**Hướng dẫn thực hành Lab**

Music Player

Lab version: 1.0.0

Last updated: 3/25/2012

[Overview 3](#_Toc304796045)

[Exercise 4](#_Toc304796046)

[Task 1 – Adding the main UI and functionality 4](#_Toc304796047)

[Task 2 – Adding Button Controls 10](#_Toc304796048)

[Task 3 – Copying music and image files to Isolated Storage 16](#_Toc304796049)

[Task 4 – Adding and Implementing the Audio Playback Agent 17](#_Toc304796050)

[Summary 24](#_Toc304796051)

Tổng quan

* 1. Trong phiên bản gốc đầu tiên của hệ điều hành Windows® Phone, chúng ta có thể tạo một trình chơi nhạc nhưng ứng dụng này sẽ không thể tiếp tục chơi nhạc khi đang chạy nền.
  2. Như chúng ta thấy trong bài lab này, với Windows® Phone Codenamed Mango, một ứng dụng sẽ có thể tiếp tục chơi nhạc ngay cả khi đang chạy nền.. Để làm được điều này, chúng ta sẽ sử dụng lớp **Background Audio Player** mới trong namespace **Microsoft.Phone.BackgroundAudio.**
  3. Bài lab này sẽ hướng dẫn cách sử dụng nhạc nên trong ứng dụng của bạn bằng cách sử dụng Windows® Phone Codenamed Mango API, phát triển trình chơi nhạc có khả năng tiếp tục chơi nhạc ngay cả khi đang chạy nền.

# Mục tiêu

* 1. Bài lab này sẽ giúp bạn :
  + Sử dụng lớp BackgroundAudio để chơi nhạc ngay cả khi ứng dụng của bạn đang chạy nền.
  + Cài đặt một background agent vào trong ứng dụng của bạn. ( Giúp cho ứng dụng của bạn có thể chạy ngầm dưới hệ thống và thông báo với người dùng khi hoàn thành một nhiệm vụ hay mỗi khi xảy ra một sự kiện nào đó)

# Yêu cầu

* + Microsoft Visual Studio 2010 hoặc Microsoft Visual C# Express 2010, và Windows® Phone Developer Tools download tại <http://go.microsoft.com/?linkid=9772716>
  + Có kiến thức về phát triển ứng dụng trên Windows® Phone (Nếu đây là lần tiên bạn phát triển ứng dụng trên Windows® Phone 7, hãy tham khảo tại <http://msdn.microsoft.com/en-us/gg266499>. Bài lab này sẽ xem như là bạn đã hiểu cơ bản và tập trung vào trình bày phần âm thanh nền.)

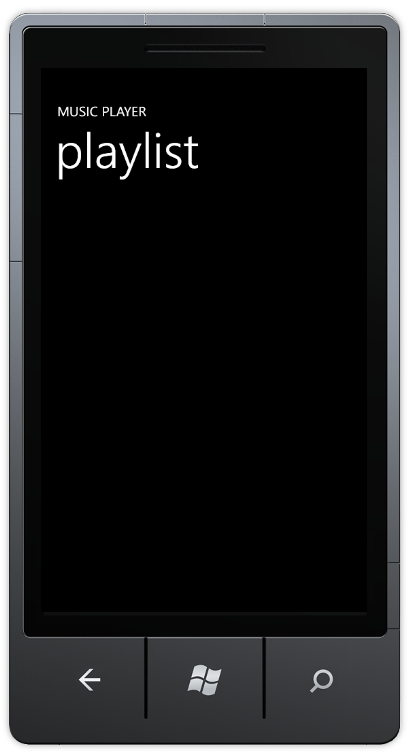
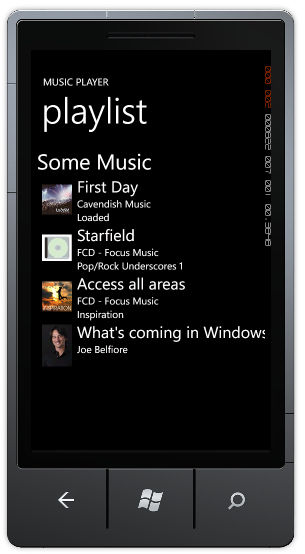
# Ước lượng thời gian hoàn thành

* 1. Bạn sẽ mất khoảng 20-40 phút để hoàn thành bài lab này.

