Creating a Voice-Controlled Windows Phone 8 Music Information app with the Nokia Music API

This tutorial explains how to combine the Windows Phone 8 voice services with Nokia Music to make a voice-controlled Windows Phone 8 music information app.

Note: This article was a winner in the Windows Phone 8 Wiki Competition 2012Q4.

Introduction

With Windows Phone 8, apps can register to handle specific commands. For instance, a calculator app could handle the "add," "subtract," "multiply," and "divide" commands to provide instant, easy access from any place in the OS. Windows Phone 8 also provides text to speech capabilities, and Nokia has provided Nokia Music API for Windows Phone which can be used to query a web service for music information.

This article explains how we can use these three features to create a voice-controlled music information app. The app will respond verbally to a voice command along the lines of "Music, what are the top songs by Adele?"

For clarity, the rest of this tutorial has been split up into three parts, Text to Speech, Nokia Music, and Voice Commands.

Before you Begin

Before you begin, we'll need to add a few references, capabilities, and using statements to our project.

Create a new Windows Phone 8 C#/XAML application and name it "Music Info."

References

To use the Nokia Music API in our app, we'll need to reference it. Right-click on **References** in the solution explorer, then choose *Manage NuGet Packages*. Use the search bar in the top-right of the NuGet window to search for *nokiamusic*, and then press the **Install** button to automatically reference the Nokia Music API and its dependencies in your application.

Capabilities

In the Visual Studio solution explorer of your new project, expand "Properties" and double-click **WMAppManifest.xml** Choose the *Capabilities* tab and check both `ID_CAP_MICROPHONE` and `ID_CAP_SPEECH_RECOGNITION`.

Using Statements

Open **MainPage.xaml.cs** and add the following using statements to the top of the page:

```
using Nokia.Music.Phone;
using Nokia.Music.Phone.Types;
using Windows.Phone.Speech.Synthesis;
using Windows.Phone.Speech.Recognition;
using Windows.Phone.Speech.VoiceCommands;
```

Text to Speech

Our music app will need to be able to respond to the user verbally, and new features in Windows Phone 8 make that extremely easy.

With Windows Phone 8, Microsoft made speech recognition and synthesization simple with the new `Windows.Phone.Speech` namespace. Our code for the `SpeakText` function is fairly straightforward:

```
public async void SpeakText(string TextToSpeak)
{
  SpeechSynthesizer synth = new SpeechSynthesizer();
  synth.SetVoice(InstalledVoices.Default); //you can change the voice to use here
  await synth.SpeakTextAsync(TextToSpeak);
}
```

Basically, we're going to create a new SpeechSynthesizer, set it's voice to the default installed voice (if you prefer another voice you can change it with `synth.SetVoice()`), and then await while it speaks whatever text is passed to the function.

(cc) Nokia 2013. All rights reserved.

Nokia Music

API Key

Before we begin, you'll need to have signed up for an API key at nokia.ly/apisignup.

Creating the music client

We'll start off by creating a blank function that will allow us to respond to an artist's name with that artist's top songs. This function will be called from our voice command handler. Place the below in your **MainPage.xaml.cs** file:

```
public void RespondWithTopSongs(string ArtistName)
{
}
```

Next, we'll add the code necessary for the Nokia Music API. Add the following line to the `RespondWithTopSongs` function we just created, filling in the App ID and App Code accordingly:

```
MusicClient nmClient = new MusicClient("App ID", "App Code"); //use the App ID and App Code values that you got from signing up at n
```

We now have a Nokia MusicClient that will allow us to query Nokia Music for information.

Searching for the Information

With the Nokia Music API, you find information by searching for artists, songs, albums, and more.

MusicClient has a `SearchArtists()` function that we will use to search for the user-specified artist. Here's the code we'll use to search for the artist:

```
nmClient.SearchArtists(
  (ListResponse<Artist> artistResponse) =>
  {
    if (artistResponse.Count() > 0)
    {
      Artist artist = artistResponse.First<Artist>(); //let's assume that the first returned artist is the one we're looking for
    }
    else
    {
      SpeakText("Sorry, we couldn't find any artist named " + ArtistName);
    }
  }, ArtistName);
```

The code uses `nmClient.SearchArtists` to search for all artists that contain the `ArtistName`, and return an error message to the user if no artist is found. The Nokia Music API will by default return the artist who best matches our search first, so that's the artist we assign to the "artist" variable.

Next, we need to use the `SearchProducts()` function to search for all the products this artist has created, and then narrow that search down to only Tracks and Singles.

