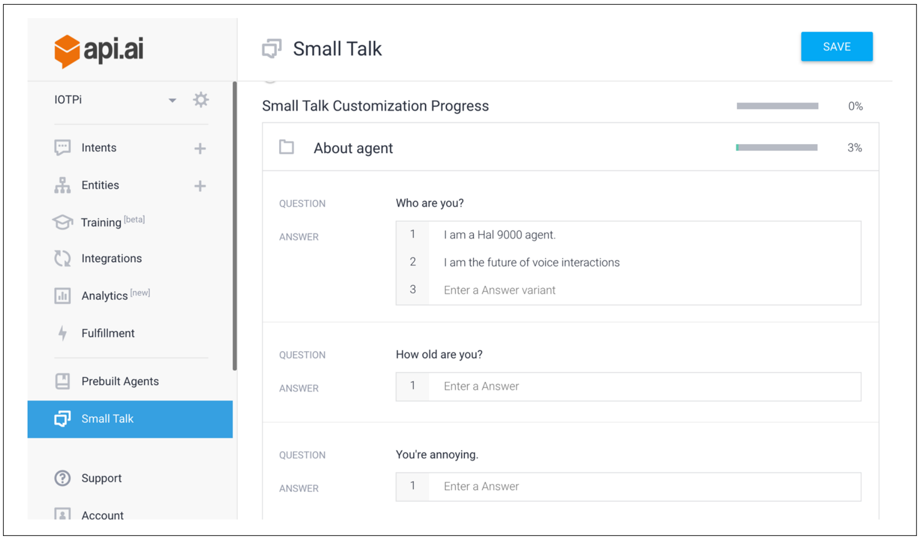
Courtesy 0%

Emotions 0%

Hello/Goodbye 0%

Figure 5-20. API.AI HelloWorldIntent response JSON

9. At this point, we've completed all of the API.AI requirements for our "Hello, World" example, but we want to quickly show you how easy it is to add Small Talk to your agent. Select Small Talk from the left menu and you will be presented with the screen shown in Figure 5-21. As you can see, API.AI provides a fairly comprehensive list of Smalltalk topics to choose from.



Small Talk Customization Progress 0%

About agent 3%

QUESTION Who are you?

ANSWER
1 I am a Hal 9000 agent.
2 I am the future of voice interactions
3 Enter a Answer variant

QUESTION How old are you?

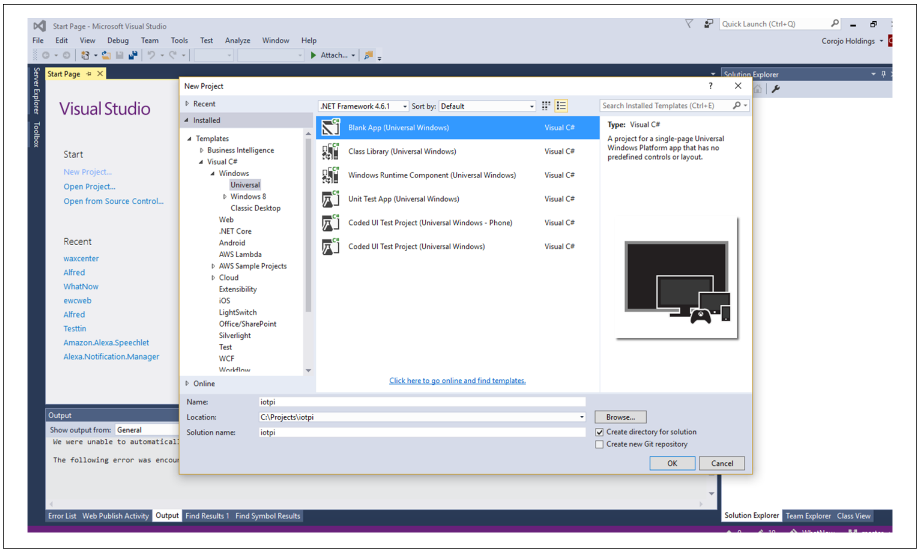
ANSWER
1 Enter a Answer

QUESTION You're annoying.

ANSWER
1 Enter a Answer

Figure 5-21. API.AI Small Talk dashboard

10. Click the "About agent" topic and under "Who are you?" (Figure 5-22), and enter a list of random response phrases you want your agent to return for that question. When you are done, click Save. That's it! Now your agent will respond with one of your provided phrases whenever it receives the "Who are you?" question. Obviously this greatly simplifies our agent's ability to handle commonly encountered phrases. Pretty powerful stuff!


Start Page - Microsoft Visual Studio
File Edit View Debug Team Tools Test Analyze Window Help
Visual Studio
Start
New Project...
Open Project...
Open from Source Control...
Recent
waxcenter
Alfred
WhatNow
ewcweb
Alfred
Testtin
Amazon.Alexa.Speechlet
Alexa.Notification.Manager
New Project
Recent
Installed
Templates
Business Intelligence
Visual C#
Windows
Universal
Windows 8
Classic Desktop
Web
.NET Core
Android
AWS Lambda
AWS Sample Projects
Cloud
Extensibility
iOS
LightSwitch
Office/SharePoint
Silverlight
Test
WCF
Workflow
Online
Blank App (Universal Windows) Visual C#
Class Library (Universal Windows) Visual C#
Windows Runtime Component (Universal Windows) Visual C#
Unit Test App (Universal Windows) Visual C#
Coded UI Test Project (Universal Windows - Phone) Visual C#
Coded UI Test Project (Universal Windows) Visual C#
Click here to go online and find templates.
Browse...
OK
Cancel
Solution Explorer Team Explorer Class View
Error List Web Publish Activity Output Find Results 1 Find Symbol Results

.NET Framework 4.6.1 Sort by: Default

Type: Visual C#

A project for a single-page Universal Windows Platform app that has no predefined controls or layout.

Name: iotpi
Location: C:\Projects\iotpi
Solution name: iotpi

[✓] Create directory for solution
[ ] Create new Git repository

Figure 5-22. API.AI Small Talk "About agent" configuration

OK, that's a wrap on our API.AI effort. We were able to create a rich set of intents and responses all without writing one line of code. We were even able to add a bit of Small Talk handing to the mix. As mentioned earlier, API.AI is a very capable NLU/NLP engine, and we highly recommend it for any voice-based project. It's especially handy for building codeless bots using API.AI's pre-built integrations. Experiment with API.AI—you won't be sorry you did.

Now, let's go build our UWP app and tie it all together.

Building Our UWP App

We're now ready to build our custom UWP app. This section assumes you are familiar with developing apps in Visual Studio (VS) and have some experience with C#. In addition, you will need Visual Studio 2015 or above (Community Edition is fine).

The Plan

The overall plan is to have fun creating a voice-activated UWP app. To help us achieve this goal, our app will leverage Microsoft's `Windows.Media` namespace. Specifically, the `Windows.Media.SpeechSynthesis` and `Windows.Media.SpeechRecognition` classes.

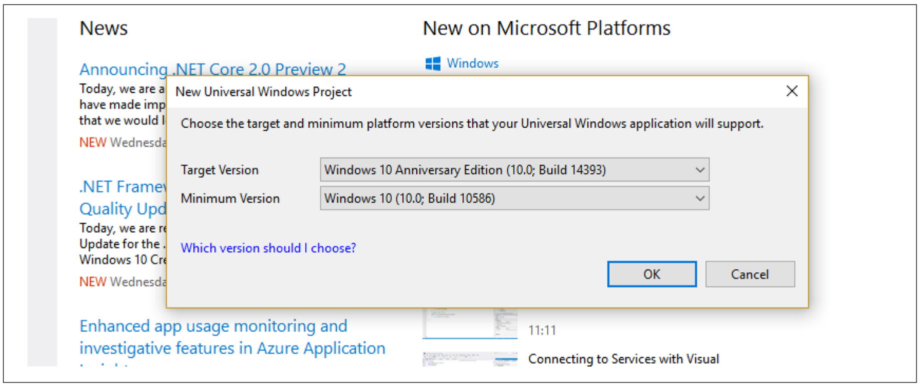
The `SpeechSynthesis` class will be used as our audio playback engine, playing the responses we receive from API.AI. The `SpeechRecognition` class will be used to listen and capture our user's voice requests (utterances). We will then forward the utterances to API.AI for processing and parse and play the received responses.

