## UWP-049 - UWP SoundBoard – Introduction

As we reach the midway point of this series of lessons we will be changing-up the format a bit from the one we have been following so far of confronting loosely related topics and then undertaking challenges to reinforce those ideas. What we will do instead is focus on building four full applications from scratch to illustrate the process of how to take an application from the conceptual stage into actual construction.

The first application we will look at is a simple “Sound Board” that basically lists a collection of predefined sounds that can be played when you click on an icon that corresponds to that sound. It will have various features such as sound categories, playing a sound via drag/drop, searching for sounds using auto-suggest, etc. The application will implement things like Hamburger Navigation, State Management, Media Elements and so forth.

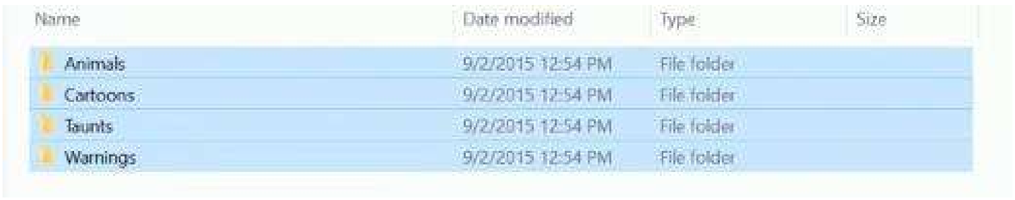
At the end we will build the application and walk through the process of submitting it to the Windows Store. As usual, it’s a good idea to follow along by writing the code on your end.

## UWP-050 - UWP SoundBoard - Setup and MainPage Layout

Without any further ado, let's begin creating our SoundBoard App.

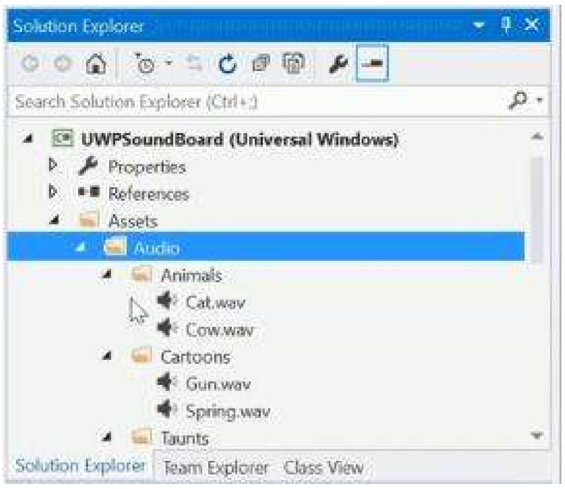
Step 1: Create a new project named "UWPSoundBoard"

Step 2: Download the zip file associated with this lesson and add the assets to the project using the following process. Start with audio assets by creating a subfolder for it in your Assets folder, then select the folders containing your audio assets as follows:



| Name | Date modified | Type | Size |
|---|---|---|---|
| Animals | 9/2/2015 12:54 PM | File folder | |
| Cartoons | 9/2/2015 12:54 PM | File folder | |
| Taunts | 9/2/2015 12:54 PM | File folder | |
| Warnings | 9/2/2015 12:54 PM | File folder | |

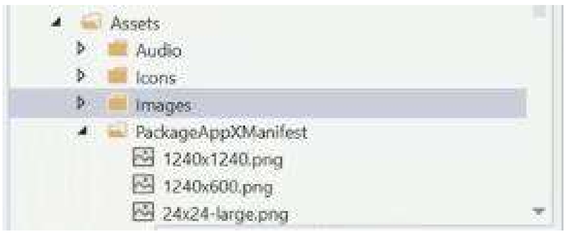
Then click and drag all of that into your Audio folder



```
UWPSoundBoard (Universal Windows)
  Properties
  References
  Assets
    Audio
      Animals
        Cat.wav
        Cow.wav
      Cartoons
        Gun.wav
        Spring.wav
      Taunts
```

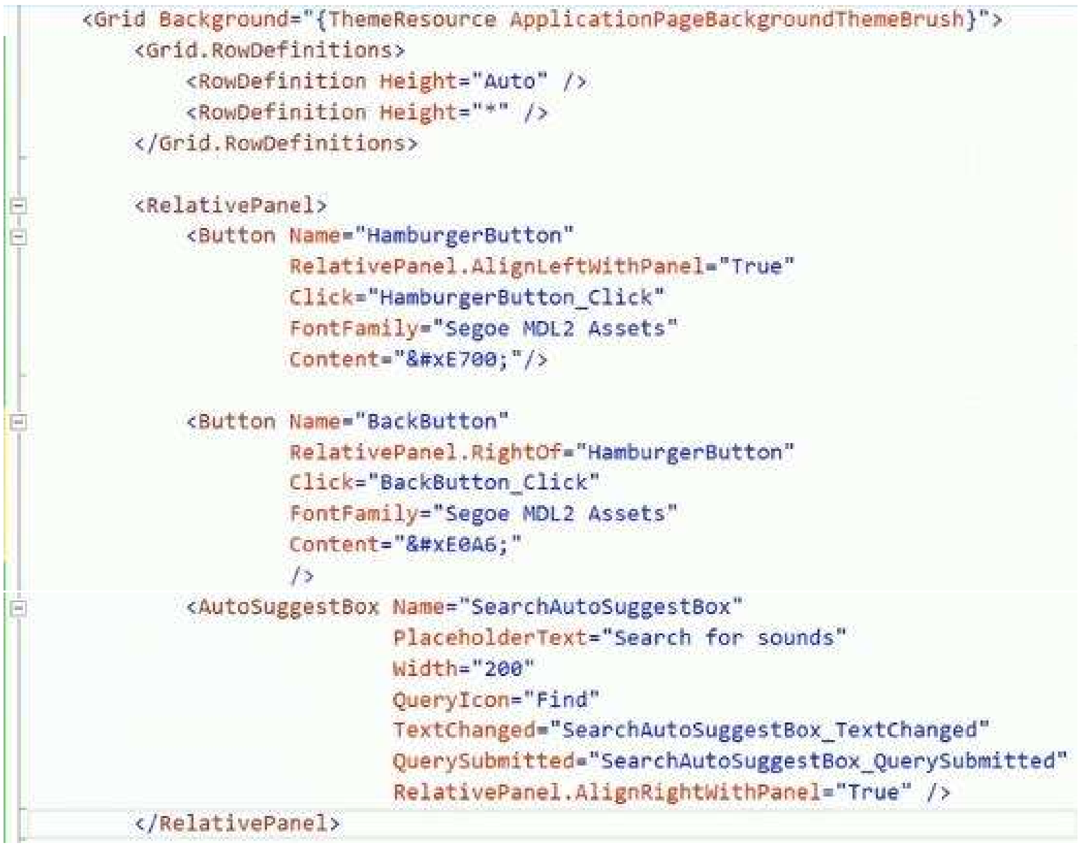
If you come across a nag screen simply select "Apply to all items."

Step 3: Repeat this process for the "Icons," "Images," and "PackageAppXManifest" folders



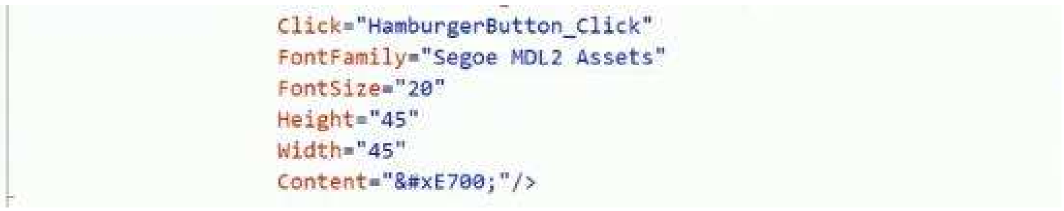
```
Assets
  Audio
  Icons
  Images
  PackageAppXManifest
    1240x1240.png
    1240x600.png
    24x24-large.png
```

Step 4: Open up MainPage.xaml and set up the Panels



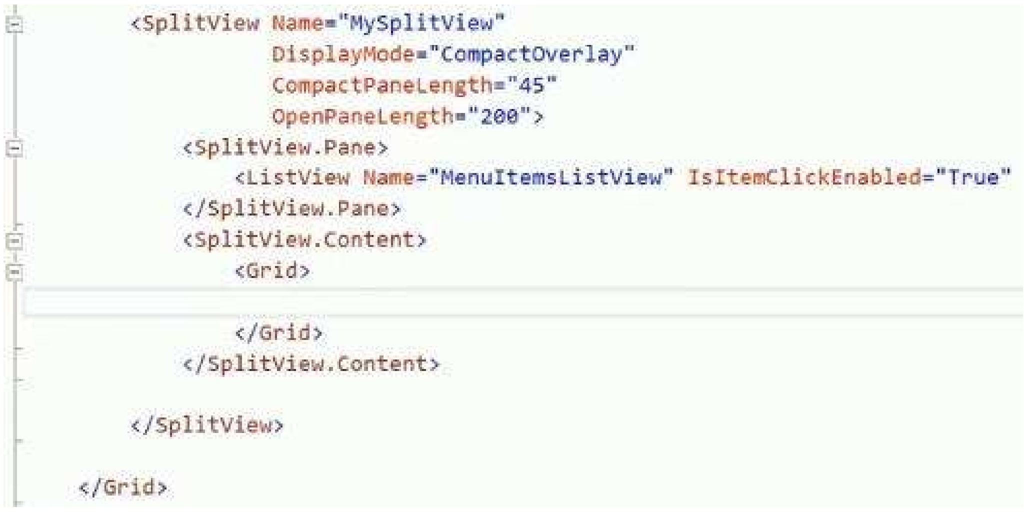
```
    <Grid Background="{ThemeResource ApplicationPageBackgroundThemeBrush}">
        <Grid.RowDefinitions>
            <RowDefinition Height="Auto" />
            <RowDefinition Height="*" />
        </Grid.RowDefinitions>

        <RelativePanel>
            <Button Name="HamburgerButton"
                    RelativePanel.AlignLeftWithPanel="True"
                    Click="HamburgerButton_Click"
                    FontFamily="Segoe MDL2 Assets"
                    Content="&#xE700;"/>

            <Button Name="BackButton"
                    RelativePanel.RightOf="HamburgerButton"
                    Click="BackButton_Click"
                    FontFamily="Segoe MDL2 Assets"
                    Content="&#xE0A6;"
                    />
            <AutoSuggestBox Name="SearchAutoSuggestBox"
                            PlaceholderText="Search for sounds"
                            Width="200"
                            QueryIcon="Find"
                            TextChanged="SearchAutoSuggestBox_TextChanged"
                            QuerySubmitted="SearchAutoSuggestBox_QuerySubmitted"
                            RelativePanel.AlignRightWithPanel="True" />
        </RelativePanel>
```

Step 5: Adjust the Height, Width and FontSize for the HamburgerButton and BackButton by adding the following



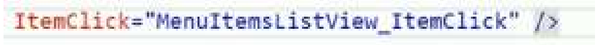
```
                    Click="HamburgerButton_Click"
                    FontFamily="Segoe MDL2 Assets"
                    FontSize="20"
                    Height="45"
                    Width="45"
                    Content="&#xE700;"/>
```

Step 6: Then for the SplitView add the following. Note that we are adding a ListView instead of a ListBox under the SplitView.Pane



