Windows 10

Hands-on lab

Lab: Launching Apps with Speech Commands

September 2015


Microsoft

CONTENTS

Overview

Windows 10 apps can leverage the Cortana interface in those regions where it is available to interact with users through convenient and customizable voice commands. In addition to accessing system features, Cortana can launch your app in the foreground or interact with app data in the background.

App-specific voice commands start with a prefix, usually the app name or a keyword, to allow for infinite combinations without needing to disambiguate from other apps that may have similar commands. You can define options to speak the app name before or after the command and choose which behaviors to implement when your app is launched.

In this lab, you will create a voice command definition file and add commands to launch your app and customize its appearance.

Objectives

This lab will show you how to:

- Create a voice command definition file
- Launch your app using a voice command
- Use a switch to target incoming voice commands
- Pass information from a voice command to the view
- Change the app's appearance based on an incoming voice command
- Run a background task from a voice command
- Return written and spoken responses to Cortana from your app

System requirements

You must have the following to complete this lab:

- Microsoft Windows 10 set to one of the languages and regions where Cortana is supported

    - In September 2015, Cortana is available in the following countries/regions: China, France, Germany, Italy, Spain, United Kingdom, and United States. Cortana is available in these languages: Chinese (Simplified), English (U.K.), English (U.S.), French, Italian, German, and Spanish
    - To use Cortana, all these settings must be set to the same language:
        - Languages (this is your device language)
        - Speech language (language pack must be installed)
        - Country or region
- Microsoft Visual Studio 2015

Setup

You must perform the following steps to prepare your computer for this lab:

1. Install Microsoft Windows 10.
2. Install Microsoft Visual Studio 2015.

Exercises

This Hands-on lab includes the following exercises:

1. Launching with Voice Commands
2. Use a Voice Command to Change the App's Appearance
3. Respond to a Voice Command with a Background Task

Estimated time to complete this lab: **30 to 45 minutes**.

Exercise 1: Launching with Voice Commands

Voice commands provide a convenient, hands-free alternative for launching your app. To implement voice commands, you will create a simple schema with a command to launch your app and the corresponding response from Cortana. You will register the commands with Cortana and use the OnActivated start case to initiate the activation of your app in the foreground.

Task 1 – Create a blank Universal Windows app

We will begin by creating a project from the Blank App template.

1. In a new instance of Visual Studio 2015, choose **File > New> Project** to open the New Project dialog. Navigate to **Installed > Templates > Visual C# > Windows > Universal** and select the **Blank App (Universal Windows)** template.

2. Name your project **SpeechRecognition** and select the file system location where you will save your Hands-on Lab solutions. We have created a folder in our **C:** directory called **HOL** that you will see referenced in screenshots throughout the labs.

   Leave the options selected to **Create new solution** and **Create directory for solution**. You may deselect both **Add to source control** and **Show telemetry in the Windows Dev Center** if you don't wish to version your work or use Application Insights. Click **OK** to create the project.

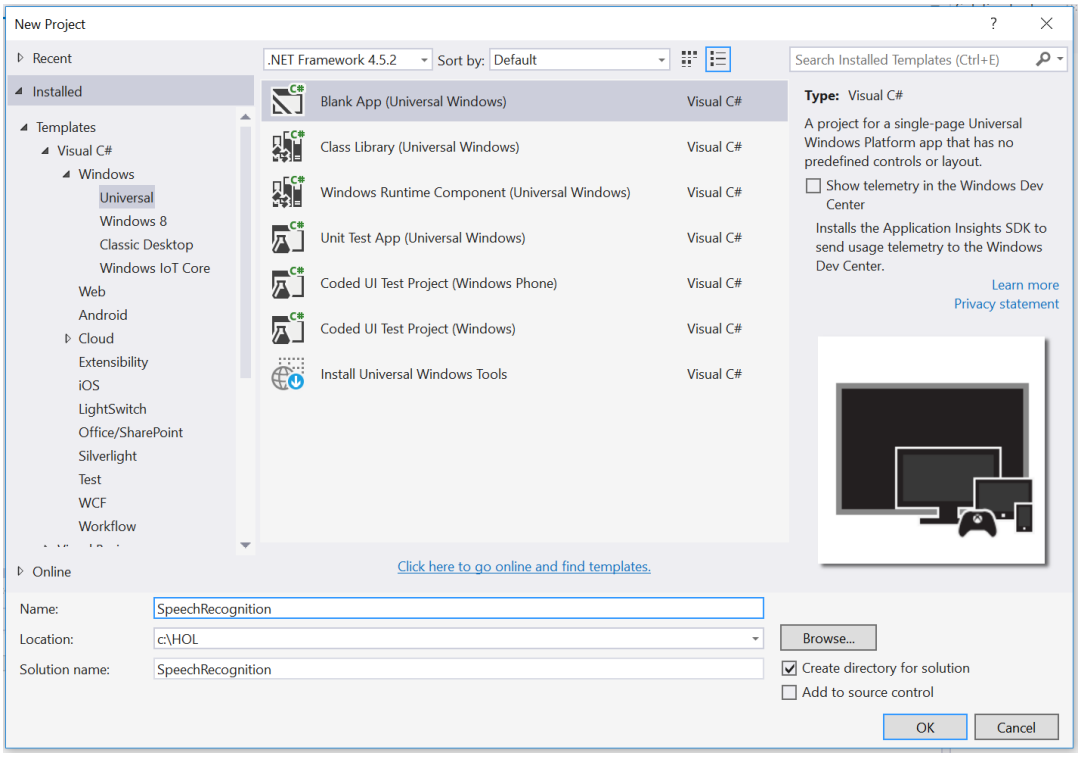

Figure 1

Create a new Blank App project in Visual Studio 2015.

3. Set your Solution Configuration to **Debug** and your Solution Platform to **x86**. Select **Local Machine** from the Debug Target dropdown menu.

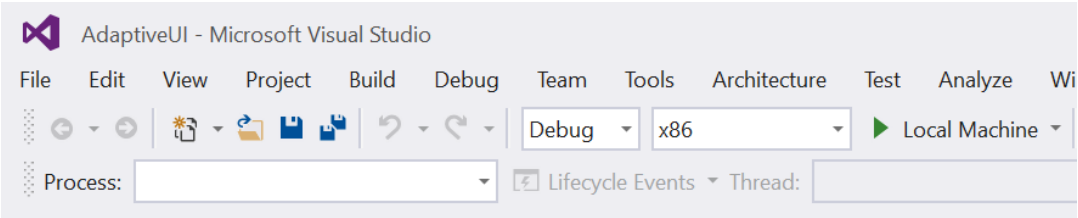

Figure 2

Configure your app to run on the Local Machine.