Thực hành

* 1. Bắt đầu bài lab với solution đã được khởi tạo sẵn trong thư mục **Source\Begin.**
  2. Solution này chứa một project với tất cả giao diện người dùng (user interface viết tắt là UI) mà chúng ta sẽ sử dụng trong bài lab này và 2 model (**Playlist** và **Track** ) mà chúng ta sẽ làm việc với chúng.

Task 1 – Thêm giao diện người dùng chính và chức năng

1. Trong task này, chúng ta sẽ làm việc trên phần chính của ứng dụng, cài đặt các xử lý sự kiện cho **Microsoft.Phone.BackgroundAudio.BackgroundAudioPlayer** để có thể tiếp tục chơi âm thanh ngay cả khi ứng dụng đang chạy nền.
   1. Open the solution under the **Source\Begin** folder and examine the project.
   2. Open the **MainPage.xaml** file.
   3. Set the **DataContext** property of the page itself by adding the following attribute to the **phone:PhoneApplicationPage** element:
      1. XAML
      2. DataContext="{Binding RelativeSource={RelativeSource Self}}"
   4. Locate the grid named **ContentPanel** and add the **HorizontalAlignment** attribute,setting it to **Stretch**.
      1. XAML
      2. <Grid x:Name="ContentPanel" Grid.Row="1" Margin="12,0,12,0" HorizontalAlignment="Stretch">
   5. Now add the following **RowDefinitions** and a new **TextBlock**. You can see that we use the new textblock to display the name of the active playlist. We will add a property to hold the active playlist later in the lab:
      1. XAML
      2. <Grid x:Name="ContentPanel" Grid.Row="1" Margin="12,0,12,0" HorizontalAlignment="Stretch">
      3. <Grid.RowDefinitions>
      4. <RowDefinition Height="Auto"/>
      5. <RowDefinition Height="\*"/>
      6. <RowDefinition Height="Auto"/>
      7. </Grid.RowDefinitions>
      8. <TextBlock Text="{Binding ActivePlaylist.Name}" FontSize="{StaticResource PhoneFontSizeExtraLarge}"/>
      9. </Grid>
   6. Add a new **ListBox** as seen in the code below. This listbox will display the list of available music tracks.
      1. XAML
      2. <Grid x:Name="ContentPanel" Grid.Row="1" Margin="12,0,12,0" HorizontalAlignment="Stretch">
      3. <Grid.RowDefinitions>
      4. <RowDefinition Height="Auto"/>
      5. <RowDefinition Height="\*"/>
      6. <RowDefinition Height="Auto"/>
      7. </Grid.RowDefinitions>
      9. <TextBlock Text="{Binding ActivePlaylist.Name}" FontSize="{StaticResource PhoneFontSizeExtraLarge}"/>
      10. <ListBox ItemsSource="{Binding ActivePlaylist.Tracks}" Grid.Row="1" x:Name="lstTracks" HorizontalContentAlignment="Stretch" Loaded="lstTracks\_Loaded">
      11. <ListBox.ItemTemplate>
      12. <DataTemplate>
      13. <Grid HorizontalAlignment="Stretch">
      14. <Grid.RowDefinitions>
      15. <RowDefinition/>
      16. <RowDefinition/>
      17. <RowDefinition/>
      18. </Grid.RowDefinitions>
      19. <Grid.ColumnDefinitions>
      20. <ColumnDefinition Width="Auto"/>
      21. <ColumnDefinition Width="\*"/>
      22. </Grid.ColumnDefinitions>
      24. <Image Margin="10,0" Source="{Binding Tile, TargetNullValue={StaticResource NoArt}}" Grid.RowSpan="3" Width="60" Stretch="Uniform" VerticalAlignment="Center"/>
      25. <TextBlock Text="{Binding Title}" Grid.Column="1" FontSize="{StaticResource PhoneFontSizeLarge}"/>
      26. <TextBlock Text="{Binding Artist}" Grid.Row="1" Grid.Column="1"/>
      27. <TextBlock Text="{Binding Album}" Grid.Row="2" Grid.Column="1"/>
      28. </Grid>
      29. </DataTemplate>
      30. </ListBox.ItemTemplate>
      31. </ListBox>
      32. </Grid>
   7. Open the file named **MainPage.xaml.cs**. This is the code behind file for the main page.
   8. Add an empty method called **lstTracks\_Loaded** as seen below:
      1. C#
      2. public partial class MainPage : PhoneApplicationPage
      3. {
      4. ...
      5. private void lstTracks\_Loaded(object sender, RoutedEventArgs e)
      6. {
      7. }
      8. ...
      9. }
   9. Compile and run the application on the emulator. At this stage, you should see the following screen.
      * 1. 
        2. Figure 1
        3. The application without an actual playlist.