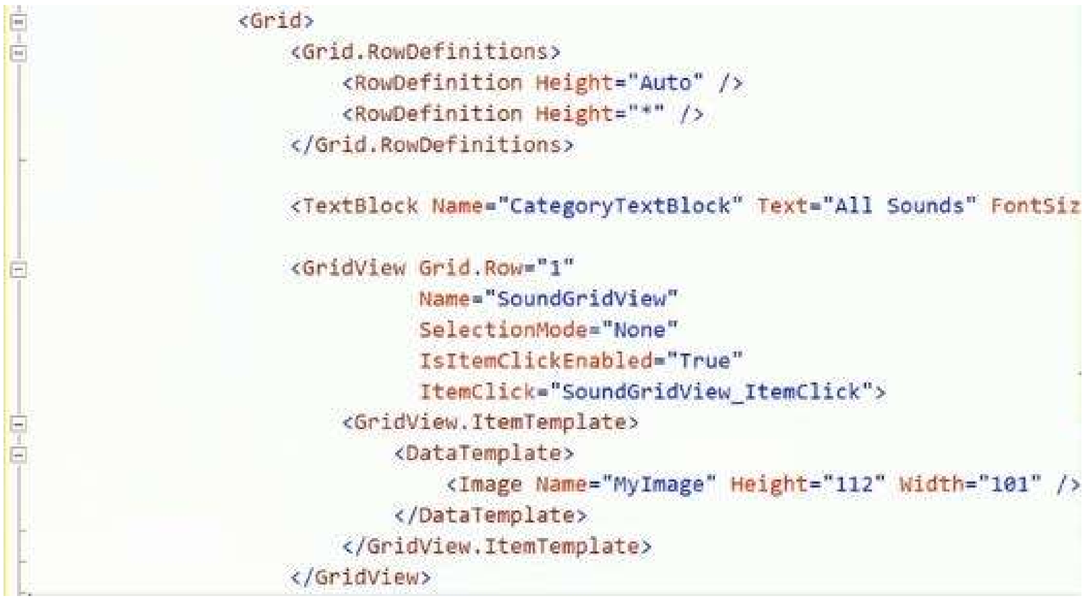
```
        <SplitView Name="MySplitView"
                   DisplayMode="CompactOverlay"
                   CompactPaneLength="45"
                   OpenPaneLength="200">
            <SplitView.Pane>
                <ListView Name="MenuItemsListView" IsItemClickEnabled="True"
            </SplitView.Pane>
            <SplitView.Content>
                <Grid>

                </Grid>
            </SplitView.Content>

        </SplitView>

    </Grid>
```

And add this at the end of `<ListView`



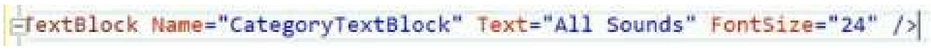
```
ItemClick="MenuItemsListView_ItemClick" />
```

Step 7: Add this code in between the SplitView.Content Grid



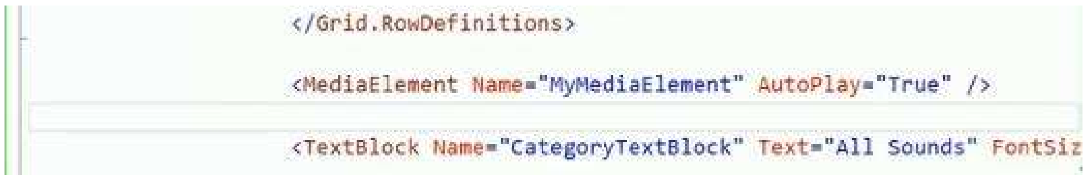
```
                <Grid>
                    <Grid.RowDefinitions>
                        <RowDefinition Height="Auto" />
                        <RowDefinition Height="*" />
                    </Grid.RowDefinitions>

                    <TextBlock Name="CategoryTextBlock" Text="All Sounds" FontSiz

                    <GridView Grid.Row="1"
                              Name="SoundGridView"
                              SelectionMode="None"
                              IsItemClickEnabled="True"
                              ItemClick="SoundGridView_ItemClick">
                        <GridView.ItemTemplate>
                            <DataTemplate>
                                <Image Name="MyImage" Height="112" Width="101" />
                            </DataTemplate>
                        </GridView.ItemTemplate>
                    </GridView>
```

And the TextBlock will look like a Header so make its FontSize fairly large



```
TextBlock Name="CategoryTextBlock" Text="All Sounds" FontSize="24" />
```

Step 8: Next we will add a Media Element, which allows you to play sounds



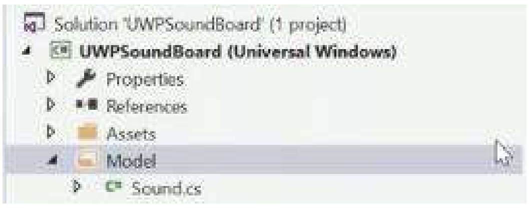
```
</Grid.RowDefinitions>

<MediaElement Name="MyMediaElement" AutoPlay="True" />

<TextBlock Name="CategoryTextBlock" Text="All Sounds" FontSiz
```

Having completed all of that, you now have something akin to the shell of an App which we will continue to build throughout the next series of lessons.

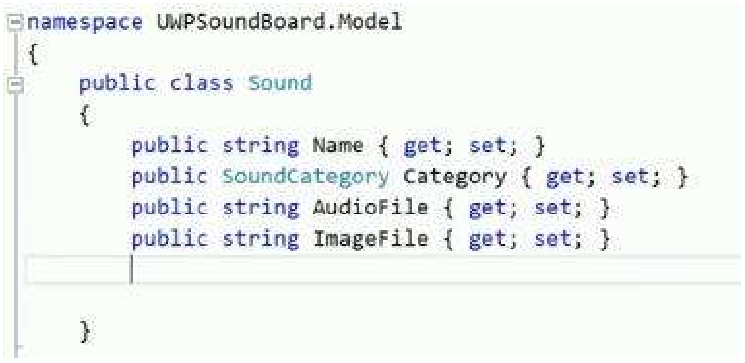
## UWP-051 - UWP SoundBoard - Creating the Data Model & Data Binding

The next thing we will do is create the Data Model with a class called “Sound,” and Sound, in turn, will have an audio file, an image file, and a category. After that we will create a Sound Manager, which will allow us to either get access to all sounds, or just a sound category.

Step 1: Create a folder called “Model” and add to it a class called “Sound”

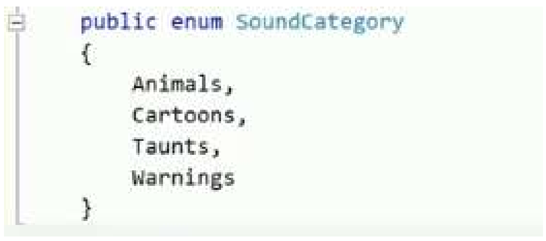

Solution 'UWPSoundBoard' (1 project)
UWPSoundBoard (Universal Windows)
Properties
References
Assets
Model
Sound.cs

Step 2: In the Sound class type in the following properties (we will create the SoundCategory enum in the next step)



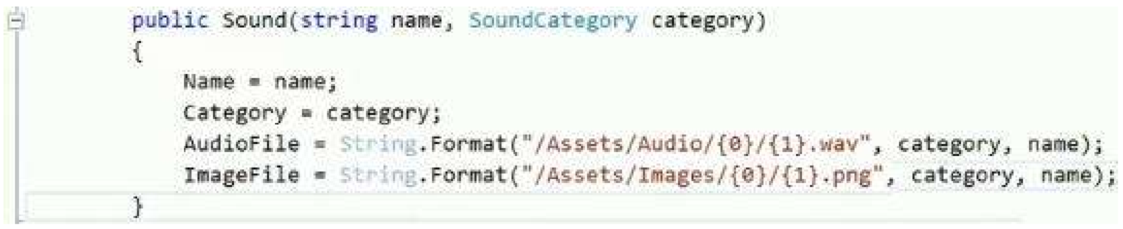
```
namespace UWPSoundBoard.Model
{
    public class Sound
    {
        public string Name { get; set; }
        public SoundCategory Category { get; set; }
        public string AudioFile { get; set; }
        public string ImageFile { get; set; }


    }
```

Step 3: Create an enum called SoundCategory which correlates to the different types of sounds we have in the Audio folder

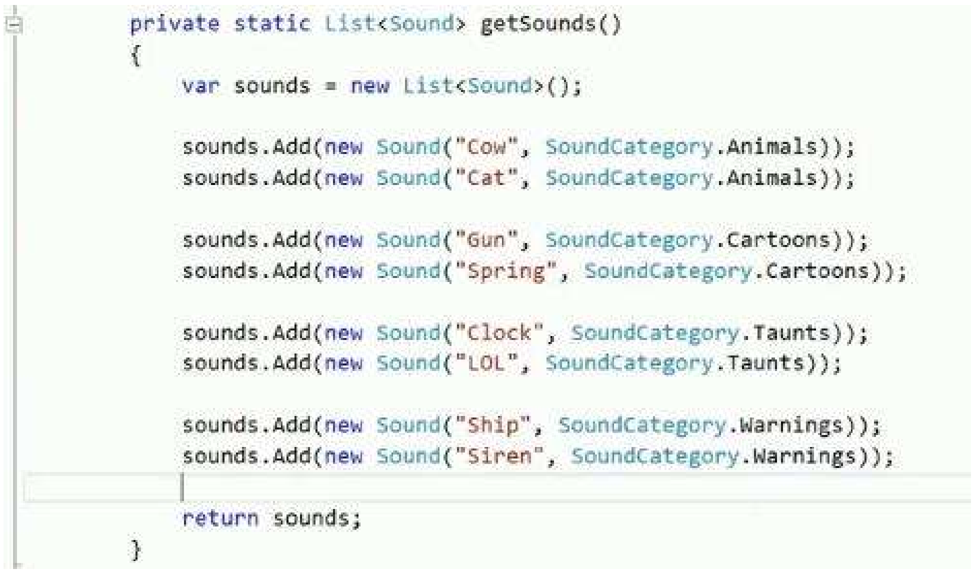


```
    public enum SoundCategory
    {
        Animals,
        Cartoons,
        Taunts,
        Warnings
    }
```

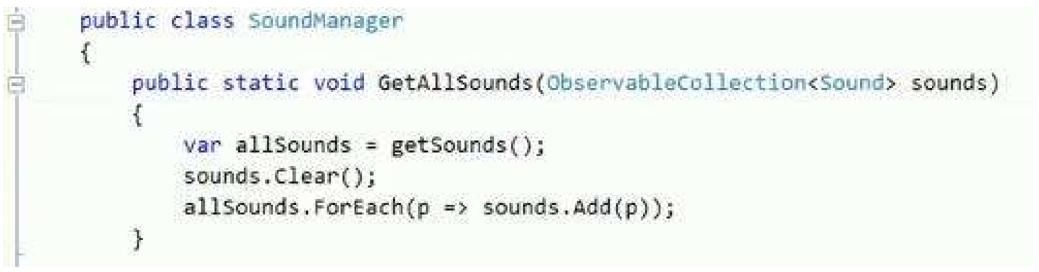
Step 4: Next, create a Constructor for Sound that contains references to the asset folders for Audio and Images


```
public Sound(string name, SoundCategory category)
{
    Name = name;
    Category = category;
    AudioFile = String.Format("/Assets/Audio/{0}/{1}.wav", category, name);
    ImageFile = String.Format("/Assets/Images/{0}/{1}.png", category, name);
}
```

Step 5: Now, add a new class to the Model folder and call it “SoundManager.” Within the SoundManager class create a static method called getSounds() that returns a List of type Sound.

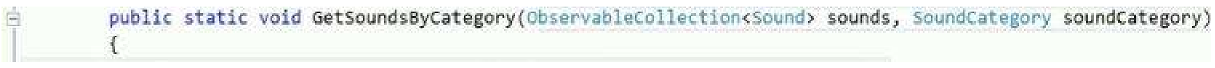

```
private static List<Sound> getSounds()
{
    var sounds = new List<Sound>();

    sounds.Add(new Sound("Cow", SoundCategory.Animals));
    sounds.Add(new Sound("Cat", SoundCategory.Animals));

    sounds.Add(new Sound("Gun", SoundCategory.Cartoons));
    sounds.Add(new Sound("Spring", SoundCategory.Cartoons));

    sounds.Add(new Sound("Clock", SoundCategory.Taunts));
    sounds.Add(new Sound("LOL", SoundCategory.Taunts));

    sounds.Add(new Sound("Ship", SoundCategory.Warnings));
    sounds.Add(new Sound("Siren", SoundCategory.Warnings));

    return sounds;
}
```

Step 6: Also add to the SoundManager class a static method as follows


```
public class SoundManager
{
    public static void GetAllSounds(ObservableCollection<Sound> sounds)
    {
        var allSounds = getSounds();
        sounds.Clear();
        allSounds.ForEach(p => sounds.Add(p));
    }
```

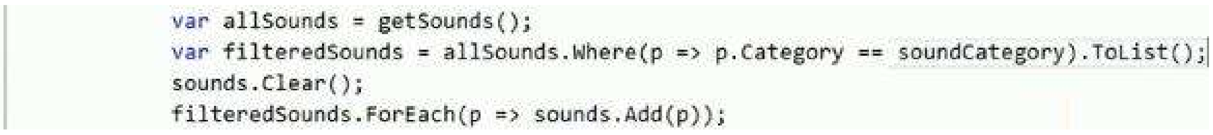
The objective with this method is to be able to pass in an ObservableCollection of type Sound as that's what will be passing in from MainPage.xaml.cs to this SoundManager. Note that sounds.Clear() is referenced because you will want to get rid of anything that's already in the Collection. Also note how the allSounds.ForEach() method allows us to pass in a lambda expression, which is used to loop through every sound that was pulled from getSounds() and stored into allSounds. This loop is responsible for iterating over each sound, and then add it to the variable called “sounds” of type ObservableCollection<Sound>.

