Kishore Gaddam

Building Bots with Microsoft Bot Framework

Build intelligent and smart conversational interfaces using Microsoft Bot Framework

Packt

WOW! eBook
www.wowebook.org

- 1: Setting up Microsoft Bot Framework Dev Environment
  - Chapter 1: Setting up Microsoft Bot Framework Dev Environment
  - Conversation as a Service (CaaS)
  - Your bot
  - The Bot Connector
  - The Bot Directory
  - Setting up the development environment
  - Messages
  - Cards and buttons
  - Groups
  - Calling
  - Summary
- 2: Developing Your First Bot Using the Connector and Builder SDK
  - Chapter 2: Developing Your First Bot Using the Connector and Builder SDK
  - Bots are evolving
  - Bots use cases
  - Developing your first bot
  - Creating our first bot
  - Post method
  - How to deploy and run the bot application in the Bot Framework emulator locally
  - How to use dialogs in bot applications
  - How to use FormFlow in the bot application
  - Summary
- 3: Developing WeatherBot Using Dialogs and LUIS
  - Chapter 3: Developing WeatherBot Using Dialogs and LUIS
  - Language Understanding Intelligent Service (LUIS)
  - Intents and Entities
  - Training your bot using utterances
  - Development of WeatherBot code
  - Calling LUIS from the bot
  - Natural speech and Intent processing bot using Microsoft Cognitive Services
  - Identifying the name of a person, place, and company using LUIS
  - Training your app
  - Calling LUIS from the bot

WOW! eBook
www.wowebook.org

    - Summary
- 4: Natural Speech and Intent Processing Bot Using Microsoft Cognitive Services
    - Chapter 4: Natural Speech and Intent Processing Bot Using Microsoft Cognitive Services
    - Microsoft Cognitive Services
    - Identifying the name of a person, place, and company using LUIS
    - Training your app using utterances
    - Calling LUIS from the bot
    - Summary
- 5: Developing Bots Using LUIS Prompt Dialogs with State and Nearby Bot Using Custom APIs
    - Chapter 5: Developing Bots Using LUIS Prompt Dialogs with State and Nearby Bot Using Custom APIs
    - Employee Enroll bot using LUIS prompt dialogs
    - Creating the C# class for LUIS response
    - Bot state service
    - Developing a Nearby Bot using custom APIs
    - Summary
- 6: Developing an IVR Bot for a Bank Using Advanced Microsoft Bot Framework Technologies
    - Chapter 6: Developing an IVR Bot for a Bank Using Advanced Microsoft Bot Framework Technologies
    - High-level architectural diagram
    - Let's start coding
    - Storing the bot conversation (new account info) data in an Azure SQL database
    - Checking your savings account balance using the bot
    - Checking your current account balance using the bot
    - Paying your credit card bill using the bot
    - Deleting an account using the bot
    - Summary
- 7: Intelligent Bots with Microsoft Bot Framework and Service Fabric
    - Chapter 7: Intelligent Bots with Microsoft Bot Framework and Service Fabric
    - Getting started using stateless microservices
    - Publishing a Service Fabric project in Azure

WOW! eBook
www.wowebook.org

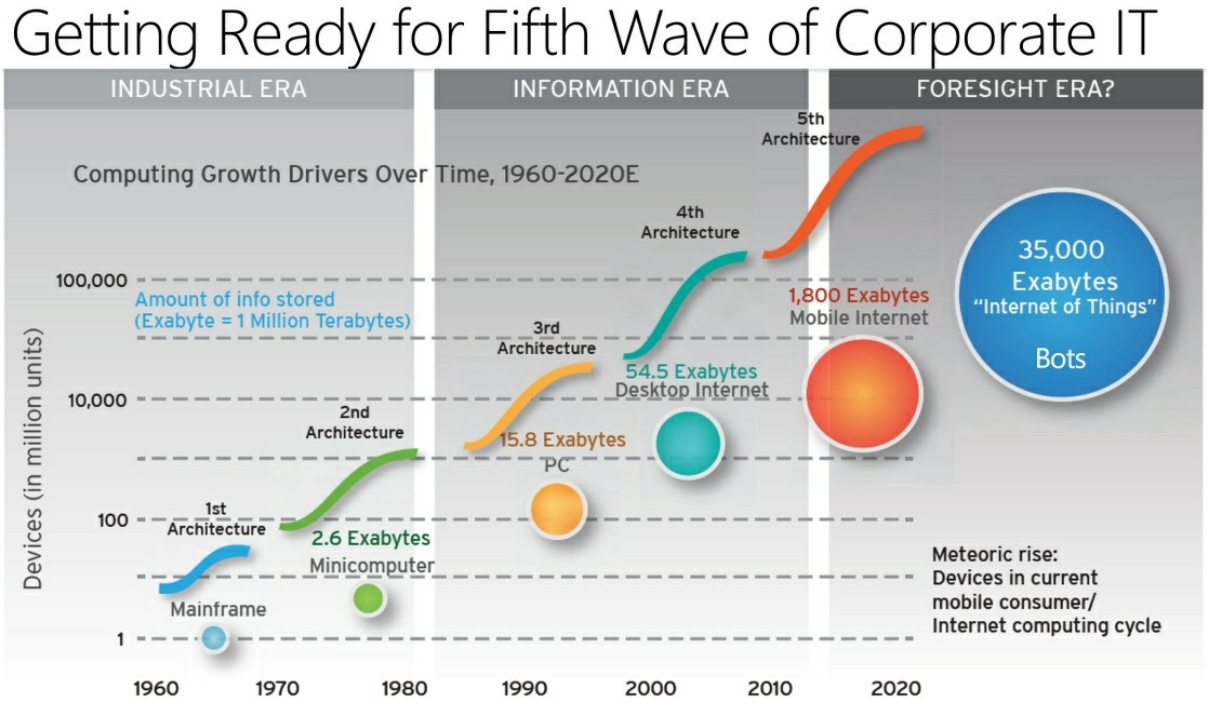
- Summary
- 8: Developing Intelligent Facial Expression Identification Bot for IoT Using Azure and Power BI
  - Chapter 8: Developing Intelligent Facial Expression Identification Bot for IoT Using Azure and Power BI
  - Before getting started
  - Configuring Raspberry Pi and sensors
  - Device identity and registry with IoT Hub
  - Face API
  - Emotion API
  - Sign Up Microsoft Cognitive Services
  - Registering your Bot in Bot Framework
  - Configure Direct Line Channel
  - Develop an UWP app for Raspberry Pi device
  - Deploy Code in to Raspberry Pi
  - Show facial analytics data in Power BI
  - Summary
- 9: Publishing a Bot to Skype, Slack, Facebook, and the GroupMe Channel
  - Chapter 9: Publishing a Bot to Skype, Slack, Facebook, and the GroupMe Channel
  - Publishing bots to various channels
  - Registering your bot with Microsoft Bot Framework
  - Configuring channels
  - Summary

WOW! eBook
www.wowebook.org

Chapter 1. Setting up Microsoft Bot Framework Dev Environment

In the past several decades, the corporate, government, and business world has experienced several waves of IT architecture foundations, moving from mainframes, to minicomputers, to distributed PCs, to the Internet, to social media / mobile, and now to the **Cloud** / **Internet of Things** (**IoT**) stack. We call this the *sixth wave* of corporate IT, and like its predecessors, cloud and IoT technologies are causing significant disruption and displacement, even while they drive new levels of productivity. Each architecture focuses on key business processes and supports **killer technology** applications to drive new levels of value. Very soon we will be looking at an enormous networked interconnection of everyday machines to one another, as well as to humans.

Lets have a look at the fifth wave of corporate IT:



The machine-to-machine-to-human connectivity will have a profound impact on the consumer and corporate IT experience. As these machines become

WOW! eBook
www.wowebook.org

social and talk to us, we have an enormous opportunity to greatly enhance their value proposition through improved product quality, customer experience, and lowered cost of operations. A heightened consumer expectation for more personal and real-time interactions is driving business to holistically embrace the next wave of technology innovation such as cloud, IoT, and bots to boost business performance. In this age of billions of connected devices, there is a need for such a technology where our apps, such as bots, could talk back. Bots that have specific purposes and talk to any device or any app or to anyone, live in the cloud, we can talk to via any communication channel such as e-mail, text, voice, chat, and many others, can go where no apps have gone before when it comes to the machine-to-machine-to-human connectivity. In order to make this happen, we will need a whole new platform, a platform for conversations.

WOW! eBook
www.wowebook.org

Conversation as a Service (CaaS)

Messaging apps in general are becoming a second home screen for many people, acting as their entry point to the Internet; where the "youngins" are, the brands will follow. Companies are coming up with messaging apps as bots and apps that offer everything from customer service to online shopping and banking.

Conversations are shaping up to be the next major human-computer interface. Thanks to advances in natural language processing and machine learning, the technology is finally getting faster and accurate enough to be viable. Imagine a platform where language is the new UI layer. When we talk about conversation as a platform, there are three parts:

- There are people talking to people. The Skype translator is an example where people can communicate across languages.
- Then, there is the opportunity to enhance a conversation by the ability to be present and interact remotely.
- Then, there are personal assistants and the bots.

The following screenshot shows the `Conversation as a Service`:

WOW! eBook
www.wowebook.org

CaaS

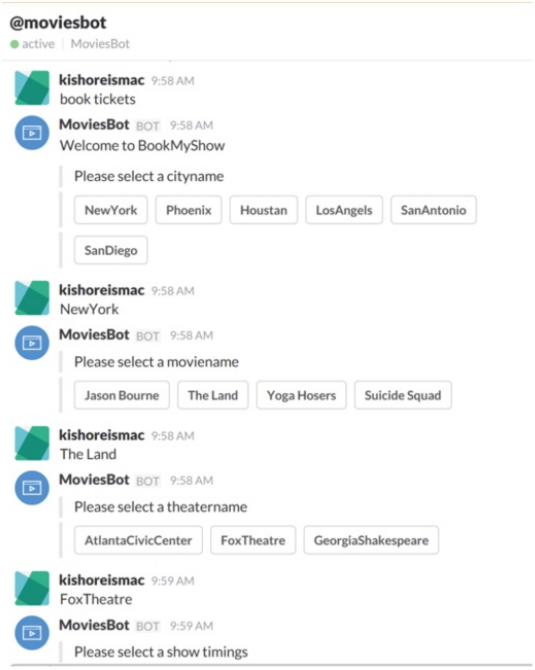
Conversation as a Service

Human language is the new UI

Bots are the new apps;

Digital Assistants are meta apps

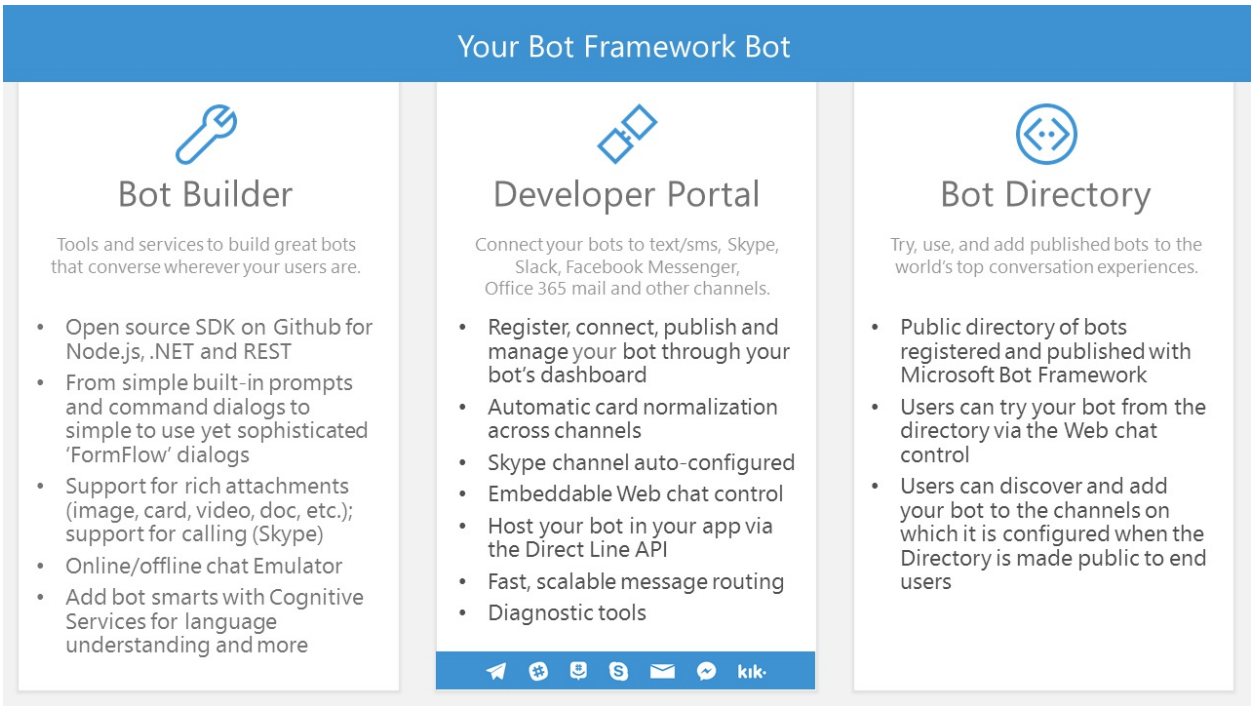
Intelligence infused into all interactions



Think of bots as the new mechanism that you can converse with. Instead of looking through multiple mobile apps or pages of websites, you can call on any application as a bot within the conversational canvas. Bots are the new apps, and digital assistants are the meta-apps. This way, intelligence is infused into all our interactions.

This leads us to the **Microsoft Bot Framework**, which is a comprehensive offering from Microsoft to build and deploy high quality bots for your users to interact using **Conversation as a Platform** (**CaaP**). This is a framework that lets you build and connect intelligent bots. The idea is that they interact naturally wherever your users are talking, such as Skype, Slack, Facebook Messenger, text/SMS, and others. Basically, with any kind of channel that you use today as a human being to talk to other people, you will be able to use them to talk to bots, all using natural language:

WOW! eBook
www.wowebook.org

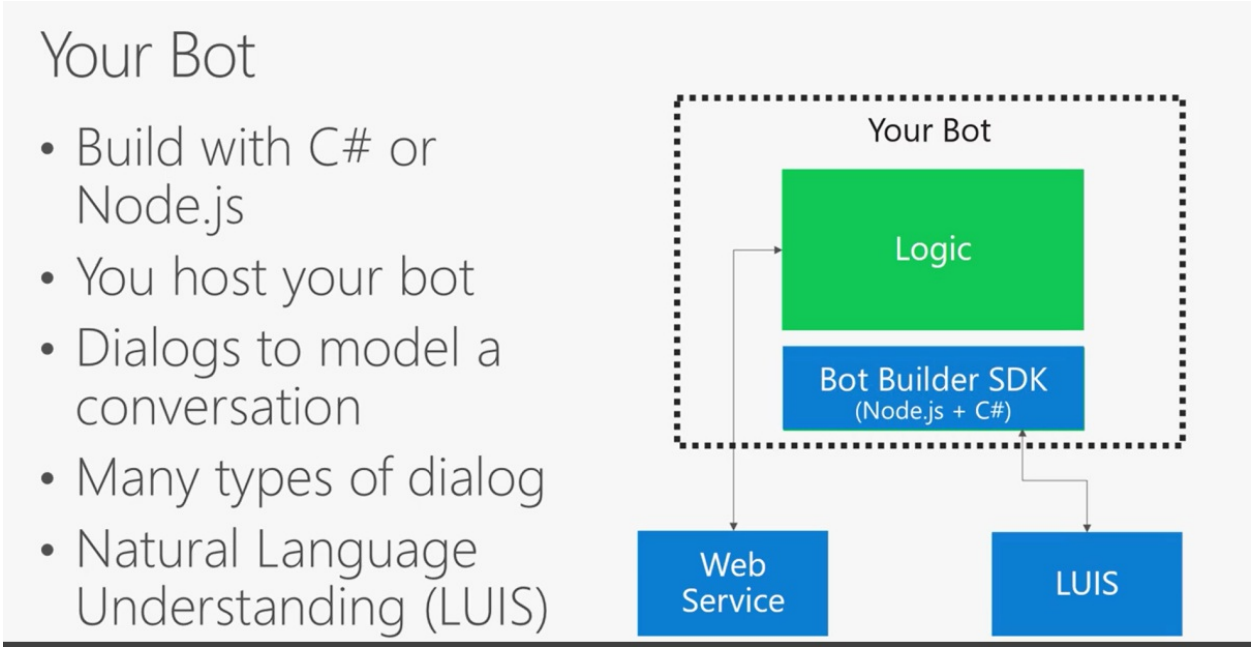


The Microsoft Bot Framework is a Microsoft operated CaaP service and an open source SDK. The Microsoft Bot Framework is one of the many tools that Microsoft is offering for building a complete bot. Other tools include **Language Understanding Intelligent Service (LUIS)**, Speech APIs, Microsoft Azure, Cortana Intelligence Suit, and many more.

WOW! eBook
www.wowebook.org

Your bot

The **Microsoft Bot Builder SDK** is one of three main components of the Microsoft Bot Framework. First, you have to build your bot. Your bot lives in the cloud and you host it yourself. You write it just like a web service component using Node.js or C#, like the **ASP.NET Web API** component. The Microsoft Bot Builder SDK is open source, so it will support more languages and web stacks over time. Your bot will have its own logic, but you also need a conversation logic using dialogs to model a conversation. The Bot Builder SDK gives you facilities for this, and there are many types of dialogs that are included, from simple yes or no questions, to full LUIS, which is one of the APIs provided by **Microsoft Cognitive Services**. This is what the architecture of bot looks like:

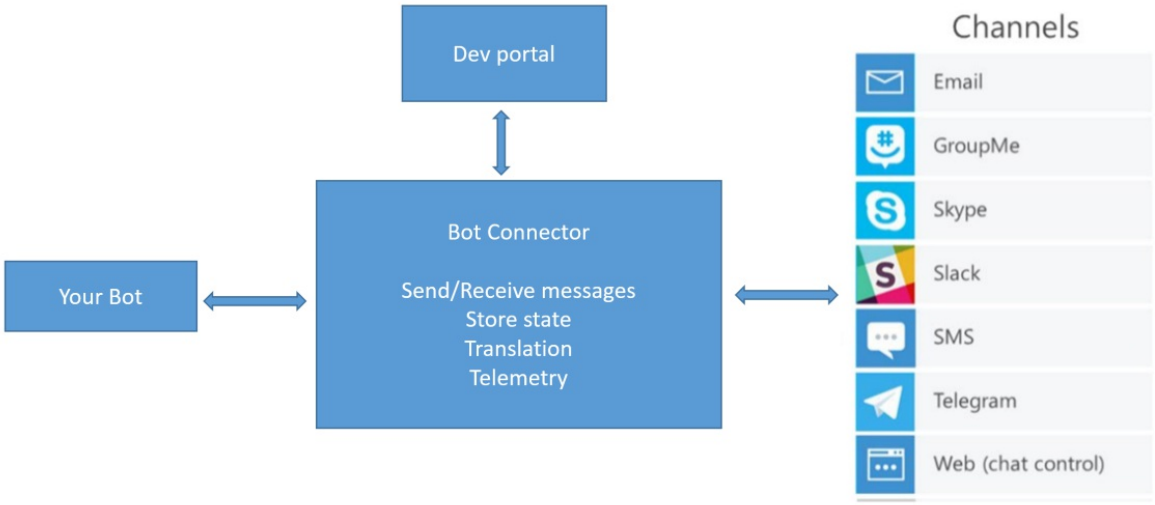


WOW! eBook
www.wowebook.org

The Bot Connector

The **Bot Connector** is hosted and operated by Microsoft. Think of it as a central router between your bots and many channels to communicate with your bots. Apart from routing messages, it manages state within the conversation. The Bot Connector is an easy way to create a single backend and then publish it to a bunch of different platforms called **channels**.

The following screenshot illustrates the Bot Connector:



WOW! eBook
www.wowebook.org

The Bot Directory

The **Bot Directory** is where the user will be able to find bots. It's like an App Store for mobile apps. The Bot Directory is a public directory of all the reviewed bots registered through the developer portal. Users will be able to discover, try, and add bots to their favorite conversation experiences from the Bot Directory. Anyone can access it and can submit bots to the directory.

As you begin your development with the Microsoft Bot Framework, you might be wondering how best to get started. Bots can be built in C#; however, Microsoft's Bot Framework can also be used to build bots using Node.js. For developing any bots, we need to first set up the development environment and have the right tools installed for successfully developing and deploying a bot. Let's see how we can set up a development environment using Visual Studio.

WOW! eBook
www.wowebook.org

Setting up the development environment

In this section, we will see how to set up the development environment but, before that, let's check out the prerequisites needed for setting it up.

Prerequisites

To use the Microsoft Bot Framework Connector, you must have the following:

- A Microsoft account (Hotmail, Live, or Outlook) to log into the Bot Framework developer portal, which you will use to register your bot.
- An Azure subscription (free trial at https://azure.microsoft.com/en-us/). This Azure subscription is essential for having an Azure-accessible REST endpoint exposing a callback for the Connector service.
- Developer accounts on one or more communication service (such as Skype, Slack, or Facebook) where your bot will communicate.

In addition, you may wish to have an Azure Application Insights account so that you can capture telemetry from your bot. There are additionally different ways to go about building a bot: from scratch, coded directly to the Bot Connector REST API, the Bot Builder SDK's for Node.js and .NET, and the Bot Connector .NET template, which is what this quick start guide demonstrates.

Setting up the Bot Framework Connector SDK .NET

This is a step-by-step guide to setting up the development environment to develop a bot in C# using the Bot Framework Connector SDK .NET template:

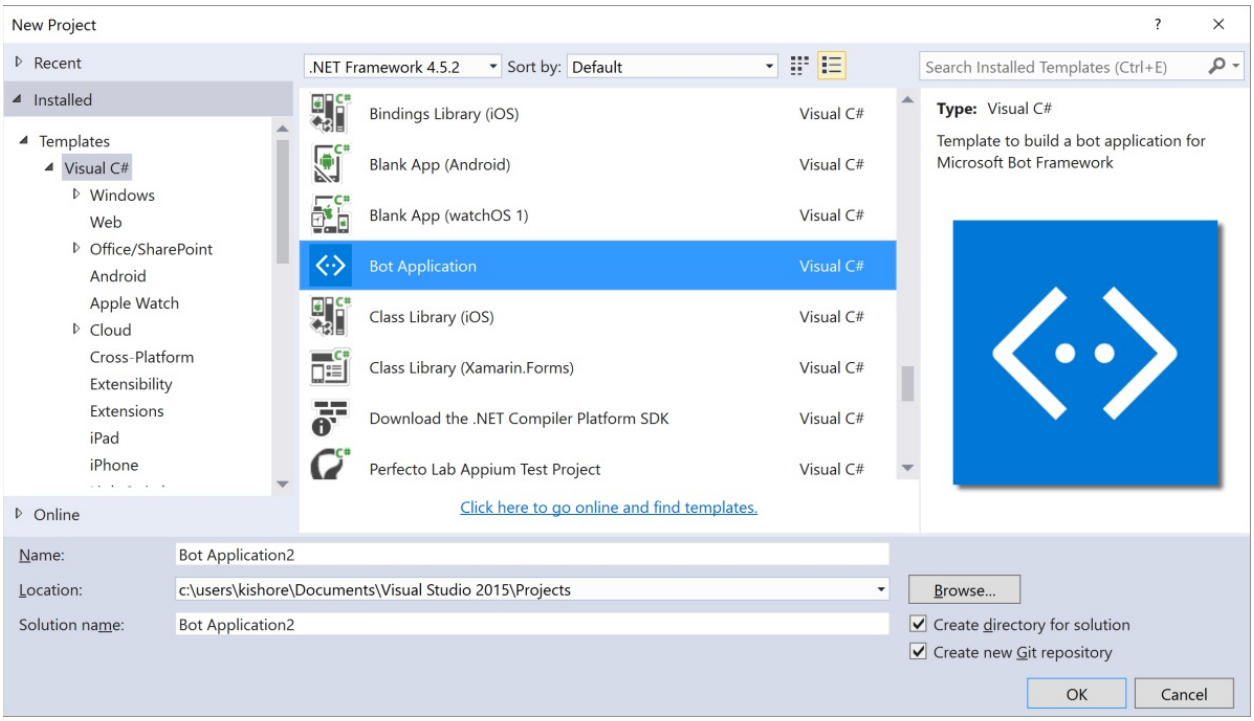
1. Install the prerequisite software:
    1. You can download the community version of Visual Studio 2015 (latest update) for free at www.visualstudio.com.
    2. Update all VS extensions to their latest versions by navigating to

WOW! eBook
www.wowebook.org

Tools | ExtensionsandUpdates | Updates.

2. Download and install the **Bot Application** template:
   1. Download the file from the direct download link, http://aka.ms/bf-bc-vstemplate.
   2. Save the ZIP file to your Visual Studio 2015 templates directory, which is traditionally in `%USERPROFILE%DocumentsVisual Studio 2015TemplatesProjectTemplatesVisual C#`.
3. Open Visual Studio.

3. Create a new C# project using the new **Bot Application** template:



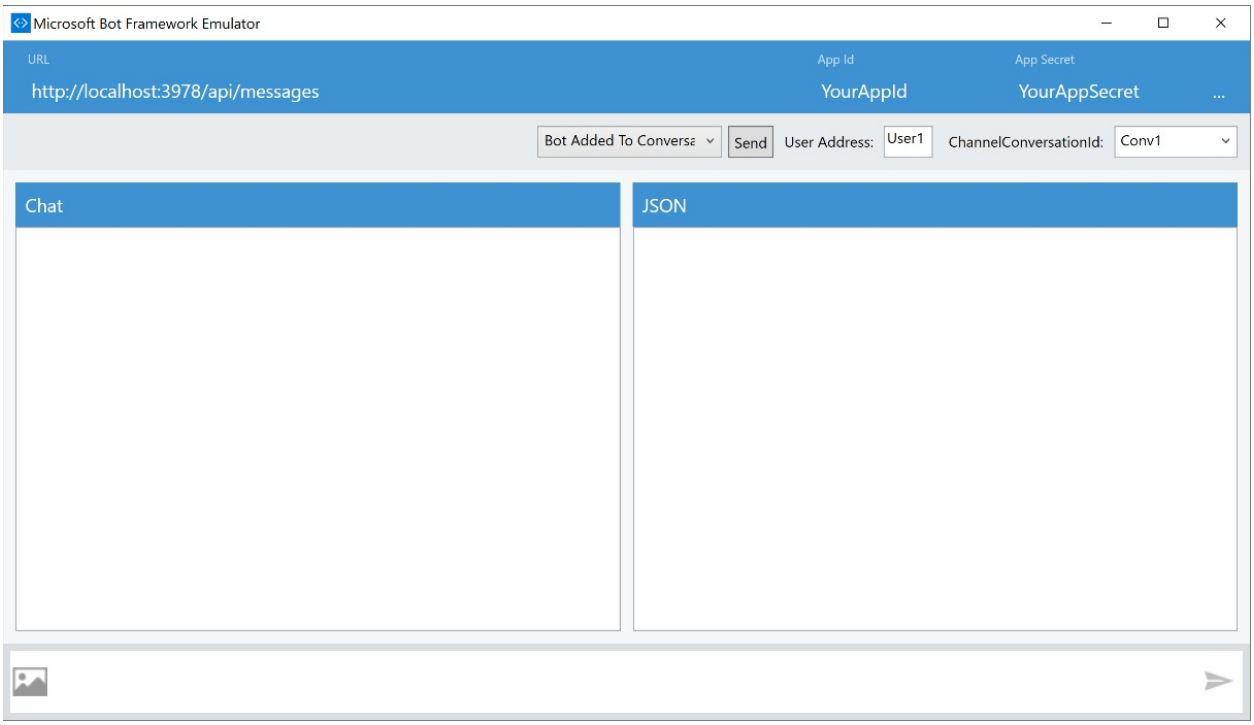
4. The template is a fully functional Echo Bot that takes the user's text utterance as input and returns it as output. In order to run, however, the following has to take place:
   1. The bot has to be registered with the Bot Connector.
   2. The AppId and AppPassword from the Bot Framework registration page have to be recorded in the project's `web.config`.

WOW! eBook
www.wowebook.org

3. The project needs to be published to the web.
4. Use the Bot Framework emulator to test your bot application.

The Bot Framework provides a channel emulator that lets you test calls to your bot as if they were being called by the Bot Framework cloud service. To install the Bot Framework emulator, download it from https://emulator.botframework.com/.

Once installed, you're ready to test, by starting your bot in Visual Studio using a browser as the application host:



WOW! eBook
www.wowebook.org

Messages

Your bot can send rich text, emoticons, pictures, and cards to a user or group. Users can send rich text and pictures to your bot. You can specify the type of content your bot can handle in the Skype settings page for your bot:

Content	From user to bot	From bot to user	Description
Rich text	✓	✓	Including emoticons :)
Pictures	✓	✓	PNG, JPEG, or GIF up to 20 Mb
Video	Coming soon	✓	MP4, AAC+h264 up to 15 Mb (approx. 1 minute), plus JPEG thumbnail

WOW! eBook
www.wowebook.org

Basic format

Each Skype user is assigned a unique ID for your bot, which is sent along with the display name with every message:


```
{
  "text": "Hello (wave)",
  "id": "1466182688092",
  "type": "message/text",
  "timestamp": "2016-06-17T16:58:08.74Z",
  "channelId": "skype",
  "serviceUrl": "https://apis.skype.com",
  "from": {
    "id": "29:2hJJkjmGn4ljB2X7YYEju-sgFwgvnISvE6G3abGde8ts",
    "name": "Display Name"
  },
  "conversation": {
    "id": "29:2hJJkjmGn4ljB2X7YYEju-sgFwgvnISvE6G3abGde8ts"
  },
  "recipient": {
    "id": "28:29415286-5a43-4a00-9dc5-bcbc2ce1f59e",
    "name": "Trivia Master"
  }
}
```


The `from` field contains the unique user ID (prefixed by `29`) and user `Display Name`. The `recipient` field contains the app ID (prefixed by `28`, which indicates a bot in Skype) and the bot's display name. In Skype, your bot is addressed using the Bot Framework App ID (a GUID).

Note

WOW! eBook
www.wowebook.org

> You cannot currently use slash (/) commands as part of conversations with your bot. This is a reserved character in Skype.

Rich text

Users can communicate with the bot using rich text format as well. Users can make the chat text as bold if needed or a bot can communicate with the user and make certain words as bold. Most of the channels support all the text properties supported by the Microsoft Bot Framework.

Skype emoticons

Skype emoticons can be sent by using the emoticon keyword in parentheses:

```
{
  "text": "(heart)"
}
```

The output of the preceding code is as follows:

> ### Note
>
> If a user sends your bot an emoticon, it may include `<ss>` tags around the emoticon, which can be ignored; for example, `<ss type="skype">(wave)</ss>`. Sending Skype Mojis (short, expressive video clips) is not currently supported.

Welcome messages

To send a welcome message to a user, listen for the `contactRelationUpdate` activity. To send a welcome message to a group, listen for the

WOW! eBook
www.wowebook.org

`conversationUpdate` activity.

Pictures and videos

Let's check out, how pictures and videos are sent:

- Pictures and videos are sent by adding attachments to a message
- Pictures can be PNG, JPEG, or GIF up to 20 Mb
- Videos can be MP4 or AAC+h264 up to 15 Mb (approx. 1 minute), plus JPEG thumbnails

WOW! eBook
www.wowebook.org

Cards and buttons

Skype supports the following cards, which may have several properties and attachments. You can find information on how to use cards in the .NET SDK and Node.js SDK docs:

- Hero card
- Thumbnail card
- Carousel card (with hero or thumbnail images)
- Sign in card
- Receipt card

Images sent to Skype cards need to be stored on an HTTPS endpoint. Skype cards do not currently support `postBack` actions.

Hero card

The hero card renders a title, subtitle, text, large image, and buttons:

WOW! eBook
www.wowebook.org

Skype Preview

MoviesBot

Online

Welcome to **Movies bot.** 19:04

Jason Bourne 19:04

Jason Bourne is
an American

View Trailer

The Land 19:04

Teenage buddies
Cisco (Jorge

View Trailer

Type a message here

WOW! eBook
www.wowebook.org

The hero card provides a very flexible layout; for example, it might contain the following:

- Image, title, subtitle, text, and three buttons
- Title, subtitle, text, and five buttons
- Title and six buttons
- Image and six buttons

The following table illustrates the flexible layout of hero card:

Property	Type	Description
`title`	Rich text	Title of the card, maximum two lines.
`subtitle`	Rich text	Subtitle appears just below the title, maximum two lines.
`text`	Rich text	Text appears just below the subtitle; two, four, or six lines depending on whether the title and subtitle are specified.
`images: []`	Array of images	Image displayed at top of the card; aspect ratio is 16:9.
`buttons: []`	Array of action objects	Set of actions applicable to the current card: three buttons, up to a maximum of six (+two if no is image is shown, +one if the title or subtitle are not included, +two if the text is not included.)