   10. Let us add the active playlist property we used for binding in the previous steps. Add a **Playlist** dependency property as seen below and call it **ActivePlaylist**:
       1. C#
       2. public partial class MainPage : PhoneApplicationPage
       3. {
       4. public Playlist ActivePlaylist
       5. {
       6. get { return (Playlist)GetValue(ActivePlaylistProperty); }
       7. set { SetValue(ActivePlaylistProperty, value); }
       8. }
       9. public static readonly DependencyProperty ActivePlaylistProperty =
       10. DependencyProperty.Register("ActivePlaylist", typeof(Playlist), typeof(MainPage), new PropertyMetadata(null));
       11. ...
       12. }
   11. Override the **OnNavigatedTo** method on the page in order to deploy the playlist supplied as one of the application’s resources to isolated storage.
       1. C#
       2. public partial class MainPage : PhoneApplicationPage
       3. {
       4. ...
       5. protected override void OnNavigatedTo(System.Windows.Navigation.NavigationEventArgs e)
       6. {
       7. Stream playlistStream = Application.GetResourceStream(new Uri("Misc/Playlist.xml", UriKind.Relative)).Stream;
       9. System.Xml.Serialization.XmlSerializer serializer = new System.Xml.Serialization.XmlSerializer(typeof(Playlist));
       10. ActivePlaylist = (Playlist)serializer.Deserialize(playlistStream);
       12. using ( IsolatedStorageFile isoStorage = IsolatedStorageFile.GetUserStoreForApplication() )
       13. {
       14. using ( IsolatedStorageFileStream file = isoStorage.OpenFile("playlist.xml", FileMode.OpenOrCreate) )
       15. {
       16. var writer = new StreamWriter(file);
       18. serializer.Serialize(writer, ActivePlaylist);
       19. }
       20. }
       22. base.OnNavigatedTo(e);
       23. }
       24. ...
       25. }
   12. In the page’s constructor, add the following highlighted code snippet to subscribe to the **PlayStateChanged** event:
       1. C#
       2. public partial class MainPage : PhoneApplicationPage
       3. {
       4. ...
       5. public MainPage()
       6. {
       7. InitializeComponent();
       9. BackgroundAudioPlayer.Instance.PlayStateChanged += new EventHandler(Instance\_PlayStateChanged);
       10. }
       11. ...
       12. }
   13. Add the event handler function specified in the previous step along with some additional methods. This will update the track number of the currently playing track whenever the background audio player, which we use to play back tracks, changes its playback state:
       1. C#
       2. public partial class MainPage : PhoneApplicationPage
       3. {
       4. ...
       5. private void Instance\_PlayStateChanged(object sender, EventArgs e)
       6. {
       7. UpdateSelection();
       8. }
       9. private void UpdateSelection()
       10. {
       11. int activeTrackNumber = GetActiveTrackIndex();
       13. if ( activeTrackNumber != -1 )
       14. {
       15. lstTracks.SelectedIndex = activeTrackNumber;
       16. }
       17. }
       19. private int GetActiveTrackIndex()
       20. {
       21. int track = -1;
       22. if ( null != BackgroundAudioPlayer.Instance.Track )
       23. {
       24. track = int.Parse(BackgroundAudioPlayer.Instance.Track.Tag);
       25. }
       27. return track;
       28. }
       29. ...
       30. }
   14. The background audio player will allow the user to change the actively playing track even while the application is not active. To handle this, update the **lstTracks\_Loaded** method inserting the following code into that method:
       1. C#
       2. public partial class MainPage : PhoneApplicationPage
       3. {
       4. ...
       5. private void lstTracks\_Loaded(object sender, RoutedEventArgs e)
       6. {
       7. UpdateSelection();
       8. }
       9. ...
       10. }
   15. Compile and run the application. At this stage the main page should look like the following:
       * 1. 
         2. Figure 2
         3. Playlist
   16. Stop the debugging and return to the code.
   17. This step concludes the current task.