Let's get started!

The Code

This section breaks the build into simple steps and explains the critical elements of the code. For convenience sake, we've made the code for this project available on GitHub. Clone this project locally. We will demonstrate how to create and configure a new UWP project, but you can leverage our existing code to get you up and running quickly. Here are the steps you'll need to follow:

1. Go ahead and open VS and create a new project. Select Blank App (Universal Windows). Name your project and solution and click OK (Figure 5-23).


News
New on Microsoft Platforms
Windows
Announcing .NET Core 2.0 Preview 2
Enhanced app usage monitoring and investigative features in Azure Application
11:11
Connecting to Services with Visual
New Universal Windows Project
OK
Cancel

Choose the target and minimum platform versions that your Universal Windows application will support.

Target Version: Windows 10 Anniversary Edition (10.0; Build 14393)
Minimum Version: Windows 10 (10.0; Build 10586)

Which version should I choose?

Figure 5-23. New UWP app

2. When creating a UWP, you will be prompted to select a target and minimum version of Windows 10. You can leave all the defaults and select OK (Figure 5-24).

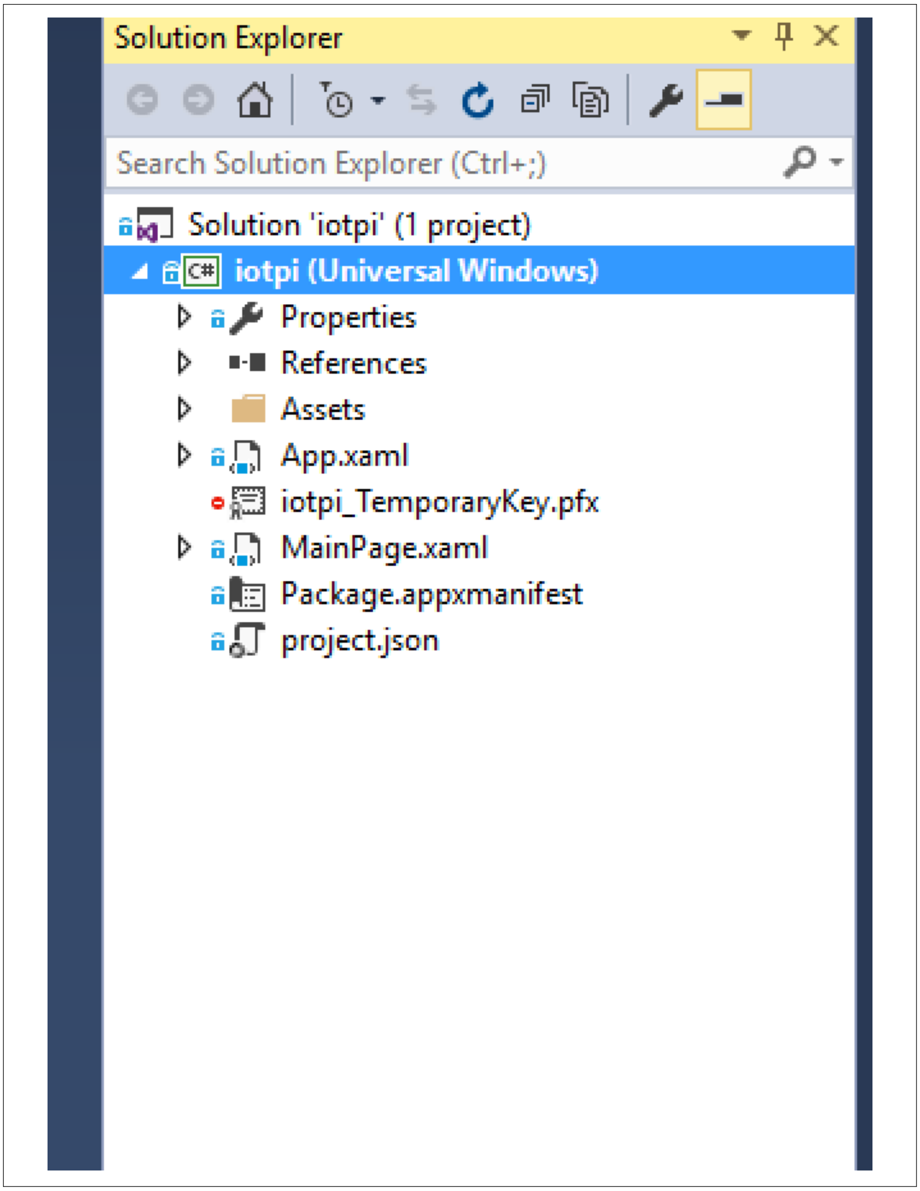
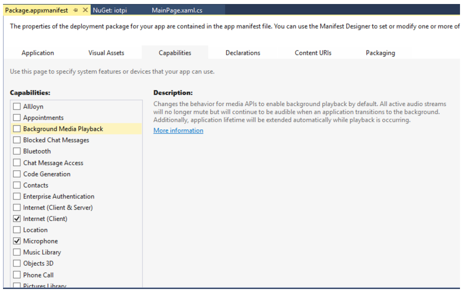


Figure 5-24. UWP targets

3. Your project should look similar to Figure 5-25. We need to do a little configuration housekeeping so double-click the *Package.appxmanifest* file.



Package.appxmanifest | NuGet: iotpi | MainPage.xaml.cs

The properties of the deployment package for your app are contained in the app manifest file. You can use the Manifest Designer to set or modify one or more of

Application | Visual Assets | Capabilities | Declarations | Content URIs | Packaging

Use this page to specify system features or devices that your app can use.

Capabilities:

- [ ] AllJoyn
- [ ] Appointments
- [ ] Background Media Playback
- [ ] Blocked Chat Messages
- [ ] Bluetooth
- [ ] Chat Message Access
- [ ] Code Generation
- [ ] Contacts
- [ ] Enterprise Authentication
- [ ] Internet (Client & Server)
- [✓] Internet (Client)
- [ ] Location
- [✓] Microphone
- [ ] Music Library
- [ ] Objects 3D
- [ ] Phone Call
- [ ] Pictures Library

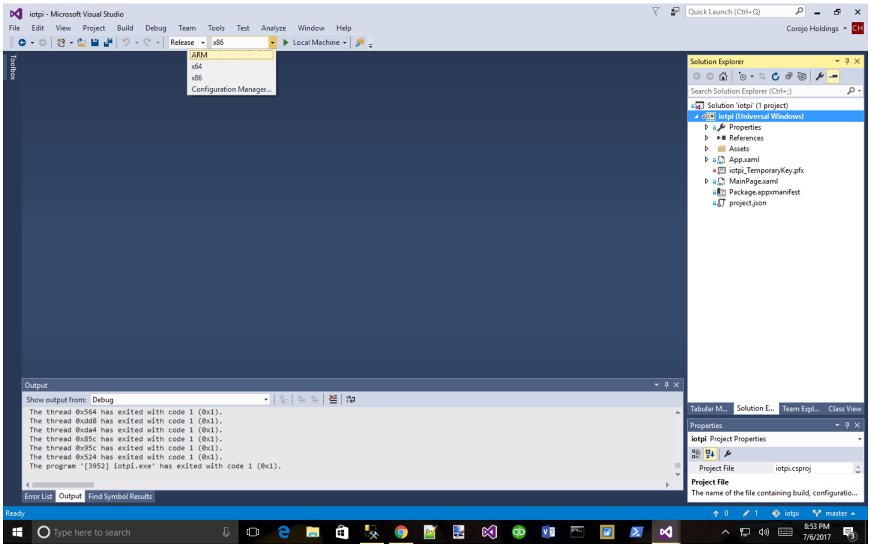
Description:

Changes the behavior for media APIs to enable background playback by default. All active audio streams will no longer mute but will continue to be audible when an application transitions to the background. Additionally, application lifetime will be extended automatically while playback is occurring.