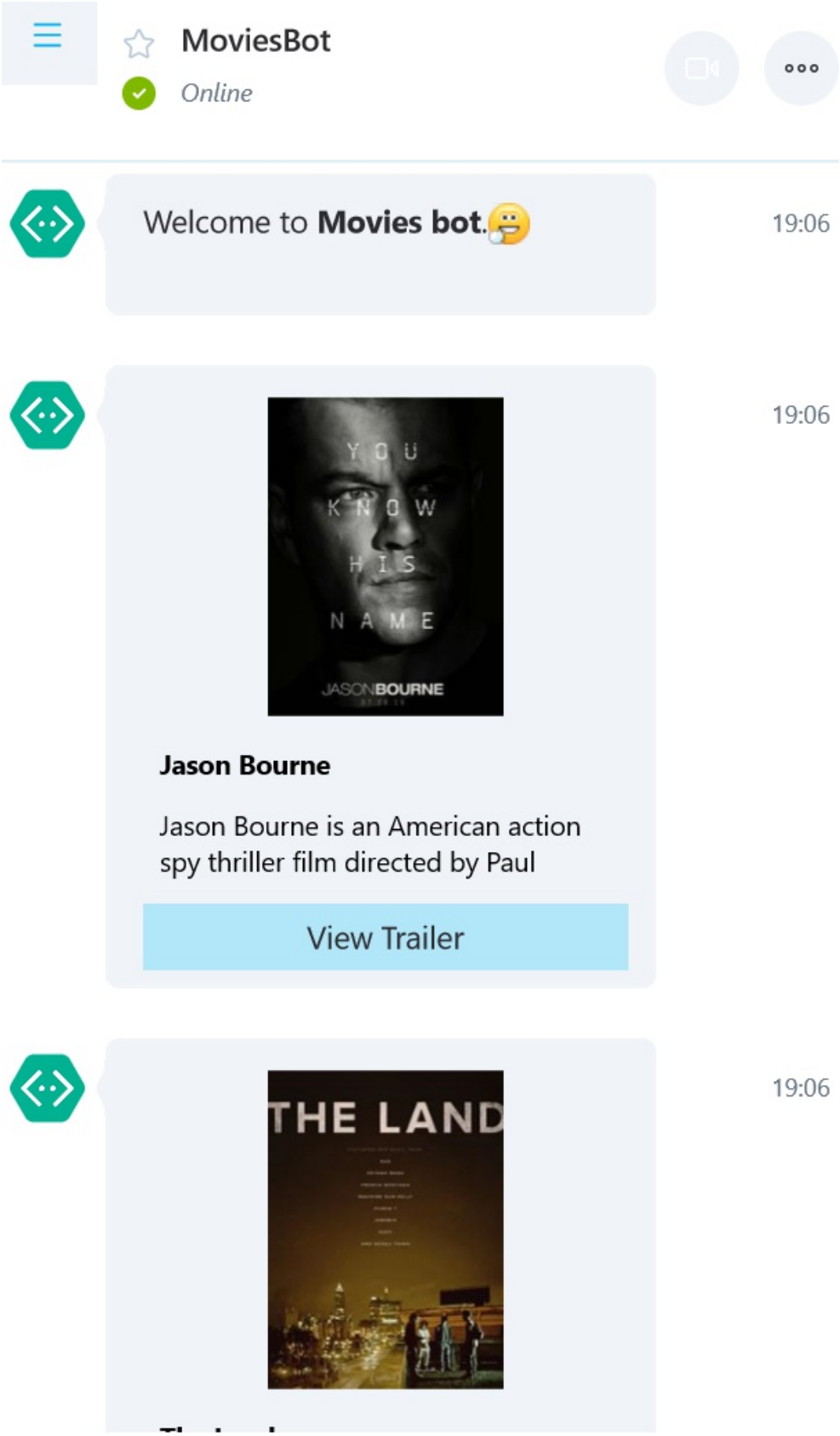
WOW! eBook
www.wowebook.org

`tap` Action This action will be activated when the user taps on the card object itself.

Thumbnail card

The thumbnail card renders a title, subtitle, text, small thumbnail image, and buttons:

WOW! eBook
www.wowebook.org



WOW! eBook
www.wowebook.org

The following table explains the properties of a thumbnail card:

Property	Type	Description
`title`	Rich text	Title of the card, maximum two lines.
`subtitle`	Rich text	Subtitle appears just below the title, maximum two lines.
`text`	Rich text	Text appears just below the subtitle: two, four, or six lines depending on whether the title and subtitle are specified.
`images: []`	Array of images	Image displayed at top of the card; the image aspect ratio in a thumbnail card is 1:1.
`buttons: []`	Array of action objects	Set of actions applicable to the current card; maximum three buttons.
`tap`	Action object	This action will be activated when the user taps on the card itself.

Carousel

The carousel card can be used to show a carousel of images and text, with associated action buttons:

WOW! eBook
www.wowebook.org

Skype Preview

MoviesBot

Online

19:31

The Land

Teenage buddies Cisco (Jorge Lendeborg Jr.), Junior (Moises Arias),

View Trailer

Type a message here

WOW! eBook
www.wowebook.org

Properties are the same as for the hero or thumbnail card.

Images

Images are scaled up or down in size while maintaining the aspect ratio to cover the image area, and then cropped from the center to achieve the image aspect ratio for the card.

Images should be stored on an HTTPS endpoint, up to 1024x1024, up to 1 MB in size, and in PNG or JPEG. The properties are explained in the following table:

Property	Type	Description
`url`	URL	URL to the image; *Must be HTTPS*.
`alt`	String	Accessible description of the image.
`value`	String	Action assigned to the image.

Buttons

Buttons are shown at the bottom of the card--in a single row if they fit, or stacked. Button text is always on a single line and will be trimmed if it is too long. If more buttons than can be supported by the card are included, they will not be shown.

Actions

The following table consists of properties, types and descriptions for actions:

WOW! eBook
www.wowebook.org

Property	Type	Description
`type`	String	Required field. One of `openURL` (opens the given URL), `imBack` (posts a message in the chat to the bot that sent the card), `call` (Skype or phone number), `showImage` (for images only, displays the image),or `signin` (sign in card only).
`title`	String	Text description that appears on the button.
`tap`	Action object	Value depending on the type of action. For `openURL` it is a URL, for `signin` it is the URL to the authentication flow, for `imBack` it is a user defined string (which may contain hidden metadata for the bot for, example, `<meta roomid='10'/>`, for `call` it can be `skype:skypeid` or `tel:telephone`, and for `showImage` it is not required.

Sign in

The sign in card can be used to initiate an authentication flow with predefined images and title:

WOW! eBook
www.wowebook.org

Skype Preview

MoviesBot

Online

movies 19:32

Welcome to **Movies bot.** 19:32

19:32

Connect

19:32

Type a message here

WOW! eBook
www.wowebook.org

The following table illustrates the properties, types and descriptions of sign in:

Property	Type	Description
`text`	Rich text	Text appears just below the subtitle: two lines maximum.
`buttons: []`	Array of action objects	Single button of type `signin`.

Receipt

The receipt card can be used to send a receipt. If the height of the card is too large, it is partially folded and a **`Show all`** action is shown to expand it:

WOW! eBook
www.wowebook.org

The following table explains the properties, types and descriptions of receipt card:

Property	Type	Description
`title`	Rich text	Title of the card. Maximum two lines.
`facts:[]`	Array of fact key-value pairs	Fact key is left aligned, value is right aligned.
`items:[]`	Array of purchased objects	Properties: title (maximum two lines), subtitle (one line), text (up to six lines depending if the title, subtitle, and price are present), price, image (1:1 aspect ratio), tap.
`total`	Rich text	Title of the card. Maximum two lines.
`tax`	Rich text	Title of the card. Maximum two lines.
`vat`	Rich text	Title of the card. Maximum two lines.
`items:[]`	Rich text	Title of the card. Maximum two lines.
`images:[]`	Array of images	Image displayed at top of the card. Aspect ratio 16:9.
`buttons:[]`	Array of action	Set of actions applicable to the current card.

WOW! eBook
www.wowebook.org

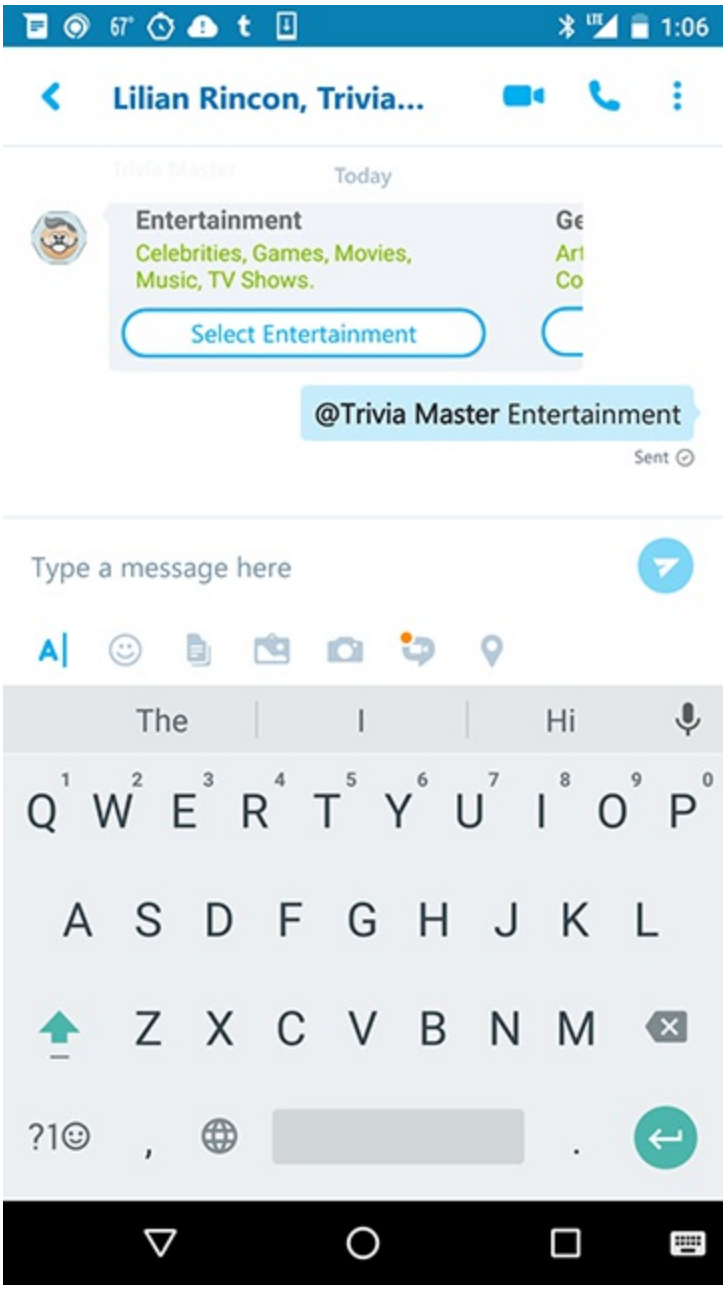
	objects	
`tap`	Action object	This action will be activated when the user taps on the card itself.

WOW! eBook
www.wowebook.org

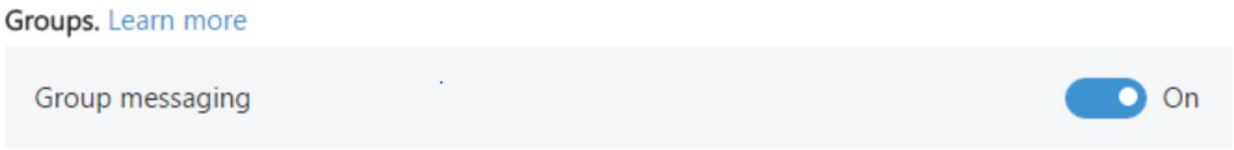
Groups

A bot can be enabled for groups in the Skype settings for the bot. It can be added to a group chat in the same way as adding a participant to a chat. In a group, the bot will only receive messages directly addressed to it--for example, `@YourBot This is the message`. It will not receive other messages sent by group participants or notifications of users joining or leaving the group:

WOW! eBook
www.wowebook.org



To enable a bot to be added to a group chat, you need to add this capability in **`Settings`**. Go to your bot **`Dashboard`** and edit the Skype channel:



Calling

You can build Skype bots that can receive and handle voice calls using the .NET SDK, Node.js SDK, or the Skype API.

Each time a Skype user places a call to your bot, the Skype bot platform will notify the bot using the calling **WebHook** you specify in `Settings`. In response, the bot can provide a set of basic actions called a workflow.

These are the supported actions:

- Answer
- Play prompt
- Record audio
- Speech to text
- DTMF tones
- Hang up

The Skype bot platform will execute the actions on the bot's behalf according to the workflow.

If the workflow is successful, Skype will post a result of the last action to your calling WebHook. For example, if the last action was to record an audio message, the result will be audio content.

During a voice call, your bot can decide, after each result, how to continue interaction with the Skype user.

Note

Skype bots with calling enabled are for preview only and cannot currently be published. To publish a bot in Skype, you will need to disable calling in the Skype settings for your bot and then set `Published` to `enable` in the bot dashboard. Bots can handle one-

WOW! eBook
www.wowebook.org

to-one calls, but not group calls.

WOW! eBook
www.wowebook.org

Summary

In this chapter, we introduced the Microsoft Bot Framework and explained how it helps in the development of bots. Also, we have seen how to set up the development environment, emulator, and the tools needed for programming. This chapter is based on the thought that programming knowledge and experience grow best when they grow together. In the next chapter, we will introduce bot programming by building and locally deploying a simple Hello World bot application. You will get to know Visual Studio, C# .NET, the Bot Framework, and the related technologies along with all the steps for creating projects.

WOW! eBook
www.wowebook.org

Chapter 2. Developing Your First Bot Using the Connector and Builder SDK

In this chapter, we will use the Microsoft Bot Connector, part of the Microsoft Bot Framework, as a way to create a single backend and then publish it to a bunch of different platforms called channels as quickly as possible. The goal is to have the user input natural language and your bot to perfectly understand and execute the action your user wants.

As we saw in Chapter 1, *Setting up the Microsoft Bot Framework Dev Environment*, two decades ago, we saw a major shift in the technology industry and consumers as well, where they moved from desktop client applications to Internet web applications. We are on the edge of a similar shift with mobile devices, with even bigger consequences--given the fact that no one can live without their mobile devices. **Conversation as a Platform** (**CaaP**) has become the new platform, incorporating the role played by the mobile apps. Just as Internet websites replaced desktop client applications in the 1990s, messaging bots will replace mobile apps now. Bots are the new apps, and the bot store is the new app store. Also, as we move...

WOW! eBook
www.wowebook.org

Bots are evolving

A bot is a piece of software designed to automate a specific task. When talked about in the context of conversation as a platform, a bot becomes the chat interface of a regular app. So, you should allow tasks that require full UI to be performed by the user only through conversation. We are at the early stages of a major evolving technology trend: the rise of conversation bots. Conversation bots can read and write messages just like a human would. Users will be able to interact with bots just as they interact with other humans using natural language.

Skype, WeChat, Kik, Facebook, GroupMe, Slack, Telegram, and so on are emerging conversation platforms, which help us in enabling interactions with any service from within the conversation platform. All these platforms enable developers to build conversation bots to provide a specific service. We can program bots to carry out specific automated actions. Conversation bots can initiate a definite action, and bots can respond to requests from other users as well as automate conversations and help complete transactions or implement workflows in the conversation.

WOW! eBook
www.wowebook.org

Bots use cases

Let's look at some of the use cases where bots can enable users to have natural conversation to meet their needs:

- We can develop some e-commerce bots that enable us to buy goods and services through Skype, Slack, Facebook, and any other conversation platforms
- We can develop bots for restaurants to order food online or to make reservations at a restaurant
- We can develop content bots that share relevant content with you (such as news and weather)
- We can develop watcher/tracker bots that can notify us when specific events happen (such as when a flight is delayed, when your car needs servicing, and when your pizza order is ready for pickup)
- We can develop banking and stock trading bots that can provide financial services
- We can develop workflow bots that can automate business workflows in marketing, sales, operations, HR/admin, finance, payroll, and others
- When it comes to the IoT domain, bots are the best fit for IoT applications, which can connect to your smart homes, sensors, cars, and more

When you have so many bots, it makes sense to let your personal digital assistant (such as Cortana) manage the communication with the...

WOW! eBook
www.wowebook.org

Developing your first bot

The Microsoft Bot Framework allows developers to develop code once and, using the Microsoft Bot Connector, deploy it onto multiple channels, including **`SMS`**, **`Slack`**, **`Facebook Messenger`**, **`Skype`**, **`GroupMe`**, and many other channels:

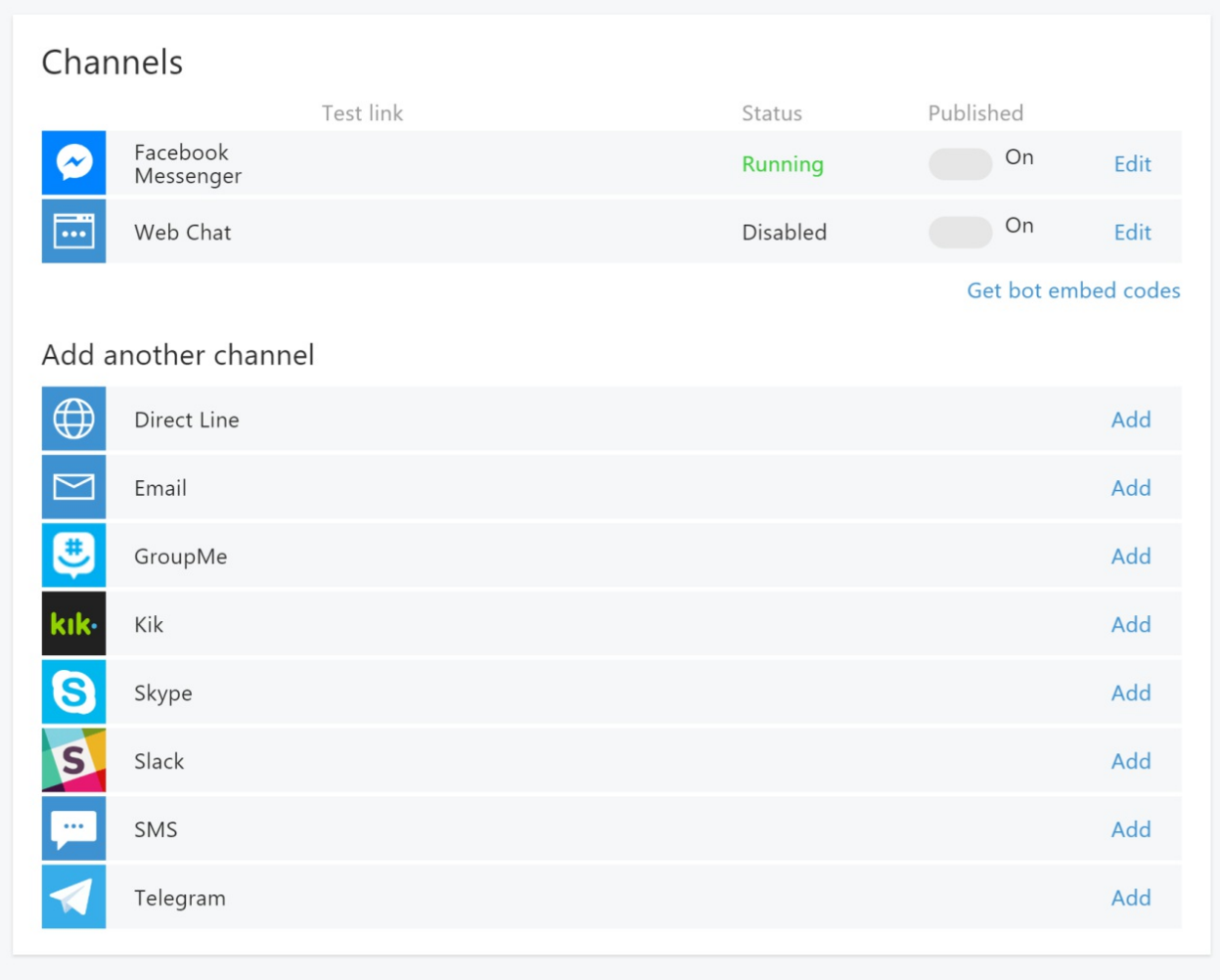


Figure 1: Various channels available in the Microsoft Bot Framework

The Microsoft Bot Framework has three main components:

- **Bot Connector**: This allows you to easily connect your bot to Slack or

WOW! eBook
www.wowebook.org

Skype, via SMS or the web

- **Bot Builder SDK**: An SDK that allows you to develop bots using C# .NET or Node.js, which is open source
- **Bot Directory**: A collection of all approved bots connected through the Bot Connector; it is a marketplace where users can search for bots to add in their chat applications

Note

It's really important for Visual Studio to be updated in order to use the Bot Directory, as well as download the web tools in the Visual Studio setup. Update all VS extensions to their latest versions by navigating to `Tools`|`ExtensionsandUpdates`|`Updates`.

In this chapter, to get started with developing bots using the Microsoft Bot Framework, we will use the following:

- The Bot Connector
-

We will build our...

WOW! eBook
www.wowebook.org

Creating our first bot

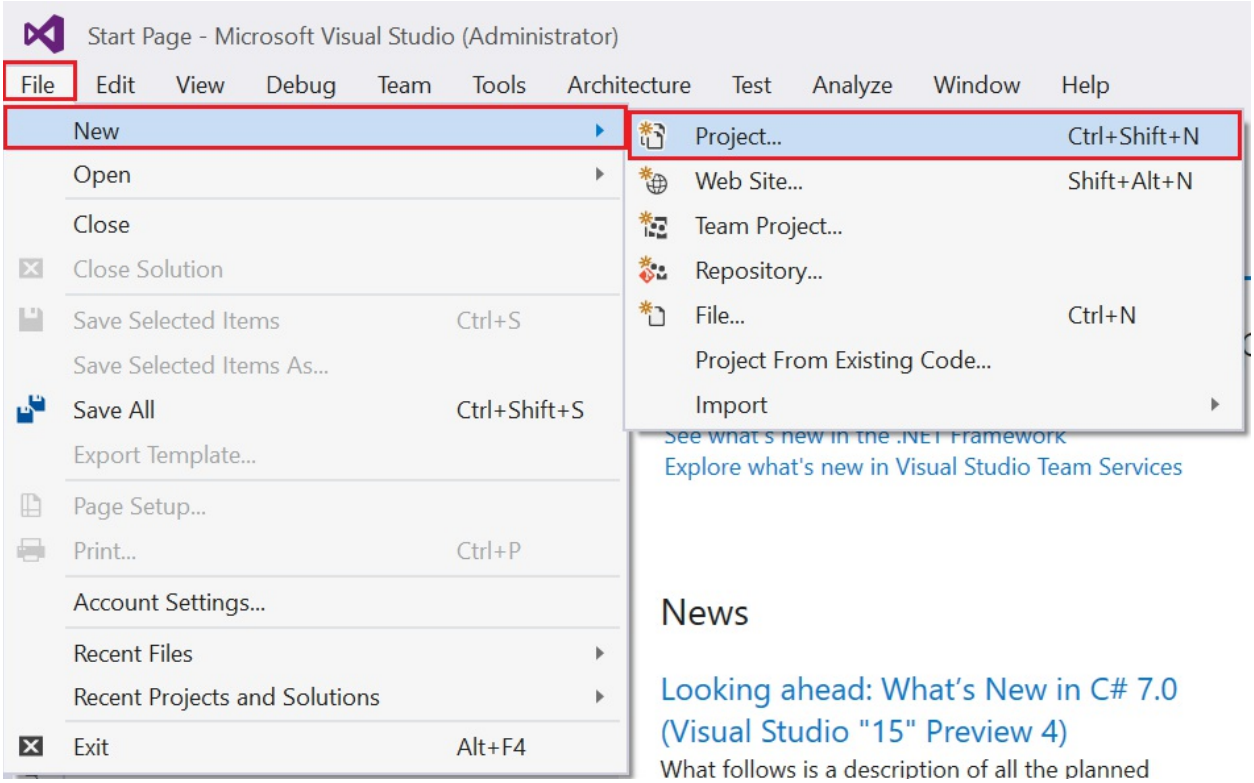
To develop a bot, perform the following steps:

1. Build a bot using the C# SDK.
2. Test it using the Bot Framework emulator.

Building a bot using the C# SDK

Let's go through the steps to create a bot application using the Bot Framework Connector SDK .NET template. They are as follows:

1. Open Visual Studio and navigate to **`File`** | **`New`** | **`Project...`** and select **`Visual C#`** from the left side **`Templates`** category. Then, from the **`Templates`** section, you will see the **`Bot Application`** template:



WOW! eBook
www.wowebook.org

Figure 2: Creating a new project in Visual Studio IDE

2. Select the **`Bot Application`** template, name the project `Hello World`, and then click on **`OK`**:

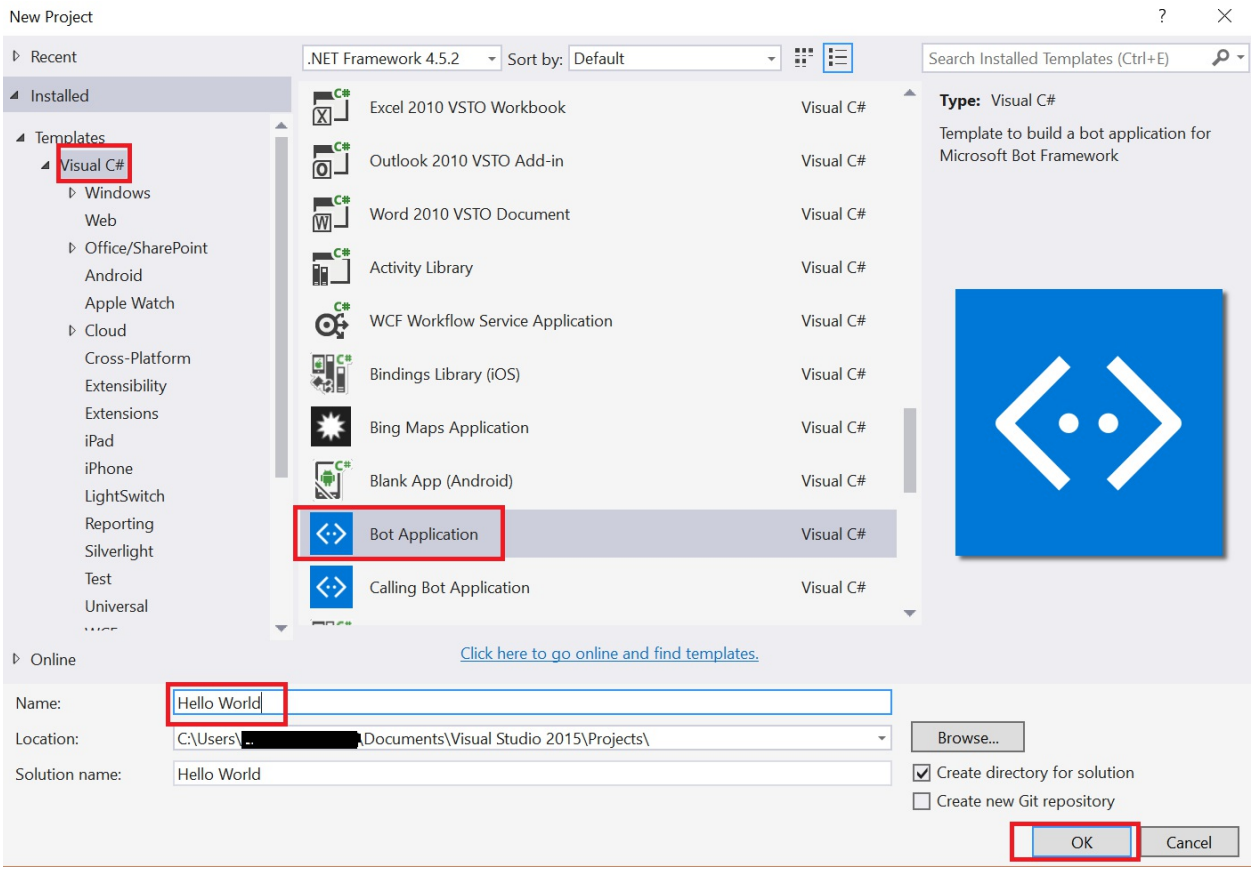


Figure 3: Selecting the Bot Application template and naming the project

A solution gets created with the `Hello World` project.

Let's go through the default files that were created by this **`Bot Application`** template in Visual Studio.

When we create a bot using the **`Bot Application`** template, it basically creates an ASP.NET Web API project, which contains all the Bot Framework SDKs and all the supporting files. The following are the files, by default:

- `Properties\AssemblyInfo.cs`
- `References`

WOW! eBook
www.wowebook.org

- App_Start\WebApiConfig.cs
-

WOW! eBook
www.wowebook.org

Post method

The `Post` method accepts messages from the user as an activity, which contains all conversation information between the user and our bot. Using this, we can ascertain what kind of information the user wants from the bot:

```
public async Task<HttpResponseMessage> Post([FromBody]Activity
```

Here, we defined a sample bot that will reply back to the user with the same as what you say to it.

The Bot Framework provides many features that include how to identify the type of incoming message and based on that, your bot can respond to the user. To identify that, we have activity enum types, which will provide information about the conversation.

To identify and apply business logic to the message sent by the user, we will write the following code in the `Post` method:

```
if (activity.Type == ActivityTypes.Message)
            {
}
```

If the user is sending a message, it means they are requesting something to the bot. So, it will receive it, process it, apply some business logic, and will reply back to the user. To reply back to the user, we need a `ConnectorClient` object, which provides connector REST API services to forward messages...

WOW! eBook
www.wowebook.org

How to deploy and run the bot application in the Bot Framework emulator locally

---

To test and debug the bot application locally, we have the Bot Framework emulator, which will provide all the rich functionalities of the Bot Framework SDK.

Download the emulator and install it from https://emulator.botframework.com/. Now, go to Visual Studio and press *F5* to run and deploy the Hello World bot application locally in your browser. You will see the welcome page `Default.htm` of your bot as shown here:

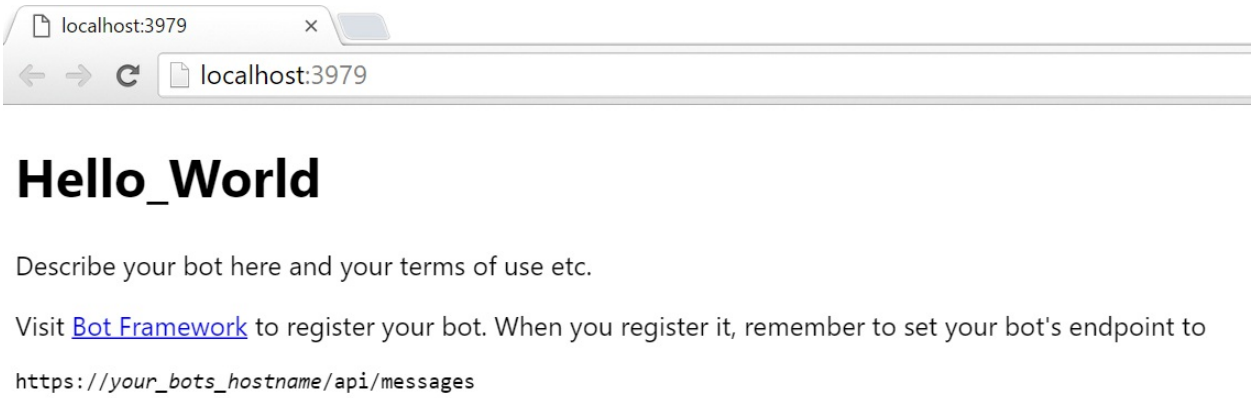


Figure 9: Your bot default page in the browser

Now, open the bot emulator that you installed in the first step. By default, the emulator sets the bot URL to localhost. Make sure that the bot application localhost port and the URL port in the emulator are the same. To check that, go to the browser where your bot application is running and open, and check the port number after the localhost word in that URL:

WOW! eBook
www.wowebook.org

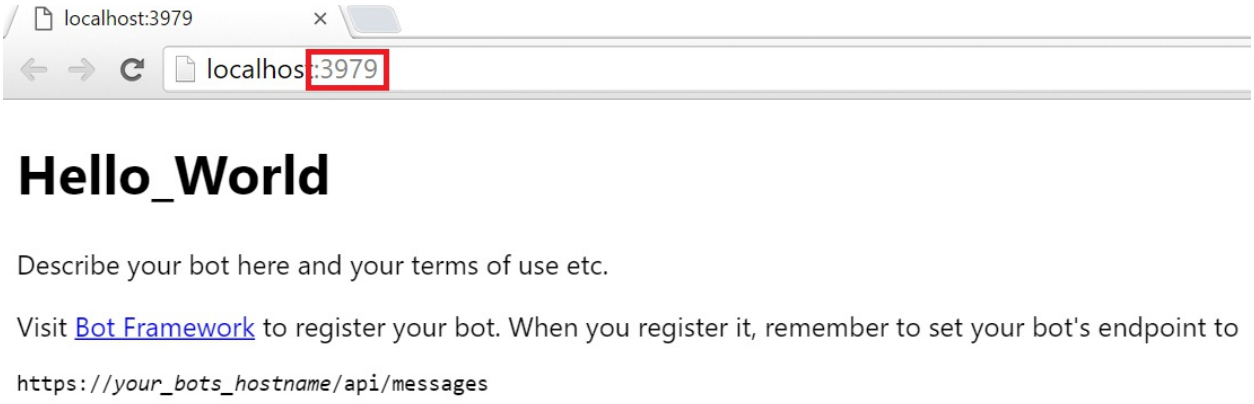


Figure 10: Your bot port number

For example, in the Hello World sample, the port number is `3979`. Now, go to the bot emulator and check whether the bot URL has the same port number or not. We need to add the path `/api/messages` to the bot URL...

WOW! eBook
www.wowebook.org

How to use dialogs in bot applications

Now, we will see how to use dialogs for the same Hello World bot application.