The following code will go after we set the `artist` variable:

```
nmClient.GetArtistProducts(
  (ListResponse<Product> productResponse) =>
  {
    var songs = productResponse.Where((x) => //productResponse will include all products, not just songs. this will only return trac
      {
        if (x.Category == Category.Track || x.Category == Category.Single)
        {
          return true;
        }
        else
        {
          return false;
        }
      }
    );
    List<Product> productList = songs.ToList<Product>();
    if (productList.Count >= 3)
    {
      SpeakText("The top songs by " + ArtistName + " are " + productList[0].Name + ", " + productList[1].Name + ", and " + productLi
    }
    else if (productList.Count >= 1)
    {
      SpeakText("The top song by " + ArtistName + " is " + productList[0].Name);
    }
    else
    {
      SpeakText("Sorry, we don't have song info for " + ArtistName);
    }
  }, artist);
```

This code will search for an artists products, narrow the results down to only real songs, then either return the top three songs, the top song, or an error message if there are no songs available. We then use the `SpeakText()` function we created earlier to speak our responses.

After you do the above, your `RespondWithTopSongs()` function should look like this:

```
public void RespondWithTopSongs(string ArtistName)
{
```


(cc) Nokia 2013. All rights reserved.

```
  MusicClient nmClient = new MusicClient("App ID", "App Code");
  nmClient.SearchArtists(
    (ListResponse<Artist> artistResponse) =>
    {
      if (artistResponse.Count() > 0)
      {
        Artist artist = artistResponse.First<Artist>(); //let's assume that the first returned artist is the one we're looking for
        nmClient.GetArtistProducts(
          (ListResponse<Product> productResponse) =>
          {
            var songs = productResponse.Where((x) => //productResponse will include all products, not just songs. this will only ret
              {
                if (x.Category == Category.Track || x.Category == Category.Single)
                {
                  return true;
                }
                else
                {
                  return false;
                }
              });
            List<Product> productList = songs.ToList<Product>();
            if (productList.Count >= 3)
            {
              SpeakText("The top songs by " + ArtistName + " are " + productList[0].Name + ", " + productList[1].Name + ", and " + p
            }
            else if (productList.Count >= 1)
            {
              SpeakText("The top song by " + ArtistName + " is " + productList[0].Name);
            }
            else
            {
              SpeakText("Sorry, we don't have song info for " + ArtistName);
            }
          }, artist);
        }
        else
        {
          SpeakText("Sorry, we couldn't find any artist named " + ArtistName);
        }
      }, ArtistName);
}
```

Voice Commands

Anatomy of a Voice Command

Voice commands consist of three parts: the app name, command, and phrase:

- App Name: The app name is simply the name of your app. This part of your command is only necessary if the user is not using your app - if they're already in your app, there's no need for them to say the app name.
- Command: This is the portion of the voice command which actually tells you what the user wants to do. For example, in our app, the command would be "what are the top songs by"
- Phrase: The phrase provides more details about what exactly the user wants to do. In our app the phrase would be "Adele."

Registering a Voice Command

Voice Command Definition

Voice commands in Windows Phone 8 are defined with a Voice Command Definition file (or VCD). The most important parts of a VCD file are:

- CommandSet: This is a group of commands. You can refer to these commands in code with the CommandSets "Name" property
- CommandPrefix: This is the "App Name" portion of your voice commmand.
- ListenFor: A phrase that, if heard, will activate this command. {PLName} designates a PhraseList, {*} designates a wildcard, and [notneccessary] designates optional words.
- PhraseList: This is a list of words that the user can say as the "Phrase" portion of your command. You can refer to a phrase list in your ListenFor element using {}.

Right-click the project name in Visual Studio and then go Add->New. Select "Voice Command Definition" and name it something like "VCD.xml"

Once the file opens, copy and paste the following:

```
<?xml version="1.0" encoding="utf-8"?>

<VoiceCommands xmlns="http://schemas.microsoft.com/voicecommands/1.0">
  <CommandSet xml:lang="en-US" Name="MusicInfoCommands">
    <CommandPrefix>Music Info</CommandPrefix>
    <Example> What are the top songs by Adele? </Example>

    <Command Name="MusicInfo">
      <Example> What are the top songs by Adele? </Example>
      <ListenFor> what are the top songs by {Artist} </ListenFor>
      <ListenFor> what are the top songs by {*} </ListenFor>
      <Feedback>Thinking...</Feedback>
      <Navigate />
    </Command>
  <PhraseList Label="Artist">
  </PhraseList>
  </CommandSet>
</VoiceCommands>
```

This definition file will look for someone saying "What are the top songs by" and then an artists name. After it receives a command, it will show "Thinking..." until our app launches, at which point we'll need to override the default `OnNavigatedTo()` handler and replace it with code to start looking for top songs.

(C) Copyright Nokia 2013. All rights reserved.