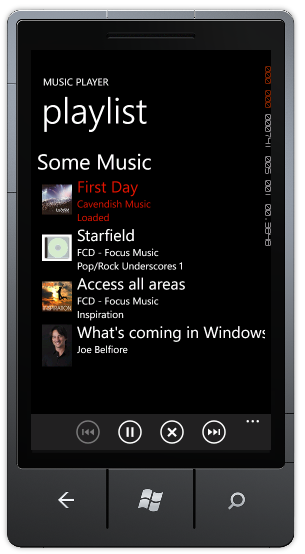
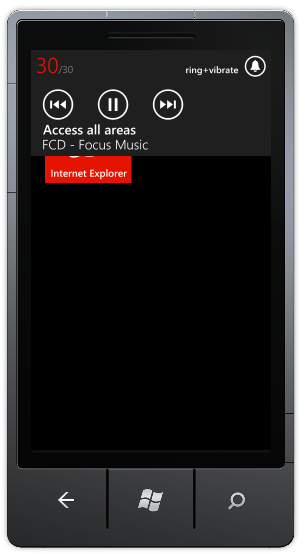
Task 2 – Adding Button Controls

* 1. This step will explain how to add **ApplicationBar** buttons, which we will use to control the player. We will add four buttons: play, stop, previous and next track.
  2. These buttons' functions will use services exposed by the **BackgroundAudioPlayer** class.
  3. Open the **MainPage.xaml** file, if it is not already opened.
  4. After the **LayoutRoot** grid ends, instead of the commented use of **PhoneApplicationPage.ApplicationBar**, add the following code:
     1. XAML
     2. <phone:PhoneApplicationPage.ApplicationBar>
     3. <shell:ApplicationBar IsVisible="True" IsMenuEnabled="False">
     4. <shell:ApplicationBarIconButton IconUri="/Images/prev.png" Text="prev" Click="appbar\_prev"/>
     5. <shell:ApplicationBarIconButton IconUri="/Images/play.png" Text="play" Click="appbar\_playpause"/>
     6. <shell:ApplicationBarIconButton IconUri="/Images/stop.png" Text="stop" Click="appbar\_stop"/>
     7. <shell:ApplicationBarIconButton IconUri="/Images/next.png" Text="next" Click="appbar\_next"/>
     8. </shell:ApplicationBar>
     9. </phone:PhoneApplicationPage.ApplicationBar>
  5. Open the file **MainPage.xaml.cs** andadd the handlers for the newly inserted buttons.
     1. C#
     2. public partial class MainPage : PhoneApplicationPage
     3. {
     4. ...
     5. #region ApplicationBar Buttons Events
     6. private void appbar\_prev(object sender, EventArgs e)
     7. {
     8. BackgroundAudioPlayer.Instance.SkipPrevious();
     9. }
     11. private void appbar\_playpause(object sender, EventArgs e)
     12. {
     13. if ( BackgroundAudioPlayer.Instance.PlayerState == PlayState.Playing )
     14. BackgroundAudioPlayer.Instance.Pause();
     15. else
     16. BackgroundAudioPlayer.Instance.Play();
     17. }
     19. private void appbar\_stop(object sender, EventArgs e)
     20. {
     21. BackgroundAudioPlayer.Instance.Stop();
     22. }
     24. private void appbar\_next(object sender, EventArgs e)
     25. {
     26. BackgroundAudioPlayer.Instance.SkipNext();
     27. }
     28. #endregion
     29. ...
     30. }
  6. Just as we updated the selected track in the previous task, we need to update the appbar buttons according to the player’s state. To do that, locate the **Instance\_PlayStateChanged** method and add a call to **UpdateAppBarStatus** method.
     1. C#
     2. public partial class MainPage : PhoneApplicationPage
     3. {
     4. ...
     5. private void Instance\_PlayStateChanged(object sender, EventArgs e)
     6. {
     7. UpdateAppBarStatus();
     8. UpdateSelection();
     9. }
     10. ...
     11. }
  7. Now add the **UpdateAppBarStatus** method itself:
     1. C#
     2. public partial class MainPage : PhoneApplicationPage
     3. {
     4. ...
     5. private void UpdateAppBarStatus()
     6. {
     7. switch ( BackgroundAudioPlayer.Instance.PlayerState )
     8. {
     9. case PlayState.Playing:
     10. //Prev Button
     11. if ( GetActiveTrackIndex() > 0 )
     12. ( ApplicationBar.Buttons[0] as ApplicationBarIconButton ).IsEnabled = true;
     13. else
     14. ( ApplicationBar.Buttons[0] as ApplicationBarIconButton ).IsEnabled = false;
     16. //Play/Pause Button
     17. ( ApplicationBar.Buttons[1] as ApplicationBarIconButton ).IsEnabled = true;
     18. ( ApplicationBar.Buttons[1] as ApplicationBarIconButton ).Text = "pause";
     19. ( ApplicationBar.Buttons[1] as ApplicationBarIconButton ).IconUri = new Uri("/Images/pause.png", UriKind.Relative);