Dialogs will be used in a conversational process, where there is an interaction or exchange of messages between the user and the bot. Each dialog is an abstraction that encapsulates its own state in a C# class that implements `IDialog`. To work with dialogs, we need to import the `Microsoft.Bot.Builder.Dialogs` namespace. Add a C# class `HelloWorldDialog` into your project. To add a class, right-click on your project and navigate to **`Add`** | **`Class...`** from the menu:

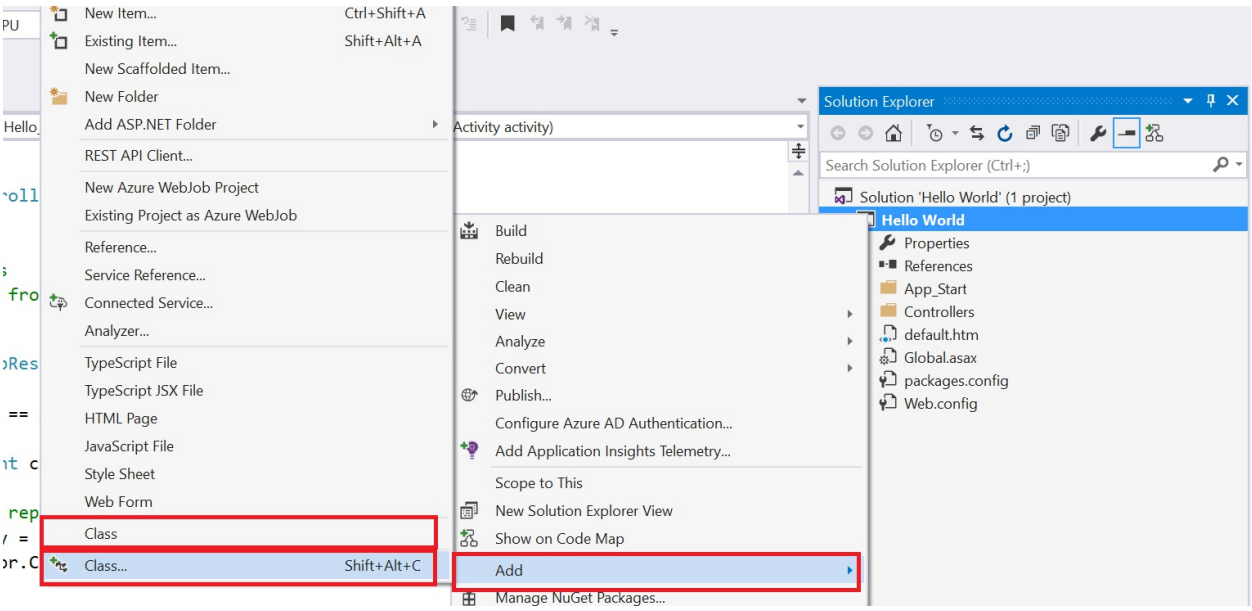


Figure 16: Using Visual Studio IDE to add a new class to an existing project

Give the **`Name`** as `HelloWorldDialog`:

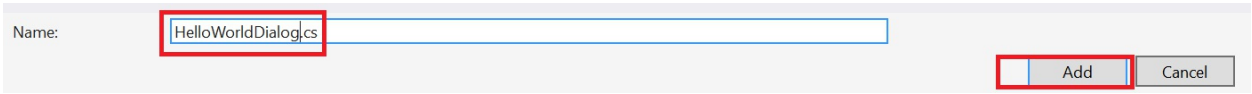


Figure 17: Using Visual Studio IDE to name a new class

WOW! eBook
www.wowebook.org

In order to change the Hello World example, add the following code in `HelloWorldDialog`. To use the Bot Builder, we first need to import the required namespace:

```
using Microsoft.Bot.Builder.Dialogs;
```

Next, we need to add a C# class to represent our conversation. You can do this by adding this class to your `HelloWorldDialog.cs` file. Replace the code in the `HelloWorldDialog.cs` file with the following code:

WOW! eBook
www.wowebook.org

How to use FormFlow in the bot application

The main purpose of FormFlow is to provide more simplified, guided conversations. This gives more flexibility and avoids ambiguity in the conversation. It has helped to review the progress so far. It has limitations compared to dialogs, but in a way that requires less effort. With the combination of dialogs and LUIS dialogs, we can get the best of both worlds.

Dialogs can be are very powerful and flexible, but it can take lot of efforts in handling a guided conversation, like ordering a pizza. At any point in dialog one can contemplate various possibilities of what's next. You may be required to provide the clarification about an ambiguity, help options, go back, or display the progress.

In order to ease out the process of building the guided conversations, the framework comes with the powerful dialog building block known as **FormFlow**. Some of the flexibilities provided by dialogs is sacrificed by the FormFlow, but that is done to ease out the efforts. A combination of the FormFlow dialogs and other kinds of dialogs would prove beneficial. For example, A combination of FormFlow and LUIS...

WOW! eBook
www.wowebook.org

Summary

In this chapter, we discussed how to build a bot by locally deploying a simple Hello World bot application. You also learned about Visual Studio, C# .NET, the Bot Framework, and the related technologies along with all the steps for creating projects. We also discussed the bot emulator and how it relates to bot development. In the next chapter, we will see how to develop a fully functional weather bot. Also, we will learn about how the bot communicates about the current weather in a given city. You will learn how the bot interacts with Skype or any other channel to know the current weather.

WOW! eBook
www.wowebook.org

Chapter 3. Developing WeatherBot Using Dialogs and LUIS

In previous chapters, we have gone through some of the concepts involved in developing and publishing bot applications. In this chapter, we will develop a bot called WeatherBot, show you how to use LUIS in dialogs and how to use third-party APIs from a bot. This involves some additional coding efforts to develop the weather bot. We will build a weather bot that is able to understand and respond to various commands, such as *What's the weather like in New York?*, *Get Weather in Seattle*, and so on. The bot will use LUIS to identify the intent of the user and reply with the appropriate message.

Before jumping into writing code, we need to configure LUIS for WeatherBot. Here, we will go through the steps on how the user gets weather data for a given location when requested by the user. We will see how LUIS can help us make the conversation between the user and your bot in more natural language, similar to how we interact with humans.

The WeatherBot will have intelligence, which will help users to interact with it, similar to how we interact with humans (in natural language). The...

WOW! eBook
www.wowebook.org

Language Understanding Intelligent Service (LUIS)

Language Understanding Intelligent Service (**LUIS**) is one of the services in **Microsoft Cognitive Services** provided by Microsoft. As mentioned earlier, natural language is a fundamental element in developing bot applications. As a result, the technology industry has seen a direct correlation between the evolution of bot platforms and NLP platforms. Although the evolution of bot technologies has been predominantly driven by messaging platform providers such as Slack or Facebook, the main advancements in NLP technologies seem to be coming from cloud platform providers such as Microsoft. As a result, to take advantage of the NLP and **Natural Language Understanding** (**NLU**) algorithms, most bot developers spend time integrating their bot applications with NLP services provided by platforms such as LUIS from Microsoft. LUIS can process natural language using pre-built or custom-trained language models.

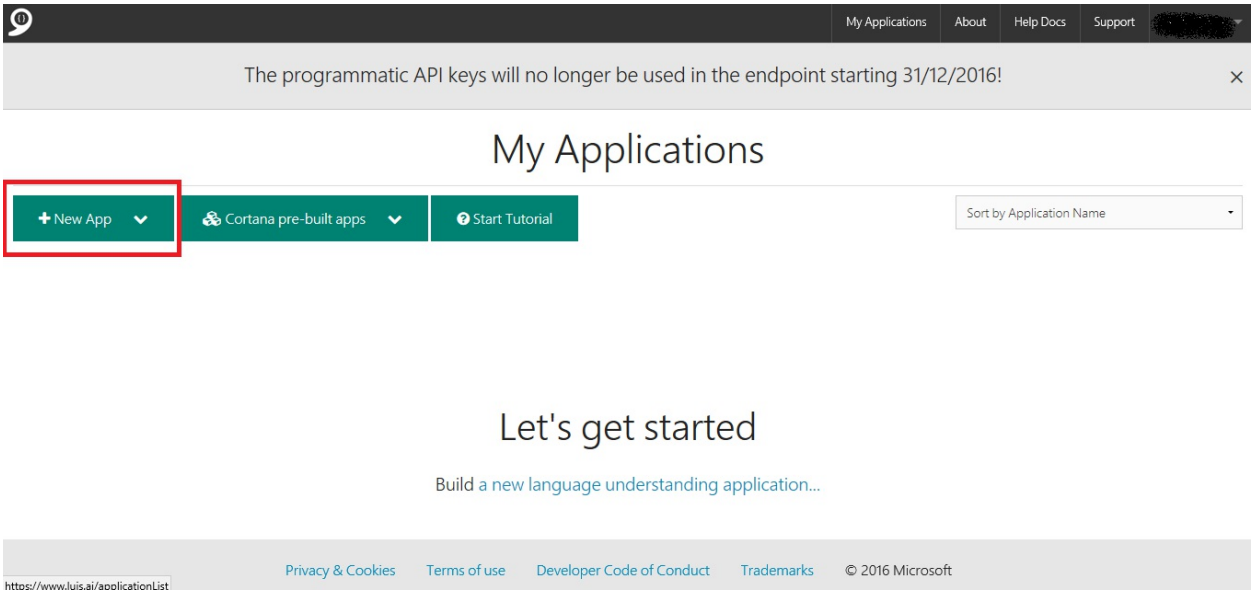
Microsoft's LUIS is a component of the Microsoft Cognitive Services Suite that helps in creating and processing natural language models. LUIS provides a sophisticated toolset that allows...

WOW! eBook
www.wowebook.org

Intents and Entities

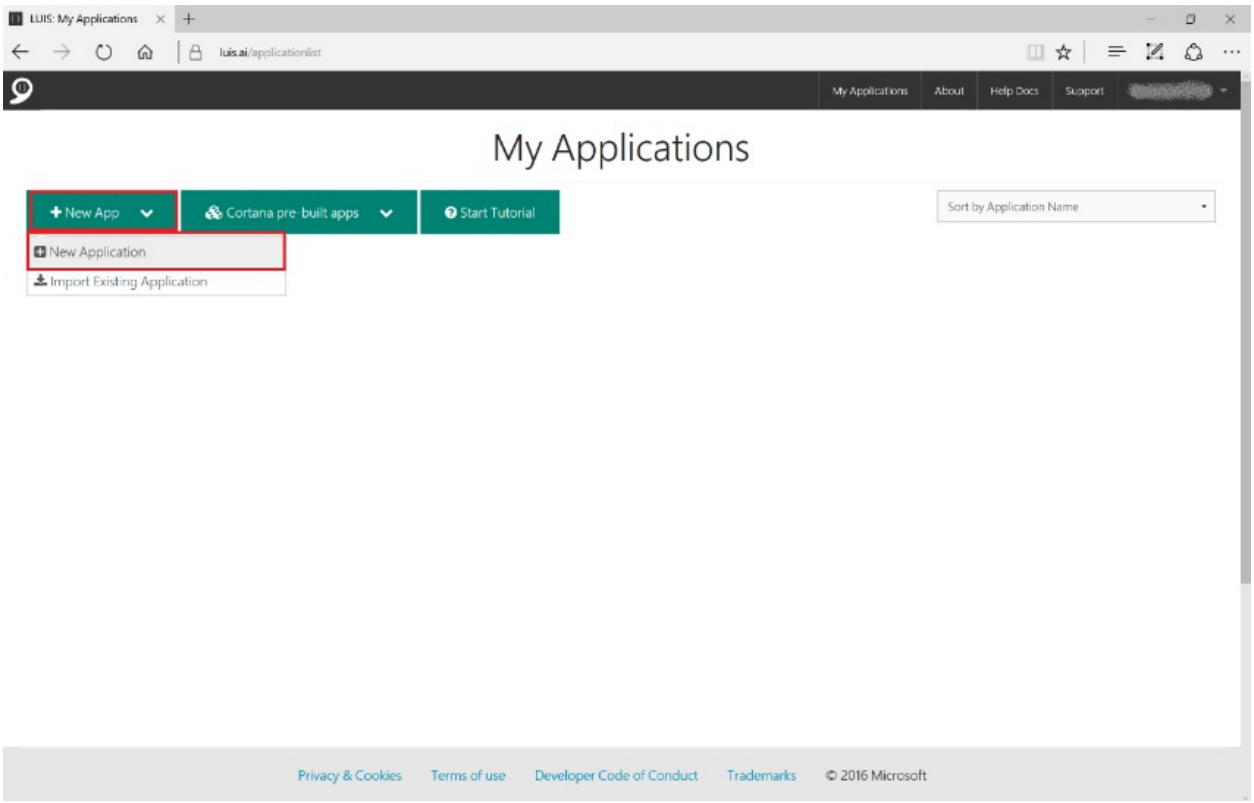
When a user enters a sentence, LUIS will interpret it and parse out the Intent and Entities. An **Intent** is an action the user wants to perform, and **Entities** are the subjects for the Intent. For example, if someone asks, *Hi, what is the current weather in Seattle?*, the Intent could be knowing weather and the entity is Seattle. Based on the complexity and requirement, you can define multiple Intents in LUIS and perform respective actions. Now, let's create an app for your WeatherBot:

1. Click on **`New App`**:



2. On the **`New App`** dropdown, select **`New Application`**:

WOW! eBook
www.wowebook.org



3. On the **`Add a new application`** popup, enter the application name, the application usage scenario as **`Bot`**, and select the category as **`Weather`**:

WOW! eBook
www.wowebook.org

Add a new application

Enter application name

WeatherBot

Enter application usage scenario

Bot

Choose application domain(s)

- ☐ Booking & Reference
- ☐ Business
- ☐ Comics
- ☐ Communication
- ☐ Education
- ☐ Entertainment
- ☐ Finance
- ☐ Gaming
- ☐ Health & Fitness
- ☐ Home Automation
- ☐ Media & Video
- ☐ Medical
- ☐ Music & Audio
- ☐ Navigation & Maps
- ☐ News & Magazines
- ☐ Personalization
- ☐ Productivity
- ☐ Real Estate
- ☐ Scheduler
- ☐ Shopping
- ☐ Social Network
- ☐ Sports
- ☐ Telecom
- ☐ Tools
- ☐ Transportation
- ☐ Translation
- ☐ Travel & Local
- ☑ Weather
- ☐ Others

Enter application description (optional)

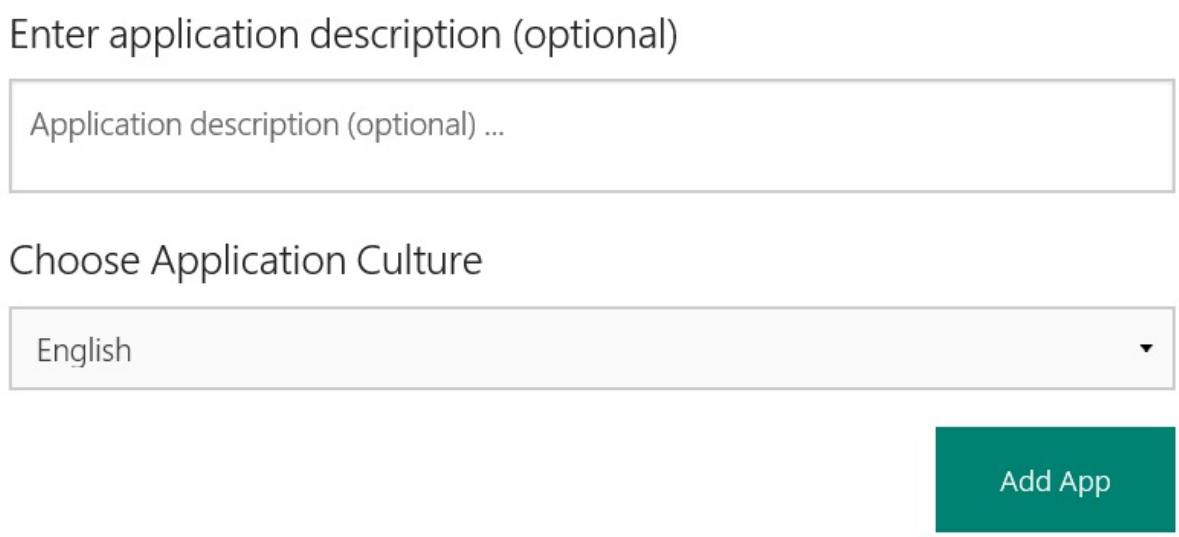
Weather Bot

Choose Application Culture

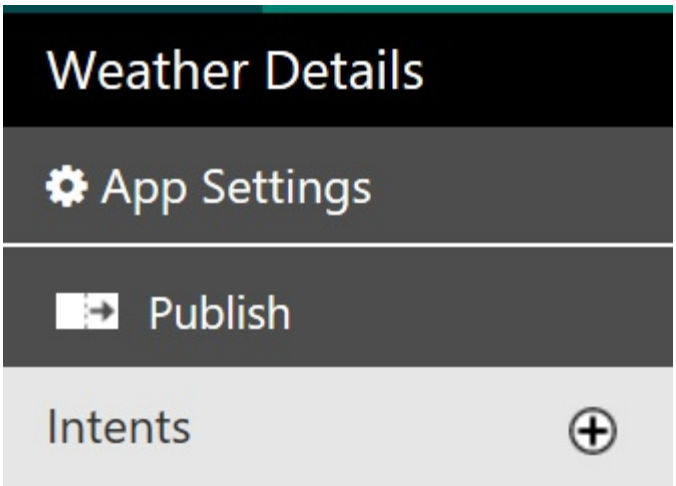
English

WOW! eBook
www.wowebook.org

4. Finally, click on the **`Add App`** button:



5. After the successful creation of the app, open it and click on the + icon of the **`Intents`** section from the left-hand side menu:

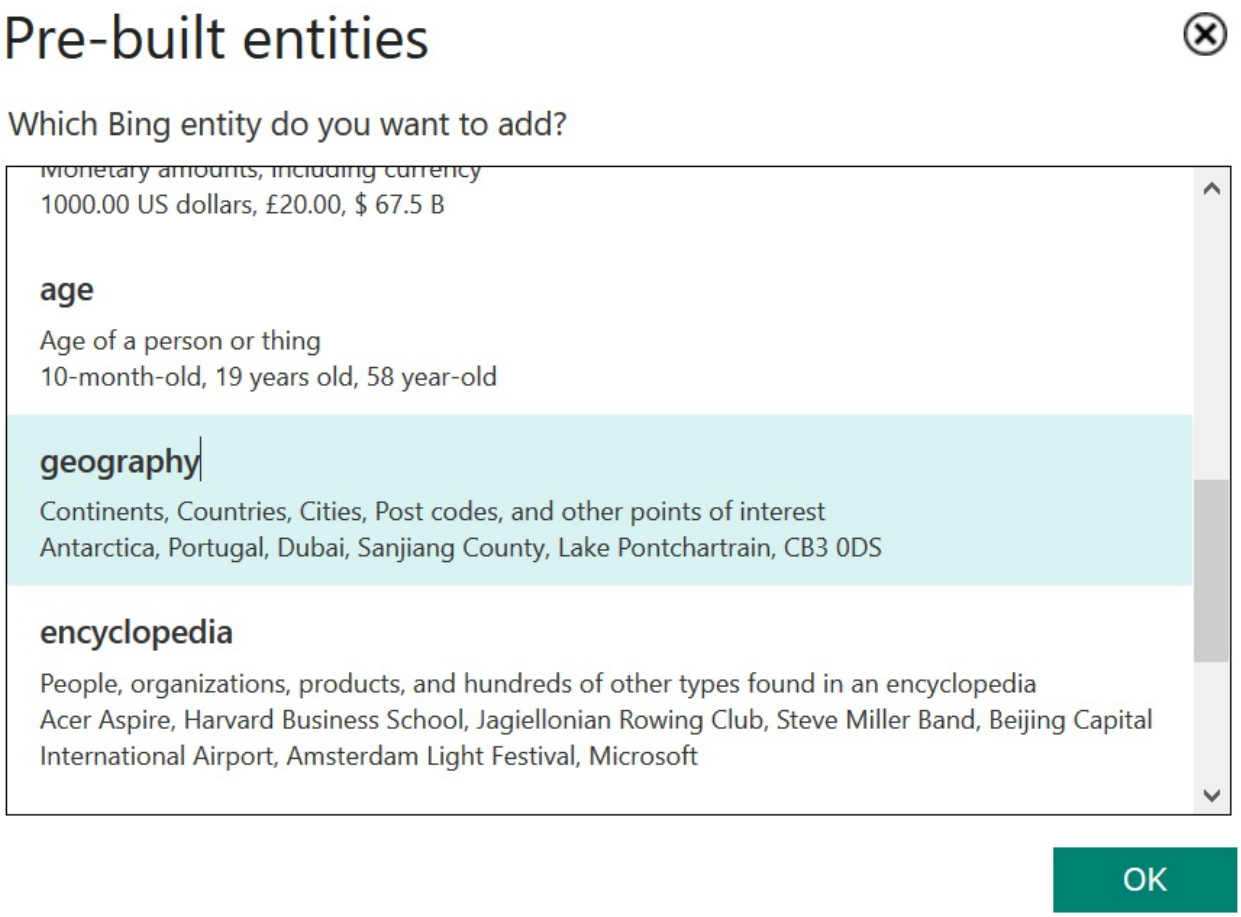


6. Enter the name for your Intent and click on the **`Save`** button:

WOW! eBook
www.wowebook.org

7. Now, add an Entity. From the left-hand side menu, click on the **+** icon of **`Pre-built Entities`**:

8. Select **`geography`** as the Entity:



WOW! eBook
www.wowebook.org

The reason why we use a pre-built entity is that LUIS already contains geography that has...

WOW! eBook
www.wowebook.org

Training your bot using utterances

Now you have to train your app using utterances to get the appropriate results from LUIS. An **utterance** is nothing but the sentence typed/asked by the user of your bot, such as *What is the current weather in Ashburn, Virginia?*. You have to enter as many utterances as possible with your bot.

Some examples of utterances include the following:

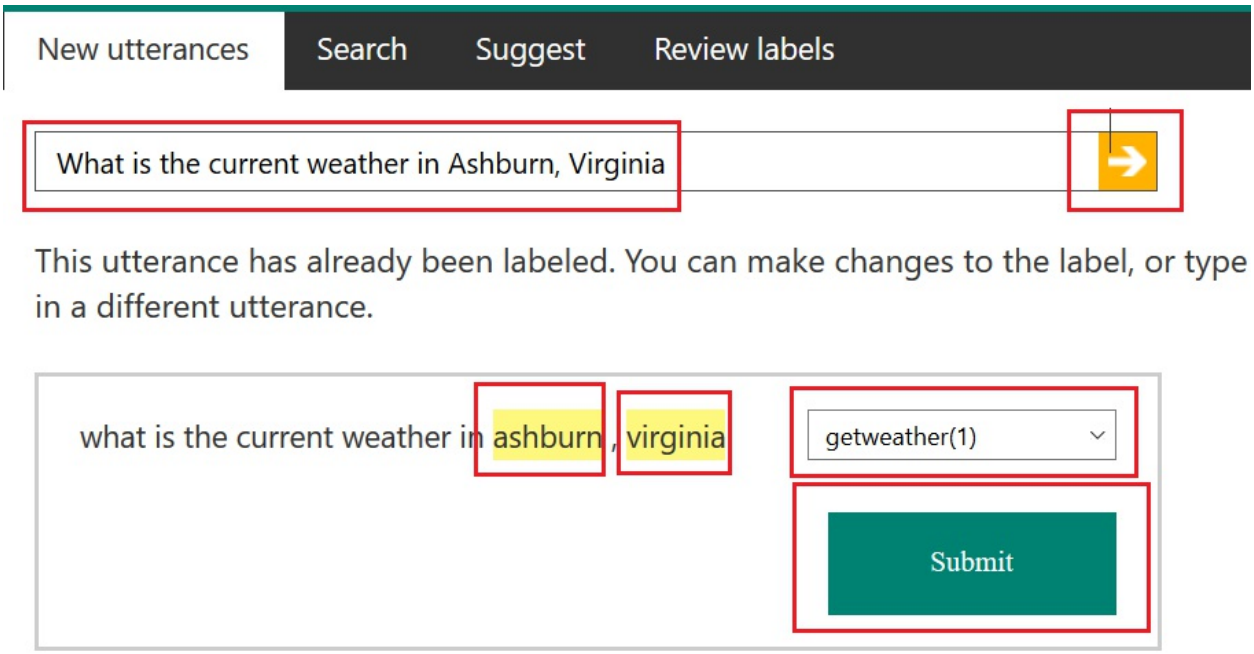
- What is the current weather in Ashburn, Virginia?
- Get weather in Boston
- Get weather in Miami

To train your app, you have to add different types of utterances in LUIS. Let's perform the following steps to add new utterances:

1. Select the `New utterances` section and then add the new utterance in the textbox:

2. After entering the utterance, press *Enter*. LUIS will automatically highlight the geography in your text, as shown in the following screenshot:

WOW! eBook
www.wowebook.org



3. Before clicking on **`Submit`**, make sure that the sentence is identified correctly and if it shows the Intent as **`getweather`** or not. If the sentence is correct and is asking about the weather, then manually select the **`getweather`** Intent from the drop-down menu. Before submitting, check whether **`geography`** is highlighted or not. If...

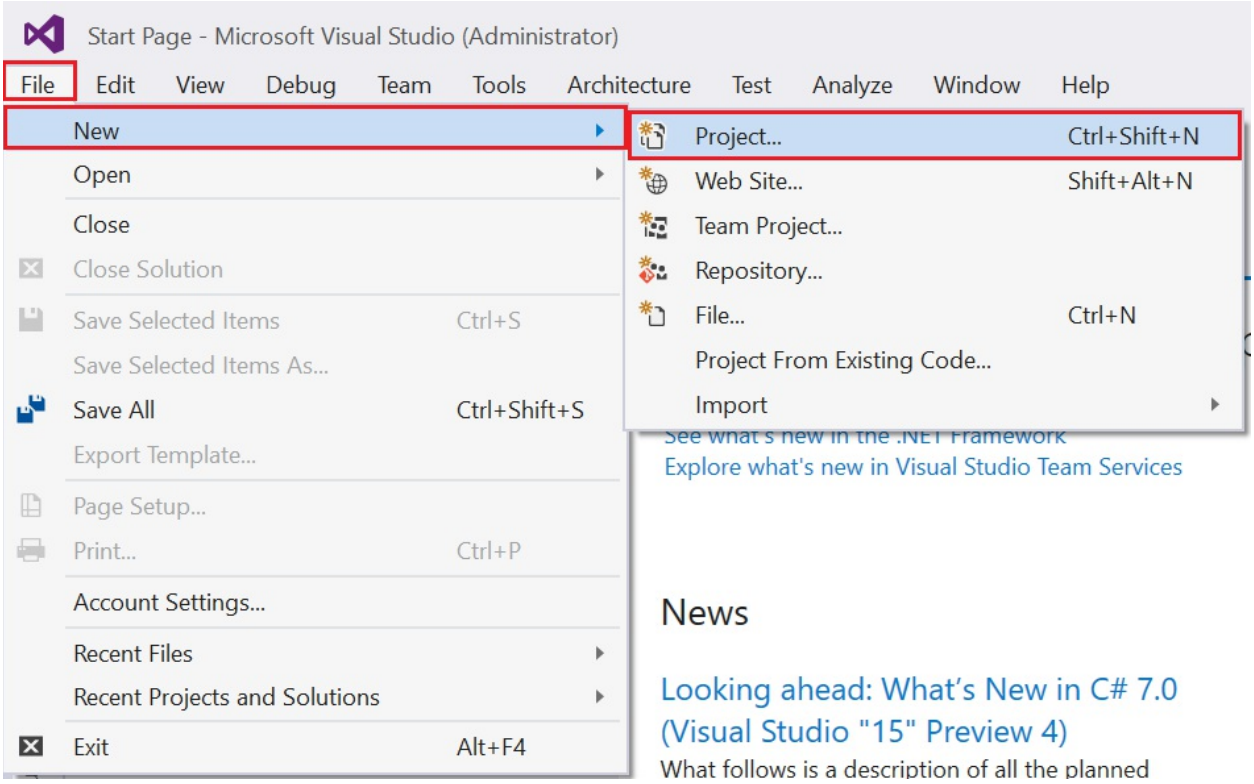
WOW! eBook
www.wowebook.org

Development of WeatherBot code

We have completed setting up LUIS. Now let's develop a bot for knowing the weather of a given geography. We will also see how LUIS can help us in identifying the geography of a given sentence.

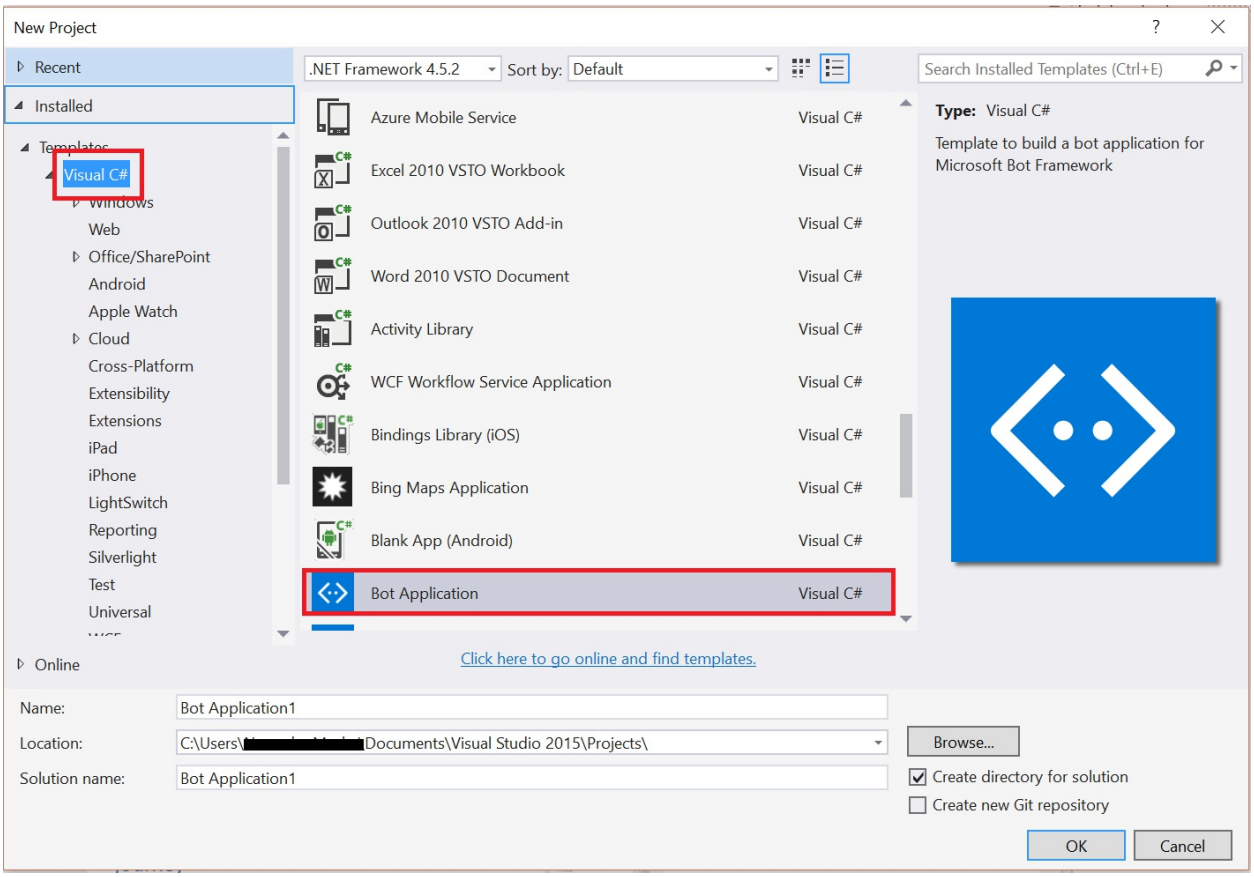
This guide is for C# using the Bot Framework Connector SDK .NET template:

1. Open Visual Studio and navigate to **`New`** | **`Project`**:



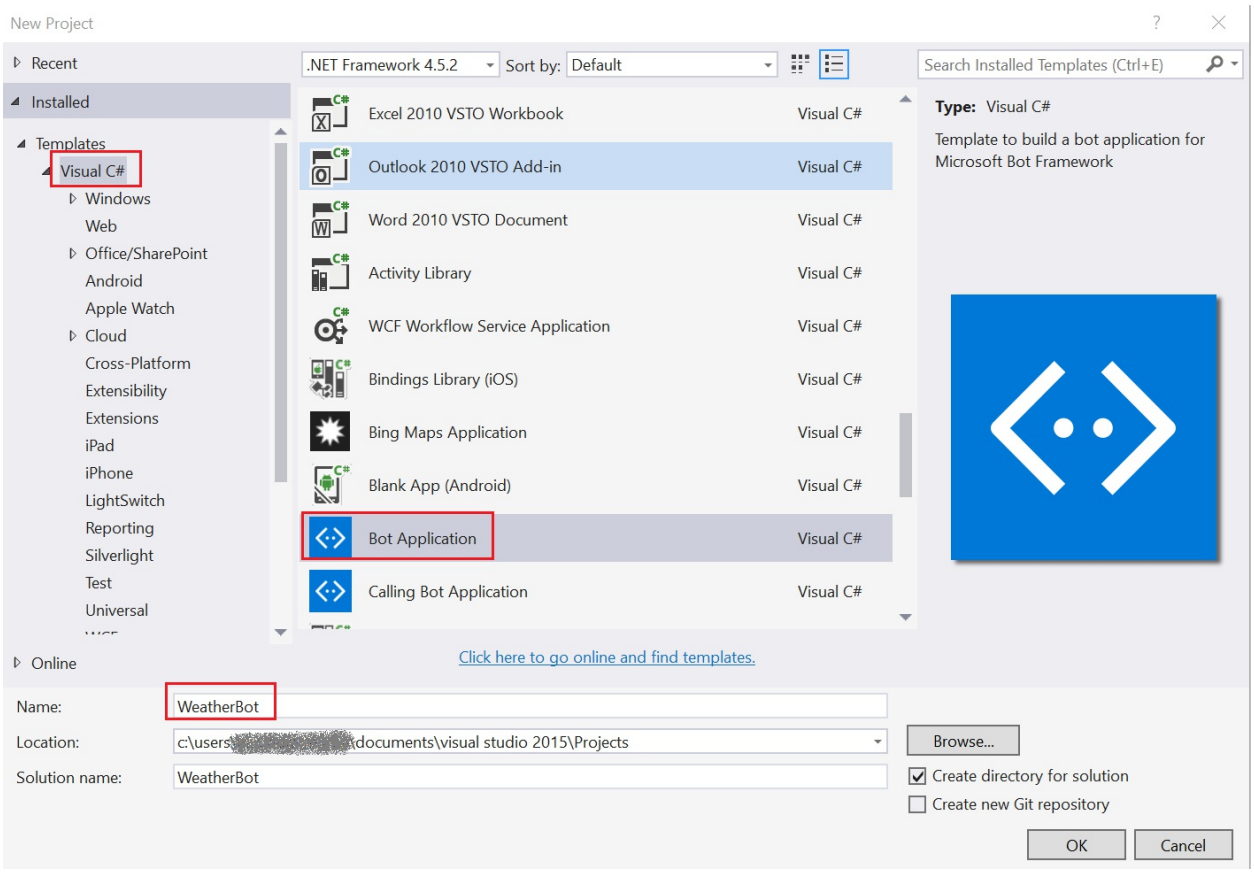
2. Select **`Visual C#`** from the left-hand side template category. From the templates section, you will see the **`Bot Application`** template:

WOW! eBook
www.wowebook.org



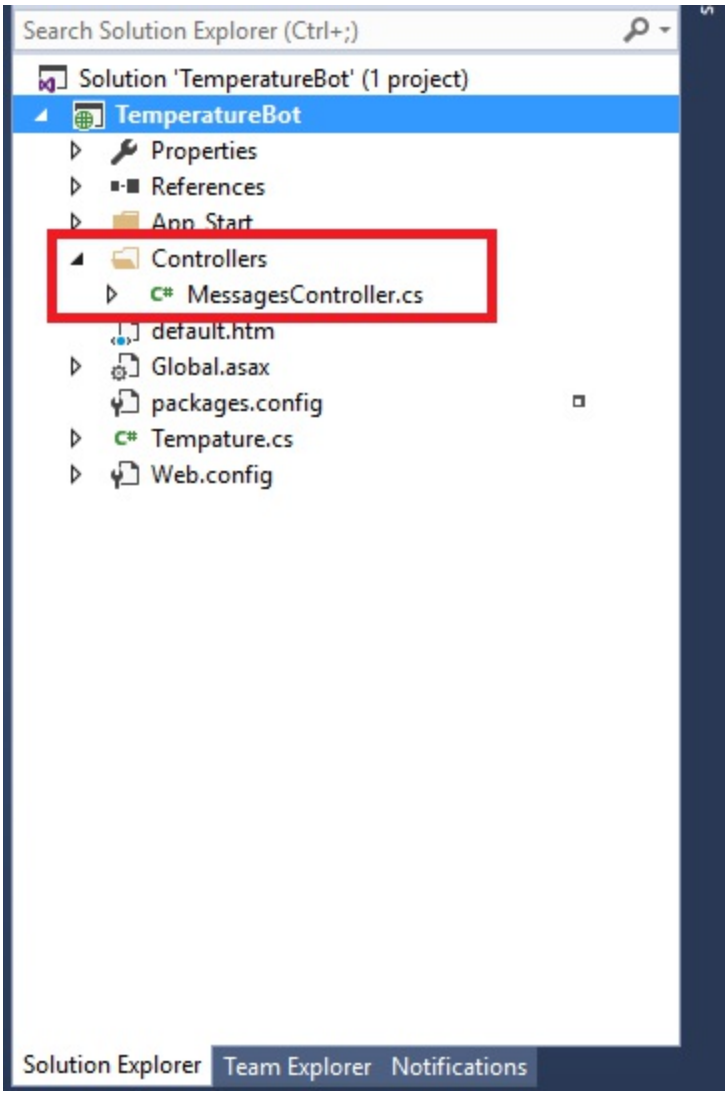
3. Select the **`Bot Application`** template, name the project `WeatherBot`, and then click on **`OK`**:

WOW! eBook
www.wowebook.org



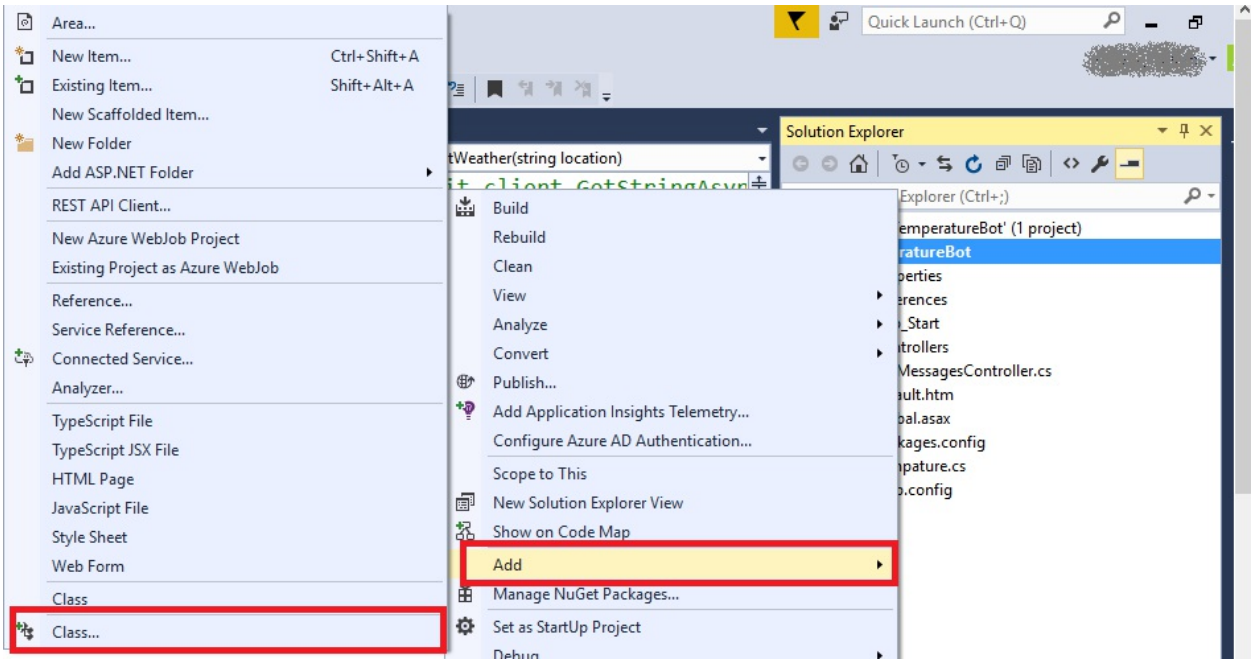
4. Select the `MessagesController.cs` file, which is located under the `Controllers` folder:

WOW! eBook
www.wowebook.org



5. Update the `Post` method to call the `Dialogs`. For that, add a class called `WeatherDialog.cs` in to your solution and extend it with `IDialog`. For that, you have to right-click on your project and select **`Add`** | **`Class...`**:

WOW! eBook
www.wowebook.org



6. Extend the class with `IDialog`, implement its interface method, `StartAsync()`, and also decorate the class with a `Serializable` annotation:

```
[Serializable]
1 reference | Kishoreismac, 17 hours ago | 1 author, 1 change
public class WeatherDialog: IDialog<object>
{
    0 references | Kishoreismac, 17 hours ago | 1 author, 1 change
    public async Task StartAsync(IDialogContext context)
    {
        context.Wait(MessageReceivedAsync);
    }
```

The core functionality of the bot template is all in the `Post` function within `Controllers\MessagesController.cs`.

In this case, the code takes the...

WOW! eBook
www.wowebook.org

Calling LUIS from the bot

To incorporate a call to LUIS, we can start by adding this function. It simply calls LUIS and returns the city, state, or country names if the message is a weather query mentioning state and country.

Create a method in the `WeatherDialog.cs` class as follows; we will call this method from the `MessageReceivedAsync` method by passing the sentence asked by the user to your bot:

```
private static async Task<string> IdentifyCityUsingLUIS(string
{
}
```

Now do a GET request to your LUIS app using the LUIS URL, which you saved in an earlier step, as follows:

```
var responseInString = await httpClient.GetStringAsync(@"REPLA
            + System.Uri.EscapeDataString(message));
               dynamic response = JObject.Parse(responseInStri
```

Once you get a response from LUIS, try to parse it and identify whether the sentence contains the required Intent and Entities. For that, write the following code:

```
var intent = response.intents?.First?.intent;
                string city="",state="",country="";
                if (intent == "getweather")
                {
                   ...
```

WOW! eBook
www.wowebook.org

Natural speech and Intent processing bot using Microsoft Cognitive Services

---

As LUIS is a new concept, we will go through one more project to understand the concepts better. This project is for understanding how to use **Cortana Intelligence Services** and **Rich Text Messaging** technology. This bot identifies the concepts and actions in the text that is sent to the bot with part-of-speech tagging, finds phrases and concepts using natural language parsers, and returns all the identified Intents, which are created and trained in the custom LUIS app. For example, if you say *Hi John, am going to New York tonight*, the bot will return part-of-speech tagging as well as parsing data for natural speech and Intent processing to know the name and location:

- Name: John
- Place: New York

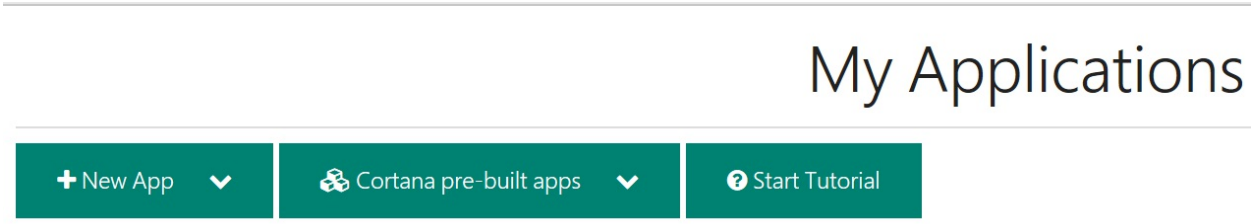
Whether you're mining customer feedback, interpreting user commands, or consuming web text, understanding the structure of the text is a critical first step, and this chapter teaches you that. Before starting the tutorial, you should know about Microsoft Cognitive Services (**Cortana Intelligence Services**), which helps you to build applications using very complex and...

WOW! eBook
www.wowebook.org

Identifying the name of a person, place, and company using LUIS

Now we will create a custom LUIS app to return all the identified Intents that are created and trained. For example, if you say *Hi John, am going to New York tonight*, the LUIS app will return natural speech and intent processing to know the name, location, and other things, such as John as name and New York as place:

1. After logging into LUIS in `luis.ai`, create an app for your `IntentProcessing`. To do that, click on **`New App`** and select **`New Application`**:



2. Enter the application's name, the usage as `Bot`, and select the category. Finally, click on the **`Add App`** button:

WOW! eBook
www.wowebook.org

Add a new application

Enter application name

Enter application usage scenario

Bot

Choose application domain(s)

- ☐ Booking & Reference
- ☐ Business
- ☐ Comics
- ☐ Communication
- ☐ Education
- ☐ Entertainment
- ☐ Finance
- ☐ Gaming
- ☐ Health & Fitness
- ☐ Home Automation
- ☐ Media & Video
- ☐ Medical
- ☐ Music & Audio
- ☐ Navigation & Maps
- ☐ News & Magazines
- ☐ Personalization
- ☐ Productivity
- ☐ Real Estate
- ☐ Scheduler
- ☐ Shopping
- ☐ Social Network
- ☐ Sports
- ☐ Telecom
- ☐ Tools
- ☐ Transportation
- ☐ Translation
- ☐ Travel & Local
- ☐ Weather
- ☐ Others

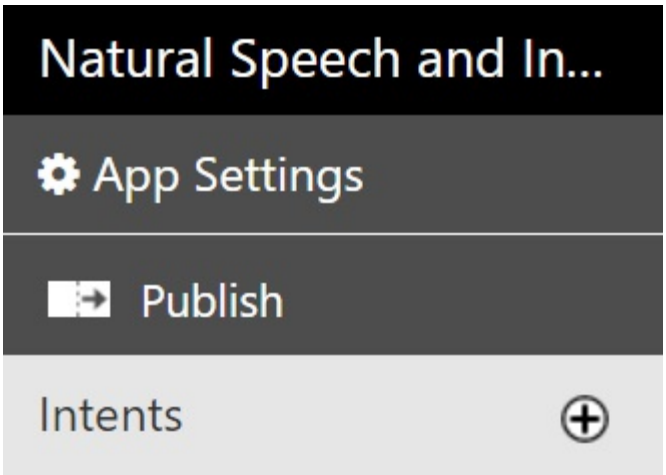
Enter application description (optional)

Choose Application Culture

English

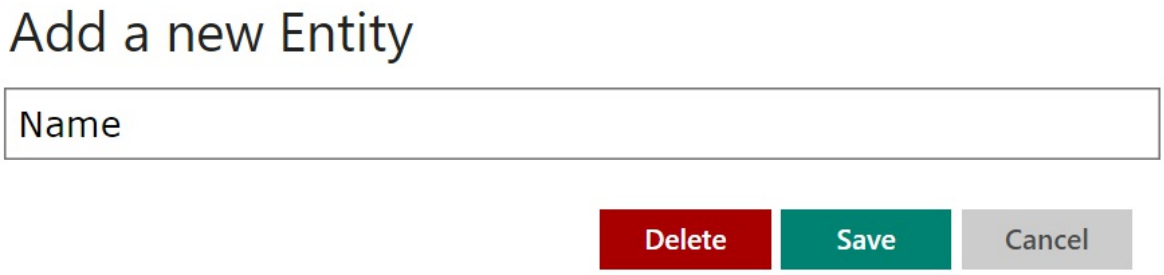
WOW! eBook
www.wowebook.org

3. After successful creation of the app, open it and click on the **+** icon of the **`Intents`** section from the left-hand side of the menu:



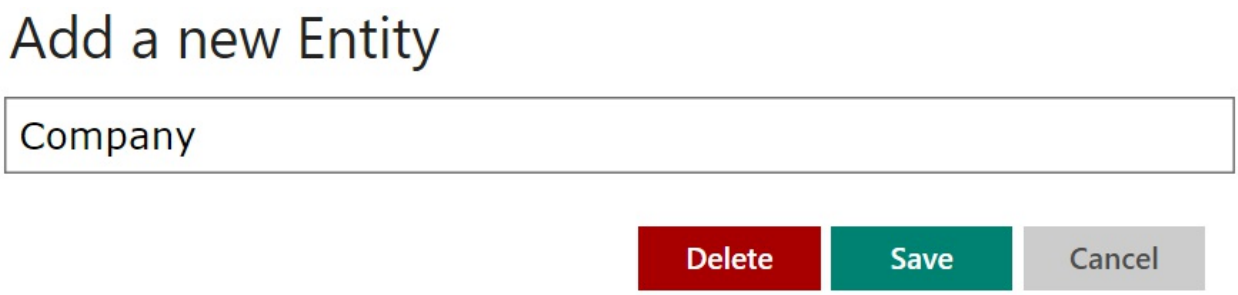
4. Enter the name for your Intent and click on the **`Save`** button:

5. Now add a custom entity from the left-hand side menu, click on the **+** icon of the Entity, and enter the name:



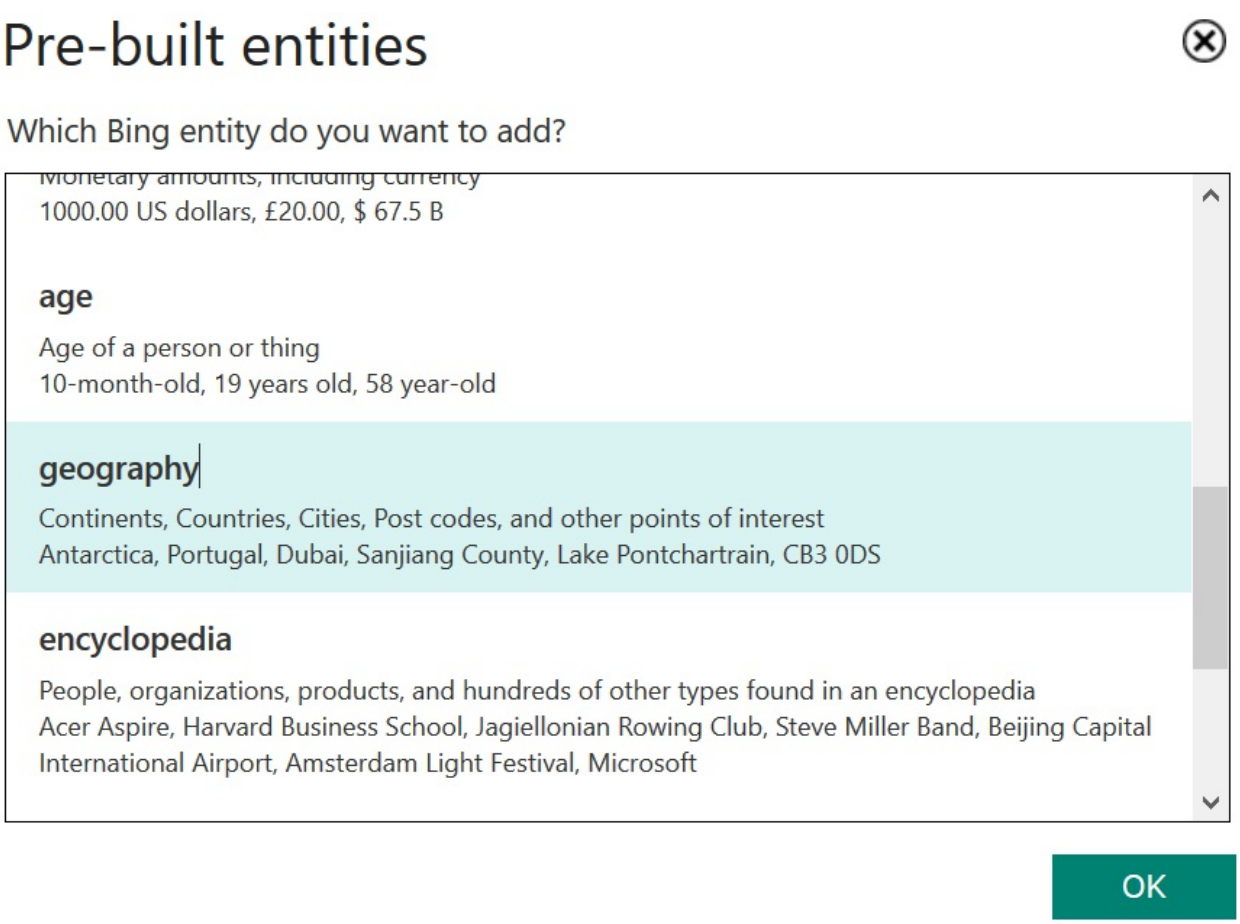
6. Repeat the preceding step for Entity `Company`:

WOW! eBook
www.wowebook.org



7. Now add an Entity. From the left-hand side menu, click on the + icon of **`Pre-Built Entities`**:

8. Select **`geography`** as the Entity:



WOW! eBook
www.wowebook.org

9. The reason why we use the pre-built Entity is that LUIS...

WOW! eBook
www.wowebook.org

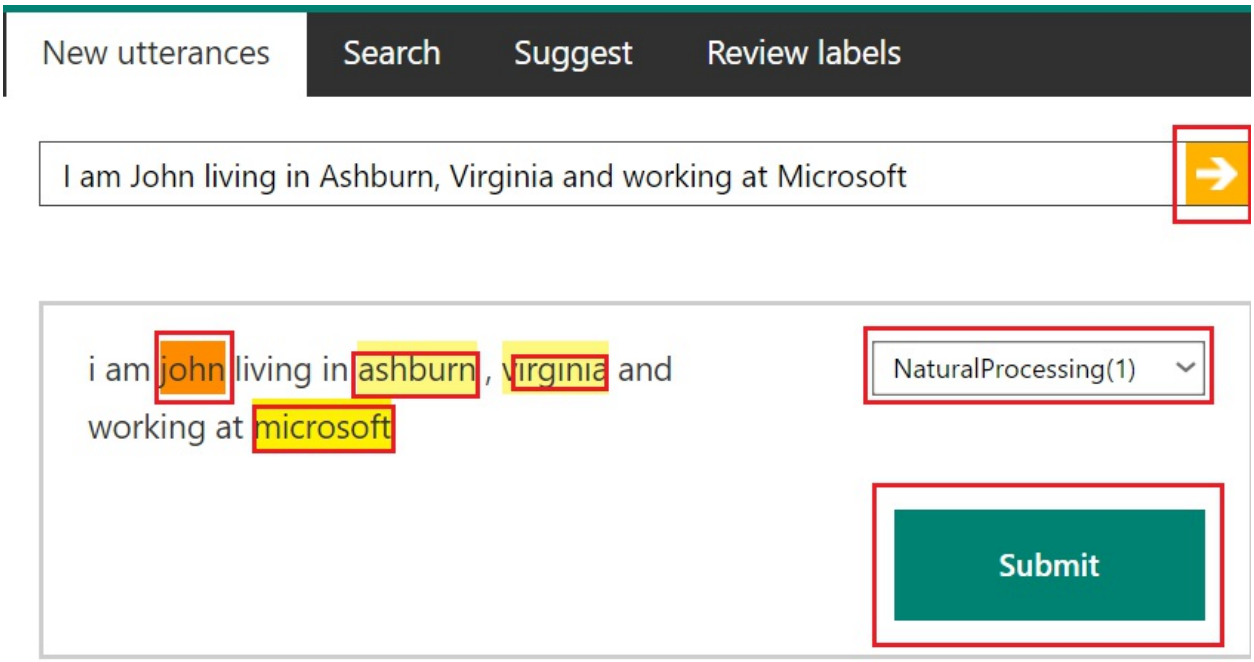
Training your app

Now you have to train your app using utterances to get the appropriate results from LUIS.

Some examples of utterances include the following:

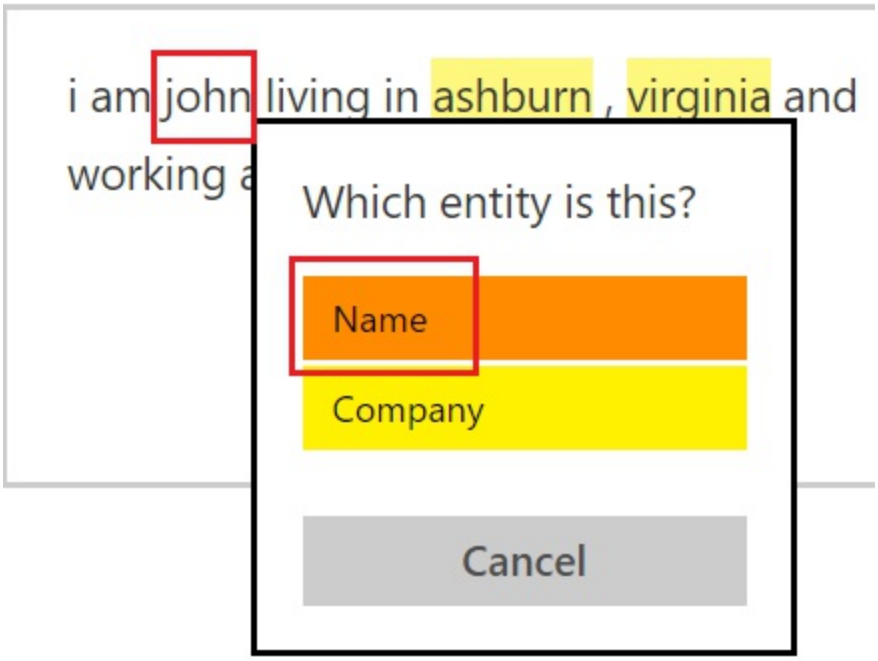
- I am John living in Ashburn, Virginia and working at Microsoft
- Jim lives in Princeton, New Jersey and works at Google

Go through the training process as we explained in previous sections:

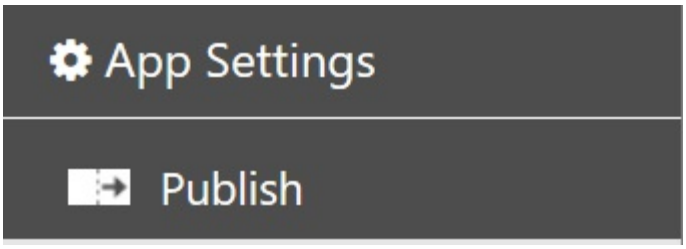


Before clicking on **`Submit`**, make sure that the sentence is identified correctly and shows the Intent as `NaturalProcessing(my intent name)`. If the name did not get highlighted, then manually click on the name. It will open a popup; select **`Name`** as the Entity. For example, here in my case, **`john`** was not highlighted by default, so I selected it manually and clicked on the **`Name`** Intent. The same applies for the company as well:

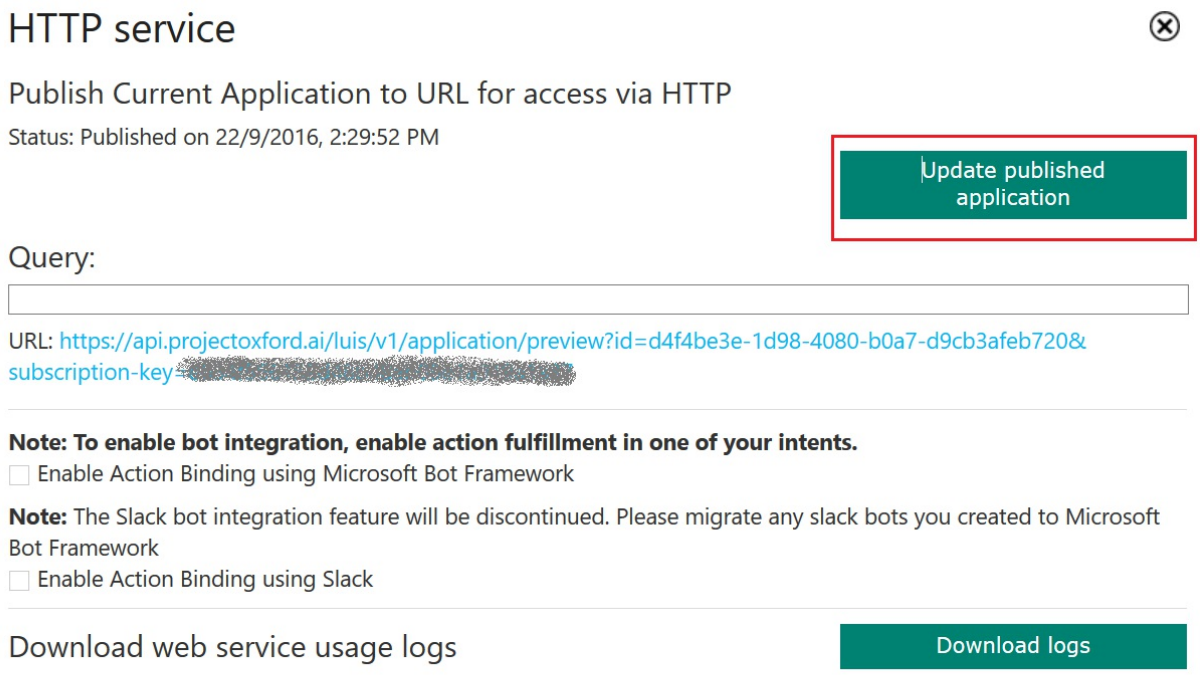
WOW! eBook
www.wowebook.org



Now publish your LUIS. Click on the **`Publish`** option on the left-hand side menu:



Now click on the **`Publish web service`**/**`Update published application`**:



Copy the URL up to query and save it in a safe place. We will need it for later steps:

Now come back to Visual Studio, open the `MessagesController.cs` file and, under the `Post` method, write the code to get the LUIS results:

WOW! eBook
www.wowebook.org

Calling LUIS from the bot

To incorporate a call to LUIS, we can start by adding this function. It simply calls LUIS and returns the phrases identified by LUIS, such as name, city, company name, and so on.

Place your LUIS app endpoint URL, which you copied from the preceding step, in to the following variable in your code:

```
var luisRequestURL =
"https://api.projectoxford.ai/luis/v1/application?id=
fbec04e7-8bda-4160-a059-a8f8b995184b&subscription-
key=ENTER_KEY_HERE";
```

Next, append the user message, which we get from the user to the `luisRequestUrl` and do a `Get` request:

```
 httpClient = new HttpClient();
 HttpResponseMessage response = await
 httpClient.GetAsync(luisRequestURL + "&q=" + messagetext);

 string luisResponseString = await
 response.Content.ReadAsStringAsync();
```

Deserialize the LUIS response and parse it to identify the Intents and Entities:

```
var luisResponse =
JsonConvert.DeserializeObject<LuisResponse>
(luisResponseString);

if (luisResponse.entities.Count > 0)
{
    foreach (var entity in luisResponse.entities)
   ...
```

WOW! eBook
www.wowebook.org

Summary

In this chapter, we have learned the following:

- **Bot dialogs**: The `Dialogs` model is a conversational process, where the exchange of messages between the bot and the user is the primary channel for interaction with the outside world
- **LUIS**: Creating language understanding models, training, and deploying/publishing a model to an endpoint
- **Cognitive Services**: Linguistic API, an advanced linguistic analysis tool for NLP, giving you access to part-of-speech tagging and parsing

WOW! eBook
www.wowebook.org

Chapter 4. Natural Speech and Intent Processing Bot Using Microsoft Cognitive Services

This chapter is for understanding how to use **Microsoft Cognitive Services**. The bot identifies the concepts and actions in the message that is sent to the bot with part-of-speech tagging, and finds phrases and concepts using **natural language parsers**.Also, it returns all the identified intents that were created and trained in the custom LUIS app. For example, if you say "Hi John, am going to New York tonight", the bot will return part-of-speech tagging as well as parses data for natural speech and Intent Processing to know the name and location--that is, Name: John, Place: New York, and so on.

You may be mining customer feedback of your application since you want to know whether the user has given a positive or negative feedback, or you may need to identify what your user is trying to communicate with your bot by interpreting user commands, such as identifying what action mentioned in the text the user wants to perform. To achieve this, first you need to have an understanding of the structure of the text, which is a critical first step, and this...

WOW! eBook
www.wowebook.org

Microsoft Cognitive Services

Initially, Microsoft Cognitive Services was known as **Microsoft Project Oxford**. It is also known as **suite of intelligent APIs** and works across platforms which provide facial recognition in images, voice recognition of speakers, language processing, Academic Knowledge, and more. All these APIs are RESTful services. Owing to REST services, you can develop and integrate with any programming language; here, we use C#.

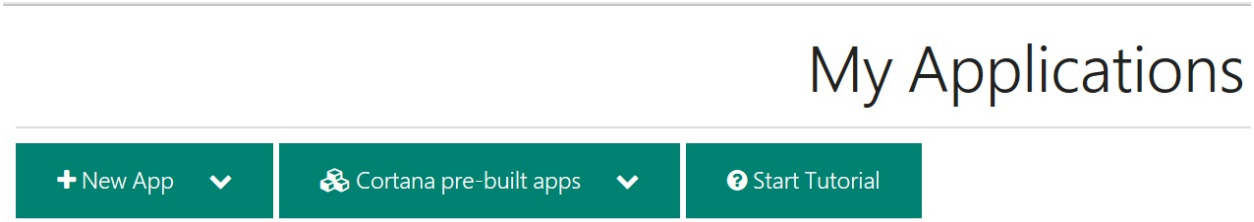
You must try out all the APIs that are available and just play around. For example, APIs such as Face, Emotion, and Speaker recognition will always return a confidence rating/value for each emotion identified in a given image from Face & Emotion APIs. In the case of Speaker recognition, it will tell you how much it accurately matches the voice to your previously registered voice. The best machine learning example API is **Language Understanding Intelligent Service** (**LUIS**). It has a potential for a much better text parser. To use this in an app, you'll need a Microsoft account to log in and get API keys. You can build an application for any platform that is easy to configure. It's free to sign...