4. Build and run your app. You will see a blank app window with the frame rate counter enabled by default for debugging.

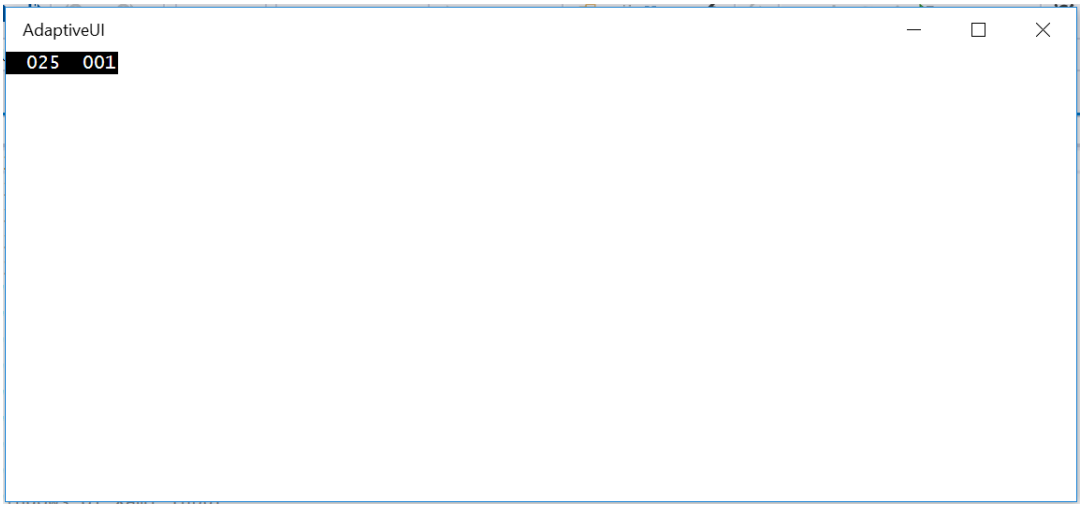

Figure 3

The blank universal app running in Desktop mode.

> **Note:** The frame rate counter is a debug tool that helps to monitor the performance of your app. It is useful for apps that require intensive graphics processing but unnecessary for the simple apps you will be creating in the Hands-on Labs.
>
> In the Blank App template, the preprocessor directive to enable or disable the frame rate counter is in **App.xaml.cs**. The frame rate counter may overlap or hide your app content if you leave it on. For the purposes of the Hands-on Labs, you may turn it off by setting **this.DebugSettings.EnableFrameRateCounter** to **false**.

5. Return to Visual Studio and stop debugging.

Task 2 – Create the voice commands definition file

The voice command schema is defined in an XML file. In this task, you will create a simple schema with a command to handle launching the app.

1. Right-click on your project name and choose **Add > New Item**.

2. Select the XML file type and give it the name **VoiceCommands.xml**.

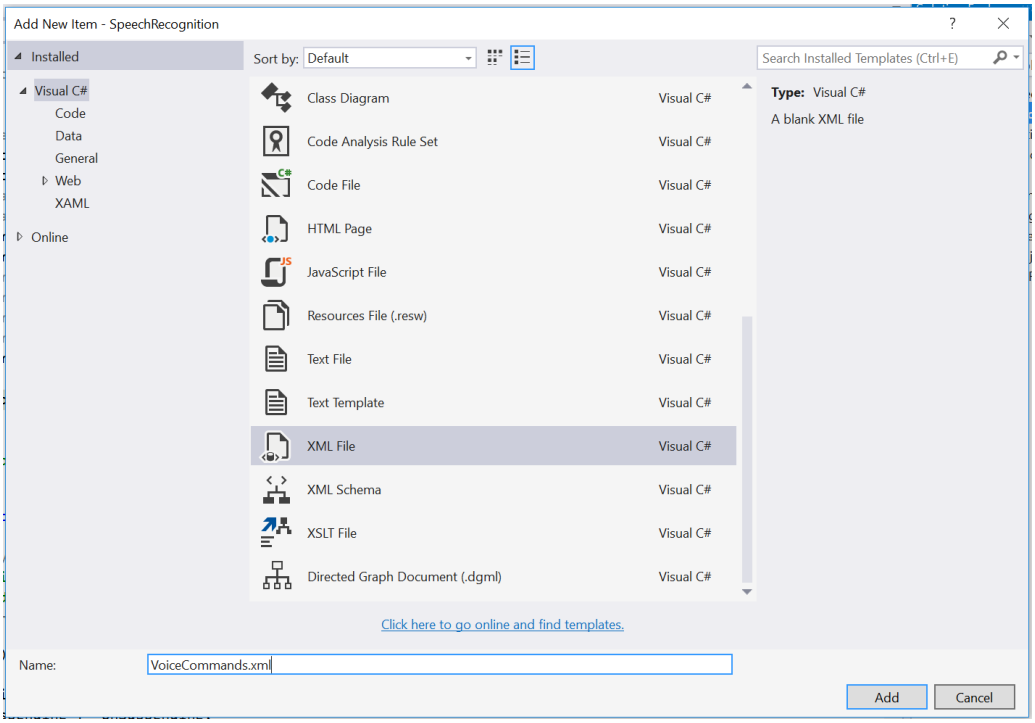

Figure 4

Create the voice commands xml file.

3. Open VoiceCommands.xml. After the xml version header, add the VoiceCommands element and the xmlns attribute referencing the voice commands v1.2 schema. Add the command set element for en-us and an additional command set for that will contain voice commands for your supported language (if different).

XML

```
<?xml version="1.0" encoding="utf-8" ?>
<VoiceCommands xmlns="http://schemas.microsoft.com/voicecommands/1.2">
    <CommandSet xml:lang="en-us" Name="HoLCommandSet_en-us">

    </CommandSet>

    <!-- Optional second command set for supported languages -->
    <!-- <CommandSet xml:lang="de-de" Name="HoLCommandSet_de-de">
    </CommandSet> -->
```

```
</VoiceCommands>
```

Note: We have added **en-us** as the language for this example, but you may add your own command set. For instance, the language tag for Germany would be **xml:lang="de-de"**. The list of regions and languages that Cortana supports is at http://windows.microsoft.com/en-us/windows-10/cortanas-regions-and-languages

If you choose to add another command set in a supported language, make sure to add an equivalent command in that language every time you add one to the en-us command set throughout this demo.