Step 7: Next, create a static method within the class called GetSoundsByCategory(). This will have a very similar task as the GetAllSounds() method, passing in an ObservableCollection<Sound> called sounds, and also a SoundCategory called soundCategory.



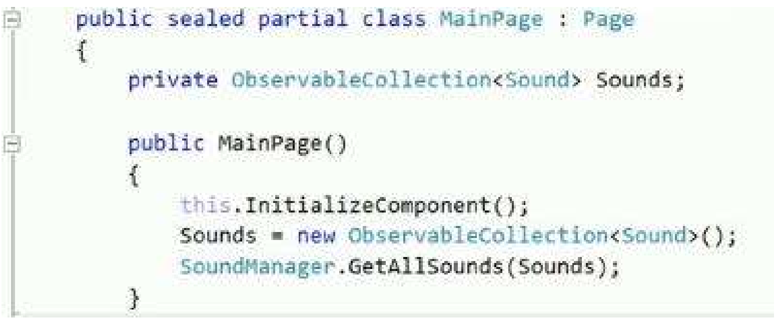
```
        public static void GetSoundsByCategory(ObservableCollection<Sound> sounds, SoundCategory soundCategory)
        {
```

And inside of the GetSoundsByCategory() method write the following code



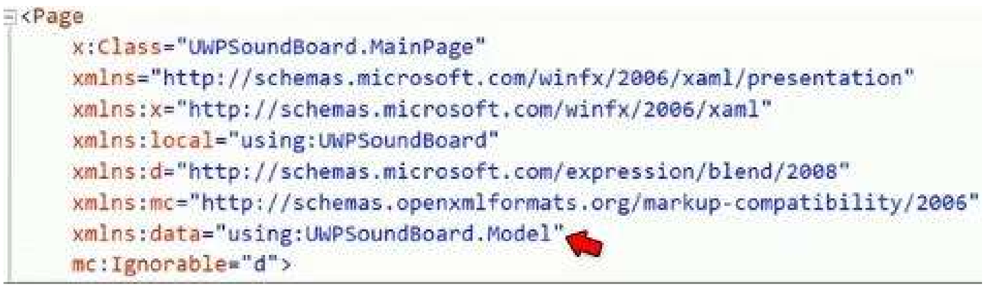
```
            var allSounds = getSounds();
            var filteredSounds = allSounds.Where(p => p.Category == soundCategory).ToList();
            sounds.Clear();
            filteredSounds.ForEach(p => sounds.Add(p));
```

Step 8: Now, in MainPage.xaml.cs we will want to create a private reference to ObservableCollection<Sound> and then in the Constructor initialize the Component and then call SoundManager.GetAllSounds() to populate the List of sounds.



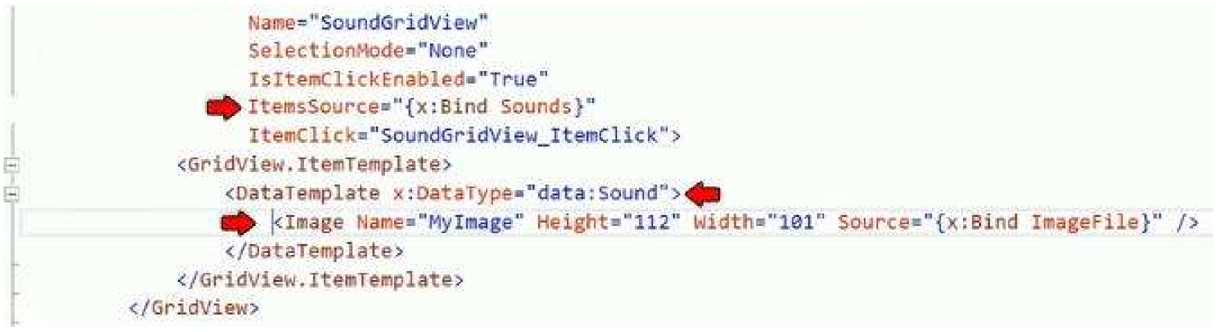
```
    public sealed partial class MainPage : Page
    {
        private ObservableCollection<Sound> Sounds;

        public MainPage()
        {
            this.InitializeComponent();
            Sounds = new ObservableCollection<Sound>();
            SoundManager.GetAllSounds(Sounds);
        }
```

Step 9: And then in MainPage.xaml create an XML namespace for data



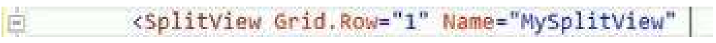
```
<Page
    x:Class="UWPSoundBoard.MainPage"
    xmlns="http://schemas.microsoft.com/winfx/2006/xaml/presentation"
    xmlns:x="http://schemas.microsoft.com/winfx/2006/xaml"
    xmlns:local="using:UWPSoundBoard"
    xmlns:d="http://schemas.microsoft.com/expression/blend/2008"
    xmlns:mc="http://schemas.openxmlformats.org/markup-compatibility/2006"
    xmlns:data="using:UWPSoundBoard.Model"
    mc:Ignorable="d">
```

And in the same file also add these edits



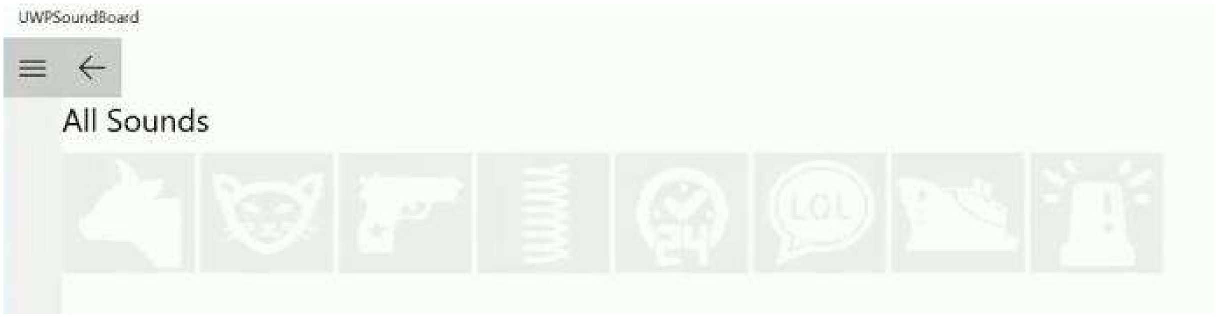
```
                Name="SoundGridView"
                SelectionMode="None"
                IsItemClickEnabled="True"
                ItemsSource="{x:Bind Sounds}"
                ItemClick="SoundGridView_ItemClick">
            <GridView.ItemTemplate>
                <DataTemplate x:DataType="data:Sound">
                    <Image Name="MyImage" Height="112" Width="101" Source="{x:Bind ImageFile}" />
                </DataTemplate>
            </GridView.ItemTemplate>
        </GridView>
```

Finally, adjust the SplitView by adding Grid.Row=”1”

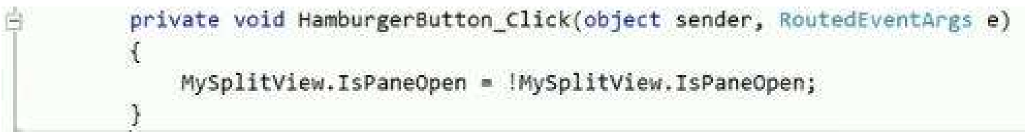


```
        <SplitView Grid.Row="1" Name="MySplitView"
```

At the end of all of that, you should run the program and see something that looks like this

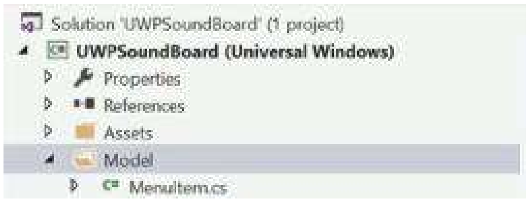


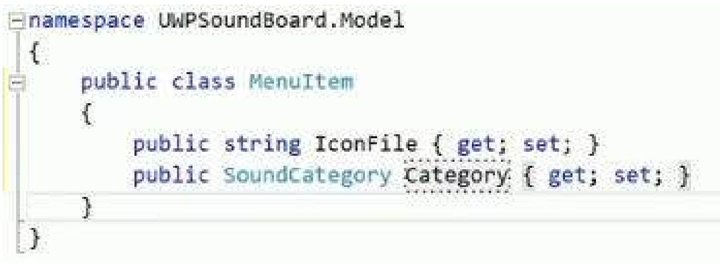
Step 10: Now, to get the HamburgerButton_Click() working in MainPage.xaml.cs add the following code



```
        private void HamburgerButton_Click(object sender, RoutedEventArgs e)
        {
            MySplitView.IsPaneOpen = !MySplitView.IsPaneOpen;
        }
```

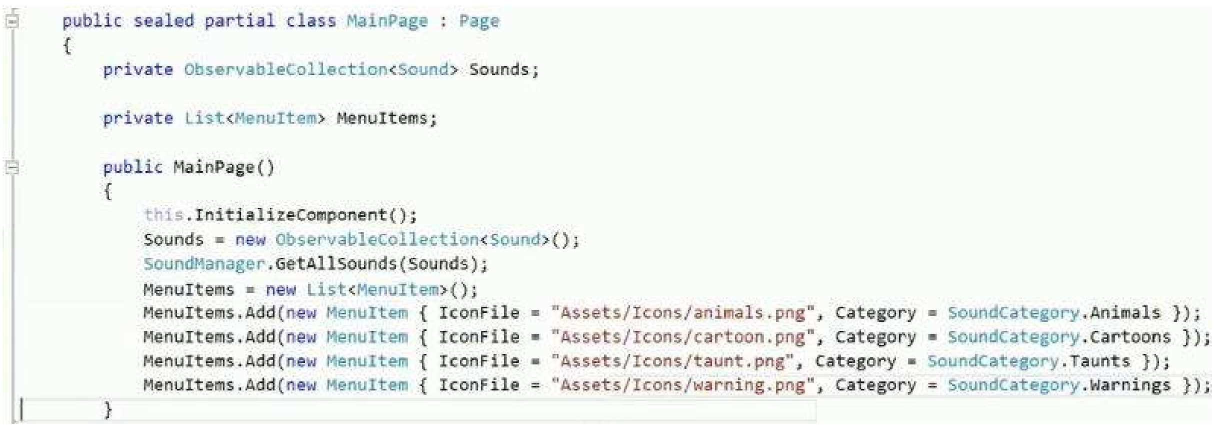
Step 11: Let’s add a class called MenuItem, and then inside the class write some code to help filter the sounds based on the selected menu item in the SplitView.Pane


Solution 'UWPSoundBoard' (1 project)
UWPSoundBoard (Universal Windows)
Properties
References
Assets
Model
MenuItem.cs



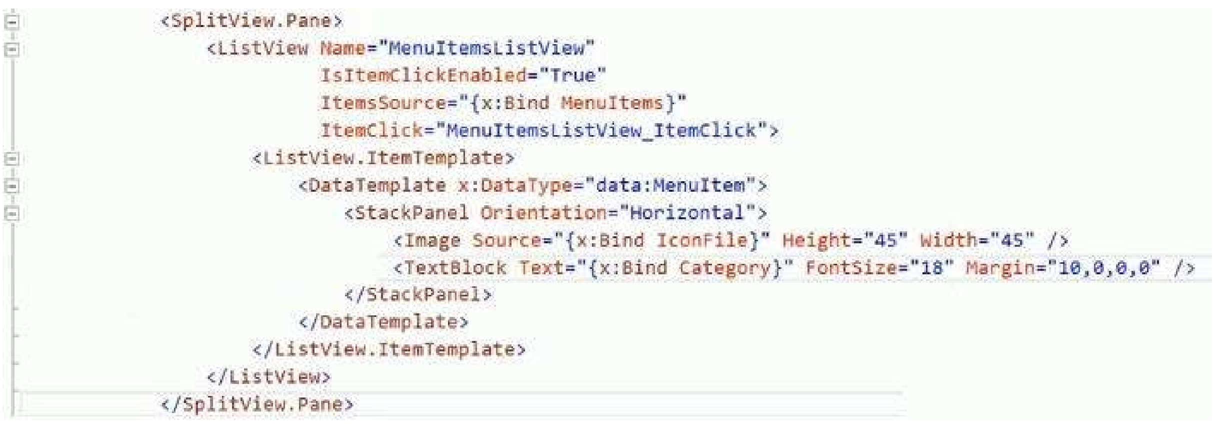
```
namespace UWPSoundBoard.Model
{
    public class MenuItem
    {
        public string IconFile { get; set; }
        public SoundCategory Category { get; set; }
    }
}
```

And then add a private List called MenuItems, and populate it in the Constructor by adding the following code.