     21. //Stop Button
     22. ( ApplicationBar.Buttons[2] as ApplicationBarIconButton ).IsEnabled = true;
     24. //Next button
     25. if ( GetActiveTrackIndex() < ActivePlaylist.Tracks.Count - 1 )
     26. ( ApplicationBar.Buttons[3] as ApplicationBarIconButton ).IsEnabled = true;
     27. else
     28. ( ApplicationBar.Buttons[3] as ApplicationBarIconButton ).IsEnabled = false;
     29. break;
     31. case PlayState.Paused:
     32. //Prev Button
     33. if ( GetActiveTrackIndex() > 0 )
     34. ( ApplicationBar.Buttons[0] as ApplicationBarIconButton ).IsEnabled = true;
     35. else
     36. ( ApplicationBar.Buttons[0] as ApplicationBarIconButton ).IsEnabled = false;
     38. //Play/Pause Button
     39. ( ApplicationBar.Buttons[1] as ApplicationBarIconButton ).IsEnabled = true;
     40. ( ApplicationBar.Buttons[1] as ApplicationBarIconButton ).Text = "play";
     41. ( ApplicationBar.Buttons[1] as ApplicationBarIconButton ).IconUri = new Uri("/Images/play.png", UriKind.Relative);
     43. //Stop Button
     44. ( ApplicationBar.Buttons[2] as ApplicationBarIconButton ).IsEnabled = true;
     46. //Next button
     47. if ( GetActiveTrackIndex() < ActivePlaylist.Tracks.Count - 1 )
     48. ( ApplicationBar.Buttons[3] as ApplicationBarIconButton ).IsEnabled = true;
     49. else
     50. ( ApplicationBar.Buttons[3] as ApplicationBarIconButton ).IsEnabled = false;
     51. break;
     53. case PlayState.Stopped:
     54. //Prev Button
     55. if ( GetActiveTrackIndex() > 0 )
     56. ( ApplicationBar.Buttons[0] as ApplicationBarIconButton ).IsEnabled = true;
     57. else
     58. ( ApplicationBar.Buttons[0] as ApplicationBarIconButton ).IsEnabled = false;
     60. //Play/Pause Button
     61. ( ApplicationBar.Buttons[1] as ApplicationBarIconButton ).IsEnabled = true;
     62. ( ApplicationBar.Buttons[1] as ApplicationBarIconButton ).Text = "play";
     63. ( ApplicationBar.Buttons[1] as ApplicationBarIconButton ).IconUri = new Uri("/Images/play.png", UriKind.Relative);
     65. //Stop Button
     66. ( ApplicationBar.Buttons[2] as ApplicationBarIconButton ).IsEnabled = false;
     68. //Next button
     69. if ( GetActiveTrackIndex() < ActivePlaylist.Tracks.Count - 1 )
     70. ( ApplicationBar.Buttons[3] as ApplicationBarIconButton ).IsEnabled = true;
     71. else
     72. ( ApplicationBar.Buttons[3] as ApplicationBarIconButton ).IsEnabled = false;
     73. break;
     75. case PlayState.Unknown:
     76. //Prev Button
     77. ( ApplicationBar.Buttons[0] as ApplicationBarIconButton ).IsEnabled = false;
     79. //Play/Pause Button
     80. ( ApplicationBar.Buttons[1] as ApplicationBarIconButton ).IsEnabled = true;
     81. ( ApplicationBar.Buttons[1] as ApplicationBarIconButton ).Text = "play";
     82. ( ApplicationBar.Buttons[1] as ApplicationBarIconButton ).IconUri = new Uri("/Images/play.png", UriKind.Relative);
     84. //Stop Button
     85. ( ApplicationBar.Buttons[2] as ApplicationBarIconButton ).IsEnabled = false;
     87. //Next button
     88. if ( GetActiveTrackIndex() < ActivePlaylist.Tracks.Count - 1 )
     89. ( ApplicationBar.Buttons[3] as ApplicationBarIconButton ).IsEnabled = true;
     90. else
     91. ( ApplicationBar.Buttons[3] as ApplicationBarIconButton ).IsEnabled = false;
     92. break;
     94. default:
     95. break;
     96. }
     97. }
     98. ...
     99. }
  8. Compile and run the application. At this stage the main page should look like the following:
     + 1. 
       2. Figure 3
       3. Playlist with playback controls
  9. Stop the debugging and return to the code.