4. Add a command prefix to your definition. The command prefix is the word or phrase your users can speak to tell the system to start listening for commands from your app.

XML

```
<?xml version="1.0" encoding="utf-8" ?>
<VoiceCommands xmlns="http://schemas.microsoft.com/voicecommands/1.2">
  <CommandSet xml:lang="en-us" Name="HoLCommandSet_en-us">
    <CommandPrefix> Hands on Labs, </CommandPrefix>


</CommandSet>
```

Note: The comma after the command prefix is optional. If you choose to add it, it will indicate a slight pause between the command prefix and the command itself. Cortana will also briefly display the prefix and the command exactly as they are written here when the command is recognized.

If you chose to add an additional voice command set, add a command prefix in that language to the appropriate command set.

5. Our goal is to launch the app from a voice command. Create a launch command with a **ListenFor** element that defines the word or words that will be recognized for this command, a **Feedback** element that defines the confirmation text that Cortana will speak back at the user when the command is recognized, and a **Navigate** element. Add an **Example** element for this new command to the containing **CommandSet**.

XML

```
<?xml version="1.0" encoding="utf-8" ?>
<VoiceCommands xmlns="http://schemas.microsoft.com/voicecommands/1.2">
  <CommandSet xml:lang="en-us" Name="HoLCommandSet_en-us">
    <CommandPrefix> Hands on Labs, </CommandPrefix>
    <Example> Launch </Example>

    <Command Name="LaunchApp">
      <Example>launch</Example>
      <ListenFor>launch</ListenFor>
```

```
      <Feedback>Opening your speech recognition app</Feedback>
      <Navigate />
    </Command>

</CommandSet>
</VoiceCommands>
```

Note: The Navigate element signifies that the app will launch in the foreground. The alternative to launching in the foreground is to define a WinRT component to handle behind-the-scenes interactions with app data through Cortana. You can learn more about Voice Command Definitions at https://msdn.microsoft.com/en-us/library/windows/apps/dn722331.aspx

Task 3 – Install the voice command definitions

We will install the voice command definitions (VCD) in the **OnLaunched** override. There is no simple way to test if the VCD has been imported, or if it is the most recent version, so it is convenient to install it whenever the app is launched.

1. Open **App.xaml.cs** and add the **async** keyword to the **OnLaunched** override and add the following lines:

C#

```
protected override async void OnLaunched(LaunchActivatedEventArgs e)
{

    //#if DEBUG
    //            if (System.Diagnostics.Debugger.IsAttached)
    //            {
    //                this.DebugSettings.EnableFrameRateCounter = true;
    //            }
    //#endif

    var storageFile = await
Windows.Storage.StorageFile.GetFileFromApplicationUriAsync(new Uri("ms-
appx:///VoiceCommands.xml"));

    await
Windows.ApplicationModel.VoiceCommands.VoiceCommandDefinitionManager.InstallCo
mmandDefinitionsFromStorageFileAsync(storageFile);

    Frame rootFrame = Window.Current.Content as Frame;
```

Note: The voice command definition (VCD) will be installed the first time the app is launched with a start kind of **Launched**. You can open your app from the Start menu to ensure the voice commands are registered.

Task 4 – Handle voice command activation

When your app is launched via voice command, it has a start kind of **Activated**. Accordingly, we will handle incoming voice commands in the **OnActivated** override. In this task, you will create a switch to evaluate the **LaunchApp** command and navigate to the MainPage view if the command has been used to start the app.

1. Create an **OnActivated** override in **App.xaml.cs**. A voice command launch has a start kind of **OnActivated**, so you will handle the incoming voice command here.

   C#

   ```
   protected override void OnActivated(IActivatedEventArgs args)
   {
       base.OnActivated(args);
   }
   ```

2. Add a switch to handle the **ActivationKind.VoiceCommand** case and call a method named **HandleVoiceCommand**. You will create the **HandleVoiceCommand()** method in the next step.

   C#

   ```
   protected override void OnActivated(IActivatedEventArgs args)
   {
       switch (args.Kind)
       {
           case ActivationKind.VoiceCommand:
               HandleVoiceCommand(args);
               break;

           default:
               break;
       }
       base.OnActivated(args);
   }
   ```

3. Add the **System.Diagnostics** namespace to **App.xaml.cs**.

   C#

   ```
   using System.Diagnostics;
   ```

4. Create the **HandleVoiceCommand()** method. This method determines the incoming voice command and implements a switch based on the voice command name. You defined the **LaunchApp** command in your voice command definitions file in Task 2.

   C#

   ```
   private void HandleVoiceCommand(IActivatedEventArgs args)
   {
       var commandArgs = args as VoiceCommandActivatedEventArgs;
       var speechRecognitionResult = commandArgs.Result;
       var command = speechRecognitionResult.Text;
   ```

```
    var voiceCommandName = speechRecognitionResult.RulePath[0];
    var textSpoken = speechRecognitionResult.Text;

    Debug.WriteLine("Command: " + command);
    Debug.WriteLine("Text spoken: " + textSpoken);

    switch (voiceCommandName)
    {
        case "LaunchApp":
            break;
        default:
            break;
    }
}
```

Note: The Debug.WriteLine options will be useful later when debugging code. You may add additional output messages if you would like to see the results of commandArgs and speechRecognitionResult. We will explore debugging apps launched by speech commands later in this task.

5. To successfully launch the app and navigate to a page, we will need to recreate some of the behavior that currently lives in the OnLaunched override. The Blank App template creates the root frame and activates the window when the app is launched, but does not provide this behavior for OnActivated cases. Add the code to handle these start up tasks.

C#

```
protected override void OnActivated(IActivatedEventArgs args)
{
    Frame rootFrame = Window.Current.Content as Frame;

    if (rootFrame == null)
    {
        // Create a Frame to act as the navigation context and navigate to the
first page
        rootFrame = new Frame();

        rootFrame.NavigationFailed += OnNavigationFailed;

        // Place the frame in the current Window
        Window.Current.Content = rootFrame;
    }


    switch (args.Kind)
    {
        case ActivationKind.VoiceCommand:
        HandleVoiceCommand(args);
```

```
        break;

        default:
        break;
    }

    Window.Current.Activate();

    base.OnActivated(args);
}
```

Note: To avoid duplication in a real-world app, you may want to create common startup code that will run for both launched and activated apps. Template10 demonstrates a more unified way of handling these essential steps on app startup. For more on Template10, visit https://github.com/Windows-XAML/Template10

6. Pass the root frame into the **HandleVoiceCommand** method in addition to **args**. You will need the context of the rootFrame to navigate to a page.