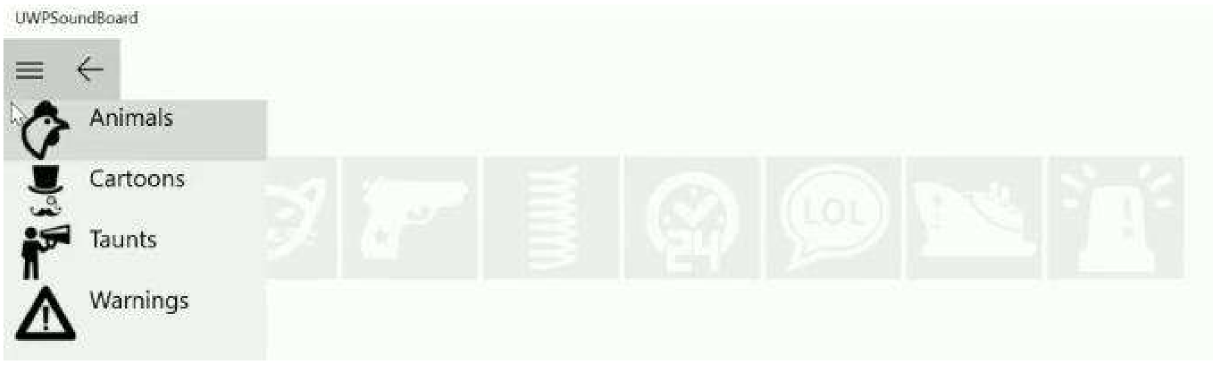
```
    public sealed partial class MainPage : Page
    {
        private ObservableCollection<Sound> Sounds;

        private List<MenuItem> MenuItems;

        public MainPage()
        {
            this.InitializeComponent();
            Sounds = new ObservableCollection<Sound>();
            SoundManager.GetAllSounds(Sounds);
            MenuItems = new List<MenuItem>();
            MenuItems.Add(new MenuItem { IconFile = "Assets/Icons/animals.png", Category = SoundCategory.Animals });
            MenuItems.Add(new MenuItem { IconFile = "Assets/Icons/cartoon.png", Category = SoundCategory.Cartoons });
            MenuItems.Add(new MenuItem { IconFile = "Assets/Icons/taunt.png", Category = SoundCategory.Taunts });
            MenuItems.Add(new MenuItem { IconFile = "Assets/Icons/warning.png", Category = SoundCategory.Warnings });
        }
```

Then, make the following changes to the SplitView.Pane



```
            <SplitView.Pane>
                <ListView Name="MenuItemsListView"
                          IsItemClickEnabled="True"
                          ItemsSource="{x:Bind MenuItems}"
                          ItemClick="MenuItemsListView_ItemClick">
                    <ListView.ItemTemplate>
                        <DataTemplate x:DataType="data:MenuItem">
                            <StackPanel Orientation="Horizontal">
                                <Image Source="{x:Bind IconFile}" Height="45" Width="45" />
                                <TextBlock Text="{x:Bind Category}" FontSize="18" Margin="10,0,0,0" />
                            </StackPanel>
                        </DataTemplate>
                    </ListView.ItemTemplate>
                </ListView>
            </SplitView.Pane>
```

And, now when you run the program you should now see the icons on the left-hand side.



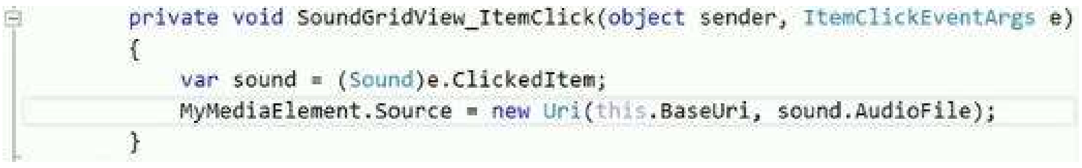
There will be some minor issues needing to be fixed, but this should be a good starting point for now. In the next lesson we will hook things up to the Media Element in order to start playing sounds.

## UWP-052 - UWP SoundBoard - Playing Sounds with the Media Element

Now comes the fun part of wiring up the Media Element in order to play sounds, and setting up filtering by category.

Step 1: First let's grab the Item when it becomes clicked, which will be referenced by ItemClickEventArgs e in the SoundGridView_ItemClick() method within MainPage.xaml.cs

Write the following code within that method



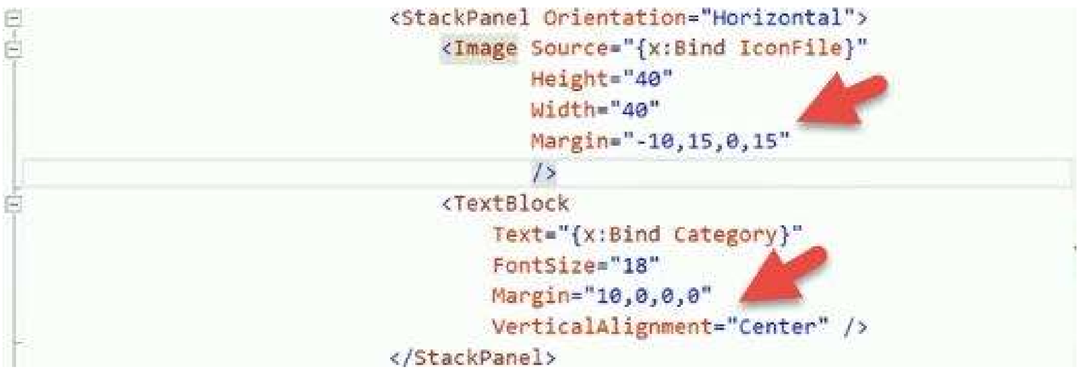
```
        private void SoundGridView_ItemClick(object sender, ItemClickEventArgs e)
        {
            var sound = (Sound)e.ClickedItem;
            MyMediaElement.Source = new Uri(this.BaseUri, sound.AudioFile);
        }
```

The first step grabs the clicked item referenced in e.ClickedItem, which then immediately becomes cast to type Sound, and then it stores that in a temporary variable called "sound."

Notice when you hover over the Source property that it accepts a URI (Uniform Resource Indicator), which is just going to be a location for the file. The easiest way to handle this is just to specify the base URI - that will give us the root of the project - and then we can give it the actual audio file that will contain the specific location of that /Assets/Audio/Category/Sound in question.

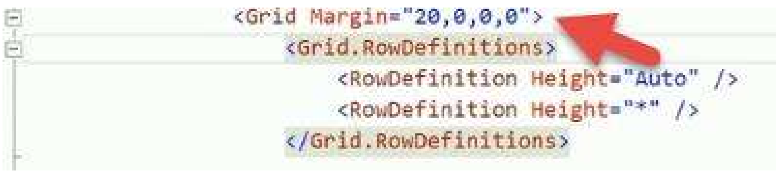
When you run the program you should notice that clicking on the items now produces the corresponding sound.

Step 2: Next, let's fix some issues with the visual elements by making the following changes to the StackPanel



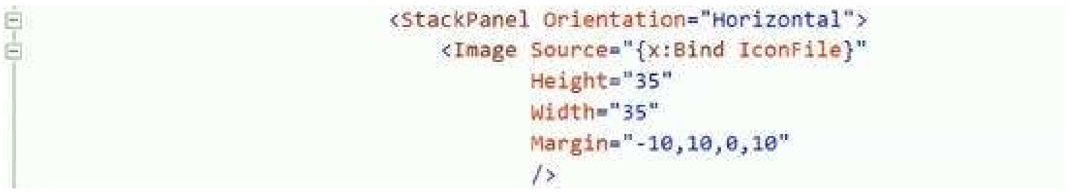
```
                        <StackPanel Orientation="Horizontal">
                            <Image Source="{x:Bind IconFile}"
                                   Height="40"
                                   Width="40"
                                   Margin="-10,15,0,15"
                                   />
                            <TextBlock
                                Text="{x:Bind Category}"
                                FontSize="18"
                                Margin="10,0,0,0"
                                VerticalAlignment="Center" />
                        </StackPanel>
```

And also adjust the Grid Margin as follows

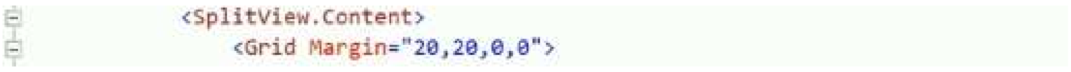


```
                <Grid Margin="20,0,0,0">
                    <Grid.RowDefinitions>
                        <RowDefinition Height="Auto" />
                        <RowDefinition Height="*" />
                    </Grid.RowDefinitions>
```

Step 3: It's looking a lot better now but we can make a few minor adjustments – tinkering with the margins, and sizing, until it all starts looking right

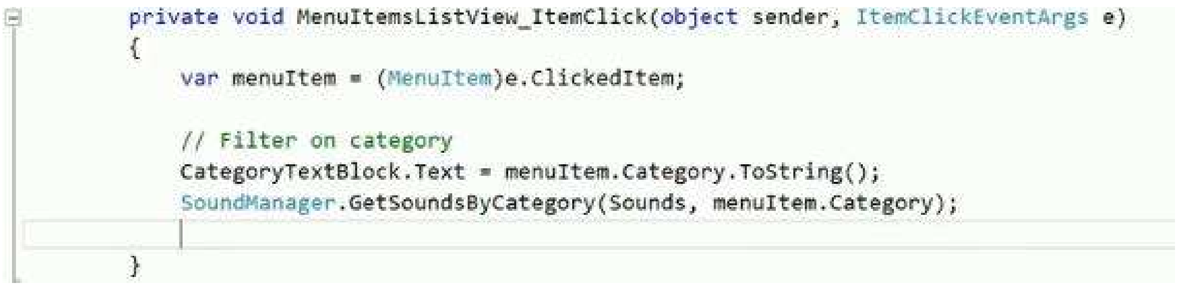


```
                        <StackPanel Orientation="Horizontal">
                            <Image Source="{x:Bind IconFile}"
                                   Height="35"
                                   Width="35"
                                   Margin="-10,10,0,10"
                                   />
```



```
            <SplitView.Content>
                <Grid Margin="20,20,0,0">
```

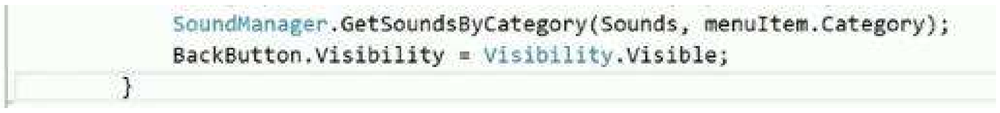
Step 4: Next, let's get the category filtering to work. Go back to MainPage.xaml.cs and this time reference ItemClickEventArgs e within the MenuItemsListView_ItemClick() method, similar to how we did it before



```
        private void MenuItemsListView_ItemClick(object sender, ItemClickEventArgs e)
        {
            var menuItem = (MenuItem)e.ClickedItem;

            // Filter on category
            CategoryTextBlock.Text = menuItem.Category.ToString();
            SoundManager.GetSoundsByCategory(Sounds, menuItem.Category);

        }
```

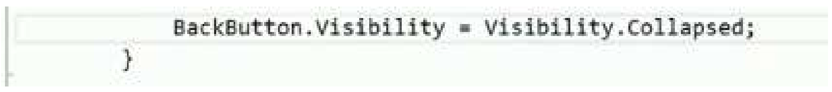
Notice how we call SoundManager.GetSoundsByCategory() this time, passing in our ObservableCollection<Sound> Sounds, and then the category as arguments. When you run the program you should now be able to click on the icons to filter by category.