More information

Figure 5-25. A typical UWP project structure in Visual Studio

4. Select the Capabilities tab and then ensure that both Internet (Client) and Microphone are selected under the Capabilities list (Figure 5-26). This grants our app access to the microphone hardware.

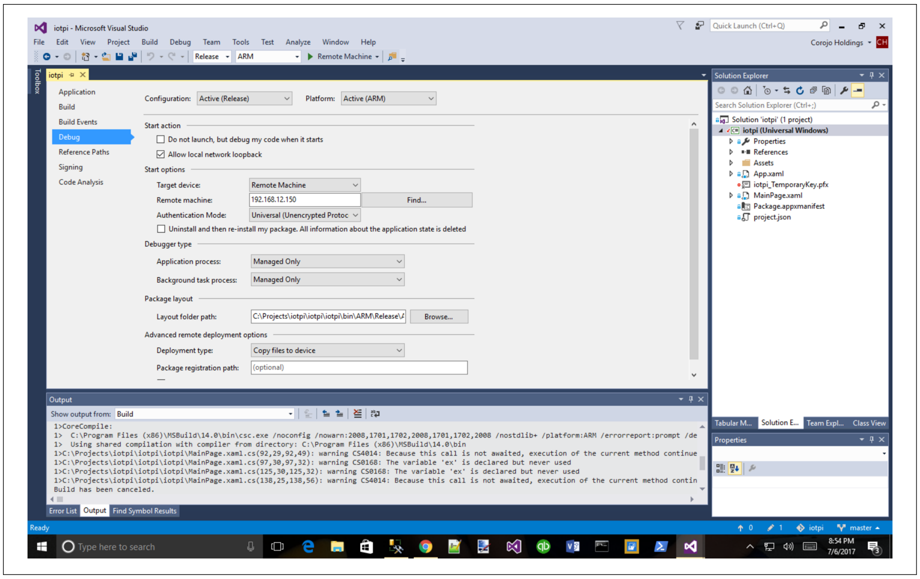


iotpi - Microsoft Visual Studio
File Edit View Project Build Debug Team Tools Test Analyze Window Help
Release
x86
ARM
x64
x86
Configuration Manager...
Local Machine
Quick Launch (Ctrl+Q)
Corojo Holdings
Solution Explorer
Search Solution Explorer (Ctrl+;)
Solution 'iotpi' (1 project)
iotpi (Universal Windows)
Properties
References
Assets
App.xaml
iotpi_TemporaryKey.pfx
MainPage.xaml
Package.appxmanifest
project.json
Tabular M... Solution E... Team Expl... Class View
Properties
iotpi Project Properties
Project File iotpi.csproj
Project File
The name of the file containing build, configuratio...
Output
Show output from: Debug
Error List Output Find Symbol Results
Ready
master
Type here to search
8:53 PM
7/6/2017

```
The thread 0x564 has exited with code 1 (0x1).
The thread 0xdd8 has exited with code 1 (0x1).
The thread 0xda4 has exited with code 1 (0x1).
The thread 0x85c has exited with code 1 (0x1).
The thread 0x95c has exited with code 1 (0x1).
The thread 0x524 has exited with code 1 (0x1).
The program '[3952] iotpi.exe' has exited with code 1 (0x1).
```

Figure 5-26. UWP package manifest settings

5. Now we need to set the Target platform from x86 to ARM (the Pi is an ARM-based processor). On the main toolbar, click the drop-down box adjacent to the build mode (debug/release) and select ARM. Next, click the arrow next to Local Machine and select Remote Machine (this will be our Pi), as shown in Figure 5-27.



Configuration: Active (Release) Platform: Active (ARM)

Start action
- [ ] Do not launch, but debug my code when it starts
- [x] Allow local network loopback

Start options
- Target device: Remote Machine
- Remote machine: 192.168.12.150
- Authentication Mode: Universal (Unencrypted Protoc
- [ ] Uninstall and then re-install my package. All information about the application state is deleted

Debugger type
- Application process: Managed Only
- Background task process: Managed Only

Package layout
- Layout folder path: C:\Projects\iotpi\iotpi\iotpi\bin\ARM\Release\A

Advanced remote deployment options
- Deployment type: Copy files to device
- Package registration path: (optional)

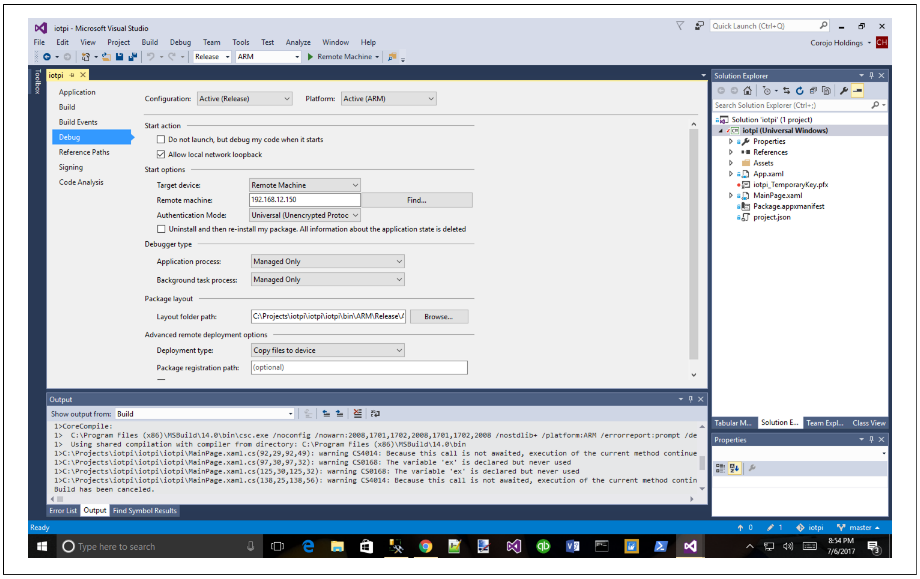
Output — Show output from: Build

```
1>CoreCompile:
1>  C:\Program Files (x86)\MSBuild\14.0\bin\csc.exe /noconfig /nowarn:2008,1701,1702,2008,1701,1702,2008 /nostdlib+ /platform:ARM /errorreport:prompt /de
1>  Using shared compilation with compiler from directory: C:\Program Files (x86)\MSBuild\14.0\bin
1>C:\Projects\iotpi\iotpi\iotpi\MainPage.xaml.cs(92,29,92,49): warning CS4014: Because this call is not awaited, execution of the current method continue
1>C:\Projects\iotpi\iotpi\iotpi\MainPage.xaml.cs(97,30,97,32): warning CS0168: The variable 'ex' is declared but never used
1>C:\Projects\iotpi\iotpi\iotpi\MainPage.xaml.cs(125,30,125,32): warning CS0168: The variable 'ex' is declared but never used
1>C:\Projects\iotpi\iotpi\iotpi\MainPage.xaml.cs(138,25,138,56): warning CS4014: Because this call is not awaited, execution of the current method contin
Build has been canceled.
```

Figure 5-27. Set platform to ARM and select Remote Machine

6. Upon selecting Remote Machine, you should be prompted to enter a Target Device, Remote Machine, and Authentication Mode. Ensure the settings are configured as indicated in Figure 5-28. Be sure to enter you Pi's IP address for the Remote Machine setting.