C#

```
switch (args.Kind)
    {
        case ActivationKind.VoiceCommand:
        HandleVoiceCommand(args, rootFrame);
        break;

        default:
        break;
    }
```

7. In the **HandleVoiceCommand** method, add the incoming **frame** parameter and use it to navigate to the **MainPage** view when the **LaunchApp** command is detected.

C#

```
private void HandleVoiceCommand(IActivatedEventArgs args, Frame frame)
{
    var commandArgs = args as VoiceCommandActivatedEventArgs;
    var speechRecognitionResult = commandArgs.Result;
    var command = speechRecognitionResult.Text;

    var voiceCommandName = speechRecognitionResult.RulePath[0];
    var textSpoken = speechRecognitionResult.Text;

    switch (voiceCommandName)
    {
        case "LaunchApp":
            frame.Navigate(typeof(MainPage));
```

```
            break;
        default:
            break;
    }
}
```

8. Open MainPage.xaml and add a page title. The text will help to determine that navigation has taken place when your app is launched.

XAML

```
<Grid Background="{ThemeResource ApplicationPageBackgroundThemeBrush}">
    <TextBlock Text="Speech Recognition and Voice Commands" FontWeight="Light"
FontSize="20" Margin="12" />
</Grid>
```

9. Build and run your app on the Local Machine.

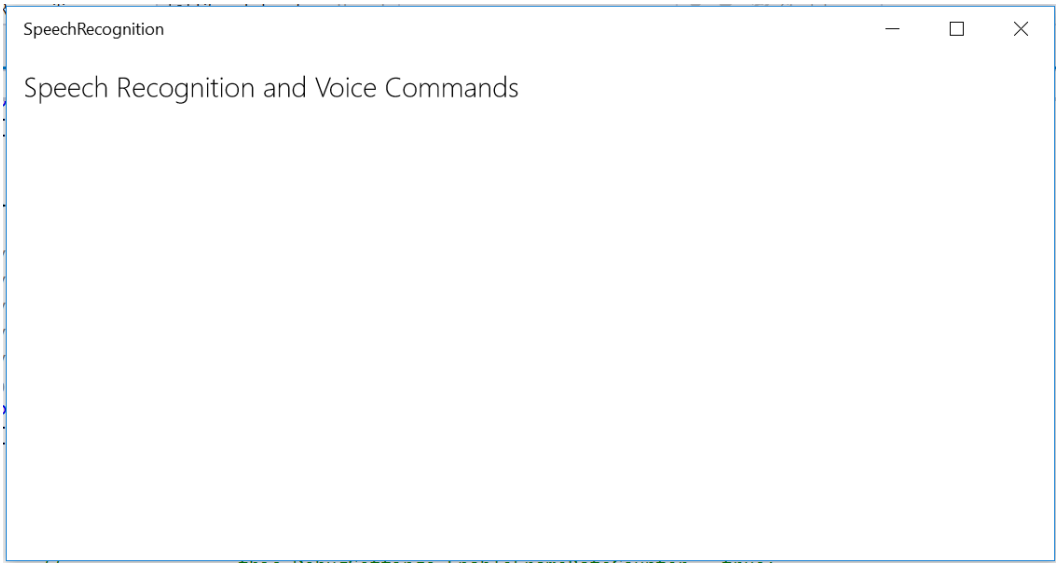

Figure 5

The app running on the Local Machine.

10. Close your app. Open the project **Properties** from the Solution Explorer and browse to the Debug tab. Choose the start action **Do not launch, but debug my code when it starts** and save the properties file. You can use this option to debug an app that you are not launching directly from Visual Studio.

11. Use the **Start Debugging** button to start debugging without launching your app.

12. Set a breakpoint in **App.xaml.cs** after the Debug.WriteLine messages.

13. Click the microphone button in your task bar to prepare to launch via voice command.

14. Say the words “Hands-on Labs, launch.” Cortana will verbally confirm that she is opening your Speech Recognition app.

> **Note:** If your region, language, and speech settings are set to another supported language for which you made a voice command set, you may use those commands instead. If you change your region, language, and speech settings, you may need to log out and log back in for the change to fully take effect.

15. When you hit your breakpoint, open the **Output** window to view your debug messages. You will see the command Cortana recognized as well as the actual text spoken. The two may differ depending on how well Cortana interpreted your speech. Use the **Continue** button to resume the app launch when you are done.

16. Your app will launch and navigate to the MainPage view.

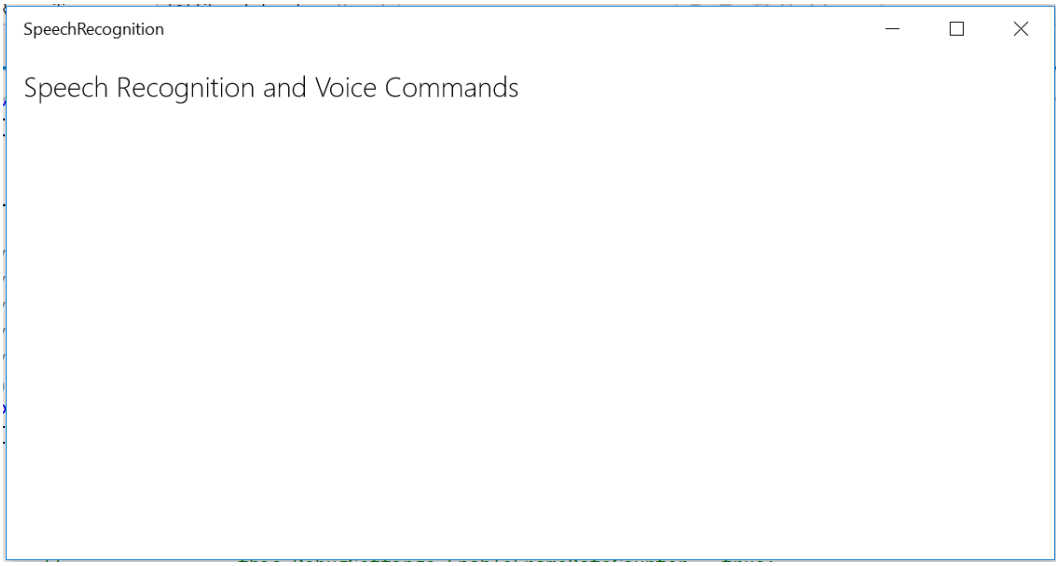

Figure 6
The app launches via voice command.