Step 5: Now, let's get the back button to work – which should really only be displayed when we click on one of the items. We do that by setting the visibility for it once an item is clicked, again inside of the ~~SoundGridView_ItemClick()~~ method MenuItemsListView_ItemClick



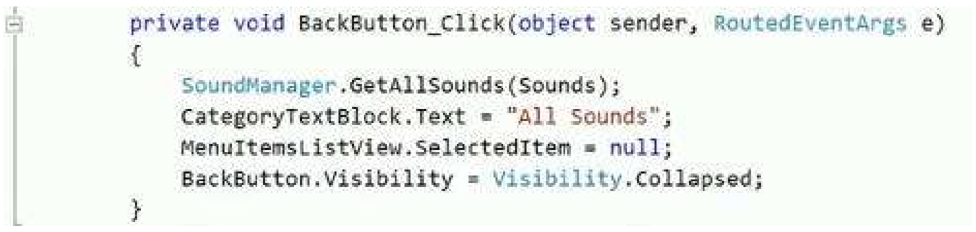
```
            SoundManager.GetSoundsByCategory(Sounds, menuItem.Category);
            BackButton.Visibility = Visibility.Visible;
        }
```

And, since we do not want the back button visible otherwise, set it to Visibility.Collapsed in the MainPage() Constructor



```
            BackButton.Visibility = Visibility.Collapsed;
        }
```

And, of course, when we hit the back button we will want to navigate back to All Sounds so add this code as well



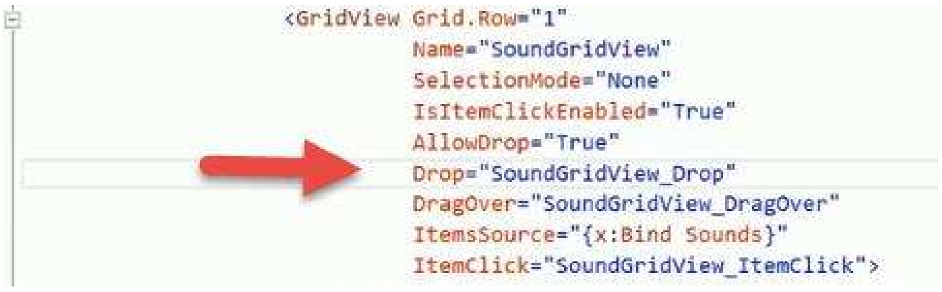
```
private void BackButton_Click(object sender, RoutedEventArgs e)
{
    SoundManager.GetAllSounds(Sounds);
    CategoryTextBlock.Text = "All Sounds";
    MenuItemsListView.SelectedItem = null;
    BackButton.Visibility = Visibility.Collapsed;
}
```

When you run the program you should now notice the back button appears and works as we intended. In the next lesson we will look into adding drag/drop functionality for adding sounds by, for example, dragging and dropping from your desktop.

## UWP-053 - UWP SoundBoard - Adding Drag and Drop

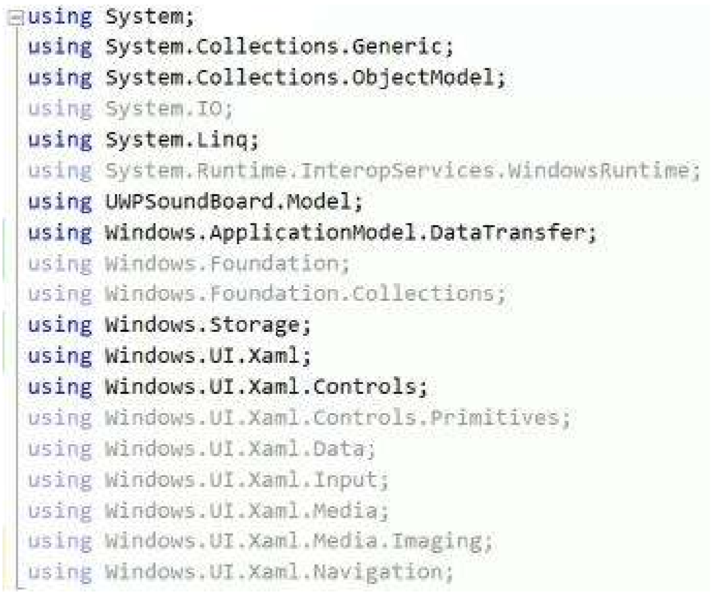
In this lesson we'll be adding drag/drop functionality, allowing us to audition sounds by placing them onto the SoundBoard app. Let's start off our drag/drop feature by going back to MainPage.xaml and modifying the GridView Control

Step 1: Add AllowDrop, Drop, and DragOver to the GridView to being referencing the drag/drop events within MainPage.xaml



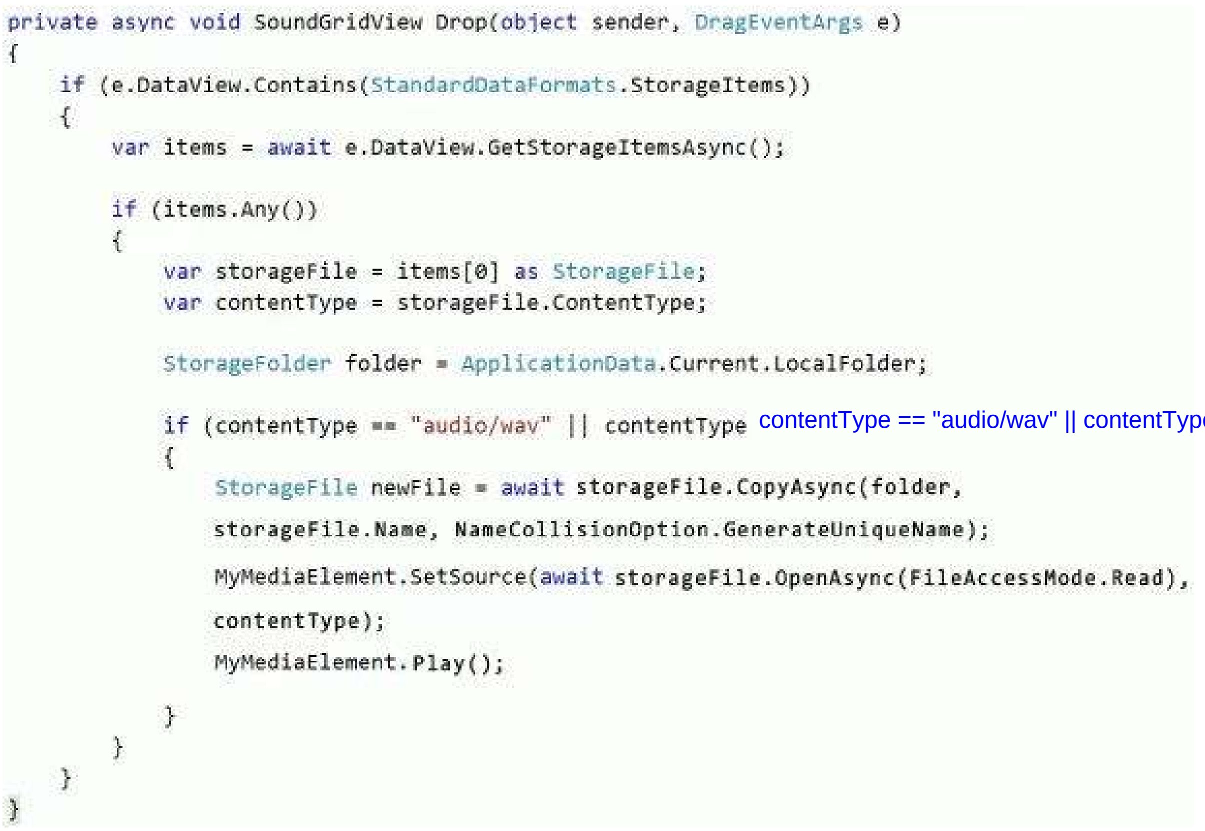
```
<GridView Grid.Row="1"
          Name="SoundGridView"
          SelectionMode="None"
          IsItemClickEnabled="True"
          AllowDrop="True"
          Drop="SoundGridView_Drop"
          DragOver="SoundGridView_DragOver"
          ItemsSource="{x:Bind Sounds}"
          ItemClick="SoundGridView_ItemClick">
```

Step 2: In MainPage.xaml.cs make sure your using statements are declared to import all of the required namespaces



```
using System;
using System.Collections.Generic;
using System.Collections.ObjectModel;
using System.IO;
using System.Linq;
using System.Runtime.InteropServices.WindowsRuntime;
using UWPSoundBoard.Model;
using Windows.ApplicationModel.DataTransfer;
using Windows.Foundation;
using Windows.Foundation.Collections;
using Windows.Storage;
using Windows.UI.Xaml;
using Windows.UI.Xaml.Controls;
using Windows.UI.Xaml.Controls.Primitives;
using Windows.UI.Xaml.Data;
using Windows.UI.Xaml.Input;
using Windows.UI.Xaml.Media;
using Windows.UI.Xaml.Media.Imaging;
using Windows.UI.Xaml.Navigation;
```

And now in the SoundGridView_ItemClick() method insert the following code



```
private async void SoundGridView_Drop(object sender, DragEventArgs e)
{
    if (e.DataView.Contains(StandardDataFormats.StorageItems))
    {
        var items = await e.DataView.GetStorageItemsAsync();

        if (items.Any())
        {
            var storageFile = items[0] as StorageFile;
            var contentType = storageFile.ContentType;

            StorageFolder folder = ApplicationData.Current.LocalFolder;

            if (contentType == "audio/wav" || contentType contentType == "audio/wav" || contentType == "audio/mpeg"
            {
                StorageFile newFile = await storageFile.CopyAsync(folder,
                storageFile.Name, NameCollisionOption.GenerateUniqueName);
                MyMediaElement.SetSource(await storageFile.OpenAsync(FileAccessMode.Read),
                contentType);
                MyMediaElement.Play();

            }
        }
    }
}
```

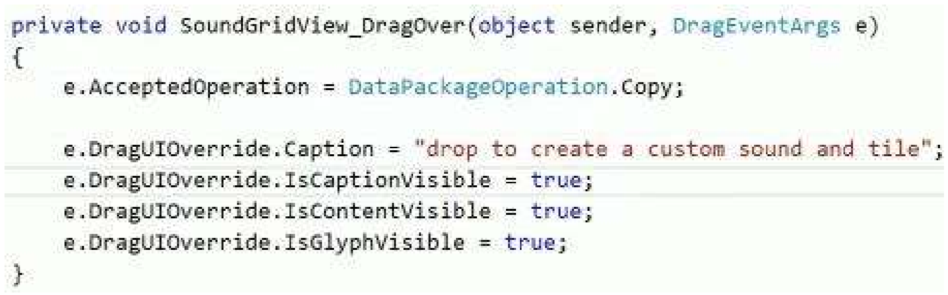
There are a few things worth mentioning about this code. First, whenever you see the await, or async, keyword, it's essentially saying that the method that it's attached to could be long-running in nature. And, instead of allowing that long-running method to block the ongoing operation of all of the other code beneath the current line (where that method is called), instead allow the rest of the code to continue running until the process is finished. And when everything is done it'll all magically work together.

There are two different ways to introduce asynchrony into your applications based on the two different types of blocking (as in "roadblock"). There's blocking due to an operation being mathematically intensive, and then there's blocking because of indeterminate lag time (such as calls made out over a network).

So, without using async/await, a request over the internet could cause an app to wait – halting all other execution - until a response came back from a remote web server. In this case, there's are not a lot of computational time being spent, it's just lag time that will appear to the user as though the application is unresponsive and stalling.

Those are the basics regarding async/await. Now, the rest of the code is basically responsible for pulling off the ContentType of the sound file, getting a reference to that local folder, and then checking the contentType. And if its type is audio/wav tell the media player to go ahead and play that file. But, before it plays, what happens is the application actually copies that file into your local storage area, and then reads from that location.