If you are not immediately prompted with a pop-up box when selecting Remote Machine, you can make the adjustments by navigating to Debug→Properties and then selecting the Debug menu option (see Figure 5-28).



Configuration: Active (Release) Platform: Active (ARM)

Start action
- [ ] Do not launch, but debug my code when it starts
- [x] Allow local network loopback

Start options
- Target device: Remote Machine
- Remote machine: 192.168.12.150
- Authentication Mode: Universal (Unencrypted Protoc
- [ ] Uninstall and then re-install my package. All information about the application state is deleted

Debugger type
- Application process: Managed Only
- Background task process: Managed Only

Package layout
- Layout folder path: C:\Projects\iotpi\iotpi\iotpi\bin\ARM\Release\A

Advanced remote deployment options
- Deployment type: Copy files to device
- Package registration path: (optional)

Output — Show output from: Build

```
1>CoreCompile:
1>  C:\Program Files (x86)\MSBuild\14.0\bin\csc.exe /noconfig /nowarn:2008,1701,1702,2008,1701,1702,2008 /nostdlib+ /platform:ARM /errorreport:prompt /de
1>  Using shared compilation with compiler from directory: C:\Program Files (x86)\MSBuild\14.0\bin
1>C:\Projects\iotpi\iotpi\iotpi\MainPage.xaml.cs(92,29,92,49): warning CS4014: Because this call is not awaited, execution of the current method continue
1>C:\Projects\iotpi\iotpi\iotpi\MainPage.xaml.cs(97,30,97,32): warning CS0168: The variable 'ex' is declared but never used
1>C:\Projects\iotpi\iotpi\iotpi\MainPage.xaml.cs(125,30,125,32): warning CS0168: The variable 'ex' is declared but never used
1>C:\Projects\iotpi\iotpi\iotpi\MainPage.xaml.cs(138,25,138,56): warning CS4014: Because this call is not awaited, execution of the current method contin
Build has been canceled.
```

Figure 5-28. Set remote machine to our IoT Core Pi IP address

7. Now that our app is configured correctly, let's set it aside for a moment and open the solution you cloned from GitHub. This will give us an opportunity to review the code in detail. When we finish the remaining steps, you can either copy and paste our code into your new project, or simply run our cloned project.

If you choose to copy and paste our code into your project, remember to replace the Client Access Token (APIKEY) to match your API.AI Token. If you choose to simply run our cloned project, you will need to update the Client Access Token as well the Remote Machine IP Address.

8. Once the cloned solution is open, expand the *MainPage.xaml* item and then double-click the *MainPage.xaml.cs* file to reveal the source. We'll start at the very top. First, we initialize an instance variable `SpeechRecognizer` object named `rec ognizer`, which will be used to start a continuous recognition session. The continuous session will listen for an invocation word (similar to how Amazon devices listen for "Alexa/Amazon/computer"). We also have a variable to hold the base URL to API.AI (we discussed this in step 8 of creating our API.AI agent). Next, we store our API.AI Client Access Token in a variable named APIKEY, which will be passed to API.AI in the request header. Finally, we created an invocation word and prompt. Our continuous session will listen for the word "Hal," at which time we will respond with "Yes?", letting the user know we are listening and awaiting her next utterance:

```
public sealed partial class MainPage : Page
{
    // Speech Recognizer
    private SpeechRecognizer recognizer;

    private HttpClient client = new HttpClient();
    private readonly string baseURL = "https://api.api.ai/api/query?v=
      20150910&lang=en";
    private readonly string APIKEY = "092e4a9c21d34bed9e4f50e7ebf56d2d";

    private readonly string INVOCATION_NAME = "hal";
    private readonly string INVOCATION_PROMPT = "yes?";

    public MainPage()
    {
        this.InitializeComponent();
    }
    .......
```

9. The following code illustrates what occurs when our app is navigated to (similar to "loaded" in a traditional Windows app). Here, we create the Authorization Header to the Client Access Token so it is passed on each call to API.AI. Next, we call `InitializeSpeechRecognizer` to initialize the continuous recognition session:

```
async protected override void OnNavigatedTo(NavigationEventArgs e)
{
    base.OnNavigatedTo(e);

    client.DefaultRequestHeaders.Authorization =
      new AuthenticationHeaderValue("Bearer", APIKEY);
    await InitializeSpeechRecognizer();
}
        ....
```

10. Let's break down the `InitializeSpeechRecognizer` method. After we create a new `SpeechRecognizer` object, we attach a delegate handler to the `ContinuousRecgonitionSession`'s `ResultGenerated` event. Next, we create a constraint for the session to only fire when certain words or phrases are received. In our case, we only want the event to fire by our invocation word "Hal." Finally, we compile our constraints, and if successful, start the session:

```
async protected override void OnNavigatedTo(NavigationEventArgs e)
{
    base.OnNavigatedTo(e);

    client.DefaultRequestHeaders.Authorization =
      new AuthenticationHeaderValue("Bearer", APIKEY);
    await InitializeSpeechRecognizer();
}
.....
```

11. Taking a quick look at the `RecognizerResultGenerated` event delegate we see that if the spoken word isn't empty, we simply pass the value to a custom handler called `ProcessContinuousVoiceResult`:

    ```
    // Recognizer generated results
    async private void RecognizerResultGenerated(
      SpeechContinuousRecognitionSession session,
      SpeechContinuousRecognitionResultGeneratedEventArgs args)
    {
        try
        {
            if (!string.IsNullOrEmpty(args.Result.Text))
            {
                await ProcessContinousVoiceResult(args.Result.Text);
            }
        }
        catch (Exception ex)
        {
            //log
        }
    }
    .....
    ```

12. Within `ProcessContinuousVoiceResult` we double-check that we received the invocation name, and if so, we respond with "Yes?", letting our user know we are ready for her next phrase. Next, we stop the continuous speech recognizer session (temporarily) and call `InitialIntentRecognizer` to spin-up a second "burst" listening session:

    ```
    private async Task ProcessContinousVoiceResult(string target)
    {
        try
        {
            if (target.ToLower() == INVOCATION_NAME)
            {
                await this.Dispatcher.RunAsync(
                  Windows.UI.Core.CoreDispatcherPriority.Normal,
                    () =>
                {
                    PlayResponse(INVOCATION_PROMPT);
                });
                await recognizer.ContinuousRecognitionSession.StopAsync();
                await InitializeIntentRecognizer();

            }
        }
        catch (Exception ex)
    ```

```
        {
            //log
        }
    }
    ....
```