17. Close your app and return to Visual Studio. Remove the breakpoint in **App.xaml.cs** and uncheck the **Do not launch, but debug my code when it starts** option in the project properties. Save the file.

Exercise 2: Use a Voice Command to Change the App’s Appearance

In addition to launching your app, voice commands can interact with content in the app. In this exercise, you will use a voice command to change the background color of your app when it launches.

Task 1 – Set the background color

In this task, you will set an initial background color for your app and specify an **x:Name** attribute to make it easier to target the grid's properties.

1. Specify **AliceBlue** as the background color for your grid in **MainPage.xaml** and give the grid the x:Name **Container**.

XAML

```
<Grid Background="AliceBlue" x:Name="Container" >
        <TextBlock Text="Speech Recognition and Voice Commands"
FontWeight="Light" FontSize="20" Margin="12" />
</Grid>
```

2. Build and run your app on the Local Machine. You will see the page title on a light blue background.

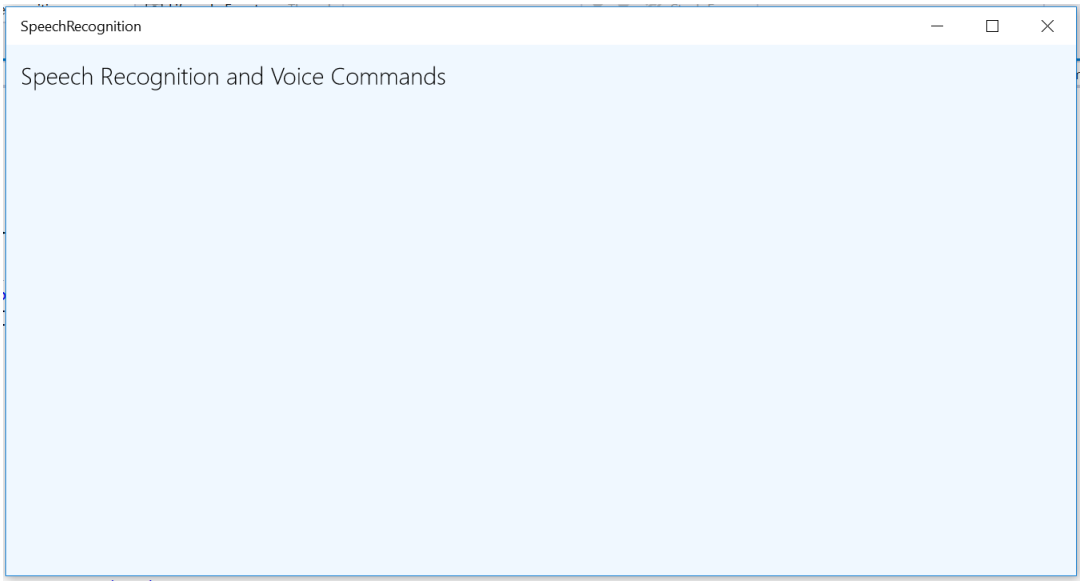

Figure 7
The app running with a colored background.

3. Stop debugging and return to Visual Studio.

Task 2 – Create a voice command to trigger the color change

In this task, you will create a voice command to change the background color of the grid to red.

1. Open your VoiceCommands.xml definition file and add a new command. Give the command the name **TurnRed.**

XML

```
<Command Name="TurnRed">
    <Example>turn red</Example>
    <ListenFor>turn red</ListenFor>
    <Feedback>My favorite color is red</Feedback>
    <Navigate />
</Command>
```

Note: If you are supporting additional languages, add an equivalent command to its command set.

2. Add a case for the **TurnRed** command to the **voiceCommandName** switch in **App.xaml.cs**. This time, pass a parameter to MainPage when navigating.

C#

```
switch (voiceCommandName)
{
    case "LaunchApp":
        frame.Navigate(typeof(MainPage));
        break;
    case "TurnRed":
        frame.Navigate(typeof(MainPage), "Red");
        break;
    default:
        break;
}
```

3. Open the MainPage code behind and create an **OnNavigatedTo** override to handle the incoming parameter. Set the grid background to a color that corresponds to the incoming parameter.

C#

```
protected override void OnNavigatedTo(NavigationEventArgs e)
{
    if (e.Parameter.ToString() == "Red")
        Container.Background = new SolidColorBrush(Windows.UI.Colors.DarkRed);

    base.OnNavigatedTo(e);
}
```

4. Build and run your app on the Local Machine to register the new voice command. You should still see the blue background. Close your app.

5. Click Cortana's microphone button and speak the command **"Hand-on Labs, turn red."** Cortana will respond that red is her favorite color. When the app launches, you will see the background is now red.

Figure 8

The app launches via voice command with a red background.

Note: If you need to debug the code you wrote in this task, refer to the instructions for the **Do not launch, but debug my code when it starts** method from Exercise 1: Task 4.

6. Stop debugging and return to Visual Studio.

Exercise 3: Respond to a Voice Command with a Background Task

Voice commands can trigger background tasks to run without launching your app. This behavior can be useful when you want to allow your users to perform simple tasks associated with your app entirely through Cortana , without the need to launch the app. In this exercise, you will create a Windows Runtime Component to respond to a user's question via a background task.

Task 1 – Create the Windows Runtime Component

We will begin by creating a Windows Runtime Component to wrap the background task that you will create in a later task.

1. Right-click on the solution name in the Solution Explorer. Choose **Add > New Project** and select the project type **Windows Runtime Component (Universal Windows)**.