Step 3: Write the following code for the SoundGridView_DragOver() method/event



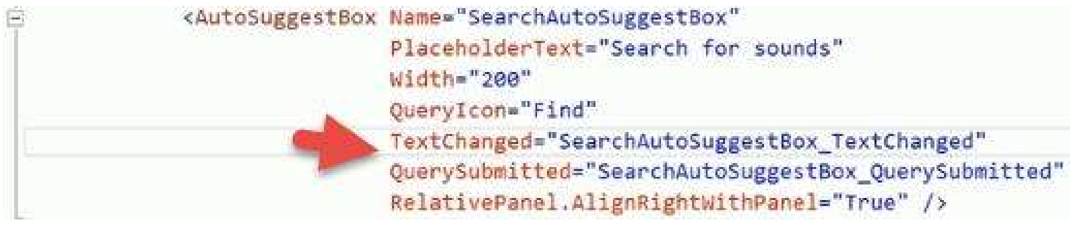
```
private void SoundGridView_DragOver(object sender, DragEventArgs e)
{
    e.AcceptedOperation = DataPackageOperation.Copy;

    e.DragUIOverride.Caption = "drop to create a custom sound and tile";
    e.DragUIOverride.IsCaptionVisible = true;
    e.DragUIOverride.IsContentVisible = true;
    e.DragUIOverride.IsGlyphVisible = true;
}
```

Now, when you run the program you should be able to drag and drop a sound from anywhere on your computer, and onto an icon and have the sound play back.

## UWP-054 - UWP SoundBoard - Finishing Touches

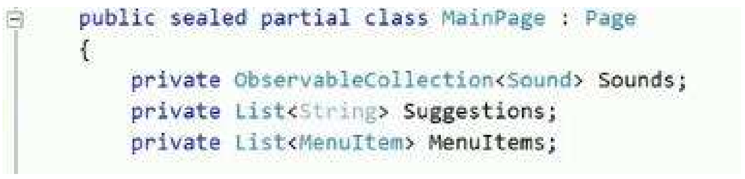
The next thing we will want to do is create an auto-suggestion search filter to refine down the items in the results to just those that match the search criteria.

Step 1: To implement this we have two Events which we will reference in MainPage.xaml via TextChanged and QuerySubmitted.



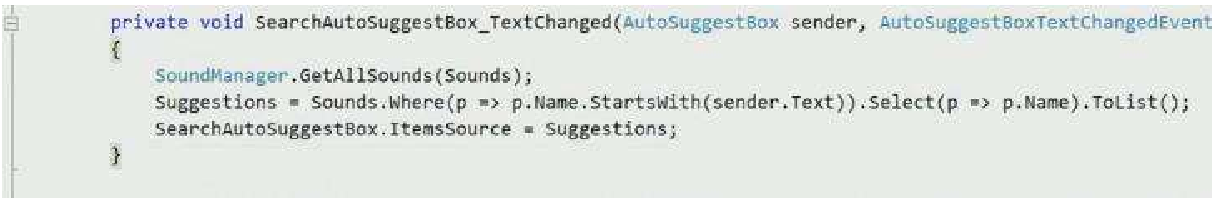
```
<AutoSuggestBox Name="SearchAutoSuggestBox"
                PlaceholderText="Search for sounds"
                Width="200"
                QueryIcon="Find"
                TextChanged="SearchAutoSuggestBox_TextChanged"
                QuerySubmitted="SearchAutoSuggestBox_QuerySubmitted"
                RelativePanel.AlignRightWithPanel="True" />
```

Step 2: Create a List<String> at the class level in MainPage.xaml.cs



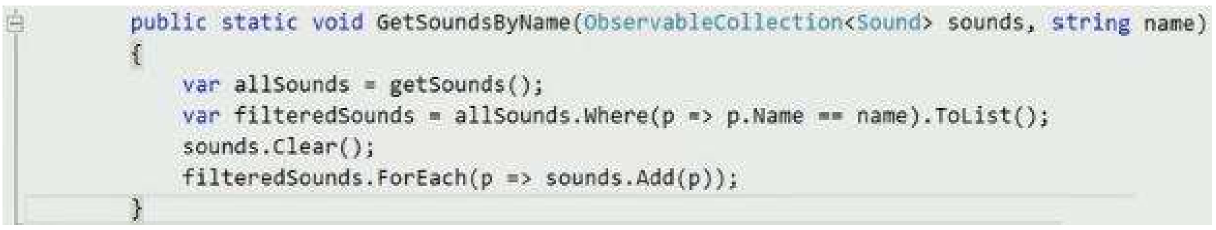
```
public sealed partial class MainPage : Page
{
    private ObservableCollection<Sound> Sounds;
    private List<String> Suggestions;
    private List<MenuItem> MenuItems;
```

Step 3: Populate this list by writing the following in the SearchAutoSuggestBox_TextChanged() method



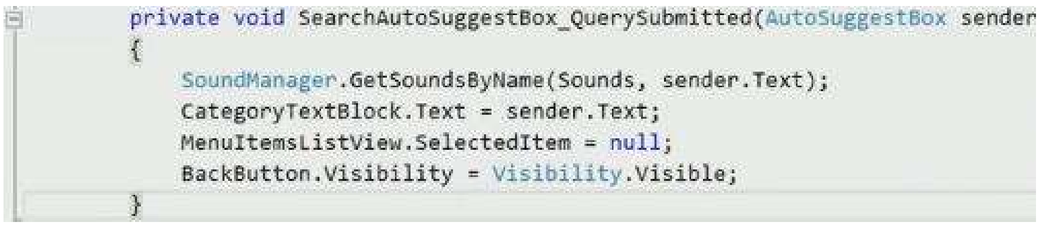
```
private void SearchAutoSuggestBox_TextChanged(AutoSuggestBox sender, AutoSuggestBoxTextChangedEvent
{
    SoundManager.GetAllSounds(Sounds);
    Suggestions = Sounds.Where(p => p.Name.StartsWith(sender.Text)).Select(p => p.Name).ToList();
    SearchAutoSuggestBox.ItemsSource = Suggestions;
}
```

Step 4: Create a custom method that will filter a set of sounds in class SoundManager.cs



```
public static void GetSoundsByName(ObservableCollection<Sound> sounds, string name)
{
    var allSounds = getSounds();
    var filteredSounds = allSounds.Where(p => p.Name == name).ToList();
    sounds.Clear();
    filteredSounds.ForEach(p => sounds.Add(p));
}
```

Step 5: Then, inside of the SearchAutoSuggestBox_QuerySubmitted() we will reference that last method as well as write the following

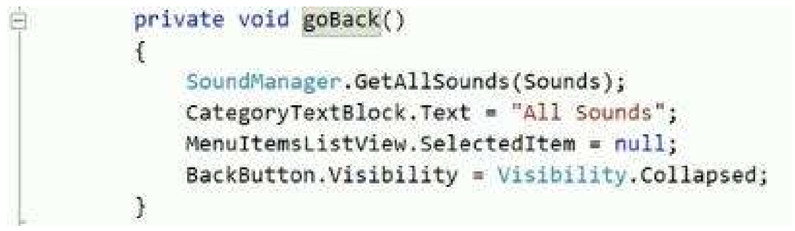


```
        private void SearchAutoSuggestBox_QuerySubmitted(AutoSuggestBox sender
        {
            SoundManager.GetSoundsByName(Sounds, sender.Text);
            CategoryTextBlock.Text = sender.Text;
            MenuItemsListView.SelectedItem = null;
            BackButton.Visibility = Visibility.Visible;
        }
```

## UWP-055 - UWP SoundBoard - Add Assets with Package.AppXManifest

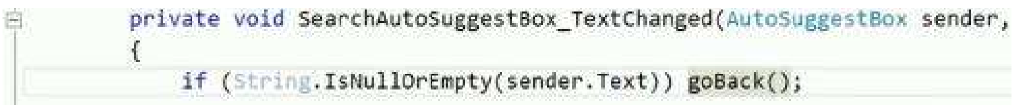
In the last lesson we created an auto-suggest search filter, however, there was a small issue with the search box retaining what we last typed in. There is a simple fix to this that you can achieve with the following code:

Step 1: Create a custom method within MainPage.xaml.cs



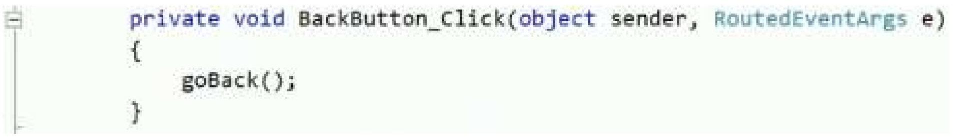
```
private void goBack()
{
    SoundManager.GetAllSounds(Sounds);
    CategoryTextBlock.Text = "All Sounds";
    MenuItemsListView.SelectedItem = null;
    BackButton.Visibility = Visibility.Collapsed;
}
```

Step 2: Call that method here



```
private void SearchAutoSuggestBox_TextChanged(AutoSuggestBox sender,
{
    if (String.IsNullOrEmpty(sender.Text)) goBack();
```

And here

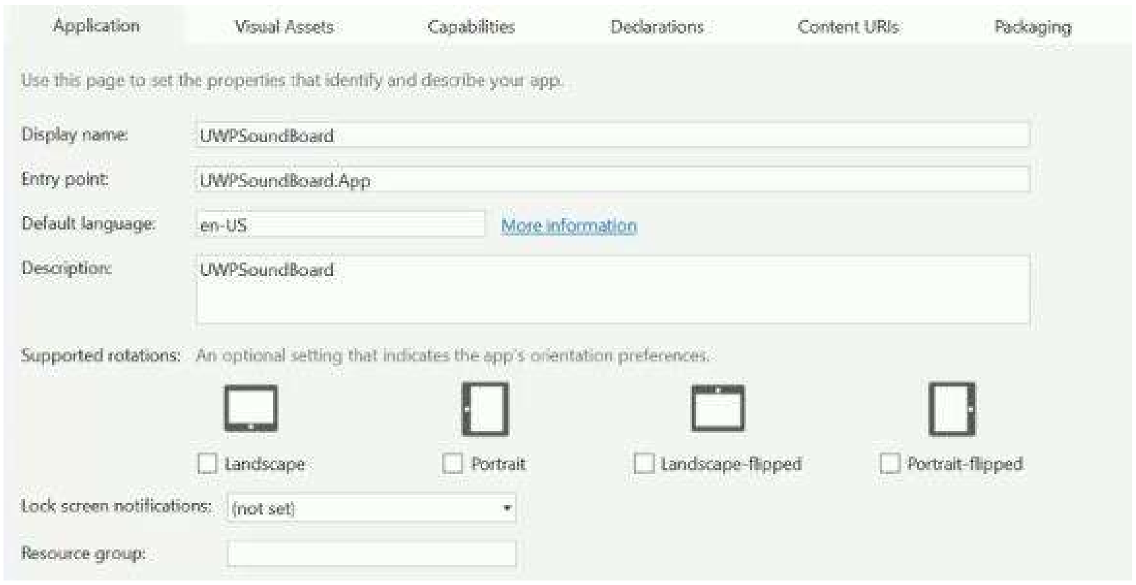


```
private void BackButton_Click(object sender, RoutedEventArgs e)
{
    goBack();
}
```

So, the GoBack() method contains all of the functionality necessary to put the application in a valid state after we move away from the search functionality, or whenever we hit the back button. It achieves this by grabbing all of the sounds, set the CategoryTextBlock.Text to “All Sounds,” and then set the MenuItemsListView.SelectedItem to no selection, and then  set the BackButton.Visibility to Collapsed.

So, the next thing is to do is touch upon deployment – covering a bit on how your applications will actually be deployed onto the end-user computer. As a developer, you don't write any installers in your routines to install/uninstall your Windows runtime app (the app that you're building).  Instead, you package your app and you submit it to the Windows Store. Then, users will acquire your app from the Store in the form of an "App Package." The operating system uses information that you supply in an app package to install the app and ensure that all traces of the app are gone from the device whenever they choose to uninstall that app.  To understand how this works, refer to the Package.appxmaninfest file in the Solution Explorer.

When you double-click that a beautiful designer pops up. But actually what this file is, is just XML - it just has a nice Visual Studio interface on top of it so you don't have to deal directly with the XML.  But essentially that Package.AppXManifest file is just a document that contains the information that the system needs to deploy, to display, or to update a Windows app.



Use this page to set the properties that identify and describe your app.