13. The `InitialIntentRecognizer` is where most of the magic happens. The first part of the method creates a new `SpeechRecognizer` object, compiles any constraints (in our case there are none), and then sets up our timeout values. The timeout settings inform the session to wait up to 10 seconds for speech (and ending the session) and to wait 5 seconds after speech is heard. Next, we start the recognition session by calling `RecognizeAysnc()`. When the session has completed, we grab the utterance from the session result, build our query string, and pass the data to API.AI. Once the response is received from API.AI, we parse the data and pass the response speech to the `PlayResponse` method:

```
    private async Task InitializeIntentRecognizer()
    {
        string spokenWord = string.Empty;

        try
        {
            // Initialize recognizer
            using (var intentRecognizer = new SpeechRecognizer())
            {
                var compilationResult =
                  await intentRecognizer.CompileConstraintsAsync();
                // If successful, display the recognition result.
                if (compilationResult.Status ==
                  SpeechRecognitionResultStatus.Success)
                {
                    // change default of 5 seconds
                    intentRecognizer.Timeouts.InitialSilenceTimeout =
                      TimeSpan.FromSeconds(10);
                    // change default of 0.5 seconds
                    intentRecognizer.Timeouts.EndSilenceTimeout =
                      TimeSpan.FromSeconds(5);
                    SpeechRecognitionResult result =
                      await intentRecognizer.RecognizeAsync();
                    if (result.Status ==
                      SpeechRecognitionResultStatus.Success)
                    {
                        spokenWord = result.Text;
                    }
                }
            }

            if (!string.IsNullOrEmpty(spokenWord))
            {
                if (!string.IsNullOrEmpty(spokenWord))
                {
                    var result = await client.GetStringAsync(baseURL +
                      "&sessionId=" + Guid.NewGuid().ToString() +
                            "&query=" + Uri.EscapeUriString(spokenWord));
                    var results = JObject.Parse(result);
                    var output = (string)results["result"]
                      ["fulfillment"]["speech"];

                    await this.Dispatcher.RunAsync(
                      Windows.UI.Core.CoreDispatcherPriority.Normal, () =>
                   {
                        PlayResponse(output);
                    });
                }
            }
        }
        catch (Exception ex)
        {
            //log
        }
        finally
        {
            //result the main recognition session to listen for trigger word
            await recognizer.ContinuousRecognitionSession.StartAsync();
        }
    }
    .....
```

14. Finally, let's review the `PlayResponse` method. This method simply builds an SSML string, selects one of the available synthesized voices, and plays the SSML value via the `MediaElement` object:

```
    private async Task PlayResponse(string text)
    {
        try
        {
            MediaElement media = new MediaElement();
            SpeechSynthesisStream stream = null;

            string Ssml =
                @"<speak version="1.0">" +
                text +
                "</speak>";

            var voices = SpeechSynthesizer.AllVoices;
            using (var speech = new SpeechSynthesizer())
            {
                speech.Voice = voices.First(gender => gender.Gender ==
                  VoiceGender.Male && gender.Description.Contains("David"));
                stream = await speech.SynthesizeSsmlToStreamAsync(Ssml);
            }

            media.SetSource(stream, stream.ContentType);
            media.Play();

            media.Stop();
        }
        catch (Exception ex)
        {
            //log
        }
    }
    }
    .....
```

15. Now that we have reviewed the critical elements of our code, we are ready to take it for test drive. As mentioned earlier, you can choose to simply deploy and run our project to your IoT Core Pi, or copy and paste our code into your project.

    If you choose to copy and paste our code into your project, remember to replace the Client Access Token (APIKEY) to match your API.AI Token. If you choose to simply run our cloned project, you will need to update the Client Access Token as well the Remote Machine IP Address.

Deploying and Testing

Whether you use our project or build your own, VS makes deploying and testing the UWP app simple. Before we begin, let's make sure our hardware is ready for testing. Ensure that your IoT Core device is powered up and the microphone and speaker are connected. Next, from the toolbar in VS, click the Remote Machine button. This will compile and deploy the app to your IoT Core device. Once the app is deployed (the VS toolbar options will update to display the Continue and Stop icons), you are ready to test the app.

If you encounter issues with deployment, double-check that the Remote Machine IP address matches your device's IP address.

While in debug mode, speak the word "Hal" into the microphone. You should hear a "Yes?" response from the device. Once you hear "Yes?", say the following into the mic: Say ‘Hello’. You should then hear the device reply with one of the response phrases created in API.AI. If everything worked as planned, pat yourself on the back —you just created a voice-activated app on IoT Core running on a Raspberry Pi!

What’s Next?

The UWP app we created can obviously be extended, and we invite you to experiment with it, perhaps even switching from API.AI to MS Cognitive Services as the NLP/NLU. Don’t stop here; really dig in and play with the various tools we discussed in earlier chapters. Above all, have fun with it!

In the next chapter we’ll pivot a bit and explore using other types of inputs and sensors and discuss security-related topics and going to production with your voice-enabled device.

CHAPTER 6

What Else Can We Do?

As you can imagine by now, there are myriad ways we can grow our voice-enabled product. We can add more inputs (e.g., sensors) and add more outputs (e.g., a digital display or different types of LEDs). Alternatively, we can also create variations of our product or expand the product line, as Amazon has done with Echo and Alexa. Initially, there was just Echo and Alexa, then Alexa appeared in Fire TV, then Echo Dot, Echo Tap, Echo Look, Dash Wand, Echo Show, and a number of other Amazon products, not to mention all the third-party Alexa-enabled devices.

We do this by first understanding how the product will be used and how to elevate that experience. For instance, does adding a motion sensor help the user experience or make it worse? Does adding a screen or extra buttons improve the overall experience and if so how would they be used? The best way to answer all these questions is to create variations of your product and get beta testers involved. Invite your friends and family over to play with the device to gather feedback. That feedback loop is critical to the success of your device. Even if you will be the only user of the device you're building, it's helpful to have other people test it—they can provide you with insight you wouldn't have otherwise thought of on your own.

There are a number of IoT sensors out there that you can use. Here's a short list of sensors you might want to consider playing with—do some research and experiment with how these sensors can be integrated into a voice interface such as Alexa:

Name	Purpose
PIR motion sensor	Senses infrared radiation to detect motion
Moisture sensor	Detects moisture levels
Gyroscope	Detects direction
Photocells	Senses light
Sonar	Detects objects by bouncing ultrasonic waves

Name	Purpose
Accelerometer	Measures motion and tilt
Force sensitive resistor	Senses pressure and force
Temperature sensor	Measures temperature
Piezo	Generally used as a buzzer but also used as a knock sensor
Ball tilt sensor	Senses orientation

If you are interested in learning more about sensors, Wikipedia keeps a great list at *https://en.wikipedia.org/wiki/List_of_sensors*. There you will find a list of over 300 sensors that you can read up on and consider incorporating in your experiments and ultimately your product. While there are multiple paths we can take from here, in this final chapter we will include one last example that entails adding a motion sensor as an input. We will then cover some of the key considerations you will need to think about if you want to move forward with taking your great invention to production.

Adding More Inputs

Now that we have our own voice-enabled device with a custom skill, let's see how we can integrate additional inputs such as a motion sensor. In this example, by adding a simple motion sensor, we can detect when people walk by and trigger an event like having Alexa say "Hello" or simply turning on an LED light. To follow along, you will need a passive infrared (PIR) sensor, sometimes referred to as pyroelectric infrared, or simply PIR (Figure 6-1). These are typically found in home security motion sensors but can also be ordered online for less than $10.

At a high level, PIR works by detecting infrared radiation levels above absolute zero; because humans and other animals typically radiate infrared wave lengths at around 10–12 μm (microns), this is a great technology for sensing people as well as creatures and other objects that radiate a decent amount of infrared light. We're not going to dive too deep into the science of a PIR sensor, so if you want to learn more, check out Adafruit's "How PIRs Work". While you're there, feel free to order a few of the PIR (motion) sensors to play with.