2. Name the project **VoiceCommandService**.

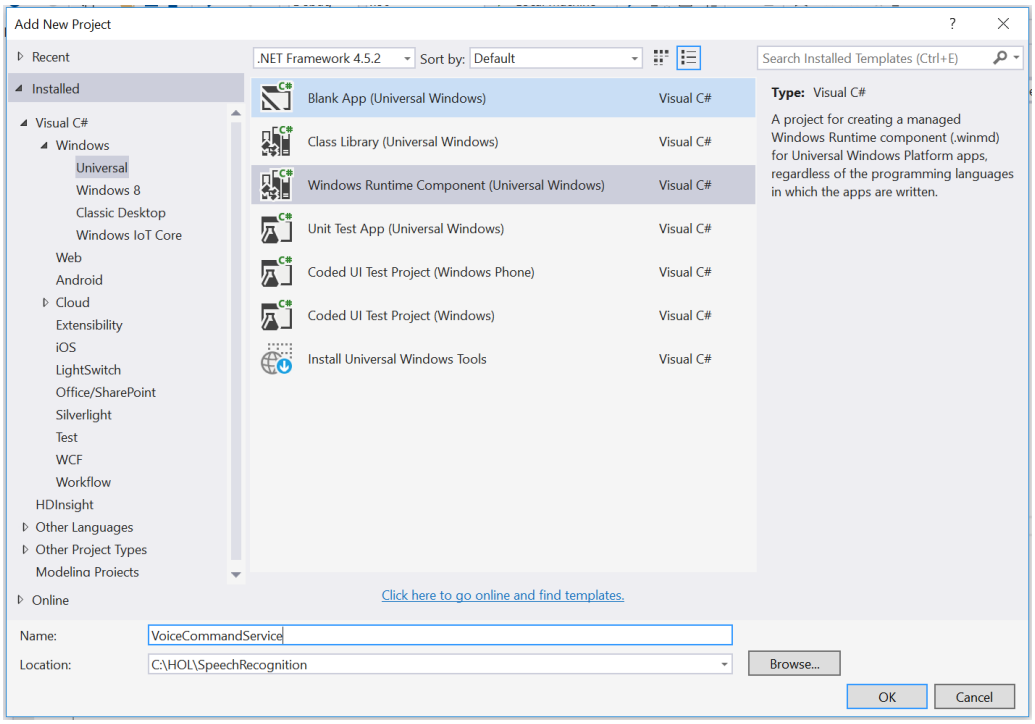

Figure 9

Add the Windows Runtime Component.

3. Right-click on Class1.cs in the Solution Explorer and use the **Rename** option to rename it to **HolVoiceCommandService**. If prompted to perform a rename in the project of all references to Class1, choose **Yes**.

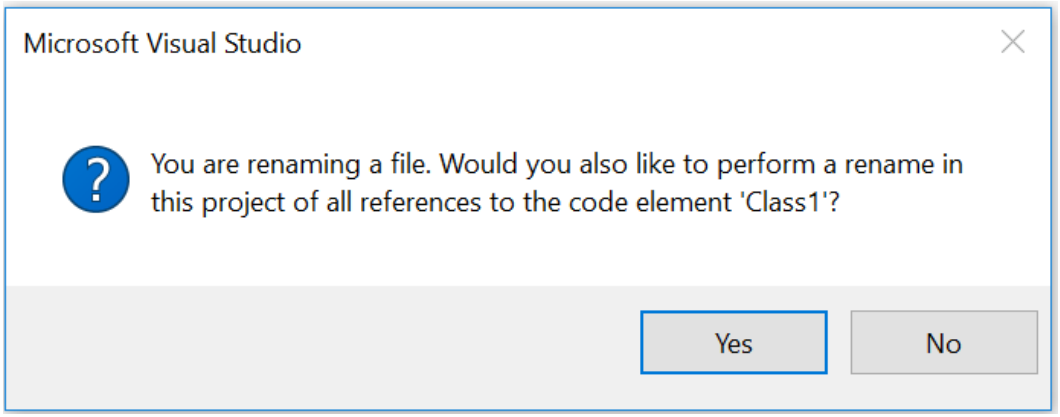

Figure 10

Rename Class1.cs to HolVoiceCommandService.cs.

4. Return to the SpeechRecognition project. Right-click on the References folder and add the VoiceCommandService as a reference.

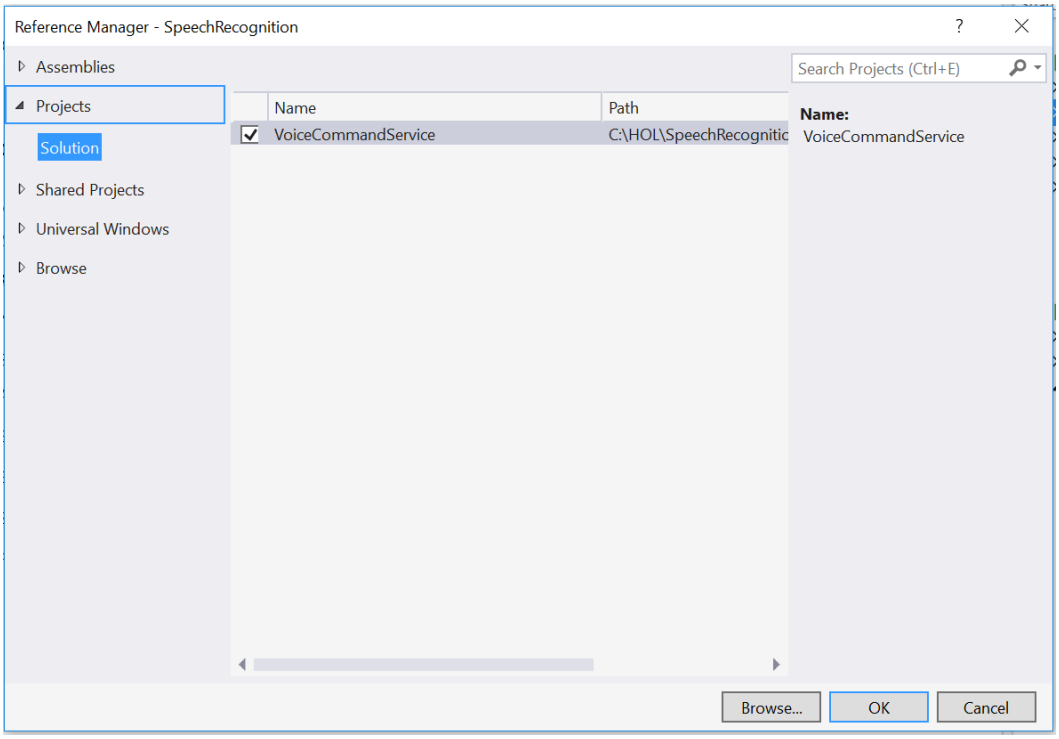

Figure 11

Reference the VoiceCommandService from the SpeechRecognition project.

Task 2 – Add a voice command to reference the VoiceCommandService

Voice commands that launch background tasks differ from the voice commands you created earlier in this lab. Instead of using a Navigate element, the command will define a custom VoiceCommandService element that has a target attribute pointing at the HolVoiceCommandService class. You will create the voice command in this task and you will also give it additional phrasing options. You can add phrasing options to any voice commands whether they are used to launch the foreground app or a background task.