WOW! eBook
www.wowebook.org

Identifying the name of a person, place, and company using LUIS

In this step, you will learn how to identify the intent in a sentence. Identifying intent is very important, and helps you to understand what users want to do with your bot. Once you know the intent, you can interpret the sentence based on it and identify the actions from it:

1. After logging in to LUIS at `luis.ai`, create an app for your `IntentProcessing`. To do that, click on **`New App`** and select **`New Application`**:



2. Enter the name of your application, the usage as Bot, and select a category. Finally, click on the **`Add App`** button:

WOW! eBook
www.wowebook.org

Add a new application

Enter application name

Enter application usage scenario

Bot

Choose application domain(s)

- ☐ Booking & Reference
- ☐ Business
- ☐ Comics
- ☐ Communication
- ☐ Education
- ☐ Entertainment
- ☐ Finance
- ☐ Gaming
- ☐ Health & Fitness
- ☐ Home Automation
- ☐ Media & Video
- ☐ Medical
- ☐ Music & Audio
- ☐ Navigation & Maps
- ☐ News & Magazines
- ☐ Personalization
- ☐ Productivity
- ☐ Real Estate
- ☐ Scheduler
- ☐ Shopping
- ☐ Social Network
- ☐ Sports
- ☐ Telecom
- ☐ Tools
- ☐ Transportation
- ☐ Translation
- ☐ Travel & Local
- ☐ Weather
- ☐ Others

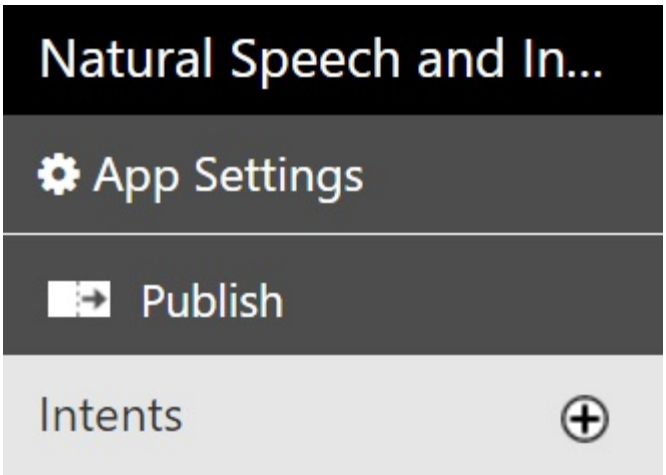
Enter application description (optional)

Choose Application Culture

English

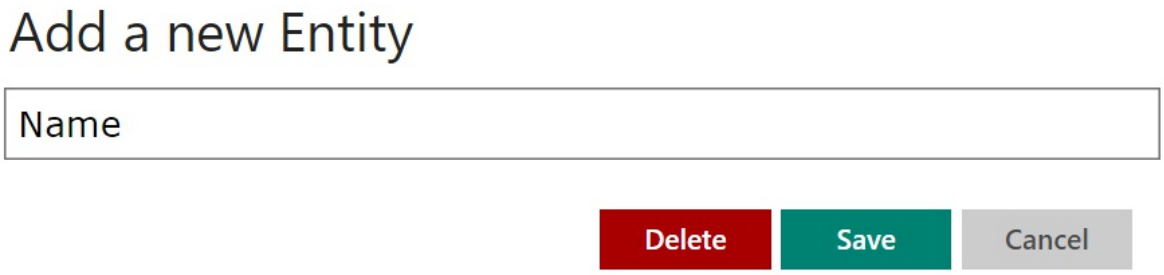
WOW! eBook
www.wowebook.org

3. After the successful creation of the app, open it and click on the **+** icon of the **`Intents`** section from the left side menu:



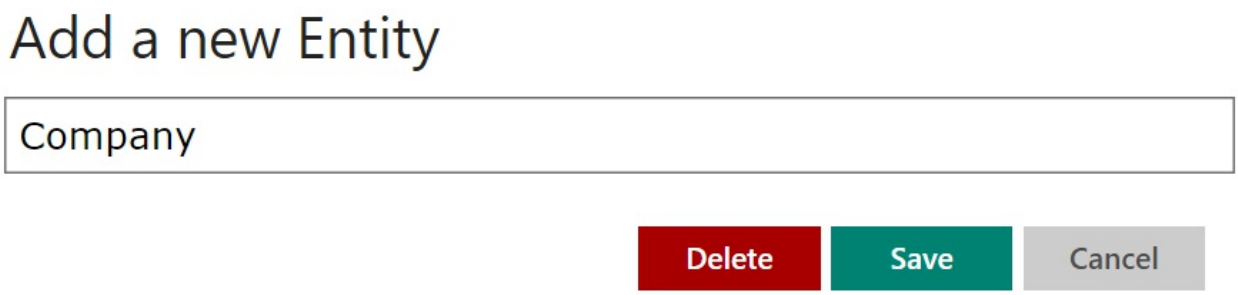
4. Enter a name for your intent and click on the **`Save`** button:

5. Now, add a custom entity, and from the left side menu, click on the **+** icon of Entity and enter **`Name`**:



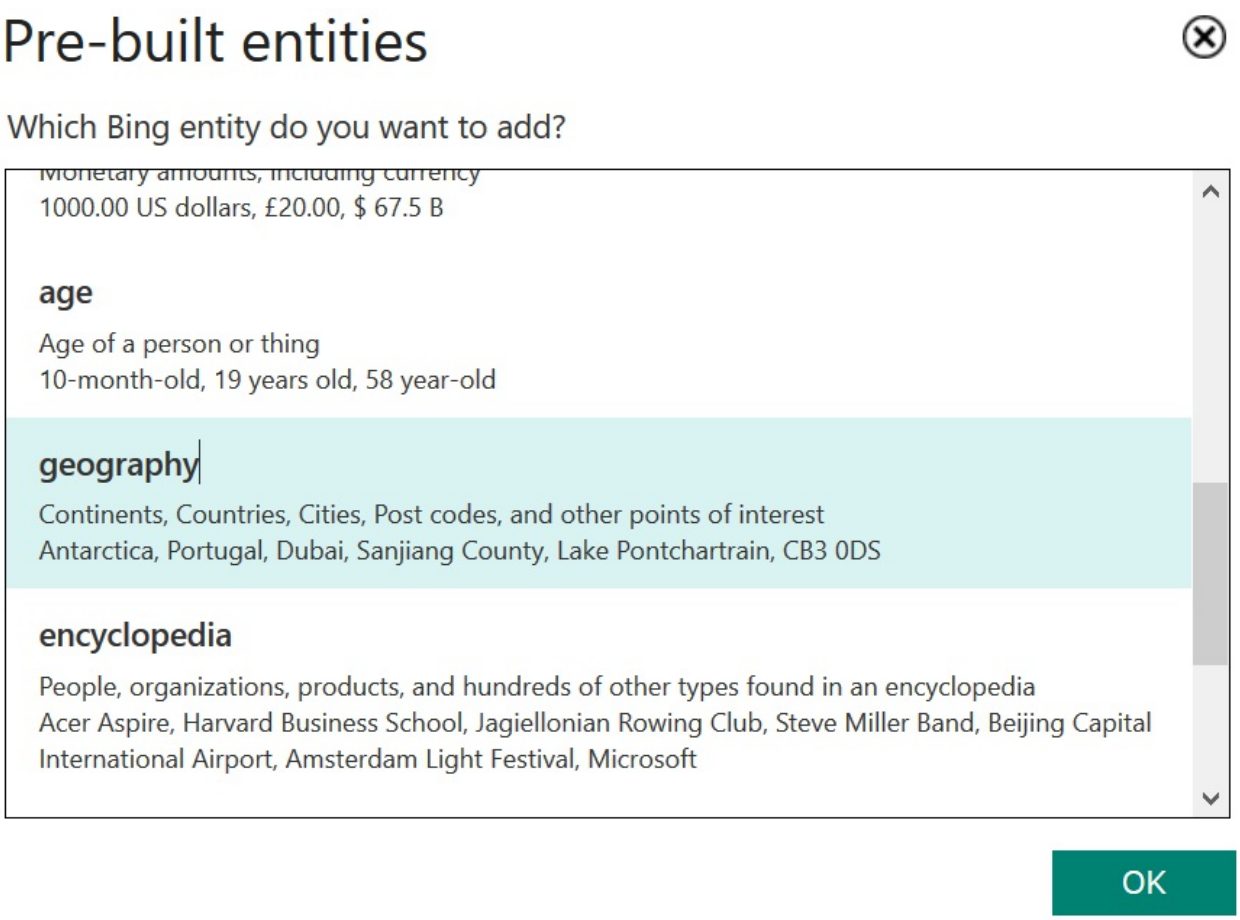
6. Repeat the above step for the company entity:

WOW! eBook
www.wowebook.org



7. Now, add an entity, and from the left side menu, click on the + icon of **`Pre-built Entities`**:

8. Select **`geography`** as the entity:



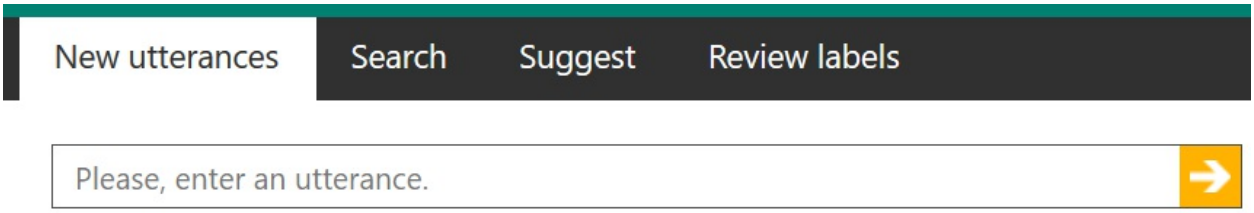
WOW! eBook
www.wowebook.org

9. The reason why we use the pre-built entity is that LUIS already contains geography, which has the complete...

WOW! eBook
www.wowebook.org

Training your app using utterances

Now, let's train your app using utterances for getting the appropriate results from the **Language Understanding Intelligent Service** (**LUIS**). To train, you have to add different types of utterances in your LUIS app. For this, select the **`New utterances`** section and then add the new utterance:



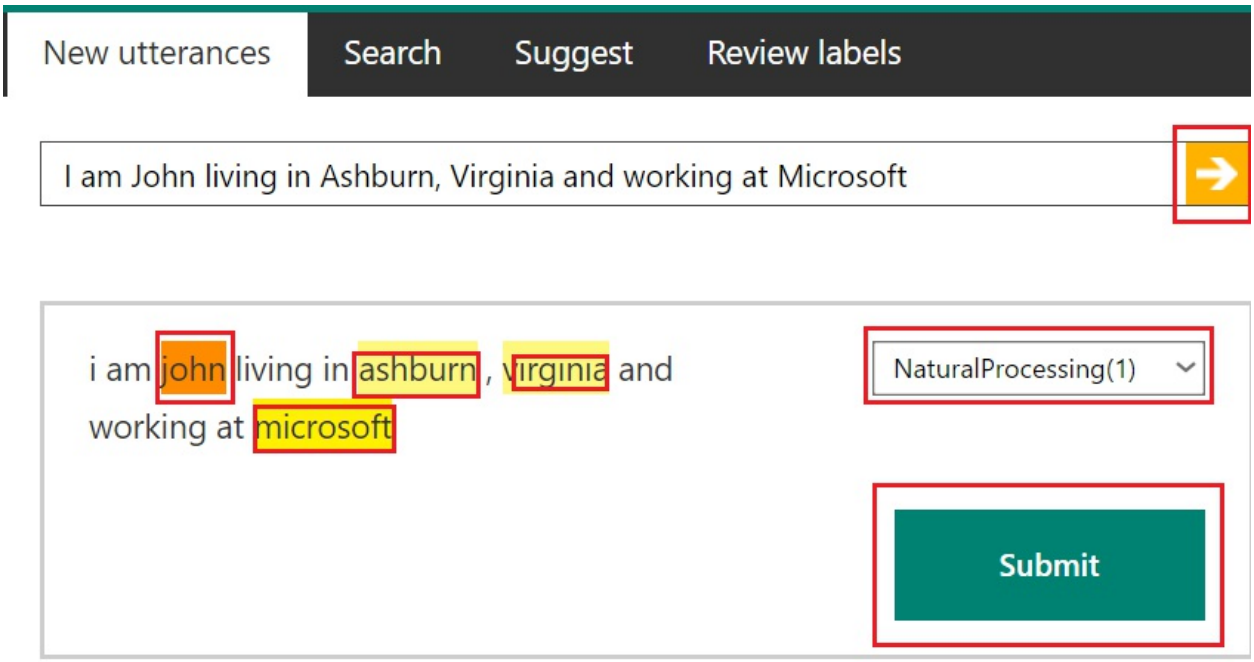
Utterance is nothing but the sentence typed/asked by the user to your bot, such as "I am Kishore living in Ashburn, Virginia and working at Microsoft." You have to enter as many utterances as possible with your bot.

Some examples of an utterance are as follows:

- I am John living in Ashburn, Virginia and working at Microsoft
- Jim lives in Princeton, New Jersey and works at Google

1. After entering the utterance, press *Enter*; now, LUIS will automatically highlight the geography and the name of the person and company in your text, as shown in the following image:

WOW! eBook
www.wowebook.org



2. Before clicking on **`Submit`**, ensure that the sentence is identified correctly and showing Intent as Natural Processing (my intent name). If the name is not highlighted then manually click on the name. It will open a popup; then select Name as the entity. For...

WOW! eBook
www.wowebook.org

Calling LUIS from the bot

In the previous step, we set up and configured the LUIS app and also trained it. Now, let's see how you can use the LUIS app in a bot application. To incorporate a call to LUIS, we can start by adding the mentioned function. It simply calls LUIS REST API and returns the phrases and intents we set up in LUIS, for example, name, city, company name, and so on.

Return to Visual Studio and open the `MessagesController.cs` file; under the `Post` method, update the code to get the LUIS results, as follows:

```
var luisOutputString = "Intent and Language Understanding Inte
```

The following line of code is where we frame a LUIS app REST API URL--if you observe, we are passing the LUIS app ID and LUIS subscription key; this is the URL you copied from the above step, publishing settings of the LUIS app:

```
var luisRequestURL =
"https://api.projectoxford.ai/luis/v1/application?
 id=fbec04e7-8bda-4160-a059-a8f8b995184b&subscript
 key=ENTER_KEY_HERE";
httpClient = new HttpClient();
HttpResponseMessage...
```

WOW! eBook
www.wowebook.org

Summary

In this chapter, we have learned the following:

- **Cognitive Services**: Linguistic API, advanced linguistic analysis tools for natural language processing, giving you access to part-of-speech tagging and parsing
- **LUIS**: Creating Language Understanding Models and Training and deploying/Publishing model to an Endpoint

In the next chapter, you will learn about developing bots using LUIS Prompt Dialogs with State and Nearby Bot using custom APIs.

If you feel like publishing your bot to other channels, refer to Chapter 9, *Publishing a bot to Skype, Slack, Facebook, and the GroupMe Channel,* for how to publish our bot to Skype, Slack, Facebook, and so on.

WOW! eBook
www.wowebook.org

Chapter 5. Developing Bots Using LUIS Prompt Dialogs with State and Nearby Bot Using Custom APIs

In this chapter, we will discuss and develop two bots. One is the Employee Enroll bot using LUIS prompt dialogs and the other is the Nearby Bot using custom APIs. Enroll bot is a basic employee registration kind of bot that will prompt users to enter their first name, last name, designation, and department. If any value is not supplied, then LUIS will prompt the user to enter the missing values. We will also discuss the implementation of bot state. Secondly, we will develop the Nearby Bot to know the attractions near you using a third-party API. So, let's get started.

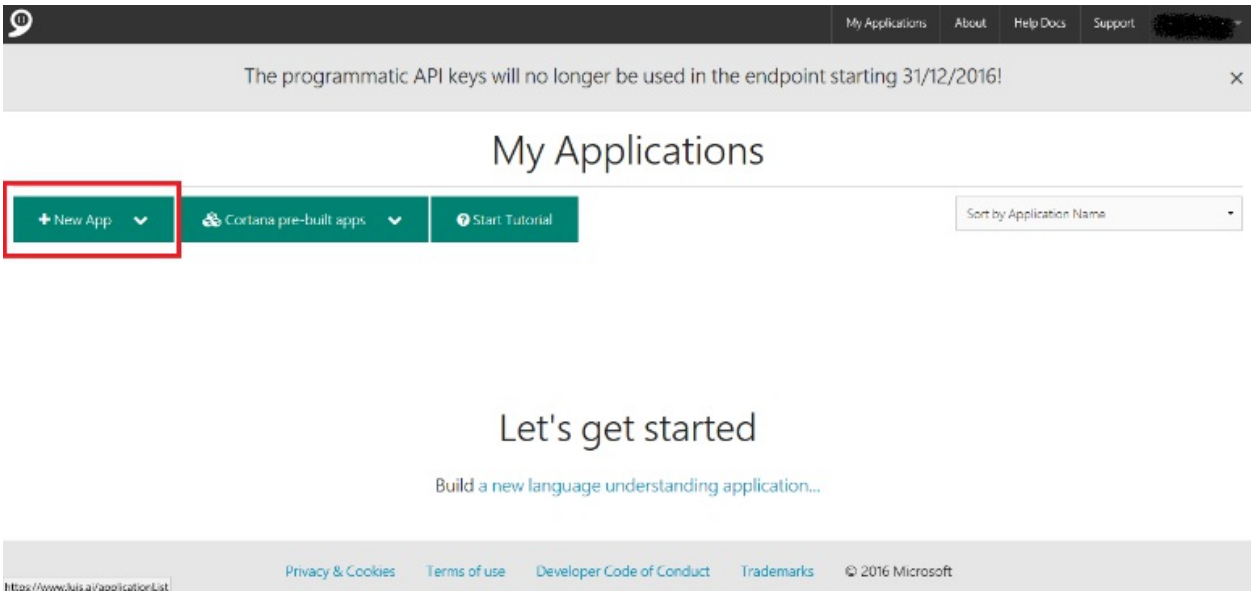
WOW! eBook
www.wowebook.org

Employee Enroll bot using LUIS prompt dialogs

The following steps will guide you to create the Enroll bot:

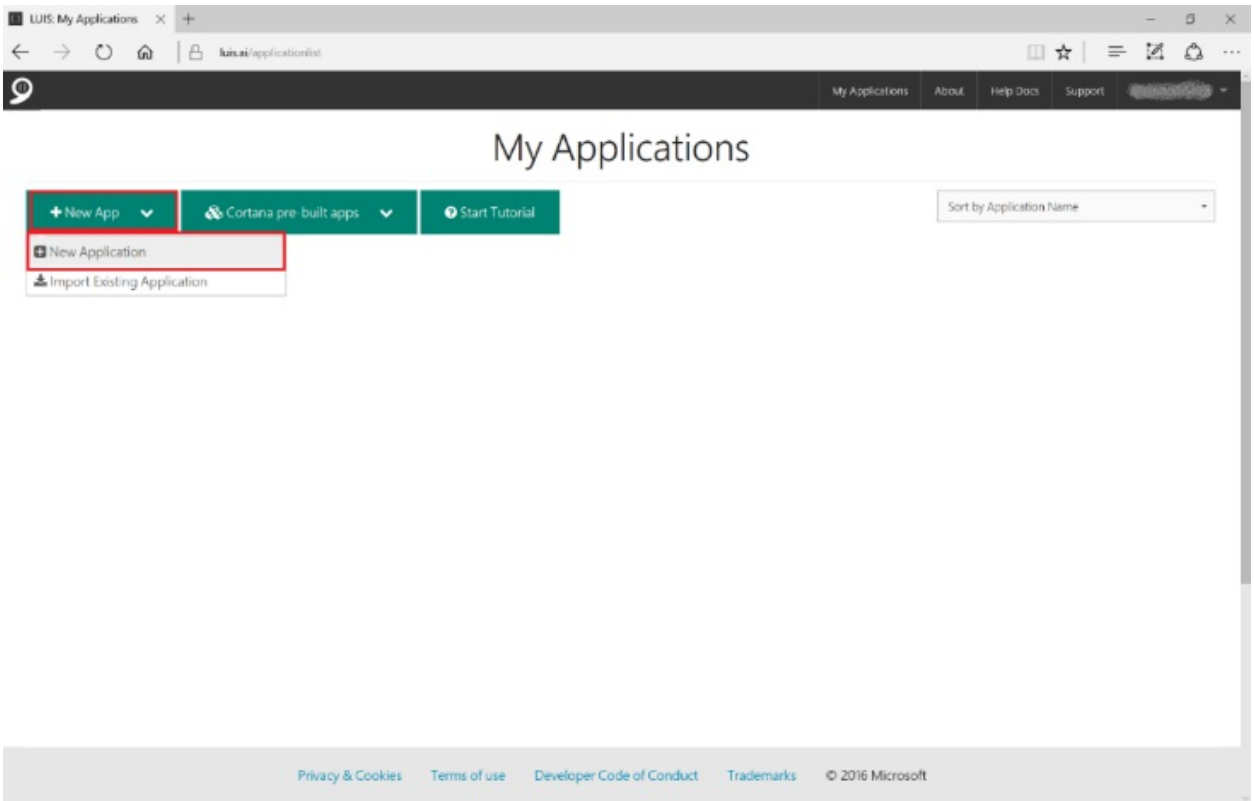
1. Login to https://www.luis.ai/; for more information on activating or signing up for Cognitive Services, check out Chapter 4, *Natural Speech and Intent Processing Bot Using Microsoft Cognitive Services*.

2. Click on **`New App`**:



3. On the **`New App`** drop-down menu, select **`New Application`**:

WOW! eBook
www.wowebook.org



4. It opens an **`Add a new application`** popup; enter the application name, the application usage scenario as `Bot`, and select the category related to your bot. Finally, click on the **`Add App`** button:

WOW! eBook
www.wowebook.org

Add a new application

Enter application name

Employee Enroll

Enter application usage scenario

Bot

Choose application domain(s)

☐ Booking & Reference	☐ Business	☐ Comics	☐ Communication
	☐ Education	☐ Entertainment	☐ Finance
☐ Gaming	☐ Health & Fitness	☐ Home Automation	☐ Media & Video
			☐ Medical
☐ Music & Audio	☐ Navigation & Maps	☐ News & Magazines	☐ Personalization
			☐ Productivity
☐ Real Estate	☐ Scheduler	☐ Shopping	☐ Social Network
☐ Sports	☐ Telecom	☐ Tools	☐ Transportation
☐ Translation	☐ Travel & Local	☐ Weather	☑ Others

Enter application description (optional)

Employee Enrollment

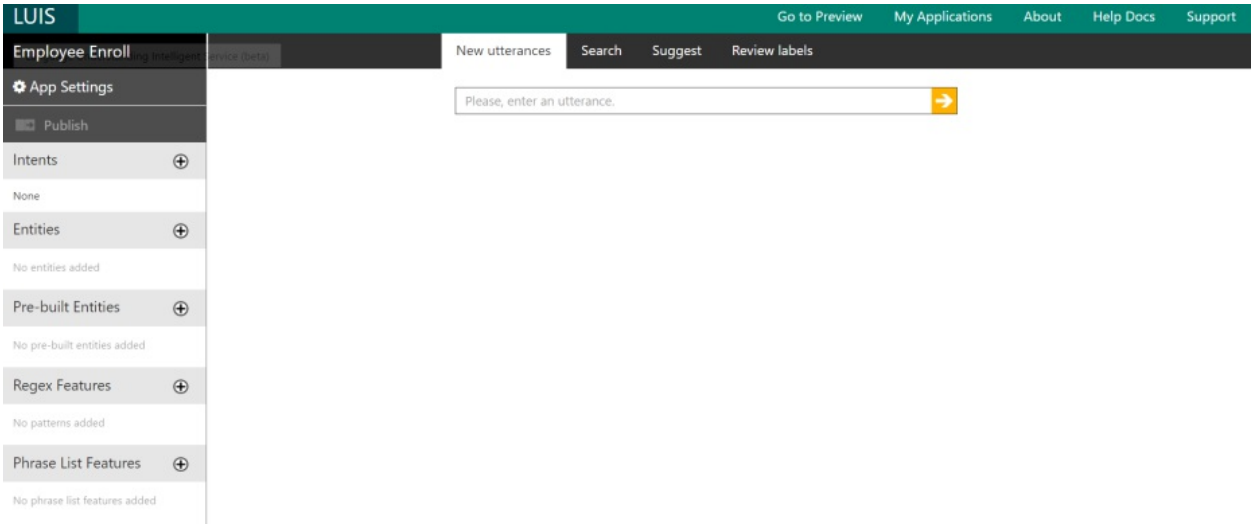
Choose Application Culture

English

Add App

WOW! eBook
www.wowebook.org

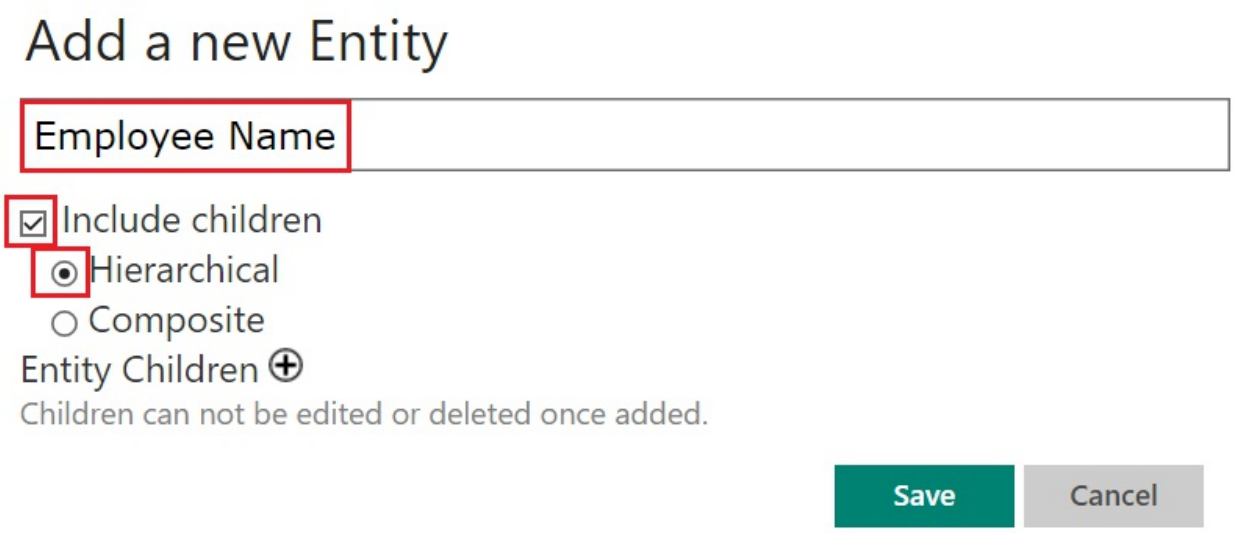
5. The following application will be created:



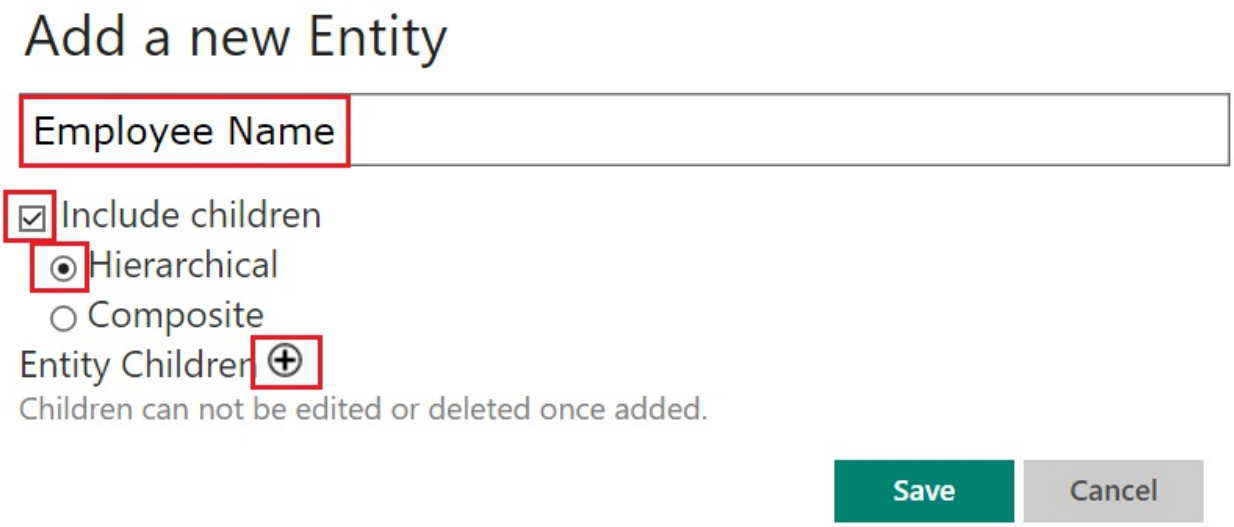
6. Our application will need to detect three entities, namely employee name (composed of first name and last name that we will define as children in the next steps), department, and location.

7. To create these, click on the plus sign next to **`Entities`**:

8. Enter **`Employee Name`** for the entity, then click on the checkbox of **`Include children`**, and select **`Hierarchical`** as the option:



9. Now, click on the plus sign next to **`Entity Children`**, as shown in the following screenshot:



10. Enter the first name in the children name box and again click on the plus...

WOW! eBook
www.wowebook.org

Creating the C# class for LUIS response

For the application, we will have to create C# classes to hold the expected results of the LUIS output.

In the following steps, when you test the LUIS endpoint as in the fifth step of the preceding section, it generates a JSON output; simply select all the contents and copy them:

1. Go to http://json2csharp.com/ or http://jsonutils.com/, or use JSON C# Class Generator at https://jsonclassgenerator.codeplex.com/ and paste the contents of the JSON, then click on **`Generate`**. These services are used to generate C# classes from a given JSON text.

2. **`Copy`** the results:

WOW! eBook
www.wowebook.org

json2csharp

generate c# classes from json

developed by Jonathan Keith
with thanks to the JSON C# Class Generator project
and James Newton-King's Json.NET

```
      "score": 0.9482315
    },
    {
      "entity": "gaddam",
      "type": "Employee Name::Last Name",
      "startIndex": 15,
      "endIndex": 20,
      "score": 0.919439
    }
  ]
}
```

Generate

? ! @

```
public class Intent
{
    public string intent { get; set; }
    public double score { get; set; }
}

public class Entity
{
    public string entity { get; set; }
    public string type { get; set; }
    public int startIndex { get; set; }
    public int endIndex { get; set; }
    public double score { get; set; }
}

public class RootObject
{
    public string query { get; set; }
    public List<Intent> intents { get; set; }
    public List<Entity> entities { get; set; }
}
```

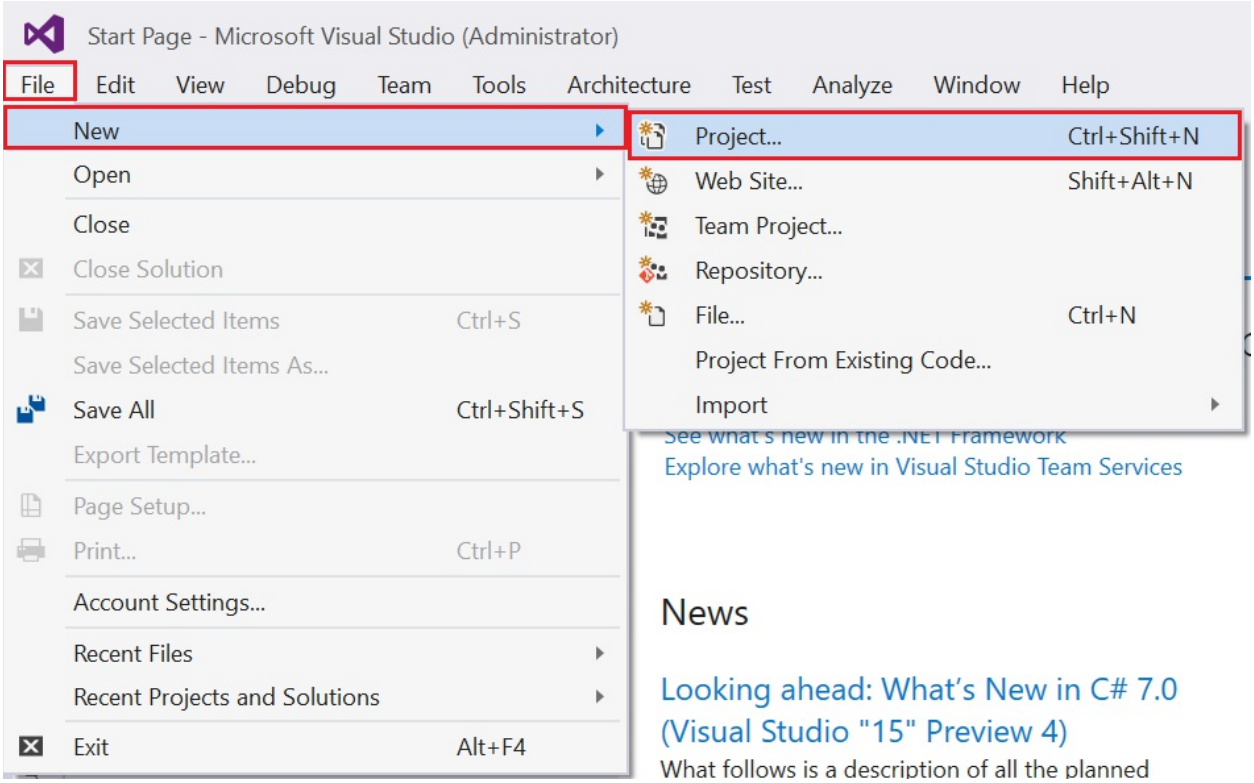
Close Copy

Creating the bot application

Let's take a look at the mentioned steps to create the bot application:

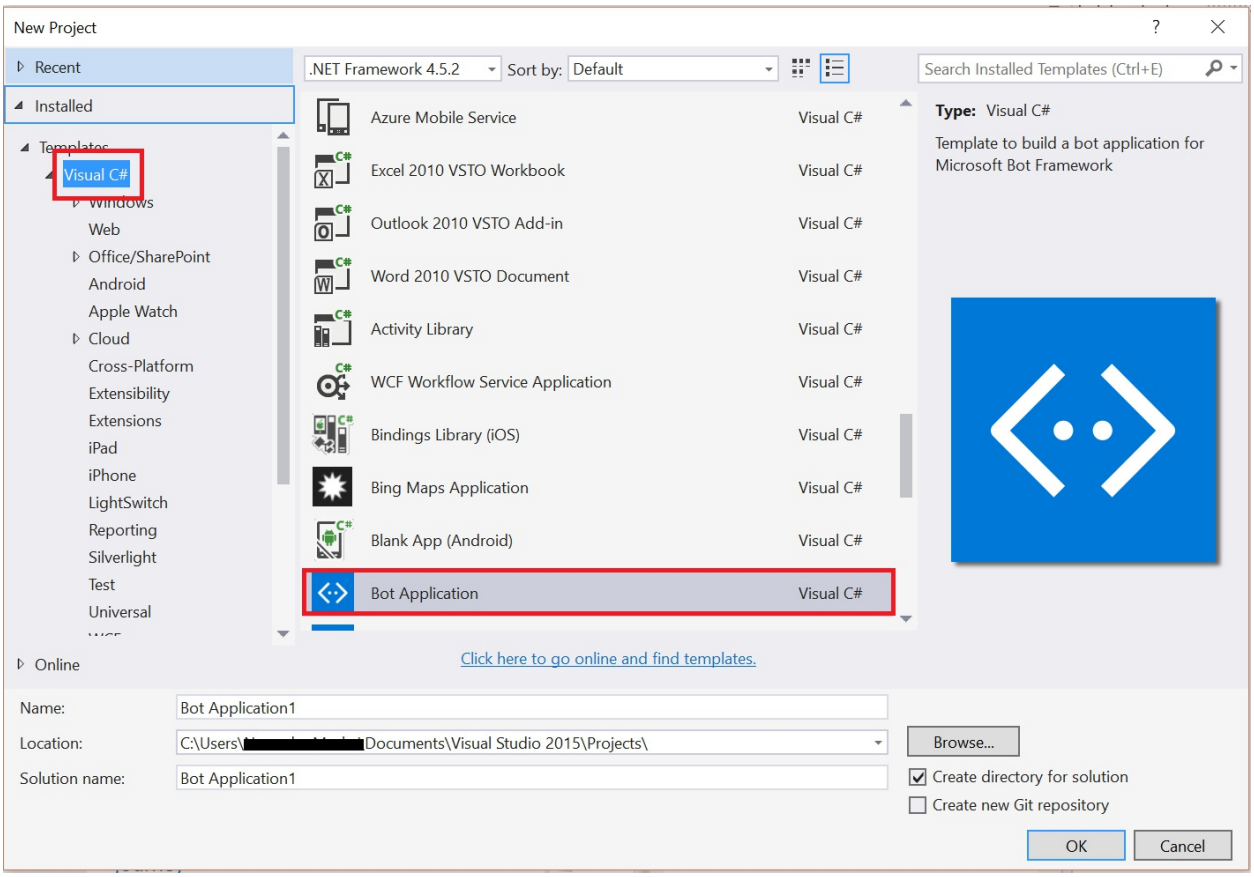
WOW! eBook
www.wowebook.org

1. Open Visual Studio, navigate to **`New`** | **`Project...`**, and select **`Visual C#`** from the left side template category. Then, from the templates section, you will see the **`Bot Application`** template:



2. Select the **`Bot Application`** template, name the project `EmployeeEnrolBot`, and then click on **`OK`**:

WOW! eBook
www.wowebook.org



3. Add a new class and name the class `LUIS`. Then, paste the C# classes that you generated for LUIS in the preceding step. Rename the `RootObject` class to `LUIS`:

WOW! eBook
www.wowebook.org

```
namespace EmployeeEnrolBot
{
    1 reference | 0 changes | 0 authors, 0 changes
    public class Intent
    {
        0 references | 0 changes | 0 authors, 0 changes
        public string intent { get; set; }
        0 references | 0 changes | 0 authors, 0 changes
        public double score { get; set; }
    }

    1 reference | 0 changes | 0 authors, 0 changes
    public class Entity
    {
        0 references | 0 changes | 0 authors, 0 changes
        public string entity { get; set; }
        0 references | 0 changes | 0 authors, 0 changes
        public string type { get; set; }
        0 references | 0 changes | 0 authors, 0 changes
        public int startIndex { get; set; }
        0 references | 0 changes | 0 authors, 0 changes
        public int endIndex { get; set; }
        0 references | 0 changes | 0 authors, 0 changes
        public double score { get; set; }
    }

    0 references | 0 changes | 0 authors, 0 changes
    public class LUIS
    {
        0 references | 0 changes | 0 authors, 0 changes
        public string query { get; set; }
        0 references | 0 changes | 0 authors, 0 changes
        public List<Intent> intents { get; set; }
        0 references | 0 changes | 0 authors, 0 changes
        public List<Entity> entities { get; set; }
    }
}
```

WOW! eBook
www.wowebook.org

4. Add the...

WOW! eBook
www.wowebook.org

Bot state service

The key to good bot design is to do the following:

- Make the web service stateless so that it can be scaled
- Make it track the context of a conversation

Since all bots have these requirements, the Bot Framework has a service for storing bot state. This lets your bot track things such as *what was the last question I asked them?*

In our case, we want to save the `contextId` of the LUIS to exchange the missed information to the LUIS app from our bot.