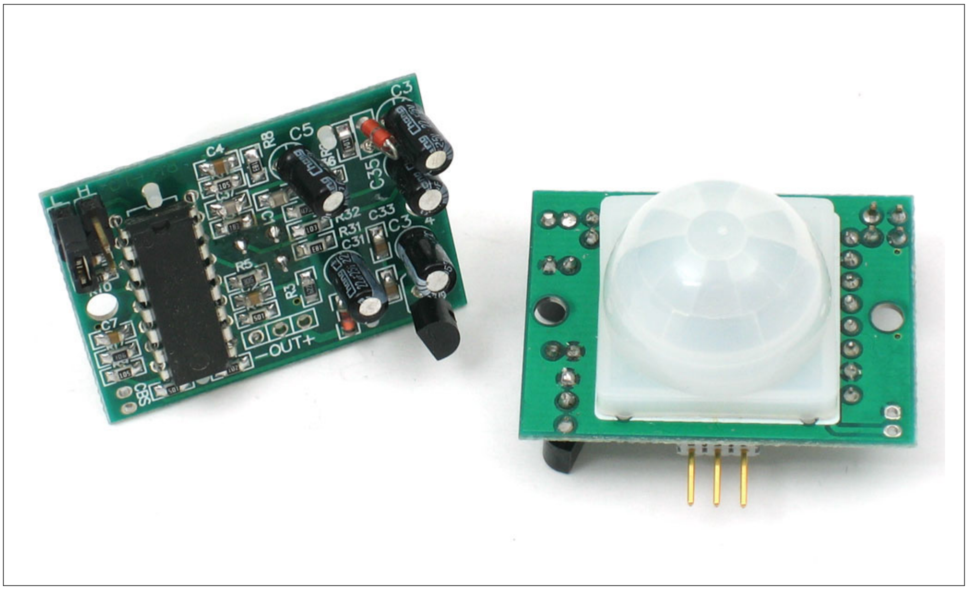


Figure 6-1. PIR Sensor - Photo Creative Commons, Adafruit Learning Systems

Generally PIR sensors will have three pins: one for ground, one for power, and one for the signal. Power is generally 3V to 5V, but can be as a high as 12V in some cases. As for the particular model shown here, you can feed it anywhere between 5V and 12V. The signal output is 3.3V, and can detect from up to 20 feet away at a 120 degree cone. The sensitivity and signal firing can be adjusted to your liking; for example, in some cases you might want to adjust the sensitivity to "min" and time to "max" (around 4 seconds) to detect when a person is standing in front of your device for a set period of time instead of just walking by.

Alternatively, you can use a sonar sensor, which uses ultrasonic waves to detect objects, or if you need something more precise for your particular application, you can look into using ToF Distance Sensor. The Adafruit VL53LOX is an amazing little sensor smaller than a US quarter dollar coin; the VL53LOX can detect objects by sensing the reflection of a very narrow field of light, unlike PIR and Sonar that detect within a wide cone or Field of View (FoV). With the wide range of motion sensors available, it's a good idea to test as many as possible in order to determine which ones work best with your device.

As we mentioned earlier, this example will use a PIR sensor, so let's get to it. First thing we'll want to do is take a look at the diagram in Figure 6-2, so we know how to wire up our motion sensor.

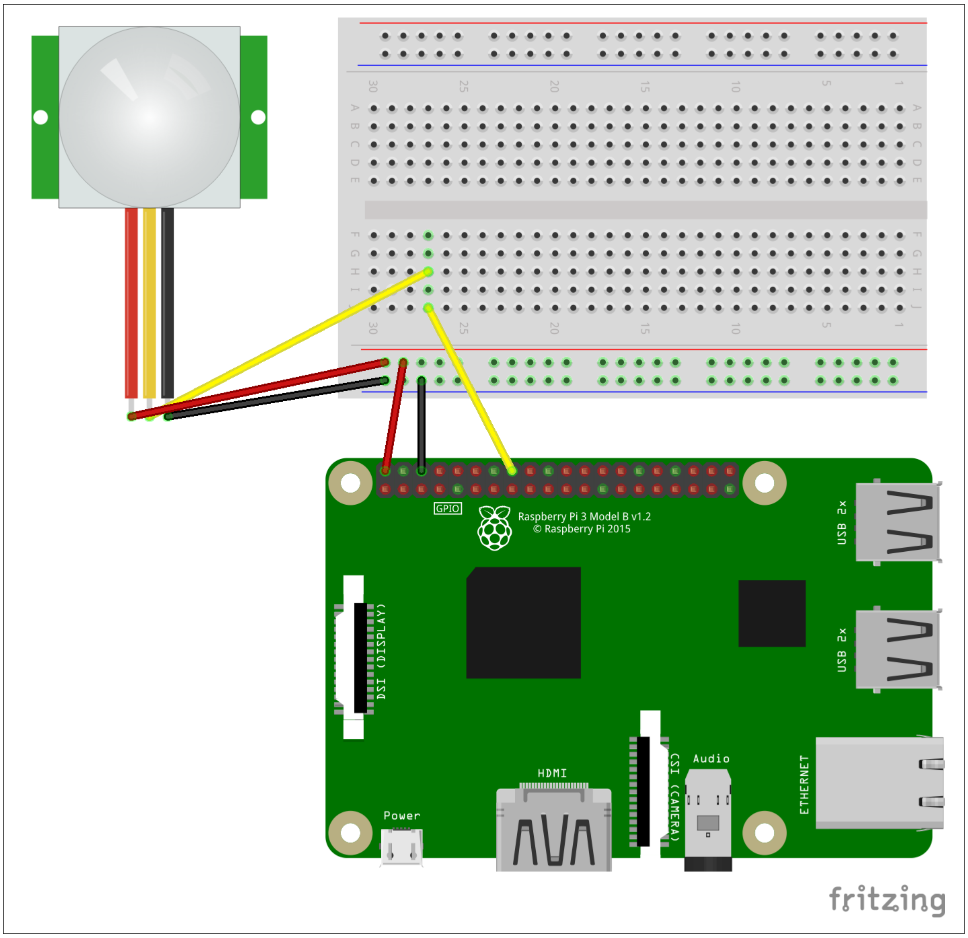


Figure 6-2. Raspberry Pi with PIR sensor wiring digram

As you can see in Figure 6-2, we have the ground (black wire) connected to the Pi's ground and the power (red wire) connected to the Pi's 5V power output. As for our PIR signal (yellow wire), we connected it to GPIO 23 but you can wire it up to another GPIO pin if 23 is taken. Once we have all our connections in place, next thing we will want to do is modify the AlexaPi code to listen for the PIR events. For this we will need to open the *rpilikeplatform.py* file we talked about earlier in "Working with a Button and LEDs" on page 56.

Additionally you can add a new key value to the configuration file; for instance, under the `raspberrypi` section, add the first line as follows:

```
raspberrypi:
  motion: 23
  button: 18
  plb_light: 24
  rec_light: 25
```

Next open up the *rpilikeplatform.py* file in the *src/alexapi/device_platforms* directory. Once the file is open, look for the definition function called `setup`. There, we will set up our GPIO pin to listen to inputs from the PIR motion sensor; for instance, in the following code, simply add the first line (bolded for clarity) under `setup`:

```
def setup(self):
  GPIO.setup(self._pconfig['motion'], GPIO.IN)
  GPIO.setup(self._pconfig['button'], GPIO.IN, pull_up_down=GPIO.PUD_UP)
  GPIO.setup(self._pconfig['rec_light'], GPIO.OUT)
  GPIO.setup(self._pconfig['plb_light'], GPIO.OUT)
  GPIO.output(self._pconfig['rec_light'], GPIO.LOW)
  GPIO.output(self._pconfig['plb_light'], GPIO.LOW)
```