1. In your SpeechRecognition project, add a command named **SayHello** to **VoiceCommands.xml**. You may define multiple **<ListenFor>** elements in a command. In this case, define two questions to ask Cortana, both of which will trigger the command.

Note: If you are supporting an additional language, add an equivalent command to its command set.

XML

```
<Command Name="SayHello">
    <Example>say hello</Example>
    <ListenFor RequireAppName="BeforeOrAfterPhrase">How's it going</ListenFor>
    <ListenFor RequireAppName="BeforeOrAfterPhrase">Say hello</ListenFor>
    <Feedback>Hold on, let me ask</Feedback>
```

```
    <VoiceCommandService Target="HolVoiceCommandService" />
</Command>
```

Note: The **RequireAppName="BeforeOrAfterPhrase"** attribute allows you to offer flexible, natural phrasing to your voice commands. Both the phrases “Hands-on Labs, How’s it going?” and “How’s it going, Hands-on Labs?” are valid when this attribute is set to BeforeOrAfterPhrase. For more on phrasing options, visit the ListenFor documentation at https://msdn.microsoft.com/en-us/library/windows/apps/dn706593.aspx

Task 3 – Register the service in the app manifest

To run in the background, the HolVoiceCommandService must be registered in the SpeechRecognition app manifest.

1. Open Package.appxmanifest in the manifest editor and navigate to the Declarations tab.
2. Using the **Available Declarations** drop-down menu, select **App Service** and click **Add** to add it to the list of Supported Declarations.

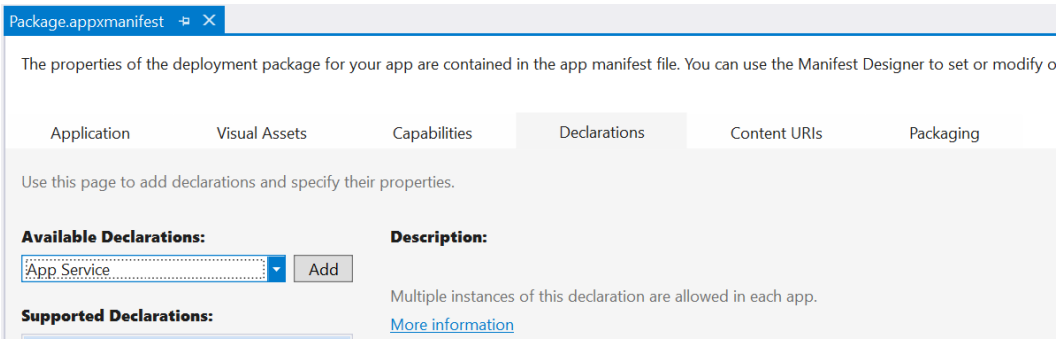

Figure 12

Add an App Service to the list of Supported Declarations.

3. In the Properties for the App Service declaration, set the **Name** property to **HolVoiceCommandService**.
4. Set the entry point property to **VoiceCommandService.HolVoiceCommandService**.

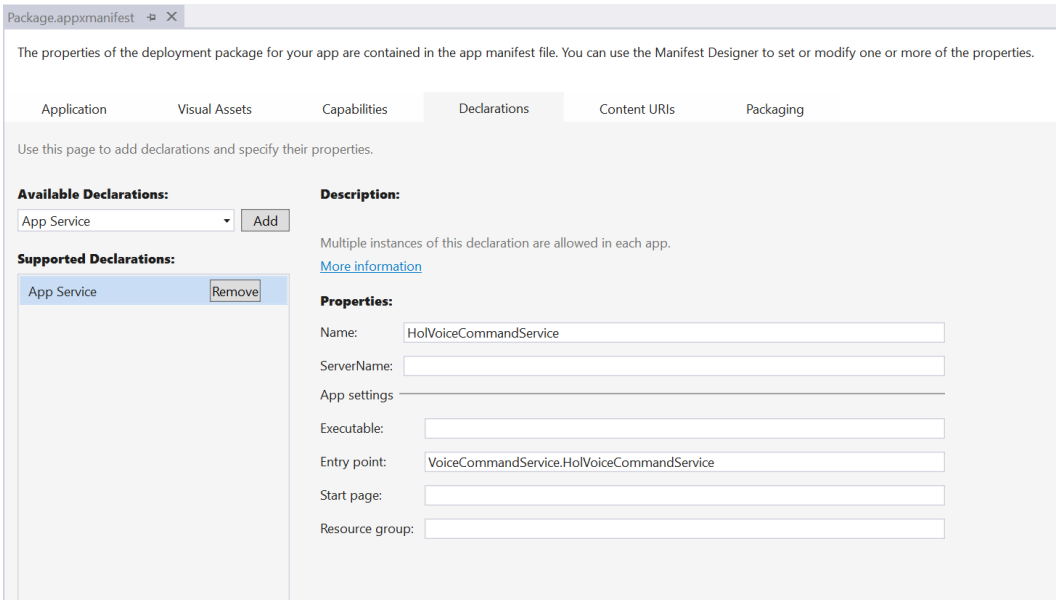

Figure 13

Register the App Service in the app manifest.

Note: When registering an app service, the Name property needs to match the name of the class within your component, not the name of the component itself. We have given the HolVoiceCommandService class a different name from the WinRT component to help to differentiate the two.

This distinction is also important when setting the Target attribute in the voice command definition, which you did in Task 2.

5. Close the manifest, saving the changes.

Task 4 – Handle the incoming command in the service

In this task, you will implement the entrypoint for all headless voice commands invoked via Cortana. The background task you will create must respond to Cortana with 0.5 seconds and report progress every 5 seconds.

1. Open **HolVoiceCommandService.cs** and add the **Windows.ApplicationModel.Background** namespace.

C#

```
using Windows.ApplicationModel.Background;
```

2. Modify the class declaration so that it implements the **IBackgroundTask** interface. Add the **Run** method required by the interface

C#

```
public sealed class HolVoiceCommandService : IBackgroundTask
    {
        public void Run(IBackgroundTaskInstance taskInstance)
        {
            throw new NotImplementedException();
        }
    }
```

3. Declare a class member called **serviceDeferral** of type **BackgroundTaskDeferral**. In your **Run** method, get a deferral for your task instance and save it in serviceDeferral. Add **OnVoiceCommandCompleted()** and **OnTaskCanceled()** methods to handle completion of the deferral. You will subscribe to the **VoiceCommandCompleted** event in a later step.