Display name: UWPSoundBoard

Entry point: UWPSoundBoard.App

Default language: en-US

Description: UWPSoundBoard

Supported rotations: An optional setting that indicates the app's orientation preferences.

☐ Landscape ☐ Portrait ☐ Landscape-flipped ☐ Portrait-flipped

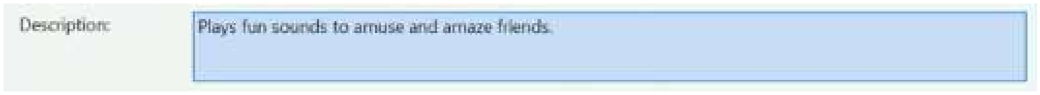
Lock screen notifications: (not set)

Resource group:

You will also notice there are tabbed sections that allow you to further configure your package for deployment. Among all of this information are things like the identity of the Package, any dependencies that it has on other DLL's or outside resources, and any capabilities that it requires in order to run. Therefore, every app package must include at least one Package.AppXManifest file.

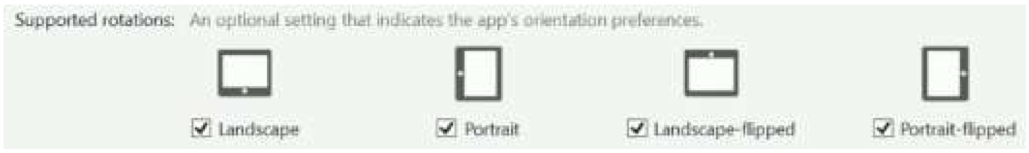
Note that the package manifest is digitally signed as a part of signing the app package. After it's been signed, you can't modify the manifest without invalidating the package signature. So it contains some security features for the end user to ensure that they're getting the application that you built and that somebody didn't hack in and make some changes to it before it gets installed on a computer.

Focusing first on the Application tab, let’s change how the app is displayed in the Store, such as its description that will be displayed



Description: Plays fun sounds to amuse and amaze friends.

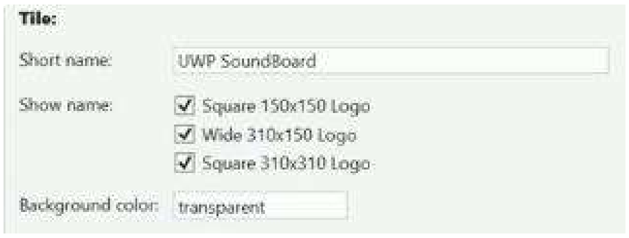
Also select all of the supported rotations



Supported rotations: An optional setting that indicates the app's orientation preferences.

☑ Landscape ☑ Portrait ☑ Landscape-flipped ☑ Portrait-flipped

Now, under the Visual Assets tab first set the tile attributes, such as the short name for your app, and the applicable logos (which we will setup in a moment).


**Tile:**

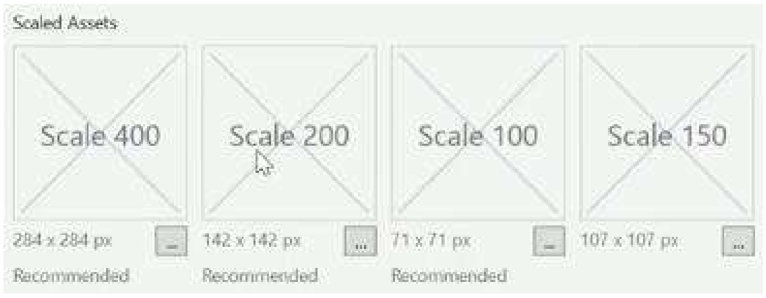
| | |
|---|---|
| Short name: | UWP SoundBoard |
| Show name: | ✓ Square 150x150 Logo<br>✓ Wide 310x150 Logo<br>✓ Square 310x310 Logo |
| Background color: | transparent |

If you are not familiar with a Windows tile, it is basically a clickable icon from which you can launch the app, and looks something like this

For a guideline on Windows tiles and badges visit the URL below:

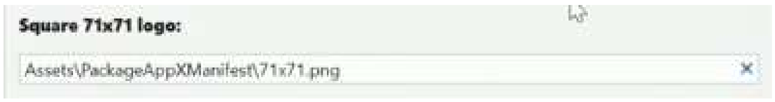
[http://bit.do/tiles-badges](http://bit.do/tiles-badges)

You can set these tiles using the Scaled Assets section, and usually what you can do is set a tile sized at one of the recommended settings and the smaller sizes will be automatically scaled down for you.


Scaled Assets

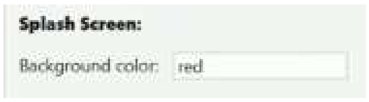
| Scale 400 | Scale 200 | Scale 100 | Scale 150 |
|---|---|---|---|
| 284 x 284 px | 142 x 142 px | 71 x 71 px | 107 x 107 px |
| Recommended | Recommended | Recommended | |

For this app you can set the logos by referring to the Assets\PackageAppManifest\ folder files as follows


**Square 71x71 logo:**

Assets\PackageAppXManifest\71x71.png

And, if you wish you can see the background color for the Splash Screen


**Splash Screen:**

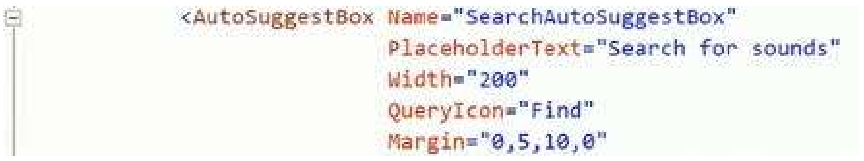
Background color: red

Now, if you right click on Package.appxmanifest

And select “Open With,” choosing the XML (Text) Editor

You will see all of the XML settings that correspond to settings we just made in the visual editor, allowing you to make changes here as well. And, now, when you run the program you should see the Splash Screen briefly run on startup. That cover the basics of getting your app ready for publishing. Note that you will probably want to test your app on multiple devices before submitting it to the Windows Store.

Now, before we move onto the next lesson, let’s make one more minor change to the AutoSuggestBox Margin, to make it a little more visually pleasing

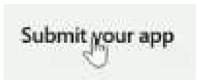


```
<AutoSuggestBox Name="SearchAutoSuggestBox"
                PlaceholderText="Search for sounds"
                Width="200"
                QueryIcon="Find"
                Margin="0,5,10,0"
```

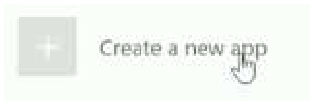
## UWP-056 - UWP SoundBoard - Submitting to the Windows Store

Now we will complete the process of submitting the app to the Windows Store. Begin by navigating to dev.windows.com and it will take you to the correct page relative to the country you are in.

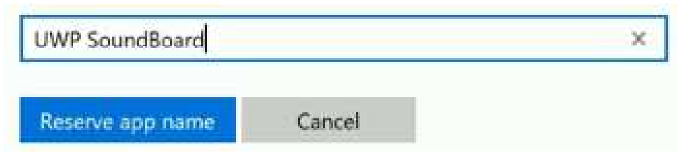
While there, find and click on the link that says "Submit your app."



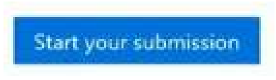
The link, as well as the other steps taken in this lesson, may not look exactly like this because web pages often change. However, you should be able to find it if you look for a link that let's you submit your app. From here, you may have to go through some additional registration steps. After that, select "Create a new app."



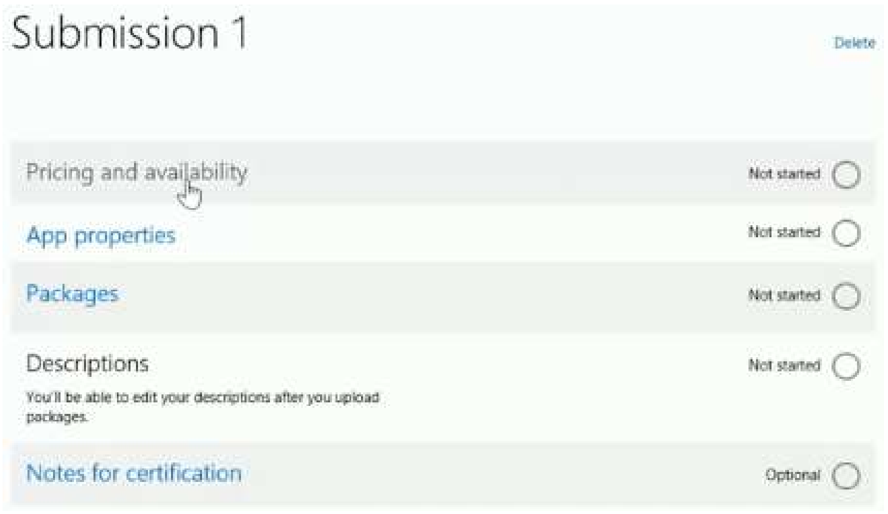
Once you sign into your Microsoft account you can reserve the app name (hoping that nobody else has taken it ahead of you).



Then, select "Start your submission"



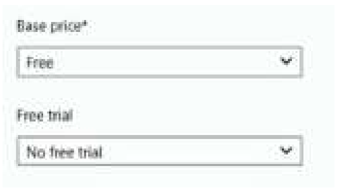
Which should lead you to a series of steps that track your progress


Submission 1

Delete

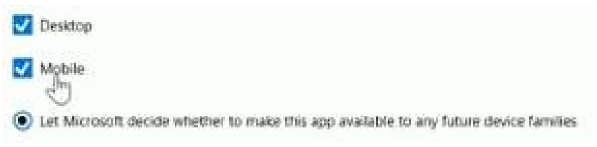
| | |
|---|---|
| Pricing and availability | Not started |
| App properties | Not started |
| Packages | Not started |
| Descriptions<br>You'll be able to edit your descriptions after you upload packages. | Not started |
| Notes for certification | Optional |

Begin by clicking on “Pricing and availability.” Here you can set your apps price, as well as you allow a free trial, as well as more detailed sales options.


Base price*: Free

Free trial: No free trial

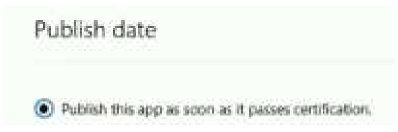
Here you can also change the visibility for the distribution of our app.  This might be useful if you’re only distributing it to a small, known clientel - but not just anybody, only our customers or people that we already know. We could also change the device families on which our app can run. Go ahead and let it run on desktop and mobile.  I'm also going to let Microsoft decide whether to make this app available for future device families.


☑ Desktop

☑ Mobile

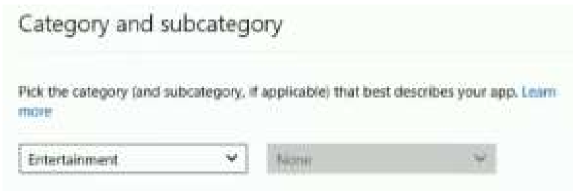
● Let Microsoft decide whether to make this app available to any future device families

You can set Organizational Licensing if you plan to sell to large corporations, etc, so we can safely ignore that right now. Next, set the publication date for as soon as the app passes certification, and click save to complete that part of the process.


Publish date

● Publish this app as soon as it passes certification.

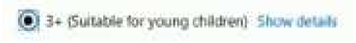
Moving on to the App Properties submission step, select the category (in this case, “Entertainment”).


Category and subcategory

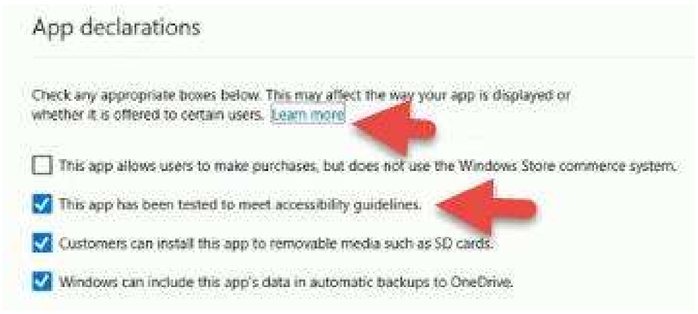
Pick the category (and subcategory, if applicable) that best describes your app. Learn more

Entertainment | None

And set the age rating to 3+


3+ (Suitable for young children) Show details

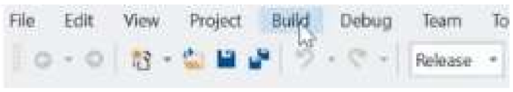
Next, set the App declarations as follows (making sure your app has been fully tested, of course). You can learn more about these options by clicking on “Learn more.” Once these settings are in place, click “save” to move on.