Now let's look for the `after_setup` definition function. There, we're going to add GPIO event detection so when the motion is sensed from the PIR sensor, a function is called. In the `after_setup` function, we add a new line to the bottom of the function that listens for a HIGH event on the motion pin specified in the config file:

```
def after_setup(self, trigger_callback=None):
  self._trigger_callback = trigger_callback
  if self._trigger_callback:
  # threaded detection of button press
  GPIO.add_event_detect(self._pconfig['button'], GPIO.FALLING,
                        callback=self.detect_button, bouncetime=100)
  GPIO.add_event_detect(self._pconfig['motion'], GPIO.HIGH,
                        callback=self.detect_motion, bouncetime=100)
```

Note that in the preceding code we set the callback to `self.detect_motion`, which doesn't currently exist. No worries, we are going to add a new function that will handle what we want to handle, when motion is detected. To test, we'll start with something easy like turning on and off one of the LED lights. Add the following code right before the `detect_button` function like so:

```
def detect_motion(self, channel=None):
  logger.debug("Motion detected!")
  GPIO.output(self._pconfig['plb_light'], GPIO.HIGH)
  time.sleep(5)
  GPIO.output(self._pconfig['plb_light'], GPIO.LOW)

def detect_button(self, channel=None):
  self._trigger_callback(self.force_recording)
  logger.debug("Button pressed!")
  time.sleep(.5)  # time for the button input to settle down
  while GPIO.input(self._pconfig['button']) == 0:
    time.sleep(.1)
  logger.debug("Button released.")
  self.button_pressed = False
  time.sleep(.5)  # more time for the button to settle down
```

One last piece of code we need to update is in the `cleanup` function. Here we will remove the GPIO event detection to free up those resources when we're not using them. In `cleanup` we will add the first line specified here, where we're simply passing in the pin specified in the config file to the `GPIO.remove_event_detect` function:

```
def cleanup(self):
  GPIO.remove_event_detect(self._pconfig['motion'])
  GPIO.remove_event_detect(self._pconfig['button'])
  GPIO.output(self._pconfig['rec_light'], GPIO.LOW)
  GPIO.output(self._pconfig['plb_light'], GPIO.LOW)
```

Now we test. Go ahead and reboot the Pi. When the Pi comes back online, you will be able to pass your hand in front of the motion sensor or walk past it and it will light up the LED. Once we know this works for certain and as expected, we can start to think about other events we can activate inside of the `detect_motion` function, such as play a sound or brighten a display in much the same way the Nest thermostat does when you walk past it.

Going to Production

We've gotten our feet wet with some fun prototypes, but if you're looking to create a product to sell to the public, there's a much longer road ahead that includes strategic planning around security, testing, assembly, procurement, logistics, support, and much more. Successful product development is a complex process that has to be carefully executed, or else the potential for failure is too high, no matter how "cool" your product is. When prototyping, we can plug in some jumpers, write some code, and it works great, but as soon as that gets into the hands of customers it will easily break.

The first step in going to production is trashing your current prototype! You want to make sure you have iterated enough times that you are satisfied with the minimal viable product (MVP) and it serves its purpose. This can't be done on the first iteration; you'll need to take some time with it, but it shouldn't be a long, drawn-out process—it's all about finding the right balance. Tony Fadell, who designed and helped launch popular products such as the iPod and Nest, was well known for saying, "Don't take longer than a year to ship a product." Whether you take six months or a year is up to you; just draw the line in the sand and stick to it. Basically, once those MVP requirements are met, it's time to launch!

A good, value-driven MVP project plan is a great way to establish when it's time to go to production, and it's important to consider the overall production lifecycle as you draw up the project outline. By now you should have an MVP in mind, based on our design thinking discussion in Chapter 1. Once you have a solid prototype ready, taking your MVP prototype to production requires some additions to that MVP requirements list. Some key considerations include:

- Security
- Quality management
- Manufacturing
- Support
- Regulations

Additionally there's logistics around warehousing and shipping, which really depends on where you are manufacturing your product. You could partner with a factory that can handle most of the product lifecycle for you or simply one aspect of it. In the beginning, however, your best option is to find a factory that can help with both PCB design, enclosure design, even specification and documentation all the way through possibly drop-shipping. The cost per unit might be much higher in the beginning, but overall costs could potentially be lower, thus reducing your risk level.

Security Concerns

For most tinkerers and makers, security is an afterthought. Security needs to be baked into your MVP way before going to production. It doesn't necessarily need to be in the first or second prototype iteration (unless, of course, you are working on a product that requires it, such as a home security system or an ATM), but it should be considered early for all product development. The last thing you want is to unwittingly deploy a thousand vulnerable units and have your product fall victim to a distributed denial-of-service (DDoS) attack—that's not the kind of press you're looking for as you head to market.

So how do you create a secure product? First, create a checklist that covers all components of your product and services. For example, database security requirements, web API security requirements, edge security requirements, and so on. The following are some key considerations to help you get started. As you iterate, you might consider adding new components and new security requirements.

Authentication

As we've seen with Alexa, we use OAuth for authentication security. This has become the standard for authenticating users. For device-level authentication, however, make sure to create a strong password for your Raspberry Pi at minimum. If you are not using AVS and are going straight to your own APIs, make sure to incorporate some kind of X.509 certificate handling to ensure the requests to your API are coming from approved and authenticated devices.

Additionally, add a threshold to login attempts. If a login counter set in your authentication method reaches this threshold (e.g., say ten login attempts in under a minute), it's highly likely that there's an automated password generator attempting to hack in and it's a good idea to lock that account.

Cryptography

This applies to both transport and storage of data. Ideally, any sensitive data such as passwords, PINs, birth dates, and other personally identifiable information (PII) are encrypted and stored in the secure database online. Then, depending on your transport protocol, say HTTPS, you use SSL/TLS to encrypt the transport of the data.

Data validation

In your web API, once you've validated the request and user (if any) make sure you validate all input data. For example, if the input includes a string and two integer parameters, make sure to check for null values, validate the value types, measure the length of the strings for an expected range, then ensure the integers fall within an expected range as well. Same goes for dates, times, decimals, and any other primitive and complex data types.

Error handling

Whether you are displaying an error message in a monitor or API response, it's important to omit any system-identifying information such as OS type, source code references, and file paths, as this information can be used by attackers. It's always best to log those details in a secure system logfile, then simply return an error code and user-friendly error message. If you are storing crash logs or debugging data, make sure it cannot be downloaded by an unauthorized user. You'd be surprised how many of us simply dump this data into a root file or directory.

While this is in no way meant to be a comprehensive security checklist, it does shine a light on the multiple aspects of security you need to consider for your voice-enabled IoT device. Make sure to do the proper research, planning, and implementation of each of the components of your specific product.

Quality Management

Quality, like security, is another one of those "I'll deal with it later" subjects when it comes to prototyping, and understandably so: the prototype simply needs to work, not necessarily work all the time or even look good—it just needs to function enough to prove that it works. That's the nature of prototyping. But when it comes time to go live, you want to make sure you have a solid quality management process in place.

This encompasses two key areas: quality assurance (QA) and testing and quality control (QC). Let's take a closer look at each.

Quality assurance

QA is the process to which one assures quality. It's your proactive plan to prevent defects and bugs in your hardware and software. Your QA plan is where you outline all the testing techniques, both automated and manual. This should be a well-written document that can be handed off to others in the event they need to implement the testing criteria so that all expectations and requirements are met.

Testing and quality control

QC is the actual implementation of testing controls that capture any defects and bugs. This can be broken down into dozens of sections once you take into account all the different types of testing you might need to run on your products and services. On the software side, there's unit testing, automated functional testing, as well as security and penetration testing. On the hardware side, you have in-circuit testing, design verification testing, optical testing, and other testing techniques designed to ensure the PCB and overall hardware was produced to spec. Then, on the usability side, you have user acceptance testing, smoke tests, A/B testing, and so on.