C#

```
public sealed class HolVoiceCommandService : IBackgroundTask
    {
        BackgroundTaskDeferral serviceDeferral;

        public void Run(IBackgroundTaskInstance taskInstance)
        {
            serviceDeferral = taskInstance.GetDeferral();

            taskInstance.Canceled += OnTaskCanceled;
        }
    }

    private void OnVoiceCommandCompleted(VoiceCommandServiceConnection sender,
VoiceCommandCompletedEventArgs args)
    {
        if (this.serviceDeferral != null)
        {
            this.serviceDeferral.Complete();
        }
    }

    private void OnTaskCanceled(IBackgroundTaskInstance sender,
BackgroundTaskCancellationReason reason)
    {
        System.Diagnostics.Debug.WriteLine("Task cancelled, clean up");
        if (this.serviceDeferral != null)
        {
            //Complete the service deferral
            this.serviceDeferral.Complete();
        }
    }
```

4. Add the **Windows.ApplicationModel.AppService** and **Windows.ApplicationModel.VoiceCommands** namespaces.

C#

```
using Windows.ApplicationModel.AppService
using Windows.ApplicationModel.VoiceCommands;
```

5. Add the voice command service connection.

C#

```
public sealed class HolVoiceCommandService : IBackgroundTask
{
    VoiceCommandServiceConnection voiceServiceConnection;

    BackgroundTaskDeferral serviceDeferral;
```

Note: The service connection is maintained for the lifetime of a Cortana session.

6. Check the trigger details to see if the name matches the name of the App Service registration from the app manifest. If so, implement a try-catch block.

C#

```
taskInstance.Canceled += OnTaskCanceled;

var triggerDetails = taskInstance.TriggerDetails as AppServiceTriggerDetails;

if (triggerDetails != null && triggerDetails.Name == "HolVoiceCommandService")
{
    try
    {
        voiceServiceConnection =

VoiceCommandServiceConnection.FromAppServiceTriggerDetails(
                            triggerDetails);

    voiceServiceConnection.VoiceCommandCompleted += OnVoiceCommandCompleted;
    }
    catch (Exception ex)
    {
    System.Diagnostics.Debug.WriteLine("Handling Voice Command failed " +
ex.ToString());
    }
}
```

Note: The subscription to the VoiceCommandCompleted event is in this code block, because it must take place after the voiceServiceConnection is set.

If you need to debug this code, use the **Do not launch, but debug my code when it starts** method described earlier and set a breakpoint in your run method.

7. Await the incoming voice command and create a switch to handle the **SayHello** case. Add the **async** keyword to the Run method.

C#

```
try
{
    voiceServiceConnection =

VoiceCommandServiceConnection.FromAppServiceTriggerDetails(
                            triggerDetails);

    voiceServiceConnection.VoiceCommandCompleted += OnVoiceCommandCompleted;

    VoiceCommand voiceCommand = await
voiceServiceConnection.GetVoiceCommandAsync();

    switch (voiceCommand.CommandName)
    {
        case "SayHello":
            break;
        default:
            break;
    }
}
```

8. If the incoming voice command is **SayHello**, create a user message and set a display message and spoken message for Cortana to relay back to the user.

C#

```
switch (voiceCommand.CommandName)
{
    case "SayHello":
        var userMessage = new VoiceCommandUserMessage();
        userMessage.DisplayMessage = "Hello!";
        userMessage.SpokenMessage = "Your app says hi. It is having a great
time.";
        var response = VoiceCommandResponse.CreateResponse(userMessage);
        await voiceServiceConnection.ReportSuccessAsync(response);
        break;
    default:
        break;
}
```

9. Save the HolVoiceCommandService.

Task 5 – Interact with your app via the background task

Now that you have set up your voice command, background task, and registered the app service, it is time to give it a try.

1. Run the SpeechRecognition project on the Local Machine to register the latest version of the voice command definition file.

2. Close the app.

3. Use the Cortana prompt to speak one of the variations of the SayHello voice command:

   - "Hands-on labs, how's it going?"
   - "How's it going, Hands-on labs?"
   - Hands-on labs, say hello."
   - "Say hello, Hands-on labs.

4. Cortana will show your app image and the displayMessage. At the same time, she will speak the spoken message you chose in the background task.

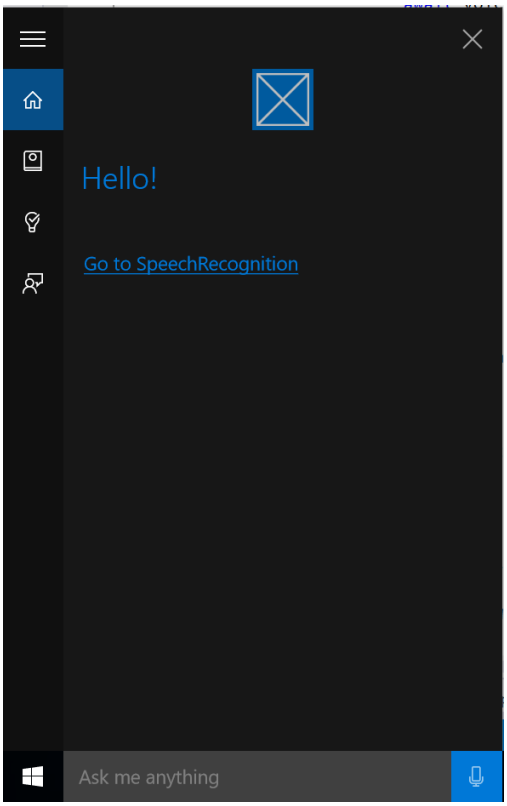

Figure 14

Cortana returns a message from the background task of the app.

Note: To debug your background task, use a similar method to the one you used to debug your voice commands. Right-click on the SpeechRecognition project in the Solution Explorer and open the **Properties** editor. On the **Debug** pane, check the option **Do not launch, but debug my code when it starts**. Save the properties file and use the Start Debugging button to start debugging. Set a breakpoint in HolVoiceCommandService and launch your app using the SayHello command. If your entrypoints are set up correctly, you will hit the breakpoint.

5. Stop debugging and return to Visual Studio.

Summary

Voice commands handle important interactions in Windows 10. In this lab, you created a voice command definition file and explored the VCD schema. You added commands to launch the app, interact with its appearance, and receive a response from a background task. You also learned how to register and differentiate between voice commands.