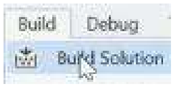

App declarations

Check any appropriate boxes below. This may affect the way your app is displayed or whether it is offered to certain users. Learn more

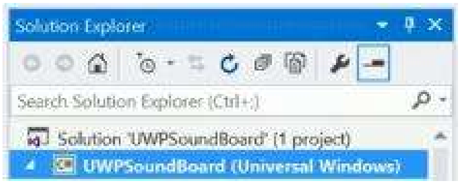
- [ ] This app allows users to make purchases, but does not use the Windows Store commerce system.
- [x] This app has been tested to meet accessibility guidelines.
- [x] Customers can install this app to removable media such as SD cards.
- [x] Windows can include this app's data in automatic backups to OneDrive.

Now, going back to the project in Visual Studio let’s change the version to a Release version, and build that solution.

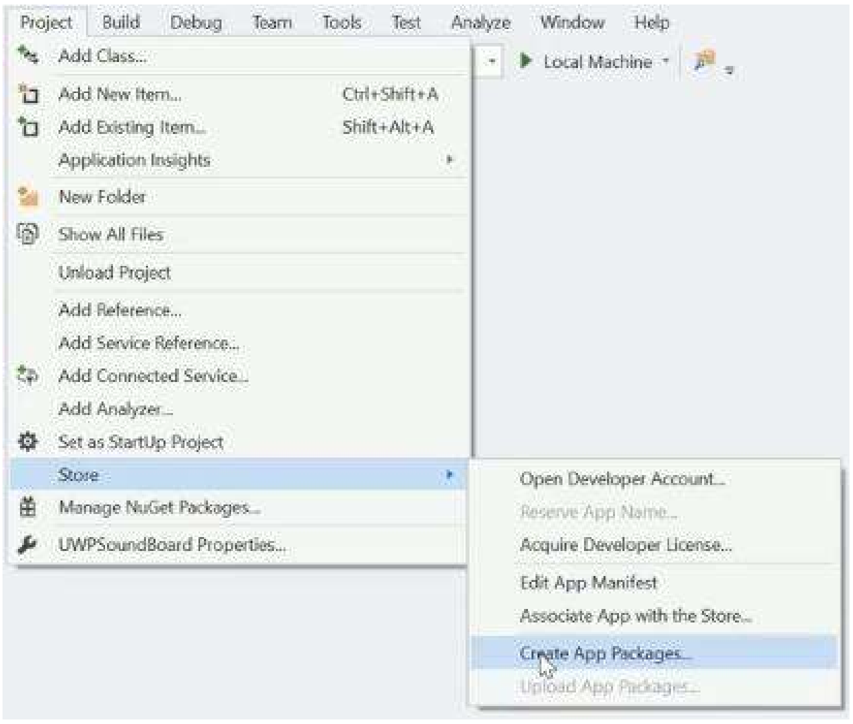




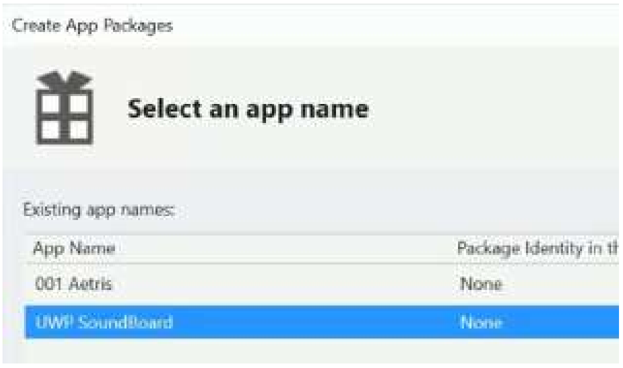
Next, with the project selected in the Solution Explorer


Solution Explorer
Search Solution Explorer (Ctrl+;)
Solution 'UWPSoundBoard' (1 project)
UWPSoundBoard (Universal Windows)

Go to Project > Store > Create App Packages…



Click "Yes" on the next screen to build the app Packages and sign into the Store with your Microsoft account if it asks you to. It will then ask you for an authentication Code which you can have sent to you ask a text message or an email. Once you're signed in select the existing app name you set for your package, and click "Next."


Create App Packages

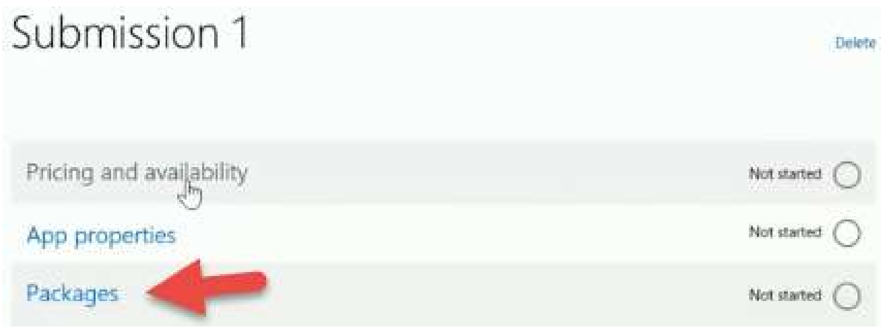
Select an app name

Existing app names:

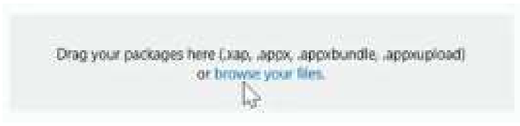
| App Name | Package Identity in th |
|---|---|
| 001 Aetris | None |
| UWP SoundBoard | None |

After that, leave the current settings as they are and click on "Create." Visual Studio will then go through the requisite steps and, if there are no errors, give you a message that the task was completed successfully.

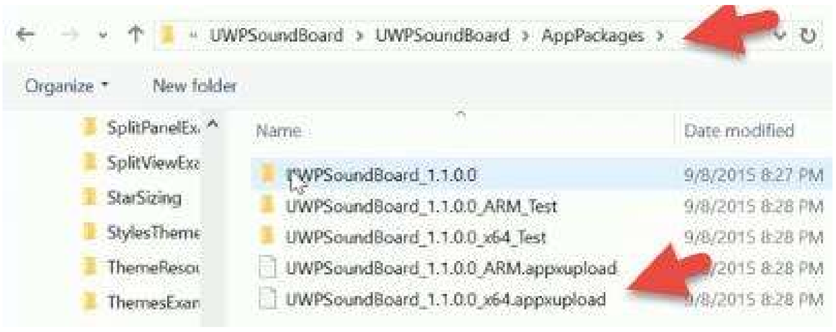
Going back to the browser where you were in the middle of completing your submission, click on "Packages."



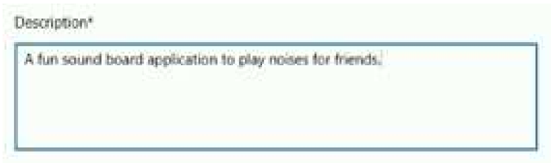
And then click on "browse your files"


Drag your packages here (.xap, .appx, .appxbundle, .appxupload)
or browse your files.

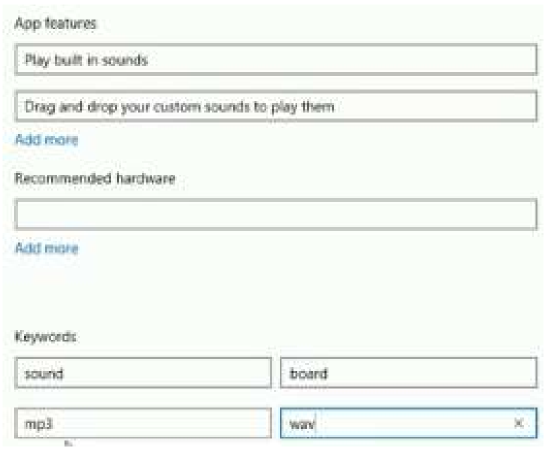
Navigating through your directories to locate your builds, you can upload your app Packages. In this case there are two Packages (ARM, x64) for different devices. Once your Packages are uploaded click on "Save" to complete this step.



Next, create a description for the app and add screenshots if you wish.


Description*

A fun sound board application to play noises for friends.



App features

Play built in sounds

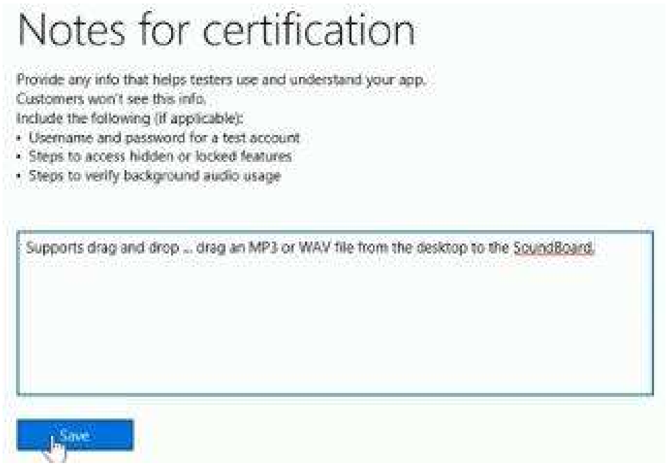
Drag and drop your custom sounds to play them

Recommended hardware

Keywords

| | |
|---|---|
| sound | board |
| mp3 | wav |

Once that is all done, continue on to "Notes for certification" at the Submission page. Include here any non-obvious feature you think would be interesting to note for the certification process.

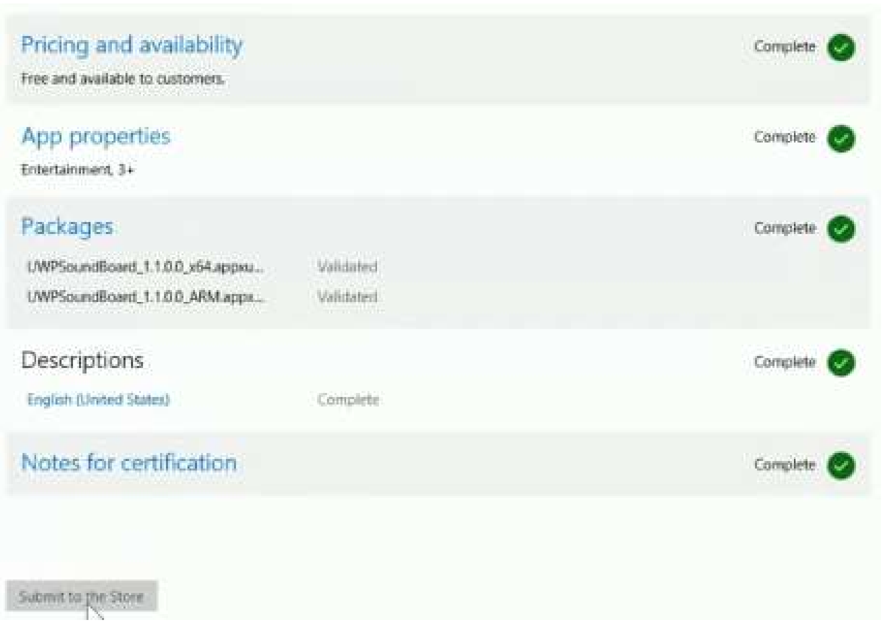


Notes for certification

Provide any info that helps testers use and understand your app.
Customers won't see this info.
Include the following (if applicable):
- Username and password for a test account
- Steps to access hidden or locked features
- Steps to verify background audio usage

Supports drag and drop ... drag an MP3 or WAV file from the desktop to the SoundBoard.

With all of the steps completed, click on "Submit to the store," and all that is left to do is wait and see if your app is accepted!



Pricing and availability
Free and available to customers.
Complete ✓

App properties
Entertainment, 3+
Complete ✓

Packages
Complete ✓

| | |
|---|---|
| UWPSoundBoard_1.1.0.0_x64.appxu... | Validated |
| UWPSoundBoard_1.1.0.0_ARM.appx... | Validated |

Descriptions
Complete ✓

| | |
|---|---|
| English (United States) | Complete |

Notes for certification
Complete ✓