Note: Looking at the above VCD file, you might be confused as to why we have two different ListenFor elements. In a perfect world, we would be able to use a wildcard ("{*}") and allow the user to say any name he would like. Unfortunantly, in Windows Phone 8 there is no way to allow the user to say any artist name; every possible response must be in a PhraseList. {*} is only useful when you want to see that the user has said something, but don't need to know what it is they said. To workaround this issue, we will load the top 200 artists into the "Artist" PhraseList when our app is first run (thus making use of the first ListenFor element), then allow the user to say something else with the second ListenFor element. In the case of the latter, our app will start and then ask the user to repeat the name of the artist.
For more information, see here and here.

Now that we have the VCD file, we have to add the commandset it contains to the system. We want to do this when the app is launched, so we'll override the default `OnNavigatedTo()` handler. Add the following to your **MainPage.xaml.cs** file:

```
protected async override void OnNavigatedTo(System.Windows.Navigation.NavigationEventArgs e)
{
  base.OnNavigatedTo(e);
  await VoiceCommandService.InstallCommandSetsFromFileAsync(new Uri("ms-appx:///VCD.xml", UriKind.RelativeOrAbsolute));
}
```

Handling the Command

Once we get a voice command, we'll need to add to our newly overridden `OnNavigatedTo()` handler to detect when it's being launched as a voice command. Copy and paste the below code as the `OnNavigatedTo()` function you just made:

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

  if (NavigationContext.QueryString.Any())
  {
    if (NavigationContext.QueryString["reco"].ToString().Contains("..."))
    {
      SpeechRecognizerUI recogUI = new SpeechRecognizerUI();
      recogUI.Settings.ExampleText = "Johnny Cash";
      recogUI.Settings.ListenText = "Please speak the name of the artist.";
      SpeechRecognitionUIResult result = await recogUI.RecognizeWithUIAsync();
      RespondWithTopSongs(result.RecognitionResult.Text.TrimEnd('.'));
    }
    else
    {
      RespondWithTopSongs(NavigationContext.QueryString["Artist"]);
    }
  }
  else
  {
    await VoiceCommandService.InstallCommandSetsFromFileAsync(new Uri("ms-appx:///VCD.xml", UriKind.RelativeOrAbsolute));
    var installedCommands = VoiceCommandService.InstalledCommandSets["MusicInfoCommands"];
    MusicClient nmClient = new MusicClient("App ID", "App Code");
    nmClient.GetTopArtists(async (artists) =>
    {
      foreach (Artist artist in artists)
      {
        await installedCommands.UpdatePhraseListAsync("Artist", new String[1] { artist.Name });
      }
    }, 0, 200);
  }
}
```

The first thing that you'll notice when looking at the code is that we've added an if/then statement. The voice command is passed through the `NaviagtionContext.QueryString` variable, so we'll first check to see if there is anything in the QueryString.

If there isn't a `QueryString` passed, we want to install the voice command set. We've moved the `await VocieCommandService.InstallCommandSetsFromFileAsync(new Uri("ms-appx:///VCD.xml", UriKind.RelativeOrAbsolute));` line into our if/then statement, which means that our voice commands will only be installed if the app is launched from the start menu (if it's being launched as a voice command, we can pretty much assume that our VCD file has already been installed). After that, we're using `VoiceCommandService.InstalledCommandSets` to get the command set we just installed with `InstallCommandSetsFromFileAsync`. We then create a MusicClient, use it to get the top 200 artists, and then add each of those artists to the `Artists` PhraseList we created in the last section (for more info on why we're doing this, see the note above).

If there is a `QueryString` passed, we want to send the artists name to our RespondWithTopSongs function. Before we do that, we need to check whether the artists name was recognized (whether it was one of the top 200 we inserted into the PhraseList). If it's not (which we can find out by using `.Contains("...")`, since {*} returns "..." in place of what the user said), we're going to launch a SpeechRecognizerUI instructing the user to repeat the artists name.

Congratulations!

Congrats - you now have a working app that incorporates a few of the awesome new features in Windows Phone 8. You can learn more about these features with the links below.

Bibliography/See Also

- What's new in Windows Phone 8 (Nokia Developer Wiki)
- How Many Lines of Code Does it Take to Build a Complete WP8 Speech App (Rob Chambers Blog)
- How to create a PhraseList in a VCD file that includes every word (wildcard)? (MSDN Forums)
- The Nokia Music WP API Client Test App (Github)
- Text-to-speech (TTS) for Windows Phone (Windows Phone Dev Center)
- Speech recognition for Windows Phone (Windows Phone Dev Center)
- Voice Commands in Windows Phone 8 (wiki)

(The Nokia Wiki) 2013. All rights reserved.

(c) Copyright Nokia 2013. All rights reserved.