To do that, first we need to create `BotStateClient`.

Creating a state client

The default state client is stored in a central service. For some channel IDs, you may want to use a state API hosted in the channel itself (for example, with the *emulator* channel) so that the state can be stored in a compliant store that the channel supplies.

We have provided a helper method on the `activity` object, which makes it easy to get an appropriate `StateClient` for a given message:

```
StateClient stateClient = activity.GetStateClient();
```

After getting the state client, we can now save our `contextId` to it. When your bot sends a reply, you simply set your object in one of the `BotData` records properties,...

WOW! eBook
www.wowebook.org

Developing a Nearby Bot using custom APIs

So far, you have learned about bot application creation using *Visual Studio*, publishing to *Azure*, *Bot* registration at dev.Botframework.com, and connecting to channels. In this Nearby Bot tutorial, we will explain how to use third-party APIs from your bot.

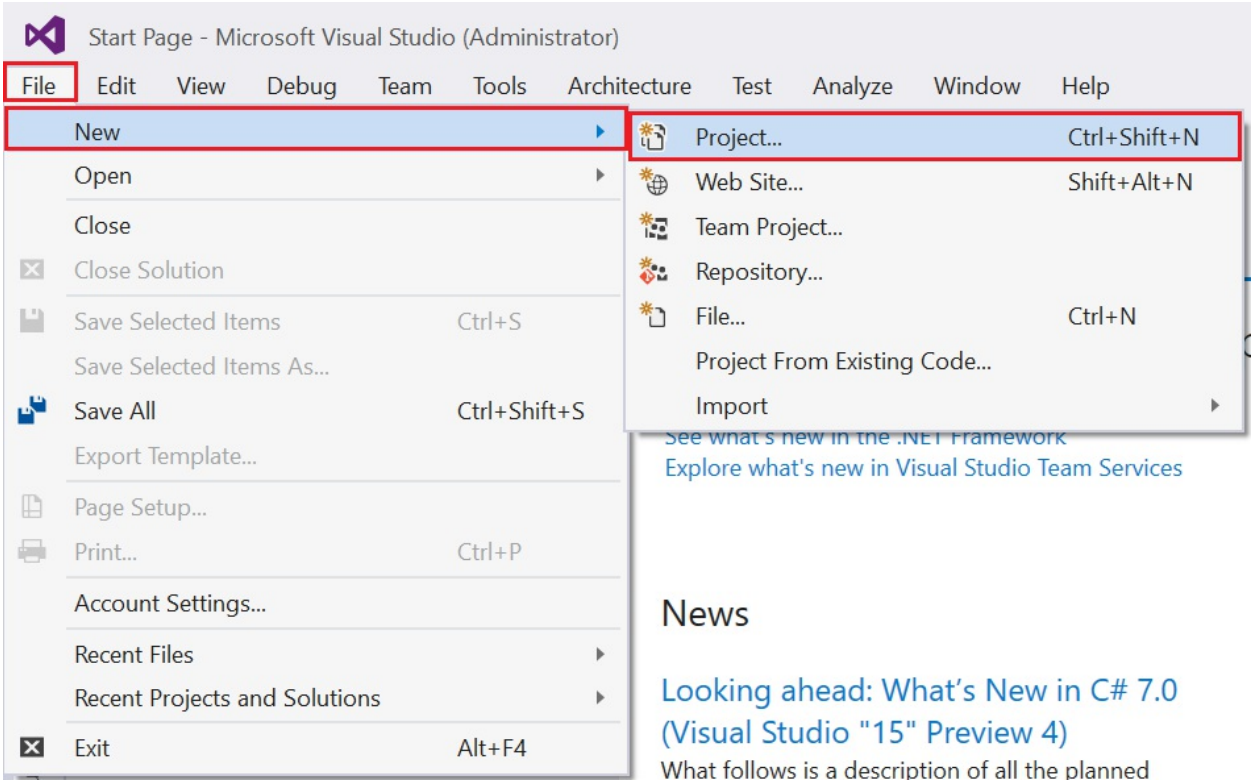
The main purpose of this bot is to provide information about the nearby amenities of a given place--for example, if you want to know the top restaurants near New York.

> ## Note
>
> This guide is for C# using the Bot Framework Connector SDK .NET template.

Let's look at the steps:

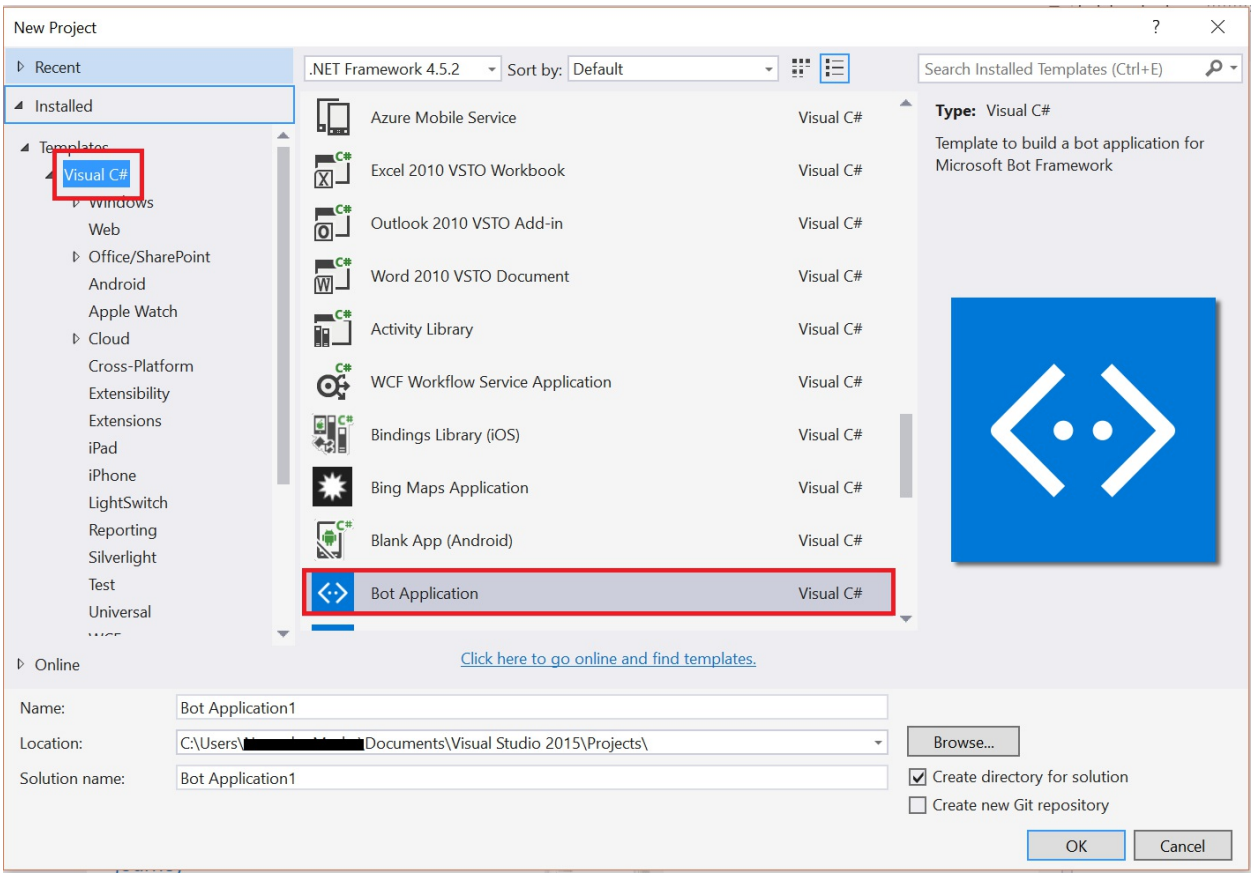
1. Open Visual Studio and click on **`New`** | **`Project...`**:

WOW! eBook
www.wowebook.org



2. Select **`Visual C#`** from the left side template category; then, from the templates section, you will see the **`Bot Application`** template:

WOW! eBook
www.wowebook.org



3. Select the **`Bot Application`** template, name the project `NearbyBot`, and then click on **`OK`**.

Before we jump into the code, first we will explain how to get the nearby places information of a given place using third-party APIs. We will do this by using the *Google Places API*. If you want to use the Google Places API, you need to have an **API key**--for that, follow the following instructions:

1. Go to the **`Google...`**

WOW! eBook
www.wowebook.org

Summary

In this chapter, we have learned the following:

- **LUIS prompt dialogs**: Using this feature, we can make our bot more mature by identifying the missing / expecting entities in a given sentence, which gives a more natural way of conversation with users.
- **Third-party APIs**: These are used for calling third-party APIs from the bot.
- **Bot state**: This will help us to store information about the user and important information in the last conversation. Based on the last conversation, we can communicate with users in a more natural way, like how we did in the Employee Enroll bot.

WOW! eBook
www.wowebook.org

Chapter 6. Developing an IVR Bot for a Bank Using Advanced Microsoft Bot Framework Technologies

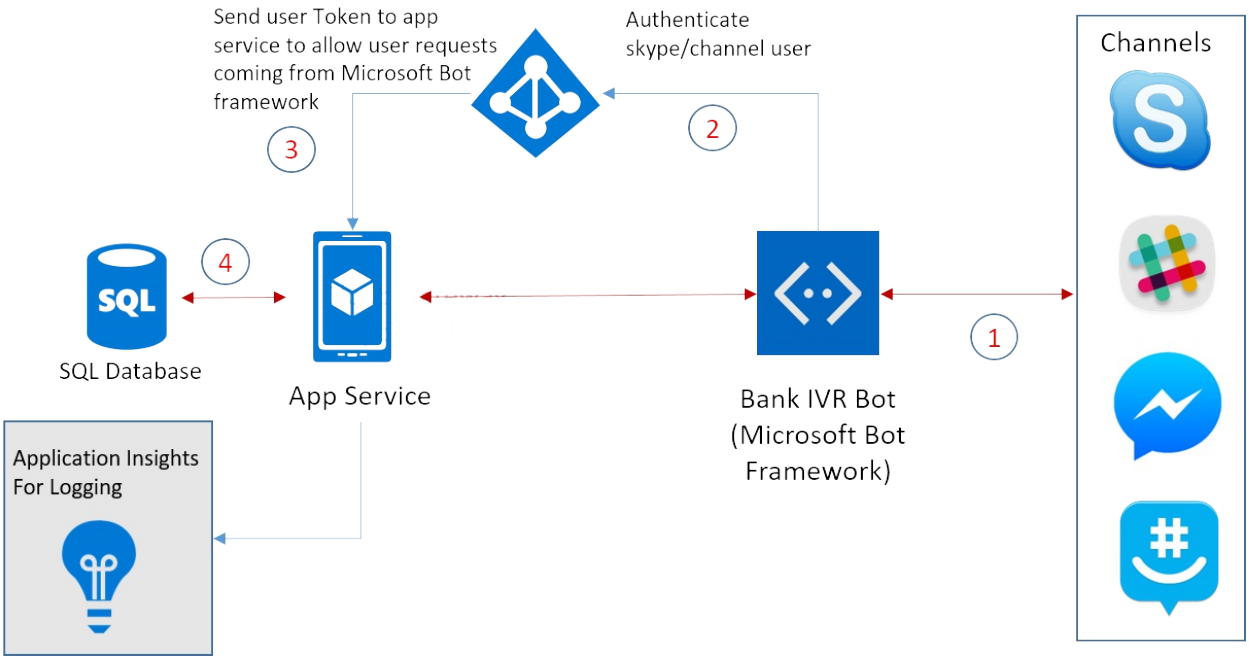
The Bank **Interactive Voice Response** (**IVR**) bot is like phone banking you can do bank transactions from within the bot itself. This bot will have options such as create account, balance enquiry, credit card payment, and delete account. The Bank IVR bot can tell you the balance of your account. It can also pay your credit card bill as well, by just selecting a few options.

In this bot, we will mainly use the **Conversation Concept** using **FormFlow** and dialogs. For example, whenever a user enters some text, the bot will immediately send a response message and also remember the entire conversation. Unlike dialogs, FormFlow helps to handle guided conversations such as ordering a sandwich, booking a movie ticket, setting up an appointment with a doctor, and so on. These types of scenarios need lots of effort.

WOW! eBook
www.wowebook.org

High-level architectural diagram

The following is the architecture diagram for the Bank IVR bot. These are the descriptions of the numbers:



1. We have the Bank IVR bot registered with Microsoft Bot Framework and configured to channels.
2. We have an AAD authentication.
3. App service is where we publish our Bot--it requests a token to allow user requests coming from Microsoft Bot Framework (from channels).
4. We also have a SQL database connected to App service.
5. For logging/tracking user operations, we use Application Insights.

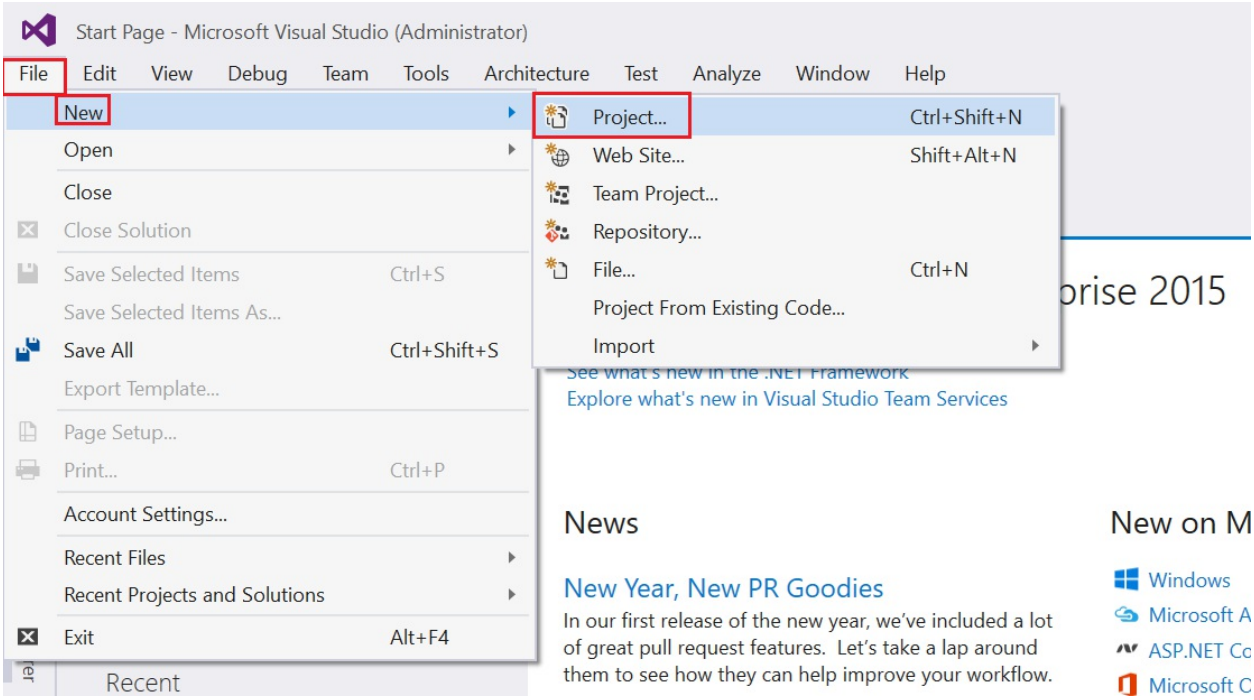
As mentioned in preceding architecture, you need to develop a Bot Application first. Perform the steps mentioned in the next section to develop a Bot Application using Visual Studio.

WOW! eBook
www.wowebook.org

Let's start coding

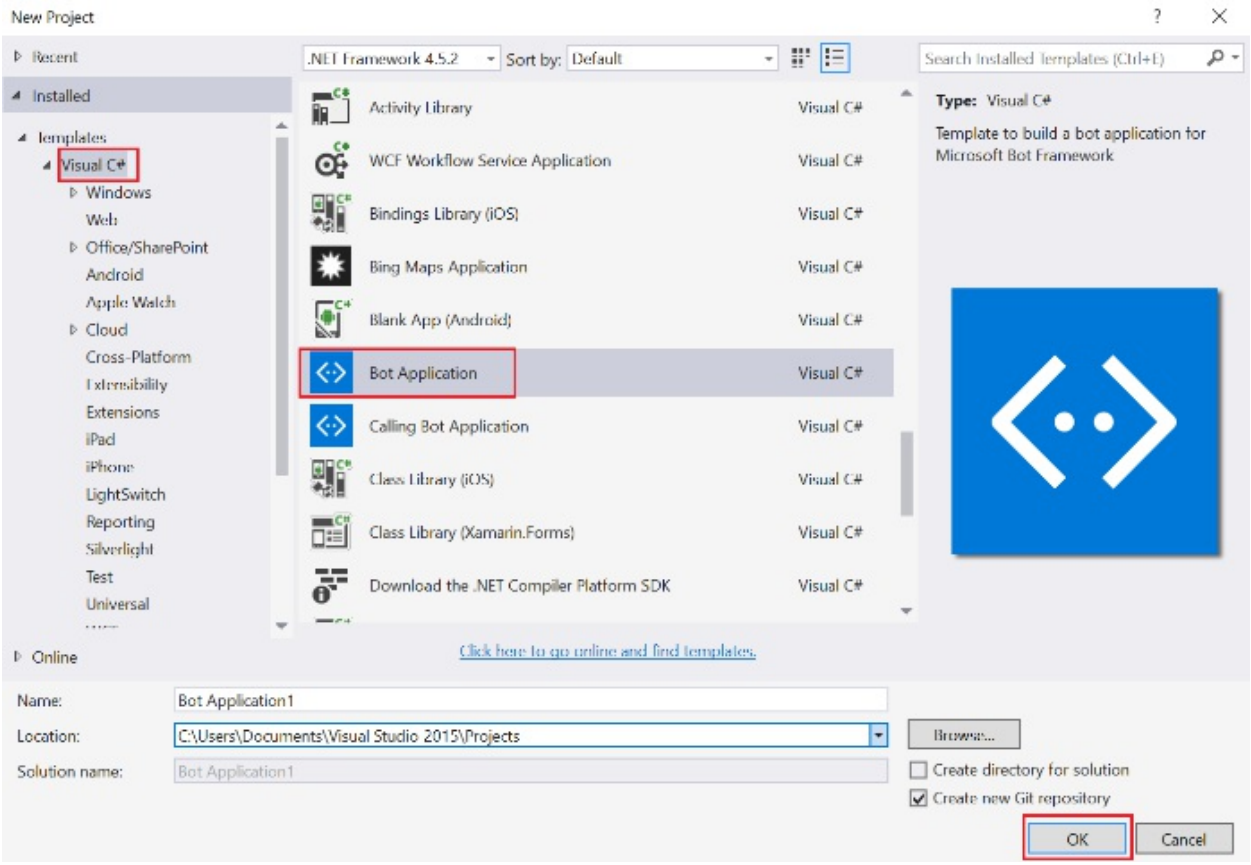
Perform the following steps to create the bot application:

1. Create a new C# project using the new **`Bot Application`** template.
2. Navigate to **`New`** | **`Project...`** in Visual Studio 2015; it will open the following window:



3. Select the **`Bot Application`** template, give it a name, and click on **`OK`**:

WOW! eBook
www.wowebook.org



Creating an account with the bot

Here, I will explain how to build a FormFlow to create an account with this bot. To start the FormFlow and create an account, you need to create a C# class to define the form you want the information to be completed.

Create an enum with all the options we are going to implement, as shown here:

```
public enum Options
{
    CreateAccount,
    [Terms(new string[] { "savings balance", "Savings A
    Balance" })]
    [Describe("Savings Account Balance")]
    SavingsAccountBalance,
    [Terms(new string[] { "current balance", "Current A
    Balance" })]
    [Describe("Current Account Balance")]
    CurrentAccountBalance,
    [Terms(new string[] { "creditcard payment", "Credit
```

WOW! eBook
www.wowebook.org

Payment"...

WOW! eBook
www.wowebook.org

Storing the bot conversation (new account info) data in an Azure SQL database

---

We are maintaining the state or bot conversation of the user. For this, we will use an Azure SQL database. We save all the transactions that the user performs, such as registering a new user, credit card payments, checking their balance, and so on. These are the steps for storing the bot conversation in the Azure SQL database:

1. In the Azure portal, create a new Azure SQL database. To see how to create a new SQL database in Azure, follow the steps shown at https://docs.microsoft.com/en-us/azure/sql-database/sql-database-create-databases.
2. After the successful creation of a database in Azure, you have to create a table in it. For that, you have to open your SQL database in Visual Studio 2015.
3. Next, create a table using the following commands:

```
CREATE TABLE [dbo].[Accountant_Information] (
         [Id]                      NVARCHAR (128)DEFAULT (newid
             NOT NULL,
         [AccountNumber]           NVARCHAR (MAX)NOT NULL,
                    [PinNo]                      INTNOT NULL,
         [AccountType]             NVARCHAR (MAX)NULL,
        ...
```

WOW! eBook
www.wowebook.org

Checking your savings account balance using the bot

In this section, I will explain about how to build a FormFlow to check your savings account balance.

To check your balance, we need an account number and PIN. For that, we will add a class called `Balance` with two fields: `AccountNumber` and `PIN`. Next, we will define a public property for the `Balance` class in the `Customer` class, and finally we will append the `Balance` field to the builder object:

```
[Serializable]
class Balance
{
    [Prompt("Please enter your account number")]
    public string AccountNumber;
    [Prompt("Please enter your pin")]
    public string PIN;
};
```

In the `Customer` class, define the `Savings_Balance` property:

```
//Savings Account Balance Template
public Balance Savings_Balance;
```

Append the `Savings_Balance` property to the builder object before the `OnCompletion` method:

```
builder.Field("Savings_Balance.AccountNumber", isBalar
validate: async (state, response) =>
                {
                    var result = new ValidateResult { I
                   ...
```

WOW! eBook
www.wowebook.org

Checking your current account balance using the bot

In this step, I will explain how to build a FormFlow for checking your current account balance. After the savings account balance logic, append the following code for the current account logic:

```
//Current Account Balance Template
        public Currentbalance Current_Balance;

                                .Field("Current_Balance.AccountNumb
                                isCurrentBalance, validate: async
                                response) =>
                                   {
                                      var result = new ValidateResu
                                      IsValid = true, Value = respo
                                      string accountnumber = (respo
                                      string);
                                      int accountnumberlength =
                                      accountnumber.Length;
                                      if (accountnumberlength < 11
                                      accountnumberlength > 17)
                                        {
                                           result.Feedback =...
```

WOW! eBook
www.wowebook.org

Paying your credit card bill using the bot

In this section, let's see how to build a FormFlow to pay a credit card bill using the bot. However, here we will not use a database; rather the bot will just contain static information.

Add the `CreditCardPayment` class:

```
[Serializable]
    class CreditCardPayment
    {
        [Prompt("Please enter your creditcard number")]
        public string CreditcardNumber;
        [Prompt("Please enter how much amount do you want to pa
        public string Pay;
        public string CreditCardPaymentSuccessMessage;
    };
```

Next, in the `Customer` class, define the following public field/property:

```
public CreditCardPayment CreditCard_Payment;
```

Append the following code to the builder object to perform the credit card payment; add it before `OnCompletion(accountStatus)`:

```
Field("CreditCard_Payment.Pay", isCreditCardPayment)
                        .Field(new FieldReflector<Customer>
                        ("CreditCard_Payment
                        .CreditCardPaymentSuccessMessage")
                        .SetType(null)
                        .SetActive((state) => state.Option ==
             ...
```

WOW! eBook
www.wowebook.org

Deleting an account using the bot

In this step, I will explain how to build a FormFlow to delete an account using this bot.

Add the following class to delete an account:

```
class DeleteAccount
    {
        [Prompt("Are you sure want to delete your account?")]
        public string DeleteConfirmationMessage;
        public string DeleteSuccessMessage;
    };
```

To delete an account, we need the account number. Define a field for the `DeleteAccount` class and one more field for the account number in the `Customer` class, as follows:

```
public DeleteAccount Delete;
        [Template(TemplateUsage.EnumSelectOne, "Please select y
        {||}", ChoiceStyle = ChoiceStyleOptions.PerLine)]
        public string AccountNumber;
```

Append the following code to the `builder` object before `OnCompletion(accountStatus)`:

```
.Field(new FieldReflector<Customer>
         (nameof(Customer.AccountNumber))
                 .SetType(null)
                  .SetActive((state) => state.Option ==
                 Options.DeleteAccount)
                  .SetDefine(async (state, field) =>
                   {
                    if (state.AccountType != null)...
```

WOW! eBook
www.wowebook.org

Summary

In this chapter, we have learned about FormFlow. With the help of the FormFlow, we can build bots that depend on guided conversations, such as ordering a sandwich, booking a movie ticket, setting up an appointment with a doctor, and so on. These types of scenario can be built with less effort using FormFlow.

WOW! eBook
www.wowebook.org

Chapter 7. Intelligent Bots with Microsoft Bot Framework and Service Fabric

In this chapter, we will learn how **Service Fabric** helps to develop intelligent bots using stateless and stateful microservices.

Azure Service Fabric is an Azure service offered by Microsoft to develop and publish microservice-based applications and perform life cycle management. Developers have the ability to select which architecture they want to use, such as stateless or stateful services. This allows developers to develop an architectural approach where complex applications are involved and composed of small, independently versioned services to scale in the cloud with Azure Service Fabric.

The name **stateless microservices** itself tells that they will not maintain state. Protocol gateways and web proxies do not maintain a mutable state outside a request and its response from the service. The best examples of stateless microservice architecture are Azure Cloud Services worker roles.

Stateful microservices will maintain a mutable state beyond a request and its response. Databases, devices, shopping carts, and queues maintain a mutable state.

The...

WOW! eBook
www.wowebook.org

Getting started using stateless microservices

First, we will learn how to develop a bot and publish/host in Service Fabric using stateless microservices.

Setting up your development environment for Service Fabric

To build and run Azure Service Fabric applications on your development machine, install the runtime, SDK, and tools. It's also necessary to enable execution of the Windows PowerShell scripts that are included in the SDK.

Prerequisites

The following operating system versions are supported for development:

- Windows 7
- Windows 8/Windows 8.1
- Windows Server 2012 R2
- Windows Server 2016
- Windows 10

Installing the SDK and tools

In Visual Studio 2015, Service Fabric tools are installed together with the SDK; if you cannot find the Service Fabric templates or tools, then you can install them with the help of **Web Platform Installer** or go through http://www.microsoft.com/web/handlers/webpi.ashx?command=getinstallerredirect&appid=MicrosoftAzure-ServiceFabric-VS2015 to download the Service Fabric SDK and tools.

Enabling PowerShell script execution

WOW! eBook
www.wowebook.org

Service Fabric uses Windows PowerShell scripts to create a local development cluster and to deploy...

WOW! eBook
www.wowebook.org

Publishing a Service Fabric project in Azure

It is important that we need to create a secure cluster in Azure. For that, we need to set up a **Key Vault** to manage keys and certificates. For more information on the Azure Key Vault and certificates, follow this link, https://docs.microsoft.com/en-us/azure/key-vault/key-vault-get-started

Create Key Vault

Create a **Key Vault** in the new resource group. The Key Vault must be enabled for deployment to allow the Service Fabric resource provider to get certificates from it and install on cluster nodes. The following is the PowerShell script:

```
New-AzureRmKeyVault -VaultName 'myvault' -ResourceGroupName
    'mycluster-keyvault' -Location 'East US' -EnabledForDeploym
```

Adding certificates to the Key Vault

This certificate is required to secure a cluster and prevent unauthorized access to it. To make this process easier, a PowerShell module is available on GitHub (https://github.com/kishoreismac/Service-Fabric/tree/master/Scripts/ServiceFabricRPHelpers).

The `ServiceFabricRPHelpers.psm1` module provides helper methods for adding certificates to the Key Vault for use in the Service Fabric cluster....

WOW! eBook
www.wowebook.org

Summary

In this chapter, we introduced Service Fabric and stateless and stateful microservices, and how these help in the development of bots using Microsoft Bot Framework. Also, we saw how to set up a development environment, the prerequisites for Service Fabric, and the tools needed for programming. This chapter is based on helping developers quickly and easily build, test, debug, deploy, and update their bot applications for single-box, test, and production deployments. It also shows how Service Fabric helps our bots to scale and be managed easily with a Service Fabric cluster. In the coming chapters, we will see how bots can help us with the **Internet of Things** (**IoT**).

WOW! eBook
www.wowebook.org

Chapter 8. Developing Intelligent Facial Expression Identification Bot for IoT Using Azure and Power BI

In previous chapters, we have gone through the concepts of Bot Framework and Cognitive Services, such as LUIS, the linguistic service, and so on that are involved in developing and publishing bot applications. In this chapter, we will mainly focus on integrating IoT, the bot, and Power BI and see how can we leverage the power of bots and Cognitive Services in IoT. Here, we will develop an IoT application that will capture photos from a USB camera connected to the Raspberry Pi, then process it using the Face API and Emotion API of Cognitive Services to identify facial expressions. Once it completes the processing and identifies the face and emotional expression, the IoT application then sends it to the Facial Expression and Identification bot and also to Power BI to show reports.

WOW! eBook
www.wowebook.org

Before getting started

I assume that you have some knowledge of the following topics. If you don't go through the following topics and links:

1. Bot Framework.
2. Raspberry Pi: In this chapter, we will not cover how to configure the Raspberry Pi, so check out the following link on how to set up the device for development:https://developer.microsoft.com/en-us/windows/iot/getstarted
3. Windows 10 IoT Core: In this chapter, we will use Windows 10 IoT Core as our operating system on the Raspberry Pi. For how to install set it up on the Raspberry Pi, check out the following link:https://developer.microsoft.com/en-us/windows/iot/getstarted
4. UWP app development: The project we are developing for the Raspberry Pi is a UWP app, so you must have some basic knowledge of UWP app development:https://docs.microsoft.com/en-in/windows/uwp/get-started/whats-a-uwp
5. Microsoft Azure IoT Hub: We need Microsoft Azure IoT Hub to easily and securely connect your IoT devices (Raspberry Pi2). Use device-to-cloud telemetry data to understand the state of your devices and assets, and be ready to take action when a device needs your attention:

WOW! eBook
www.wowebook.org

Configuring Raspberry Pi and sensors

In this project, we will use the Raspberry Pi 2 and Adafruit kit components to use a PIR (Pyroelectric/Passive Infrared Sensor) motion sensor to detect the motion of an object, which triggers a USB webcam to snap a photo and send it to the bot.

Prerequisites

Before getting started with the project, let's take a look at the hardware and software requirements.