Support

Know your limitations! As one person, you can only handle so much, especially when it comes to supporting your users. The last thing you want is to be fielding tech support calls all day long. Sure, you think you've put in some solid quality controls and all tests passed but as soon as that product hits the hands of customers, it's 100% guaranteed you will receive support inquiries.

One thing you can do to provide support and capture feedback is to build it into your voice experience. Take Alexa, for example—people can say, "Alexa, I need help." This can also be done with more custom solutions, as with the API.AI demonstration we looked at earlier. Here, we can add a response for "I need help" such as "Please visit *www.mydomain.com/support* to learn more" or potentially follow up with a question like, "What do you need help with?" and keep the conversation going from there to provide a robust chatbot-style support system.

Here are some additional things to consider when thinking about your support plan:

- Documentation
- Automated support
- Escalated support
- Returns
- Handling complaints

Documentation

As the saying goes, "Communication is key." The best way to communicate with users is through your documentation. Whether you are targeting developers or end users, the clarity and completeness of your documentation will help reduce tech support inquiries by at least 50% as the majority tech support issues out in the wild are related to user error. You can publish documentation online in a Help section of your website with an easily accessible keyword search system. A Frequently Asked Questions (FAQs) section is another great way to synthesize all those support inquiries and make it easily accessible for others to learn from.

When shipping products that require installation, including an iconography-rich installation guide works wonders, as in the case with IKEA and Apple products. Make sure to include in the printed materials any short URLs to extended online support libraries or Help sections so your users can easily navigate to it online without having to type in long, complex URLs.

Automated support

Another great way to provide help is via chatbots. Using API.AI or Microsoft's Qnamaker.ai, you can easily load up your FAQs and make them available in bots for Slack, Facebook Messenger, and other channels your users can engage in. You can even get real fancy with it by asking users key questions to rule out common issues (e.g., "Is it plugged in?"; "Is the green light on or is it red?"; etc.). Just make sure not to close users off here. You will definitely need a way to escalate the user's issue if it cannot be resolved by the chatbot. This can be done by either creating an automated support ticket or redirecting the user to chat with an actual person.

Escalated support

This is where support begins to get expensive. If a user has exhausted all other means of support and his only option is to talk to an actual person, then either online chat or phone support comes into play. In the beginning, it will most likely be you providing support. This is another great reason to move only a small amount of units at a time. For each batch you produce and ship, analyze the number of support inquiries you receive. Use this feedback loop to not only help your users but gather important ideas on how to improve your automated support as well as the quality of your product on future batches.

Returns

Strange things happen all the time. Packages might be missing parts or arrive damaged, or a customer may want to return the product because it's the wrong color and doesn't fit the feng shui of her home. Whatever the reason, you'll need to handle those returns efficiently. Set aside a dedicated space to store returned devices until they can be tested, refurbished, and resold. Refurbished devices can be sold at a marginal discount to attract entry-level customers or those on a budget who wouldn't have normally purchased your product, and at the same time, you're covered on the per unit cost.

Handling complaints

“Your product sucks!” “It doesn’t work!” “Your product is garbage!” Are you offended yet? If so, don’t go live with your product. Ever! You need to have thick skin when dealing with the public. Even the best of the best—yes, even Steve Jobs himself heard these same words about Apple products. Seriously, how many times have you wanted to throw your iPhone out the window? It happens. People complain, that’s what we do. Don’t take offense. Simply try to identify if there’s anything constructive in the complaint; for example, if it’s an irrational user blasting about how the color of the LED should be fuchsia instead of purple, then respond with something like “Great suggestion!” and move on. But if a user has a logical explanation about adjusting the positioning of the LED, then just cut out all the noise and add that suggestion to your backlog.

Some things may sound irrational to you but to others it makes perfect sense. Sometimes you’ll hear the same complaints over and over from multiple users. This should be a red flag. Consider the fuchsia color example, for instance. Why is the user so obsessed with that color? Why are multiple people asking for that color? There must be something there, right? Maybe, maybe not, but you decide to make it fuchsia and all of a sudden, you have more people asking for the purple back! Insanity can easily set in. Just remember, you cannot satisfy all people, all the time. This is why we build multiple editions, versions, variations, and additional offerings of a product. Additionally, A/B testing can be implemented to help you identify which group of people prefer certain features over other features and so on.

Overall, always remember why you built that first prototype to begin with. Was it to have fun? Build something cool? Make some serious cash? Whatever it was, always remember that purpose when things get insane.

What’s Next?

So there you have it. You’ve successfully learned to rapidly prototype a voice-enabled device. Now what? Now you keep iterating and improving. To help you along your journey in getting you through to the next steps, the following are some additional resources you can reference:

Books

- *Prototype to Product* by Alan Cohen
- *Designing Voice User Interfaces* by Cathy Pearl
- *Designing Connected Products* by Claire Rowland, Elizabeth Goodman, Martin Charlier, Ann Light, and Alfred Lui
- *Collaborative Product Design* by Austin Govella

- *Product Roadmapping* by Michael Connors, C. Todd Lombardo, Evan Ryan, and Bruce McCarthy

Online Courses and Videos

- Designing for AVS
- Designing for Voice: Conversational UI
- Learning Path: Programming the Internet of Things

Index

About the Authors

Walter Quesada is a software engineer with over 20 years of experience designing and developing applications for desktop, web, mobile, and devices for the Internet of Things. His primary focus includes integrating natural user interfaces such as touch, voice, and gestures into reimagined experiences around retail, hospitality, healthcare, and other sectors. Additionally, Walter speaks at occasional local code camps, works as a Pluralsight instructor, and is a cofounder of the South Florida Emerging Technology Meetup group. He currently resides in Miami, Florida.

Bob Lautenbach is a veteran of designing software for the hospitality and cruise industries, and he firmly believes voice technology is poised to revolutionize the way the industry operates. His main interests are around leveraging IoT devices to create new and engaging guest experiences. Bob is the cofounder and Chief "Tinker" Officer of Voceio, an Orlando-based firm that creates multichannel conversational platforms. Bob is an Alexa Champion, Pluralsight author, and founder of the Orlando Alexa Meetup group. He currently resides in Orlando, Florida.

Colophon

The animal on the cover of *Programming Voice Interfaces* are Mozambique rain frogs (*Breviceps mossambicus*). These frog can be found throughout south and central Africa in dry, wet, or subtropical savannas, greenlands, or shrublands. They are part of the Brevicipitidae family.

The Mozambique rain frog is round and stout in physique. Females can grow to be about two inches in length (from the flat snout to the base of the hind legs). The outer of the four toes on each leg are considerably shorter than the inner ones. The legs themselves are fairly short as well. Coloring in the sexes varies only slightly, with brown shades appearing in the throat of males. The rest of the underbelly is a cream color with darker blotches. The back or "top" part of the frog is also of a brownish shade, with specks of darker colors. There is also a dark streak that extends down the sides of the frog, from the eye to the back legs.

Unlike most frogs, the Mozambique rain frog doesn't croak. Instead, it emits a higher-pitched noise, like a squeal. This sound more closely resembles a kitten's meow than a traditional frog cry.

Many of the animals on O'Reilly covers are endangered; all of them are important to the world. To learn more about how you can help, go to *animals.oreilly.com*.

The cover image is from *Lydekker's Royal Natural History*. The cover fonts are URW Typewriter and Guardian Sans. The text font is Adobe Minion Pro; the heading font is Adobe Myriad Condensed; and the code font is Dalton Maag's Ubuntu Mono.