Hardware

The following list details are the hardware required for the project:

- Raspberry Pi2 Model B
- Breadboard
- Logitech USB camera (this component is not there in Adafruit kit)
- PIR motion sensor
- LED
- Resistor
- Adafruit female to male jumper wires (only five wires needed)

Software

Now, let's take a look at the software required for the project:

- Windows 10 PC
- Visual Studio 2015 Community Edition or Enterprise Edition
- Azure subscription
- Azure App Service (API App)
- Azure IoT Hub

WOW! eBook
www.wowebook.org

Now that we are equipped with all the hardware and software required for this project, let's get started with configuring the Raspberry Pi and sensors.

Setting up sensors

Before setting up the sensors, you need to know about the Raspberry Pi's GPIO pins. In the following diagram, you can...

WOW! eBook
www.wowebook.org

Device identity and registry with IoT Hub

The main purpose of device identity registration is to allow access to the device-facing endpoints. For each device, it creates resources in Azure IoT Hub, which enables device-to-cloud messages and also cloud-to-device messages, if needed.

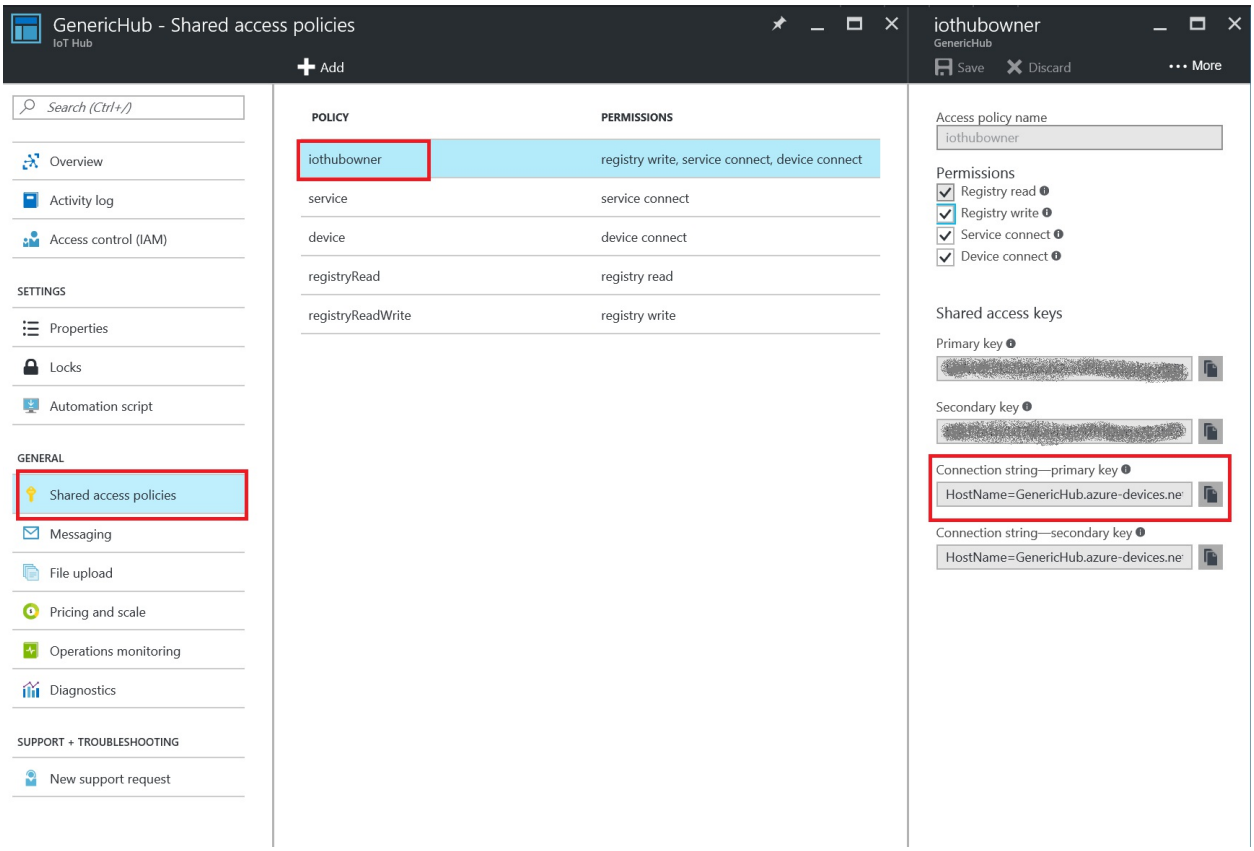
You can do this in different ways. Here, I will explain a technique using Device Explorer.

Using Device Explorer

You can use this tool to manage devices connected to your IoT Hub. For example, you can register a device with your IoT hub, monitor messages from your devices, and send messages to your devices. Device Explorer runs on your local machine and connects to your IoT hub in Azure. Following are the steps for using Device Explorer:

1. Download and install **Device Explorer** from https://github.com/Azure/azure-iot-sdks/releases.
2. Assuming that you already have IoT Hub in Azure--if not, check out the getting started link mentioned at the start of this chapter--you need an IoT Hub connection string, to get it, log in to your Azure Portal, open your IoT Hub, and under `Shared access policies`|`Iothubowner`, copy the `Connection string--primary key`.

WOW! eBook
www.wowebook.org



3. Open **`Device...`**

WOW! eBook
www.wowebook.org

Face API

The **Face API** will detect human faces; tag them as people; save people into groups based on similarity between images, such as images of the same person; and also identify the previously tagged people in images. The Face API can do face detection, identification, verification, similar face search, and face grouping. The Face API takes an image as an input, processes it to detect one or more human faces in that image, and returns face rectangles for all the faces in the image. It also returns face attributes, which contain features. The face features are age, gender, pose, smile, and facial hair, along with 27 landmarks for each face in the image. These predictions are based on the application of machine learning algorithms to facial features. For more information, refer to https://www.microsoft.com/cognitive-services/en-us/face-api/documentation/overview.

WOW! eBook
www.wowebook.org

Emotion API

The **Emotion API** analyzes faces to identify the emotions of a person. This API takes facial expression from an image as input and returns feelings/expressions for that face. If a user has already called the Face API on a particular image, they can submit the face rectangle from that image as an optional input. The emotions detected by the Emotion API are anger, contempt, disgust, fear, happiness, neutral, sadness, and surprise. For more information, refer to https://www.microsoft.com/cognitive-services/en-us/emotion-api/documentation.

Both the Face and Emotion APIs can also detect face attributes and emotions from a video. For a video, the Emotion API will detect the facial expressions of people in the video and return a summary of their emotions. In real-time scenarios, you can use these APIs to find out how a crowd responds to your speech or content.

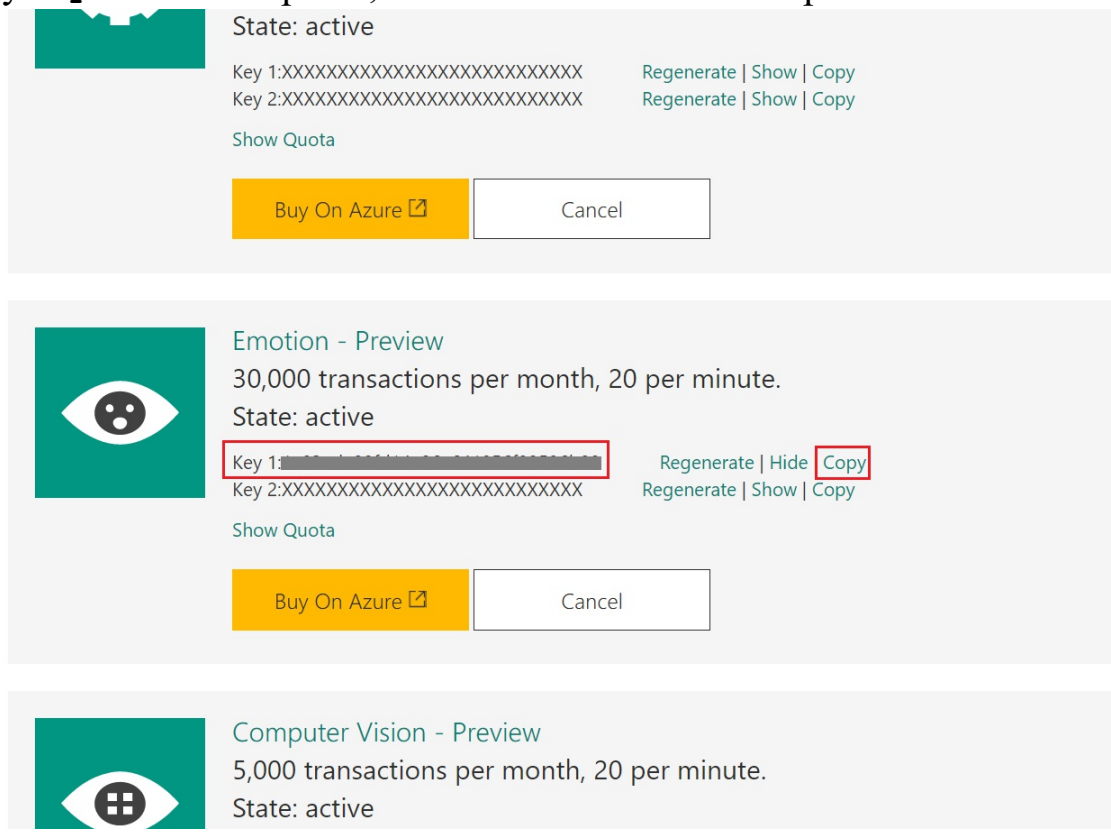
WOW! eBook
www.wowebook.org

Sign Up Microsoft Cognitive Services

These two APIs are part of the services offered in **Microsoft Cognitive Services** provided by Microsoft. It is currently free; to use these APIs, you first need to sign up for Microsoft Cognitive Services. Follow the sign-up process explained in Chapter 4, *Natural Speech and Intent Processing Bot Using Microsoft Cognitive Services* in the *Signing up for Microsoft Cognitive Services* section.

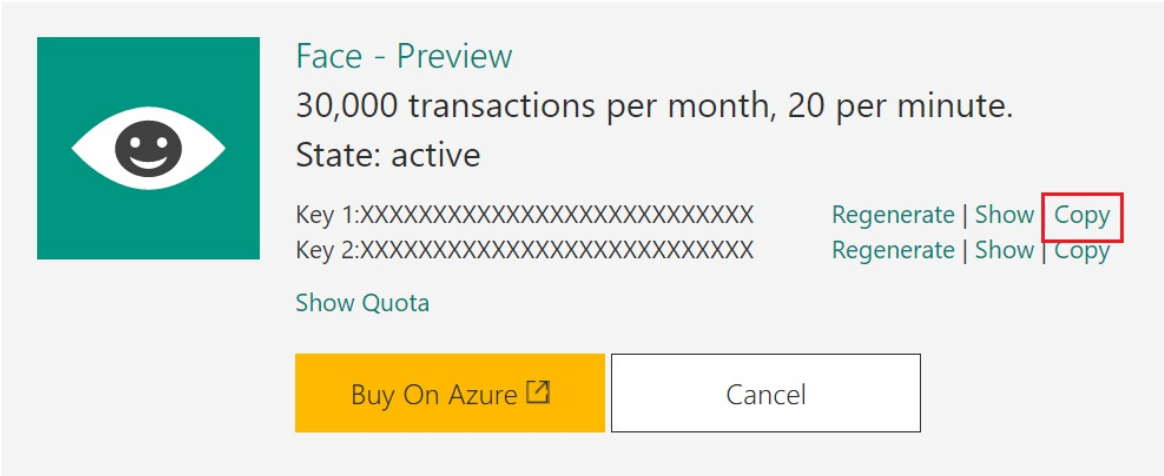
Once you complete the signup process, perform the following steps to get Face and Emotion API keys:

1. On the free trails page, navigate down to the Emotion API section and copy `Key 1` to a safe place; we will need it in later steps.



2. Similarly, copy the Face API `key 1` to a safe place.

WOW! eBook
www.wowebook.org

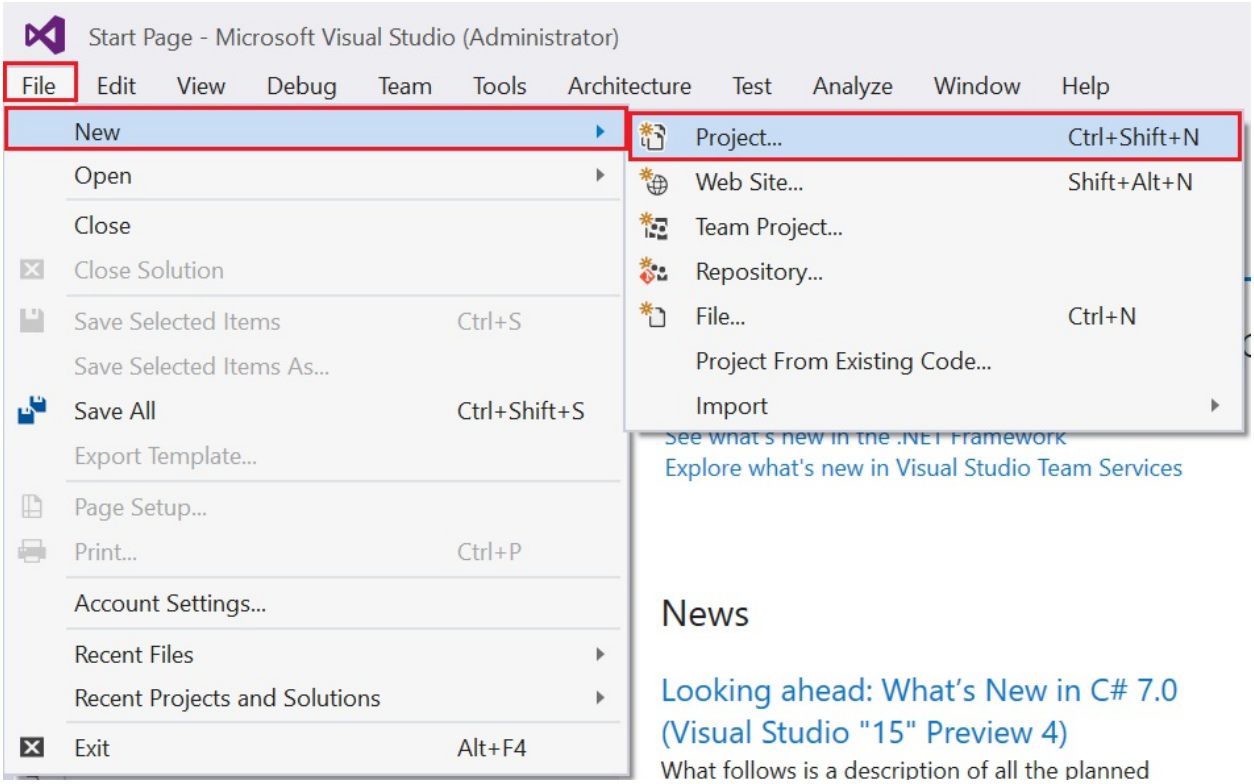


Development of facial expressions identification bot

Now, we are ready to build a bot application. Go to Visual Studio and create a new bot project using the Bot Application template, as follows:

1. Open Visual Studio, click on **`New`** | **`Project`**, and select **`Visual C#`** from the left-hand side template category. Then, in the templates section, you will see the **`Bot Application`** template.

WOW! eBook
www.wowebook.org



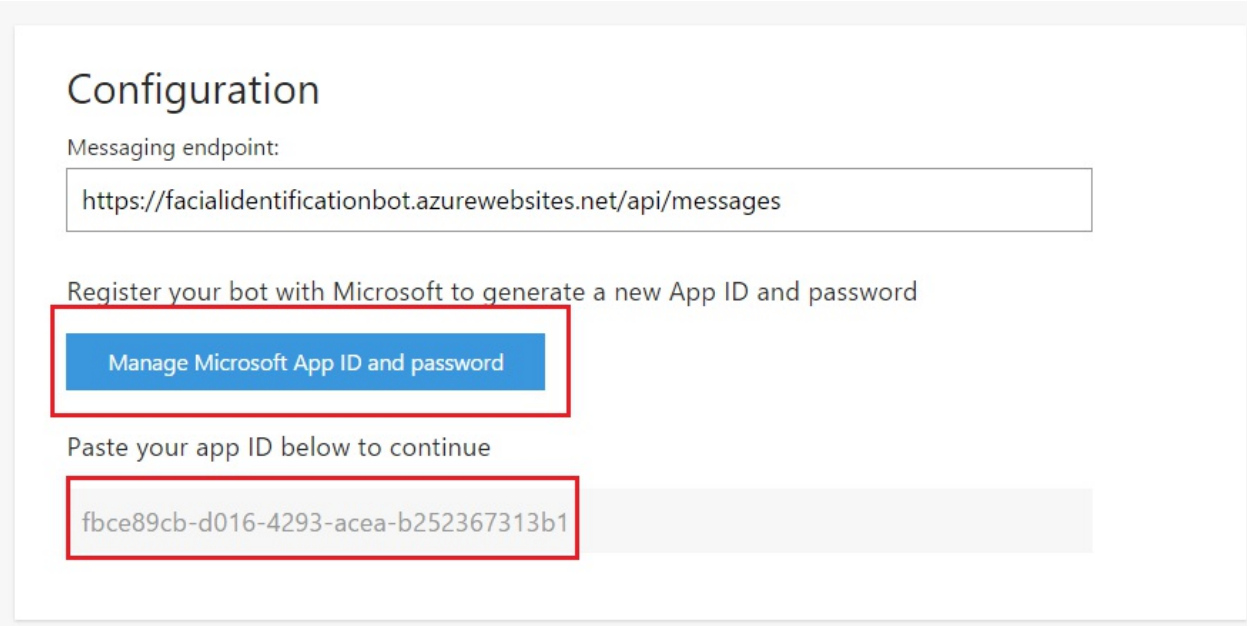
2. Select the **`Bot Application`** template, name the project, and then...

WOW! eBook
www.wowebook.org

Registering your Bot in Bot Framework

Next, you need to register your bot with Bot Framework so that you can integrate your bot with channels such as Slack, Skype, Facebook, and many more. In Chapter 9, *Publishing a Bot to Skype, Slack, Facebook, and the GroupMe Channel,* section *Registering your Bot with Microsoft Bot Framework*, we explained how to register your bot with dev.botframework, and refer to it to find out how to register your bot with Bot Framework. Following are the steps to register a bot:

1. Once you are done registering your bot, copy the *Microsoft App ID* and *Password*, which you generated when you registered your bot. You can also get these values from your existing bot--edit settings page under **`Configuration`** section--as shown in the following screenshot:



2. Now, go to bot project and open the `web.config` file. Under appSettings, update the `MicrosoftAppID`, `MicrosoftAppPassword`, and the `BotId`. The `BotId` is nothing but b*ot handle* name of your bot:

WOW! eBook
www.wowebook.org

```
<configuration>
  <appSettings>
    <!-- update these with your BotId, Microsoft App Id and your Microsoft App
    <add key="BotId" value="FacialIdentificationBot" />
    <add key="MicrosoftAppId" value="[redacted]" />
    <add key="MicrosoftAppPassword" value="[redacted]" />
    <add key="FaceKey" value="[redacted]"/>
    <add key="EmotionKey" value="[redacted]"/>
  </appSettings>
```

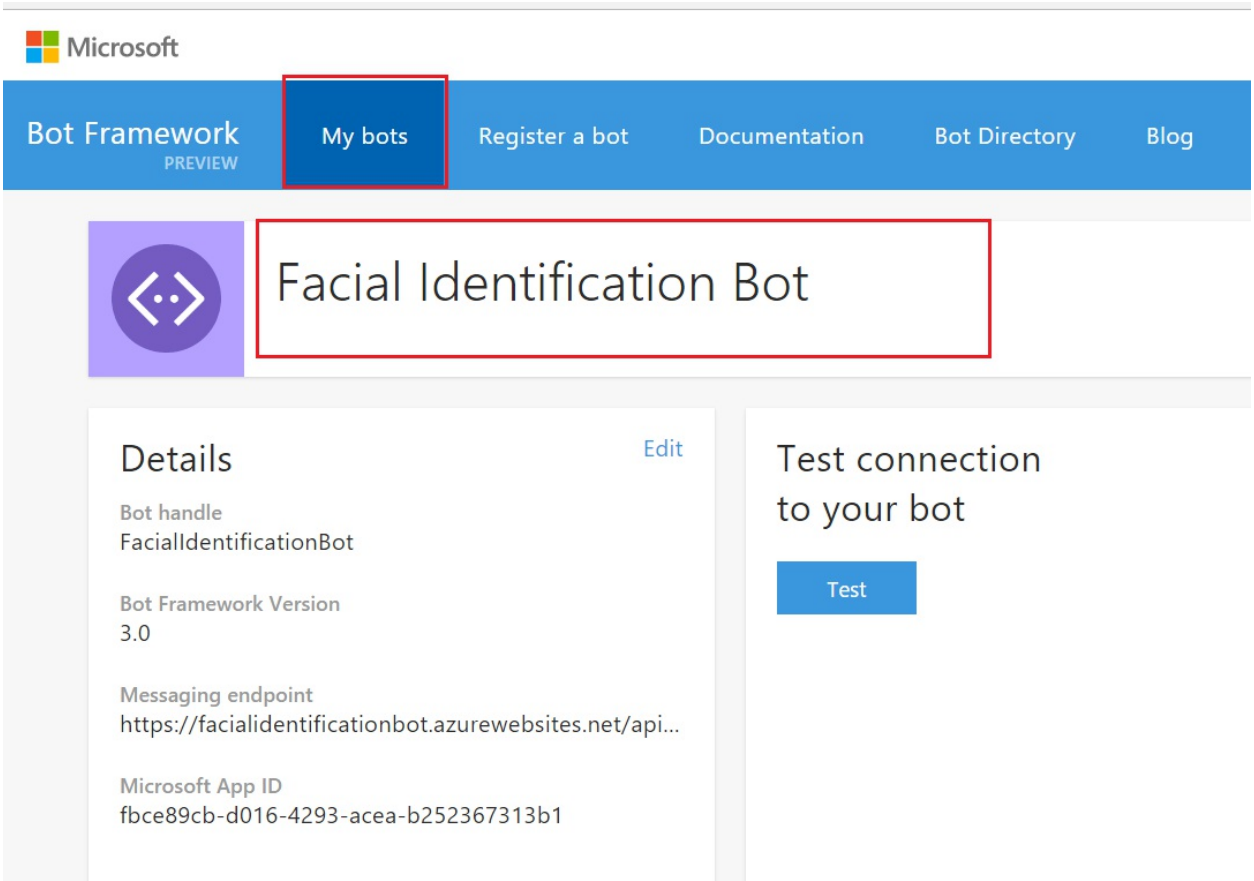
3. Next, go to the `MessageController.cs` file, `Post` method. We need to update the `Post` method to receive and support attachments/images from the user. Your bot should accept an...

WOW! eBook
www.wowebook.org

Configure Direct Line Channel

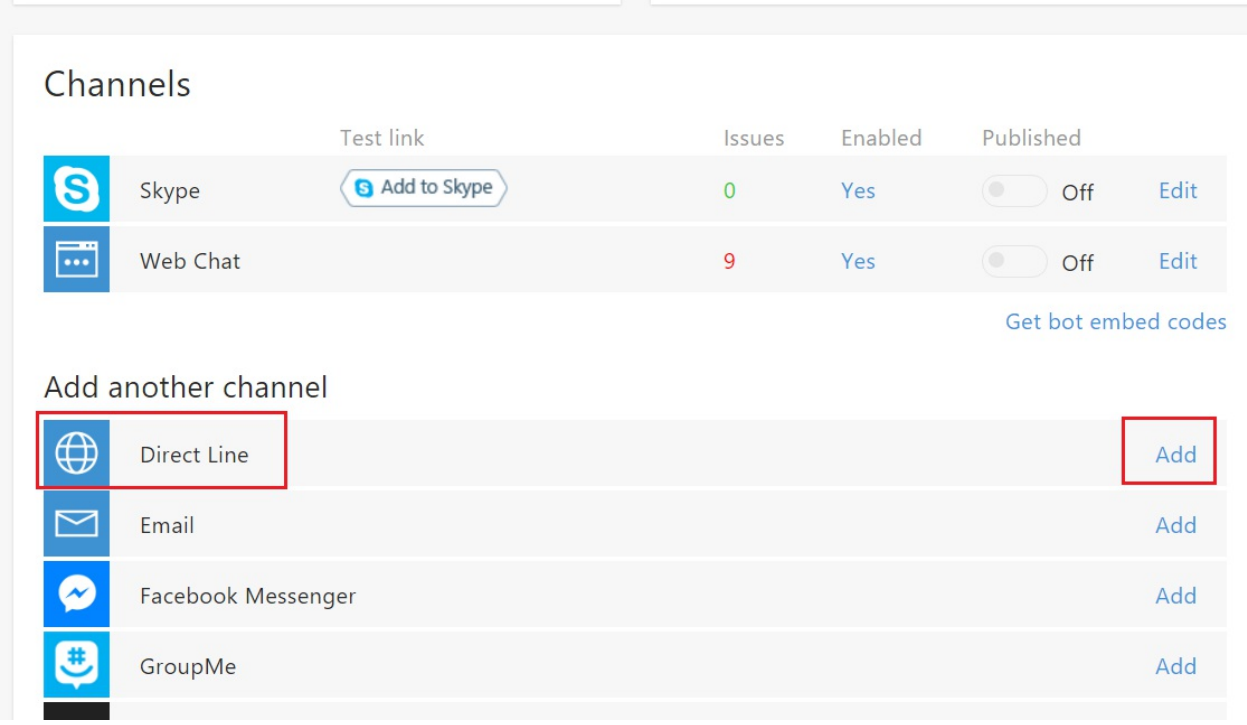
To call the bot from the IoT application, we need to configure Direct Line channel. Let's perform the following steps to do that:

1. Go to dev.botframework.com, click on the **`My Bots`** section, and select your bot:

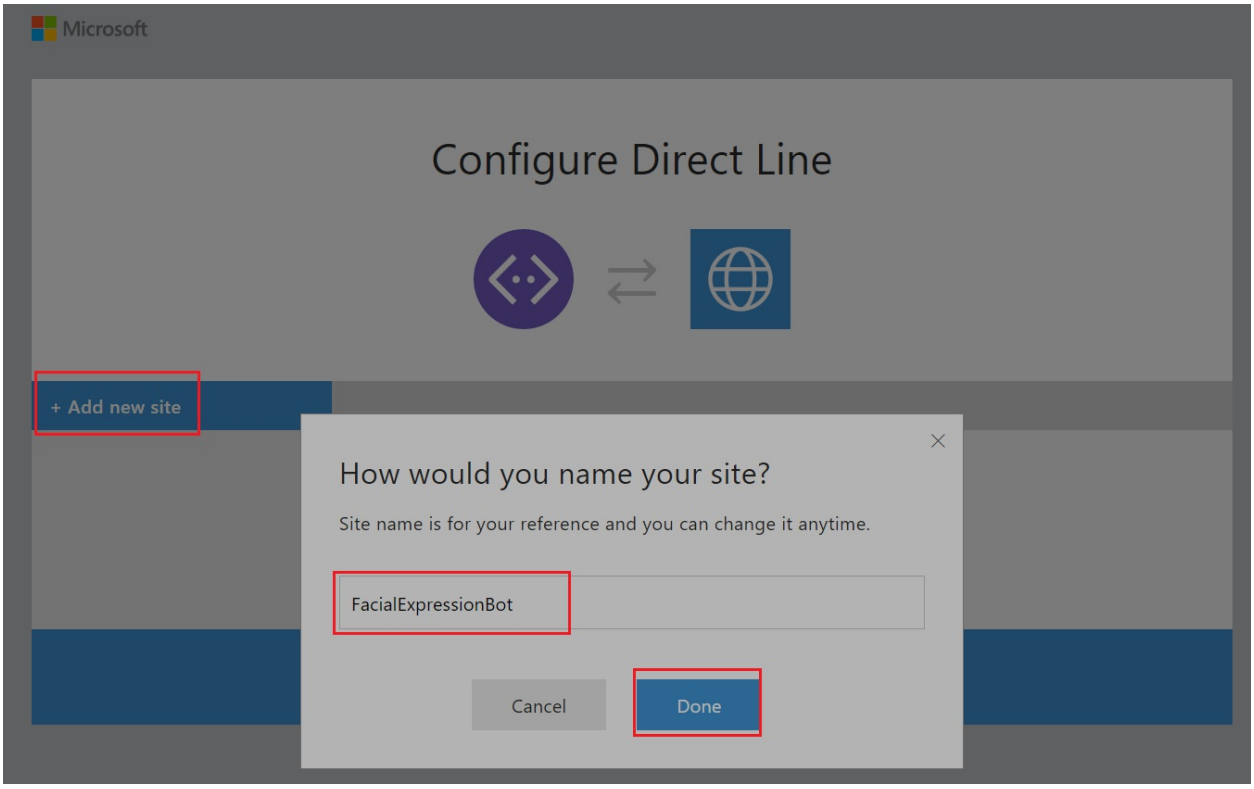


2. Under the **`Channels`** section, click on the **`Add`** option of the **`Direct Line`** channel. It opens a configuration page:

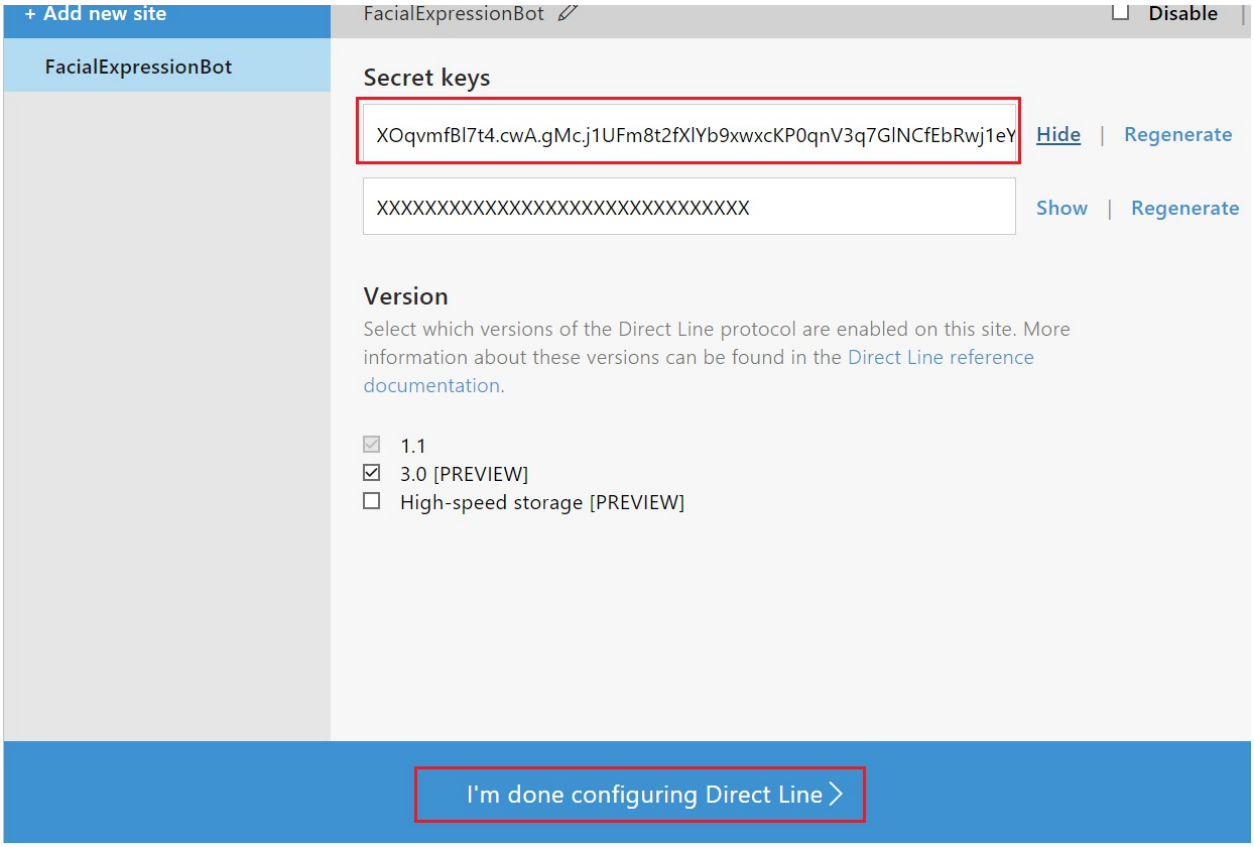
WOW! eBook
www.wowebook.org



3. On the Direct Line configuration page, click on the **`Add New Site`** option and add a name:



4. Next, copy the primary secret key we will use in later steps, as shown in the following figure, and finally click on **`I am done configuring Direct Line`**:



Next, you need to develop a **Universal Windows Platform** (**UWP**) app for Raspberry Pi2.

WOW! eBook
www.wowebook.org

Develop an UWP app for Raspberry Pi device

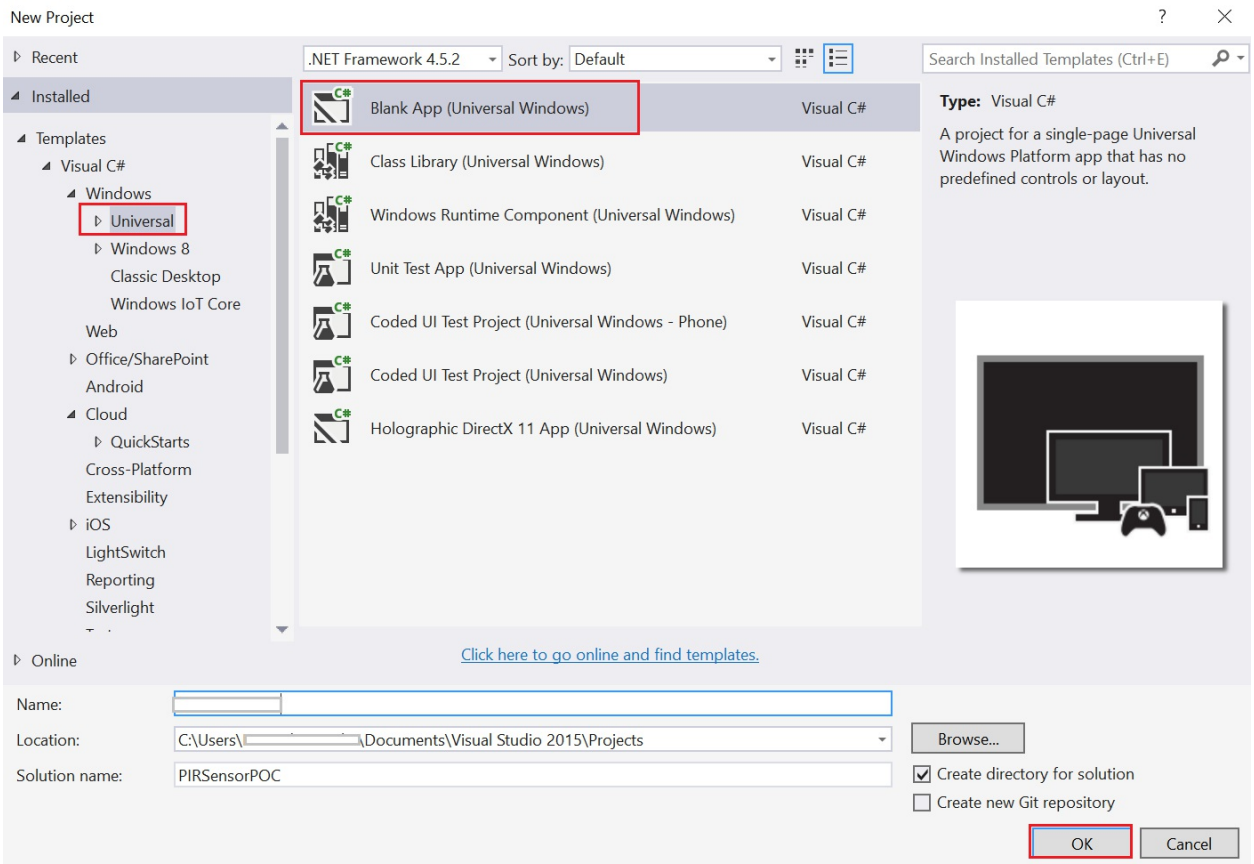
A UWP app is responsible for collecting sensor data, which means it captures a photo of a person and sends it to the bot, and the bot will process it and send back the results to the Raspberry Pi2. The processed results will be sent to IoT Hub.

Create an UWP App project

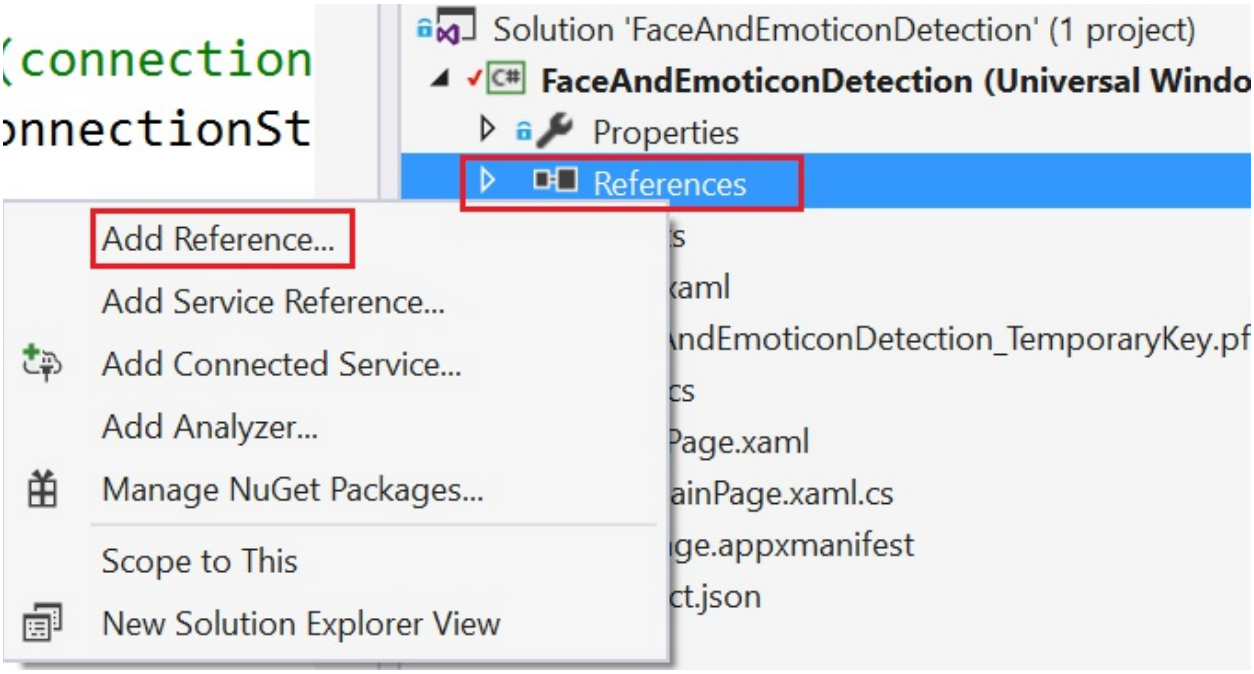
Now, let's take a look at the steps we need to follow to create the UWP app:

1. Open Visual Studio and create a new project by selecting Universal app as `Universal | Blank App` template, as follows:

WOW! eBook
www.wowebook.org

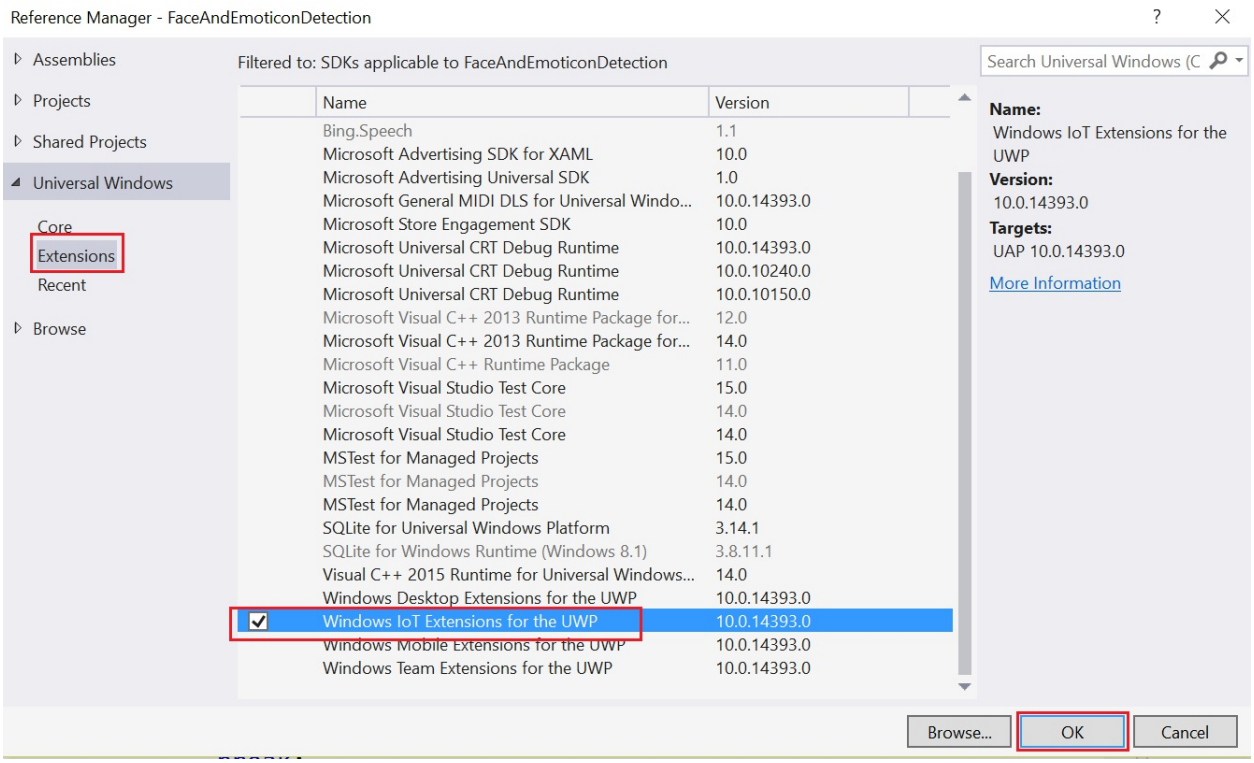


2. After the successful creation of the project, right-click on the **`Project`** and select the **`Add Reference`** option from the menu.



WOW! eBook
www.wowebook.org

3. Next, from Reference Manager windows select the **`Extensions`** category under Universal Windows and then select Windows IoT Extensions for the UWP.



4. Select the appropriate version of IoT Extensions; here I am using the `10.0.14393.0` version and my Raspberry Pi2 also has the same version of the Windows 10 IoT Core OS.

How to detect the motion of the object using PIR Sensor and How to define the LED states

To detect the motion of an object, an important sensor is PIR. For that, first you need to define a variable for...

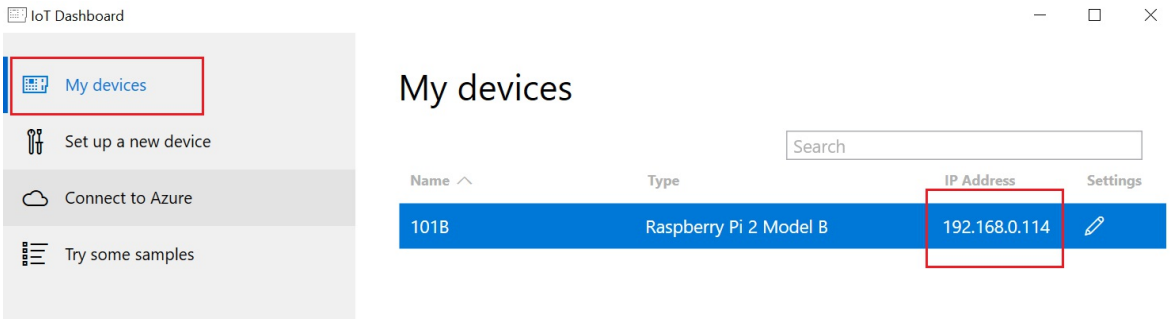
WOW! eBook
www.wowebook.org

Deploy Code in to Raspberry Pi

Now that we are done with the code for our project, let's look at the following steps to deploy code to the Raspberry Pi:

1. First, connect your Raspberry Pi to your developer machine using a LAN cable, or connect to your Wi-Fi router in the same network.
2. Download and install the *Windows 10 IoT Core Dashboard tool* from http://go.microsoft.com/fwlink/?LinkID=708576 .

3. Open it: after a few seconds it will show your Raspberry Pi device on the **`My Devices`** page shown in the following screenshot. Then, copy the IP address.

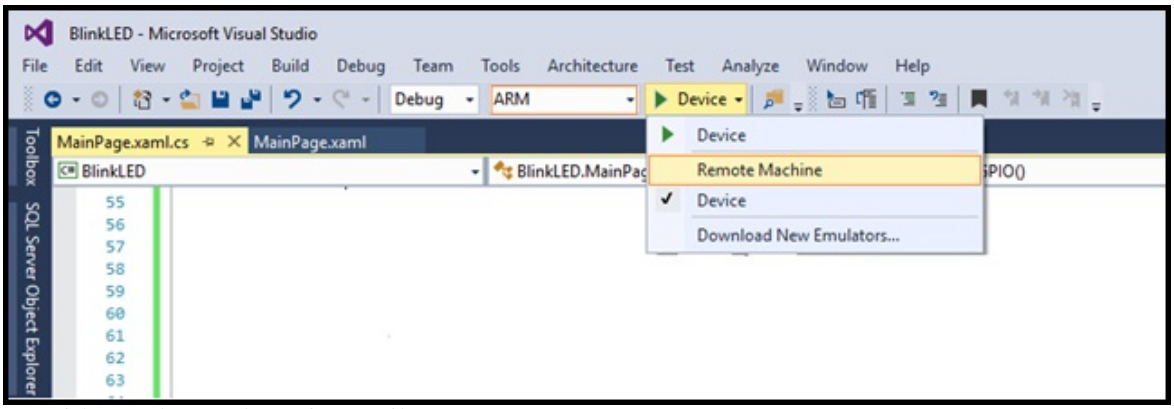


Before deploying the code, make sure that the registered device ID in Device Registry with IoT Hub step and your Raspberry Pi device name is the same. For example, in device registry step 1 given device Id as 101B and my Raspberry Pi device name 101B, both are same. So, we can identify and manage easily from IoT hub.

4. Open the Raspberry Pi UWP app solution in Visual Studio, and set the architecture in the toolbar dropdown to **`ARM`**.

5. Next, in the Visual Studio toolbar, click on the Local Machine dropdown and select **`Remote Machine`**, as shown in the following screenshot.

WOW! eBook
www.wowebook.org



6. At this point, Visual Studio...

WOW! eBook
www.wowebook.org

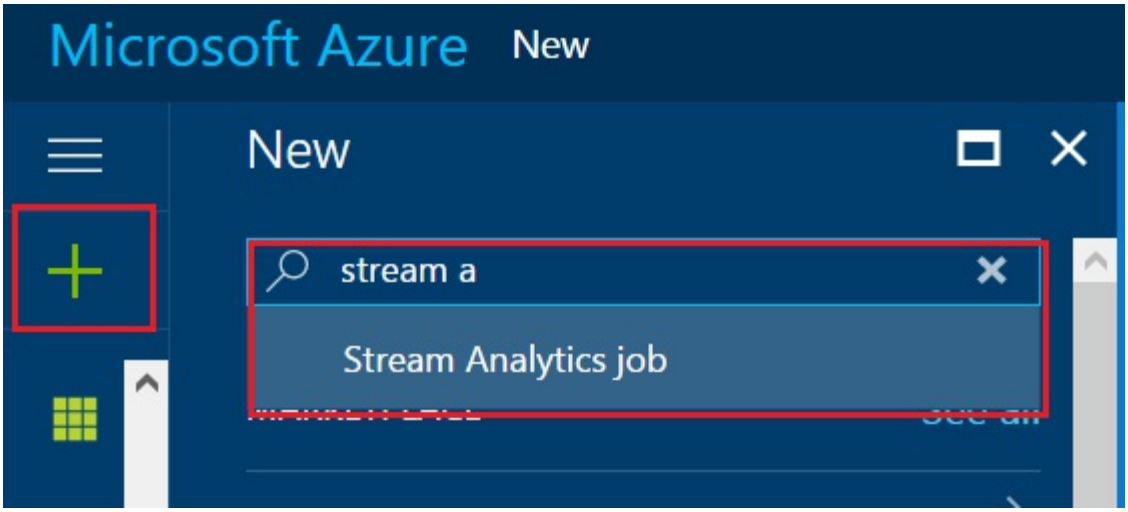
Show facial analytics data in Power BI

Power BI is a service that helps you to visualize your data in reports, charts, and interactive insights. It also has a set of software services, apps, and connectors, which all work together to help you turn your data into logical, consistent, and visually immersive. Power BI lets you easily connect to your data sources, visualize what you want, and also share with anyone. In the following steps, we will just give you a brief idea of how you can use Power BI in a real-time scenario, such as an IoT environment, where you will collect enormous amounts of data from your devices and want to see your data visually to take decisions. In this chapter, you will see how to show facial analytics data in Power BI.

Set up Azure Stream Analytics to send IoT Hub data to Power BI

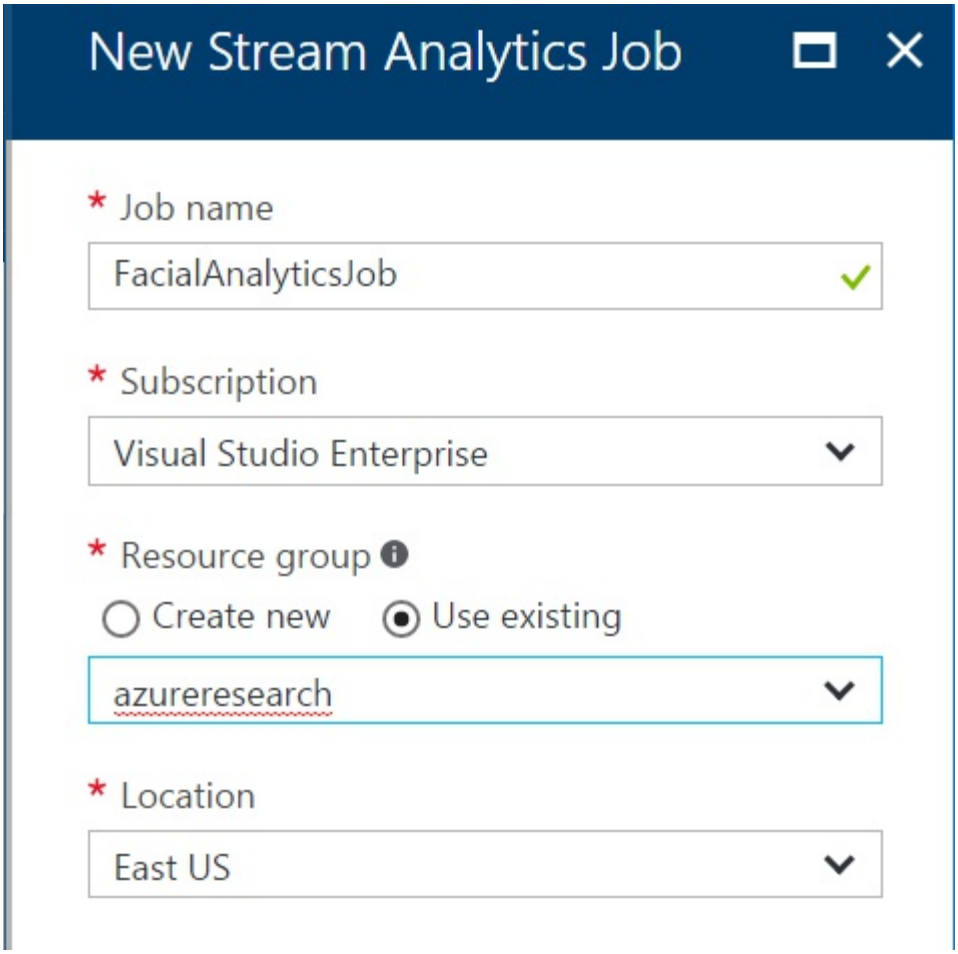
Let's follow these steps to set up Azure Stream analytics:

1. Create a `Stream Analytics Job` in azure, and log in to your Azure portal. Select `New` and search for `Stream Analytics job`.



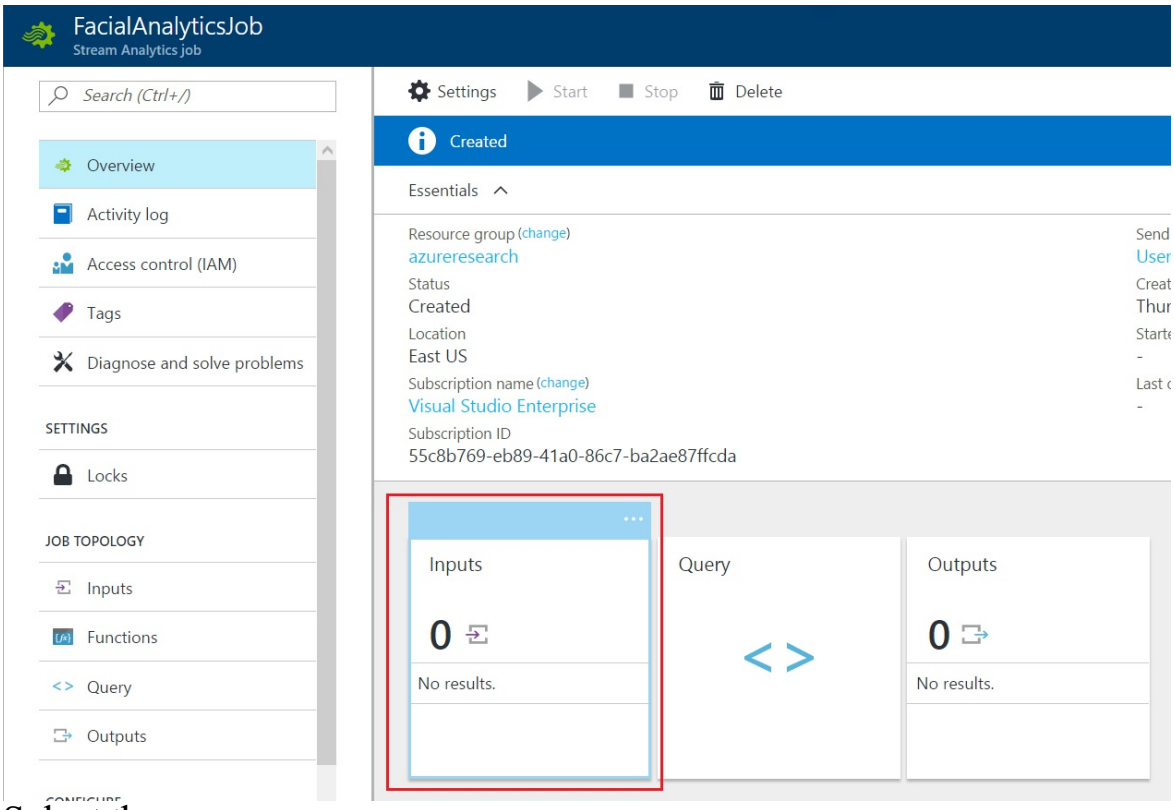
WOW! eBook
www.wowebook.org

2. Select, create, and enter **`Job name`**, and select **`Region`** and **`Resource group`**.



3. Open the newly created stream analytics, and click on **`Inputs`**.

WOW! eBook
www.wowebook.org



4. Select the **Add...**

WOW! eBook
www.wowebook.org

Summary

In this chapter, you have learned the following:

- IoT Hub: Azure IoT Hub is a fully managed service that helps us to enables reliable and secure bidirectional communications between millions of IoT devices
- Stream Analytics: It is a fully managed event-processing engine in the cloud
- Power BI: With the help of Power BI, you can analyze and visualize your important data and it always work with real-time data
- Storage Account: It provides one place to store all your data
- Cognitive Services: The Face API will detect human faces and tag them as people, also do face detection, identification, verification, similar face search, and face grouping
- Emotion API: It analyzes faces to identify the emotions of a person is feeling and also detects facial expressions in an image

In the next chapter, you will learn about registering bots with Bot framework, and also how to publish bots to Slack, Skype, GroupMe, and Facebook channels.

WOW! eBook
www.wowebook.org

Chapter 9. Publishing a Bot to Skype, Slack, Facebook, and the GroupMe Channel

In this chapter, we will use the Microsoft Bot Connector, which is a part of the Microsoft Bot Framework, as a way to create a single backend and then publish it to a bunch of different platforms called Channels as quickly as possible. The goal is to have the user input natural language and your bot to perfectly understand and execute the action your user wants.

At the confluence of the rise in messaging applications, advances in text and language processing, and mobile form factors, bots are emerging as a key area of innovation and excitement. Bots (or conversation agents) are rapidly becoming an integral part of your digital experience--they are a vital way for people to interact with a service or application, as is a website or a mobile experience. Developers writing bots all face the same problems--bots require basic I/O, they must have language and dialog skills, and they must connect to people--preferably in any conversation experience and language a person chooses. This book focuses on how to solve these problems using the Microsoft Bot...

WOW! eBook
www.wowebook.org

Publishing bots to various channels

Let's look at publishing bots to various channels in the following image:

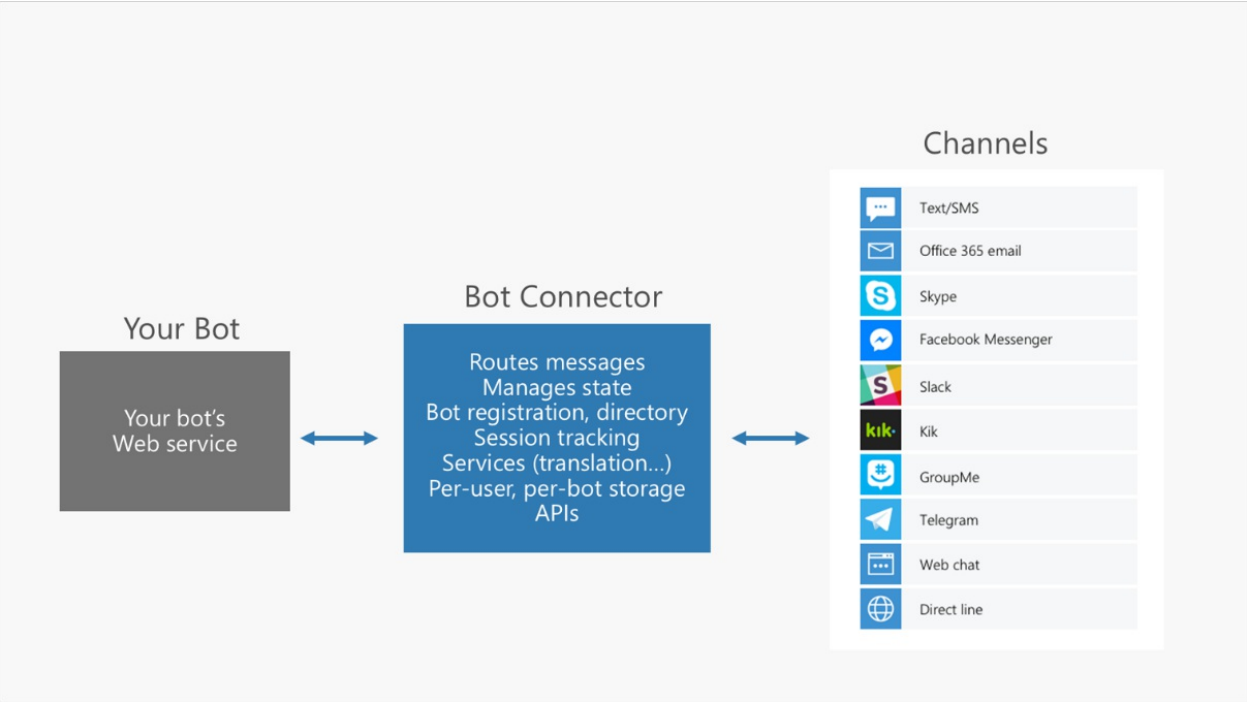


Figure1: How a single bot can be published to various channels through the bot connector

In order to publish your bot to the available directories, you need to do the following three things:

1. Publish your bot application to Microsoft Azure web app.
2. Connect your bot to at least one channel that appears in the Bot Framework.
3. Select Publish in the bot dashboard.

Publishing your bot application to Microsoft Azure web app

We use Microsoft Azure to host the bot application. To publish your bot application, you will need a Microsoft Azure

WOW! eBook
www.wowebook.org

subscription. You can get a free trial from https://azure.microsoft.com/en-us/.

In the preceding chapter, we created a bot, and now we are ready to publish the bot. By default, the bot should be published as a Microsoft Azure App Service. When publishing, keep track of the URL you chose because we'll need it when we have to register the Bot Framework endpoint, which is nothing but your bot messages, API URL. There are a few extra steps that you have to do the first time you publish, but you only have...

WOW! eBook
www.wowebook.org

Registering your bot with Microsoft Bot Framework

Registering your bot tells the connector how to call your bot's web service. Note that the Microsoft App ID and Microsoft App Password are generated when your bot is registered with the Microsoft Bot Framework Connector; the App ID and AppSecret are used to authenticate the conversation and allows the developer to configure their bot with the channels they'd like to be visible on. Let's look at the following steps to register your bot:

1. Go to the Microsoft Bot Framework portal at https://www.botframework.com and sign in with your Microsoft account.
2. Register an agent.

3. Click on the Register a bot button and fill out the Bot Profile form. You have to enter the name of your bot handle, which means a unique name that will be used in the bot connection, and then enter a description:

WOW! eBook
www.wowebook.org

Bot Framework PREVIEW | My bots | Register a bot | Documentation | Bot Directory | Blog

Tell us about your bot

Bot profile

Icon
Upload custom icon
30K max, png only

Name: *
Name

Bot handle: *
Type in your Bot handle

Description: *
Description

Configuration

Messaging endpoint:
https URL

Register your bot with Microsoft to generate a new App ID and password

Create Microsoft App ID and password

Paste your app ID below to continue

Microsoft App ID from the Microsoft App registration portal

Configuration

WOW! eBook
www.wowebook.org

Under the Configuration section, enter your published bot service endpoint that you copied during the Azure deployment step in Messaging endpoint, and don't forget that when using the bot application template, you'll need to extend the URL you pasted in which the path to the endpoint at

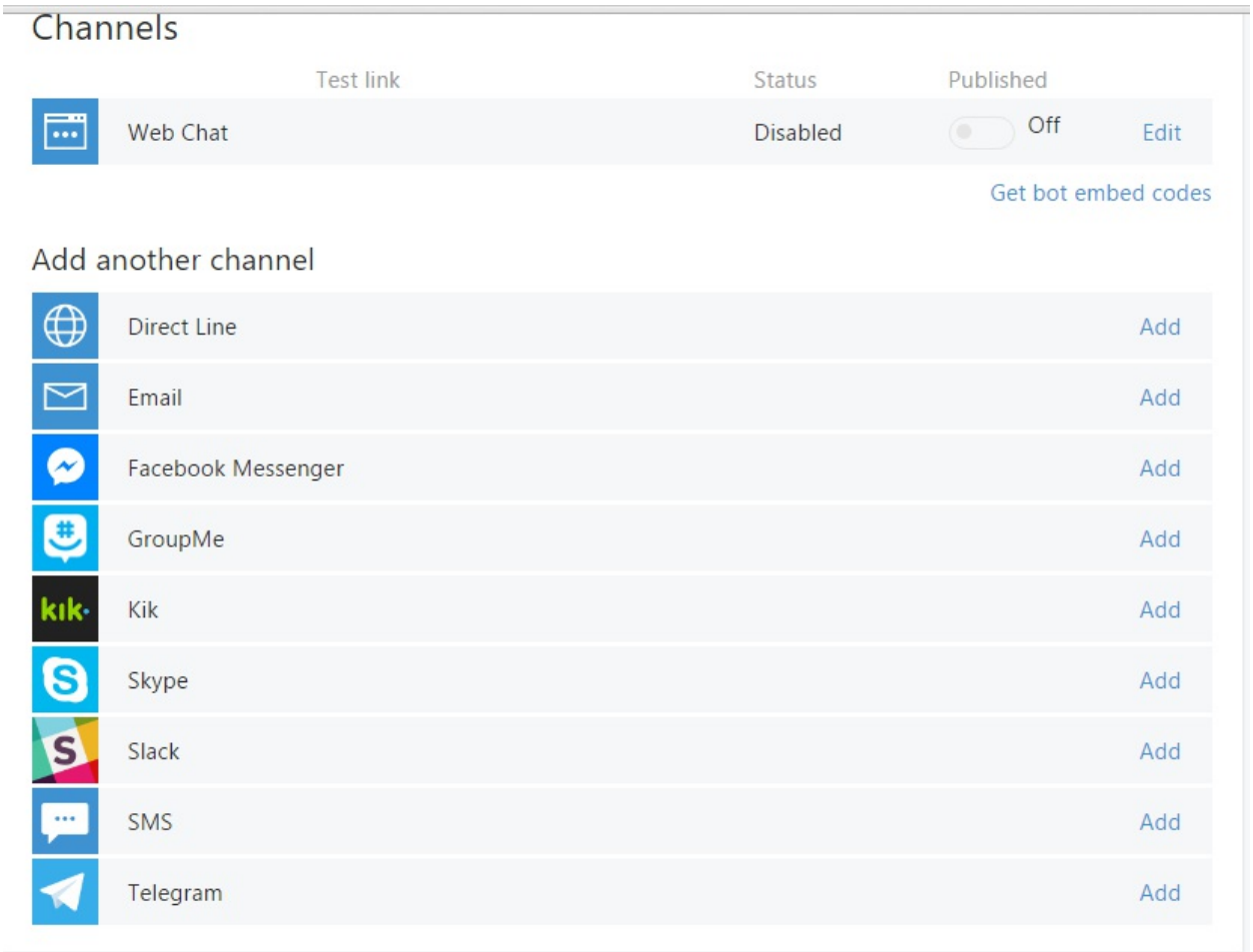
WOW! eBook
www.wowebook.org

Configuring channels

Channels are a mechanism of connecting the bot with the various communication platforms and making the channels available on those platforms.

Now that you have a bot up and running, you'll want to configure it for one or more channels that your users are using. Configuring channels is a combination of Microsoft Bot Framework workflow and conversation service workflow, and it is unique for each channel you wish to configure:

- On the right-hand side of the dashboard, you can see all the channels you can connect with your bot
- You can connect your bot to the services that I mentioned before, such as SMS, Telegram, Slack, and so on



WOW! eBook
www.wowebook.org

Configuring your bot with Slack

In this example, we will show you how to configure your bot to Slack. All channels in Microsoft Bot Framework require you to bring your own account model, so you can sign up each of these services on your own. You take your username and password for them and give them to the Bot Framework, which stores them in encrypted format, but it allows us to communicate on behalf of your bot. What this means is that if you already have an account for your bots, you can...

WOW! eBook
www.wowebook.org

Summary

In this chapter, we have learned the following:

- Registering a bot: Once registered, we use the dashboard to test the bot to ensure that it is talking to the connector service and/or use the web chat control, and an auto-configured channel, to experience what users will experience when conversing with the bot
- Connecting to channels: Connect the bot to the conversation channels such as Skype, Slack, and/or Facebook Messenger using the channel configuration page
- Testing bot: Test the bot's connection to the Bot Framework and try it out using the web chat control
- Publishing bot: We get to publish the bot
- Measuring bot: We get to learn how to link the bot to Azure Application Insights analytics directly from the bot dashboard in the Bot Framework website
- Managing a bot: Once registered and connected to channels, we can manage the bot via the bot's dashboard in the Bot Framework Developer Portal

WOW! eBook
www.wowebook.org