Task 3 – Copying music and image files to Isolated Storage

* 1. In order to play the tracks we have in the solution, we need to copy these files to **Isolated Storage.**
  2. Open the file named **App.xaml.cs** and add the following using directives:
     1. C#
     2. using System.IO.IsolatedStorage;
     3. using System.Windows.Resources;
  3. At the constructor for the **App** class, add the following highlighted code:
     1. C#
     2. public App()
     3. {
     4. ...
     5. CopyToIsolatedStorage();
     6. }
  4. Now use the following code to add the **CopyToIsolatedStorage** method itself:
     1. C#
     2. private void CopyToIsolatedStorage()
     3. {
     4. using ( IsolatedStorageFile storage = IsolatedStorageFile.GetUserStoreForApplication() )
     5. {
     6. string[] files = new string[] { "file1.mp3", "file2.mp3", "file3.mp3" };
     8. foreach ( var \_fileName in files )
     9. {
     10. if ( !storage.FileExists(\_fileName) )
     11. {
     12. string \_filePath = "Music/" + \_fileName;
     13. StreamResourceInfo resource = Application.GetResourceStream(new Uri(\_filePath, UriKind.Relative));
     15. using ( IsolatedStorageFileStream file = storage.CreateFile(\_fileName) )
     16. {
     17. int chunkSize = 4096;
     18. byte[] bytes = new byte[chunkSize];
     19. int byteCount;
     21. while ( ( byteCount = resource.Stream.Read(bytes, 0, chunkSize) ) > 0 )
     22. {
     23. file.Write(bytes, 0, byteCount);
     24. }
     25. }
     26. }
     27. }
     29. files = new string[] { "Image1.jpg", "Image3.jpg", "no-art.jpg" };
     31. foreach ( var \_fileName in files )
     32. {
     33. string \_destFilePath = "Shared/Media/" + \_fileName;
     34. ;
     35. if ( !storage.FileExists(\_destFilePath) )
     36. {
     37. string \_filePath = "Images/" + \_fileName;
     38. StreamResourceInfo resource = Application.GetResourceStream(new Uri(\_filePath, UriKind.Relative));
     40. using ( IsolatedStorageFileStream file = storage.CreateFile(\_destFilePath) )
     41. {
     42. int chunkSize = 4096;
     43. byte[] bytes = new byte[chunkSize];
     44. int byteCount;
     46. while ( ( byteCount = resource.Stream.Read(bytes, 0, chunkSize) ) > 0 )
     47. {
     48. file.Write(bytes, 0, byteCount);
     49. }
     50. }
     51. }
     52. }
     53. }
     54. }
  5. This step concludes the current task.

Task 4 – Adding and Implementing the Audio Playback Agent

* 1. The last thing we need to do is to implement an “Audio Playback Agent”. This is instantiated by the operating system to handle actions requested by the user. **AudioPlayerAgent** runs in the background and calls into an instance of the **BackgroundAudioPlayer**, which then calls into the Zune Media Queue to play the audio.
  2. add a new project to the solution using the **Windows Phone Audio Playback Agent** template located under the category and name it **AudioPlaybackAgent**.
  3. In this project, we will need to use the model defined in the **Models** project as well as XML serialization. Add references to both **Models** and **System.Xml.Serialization.**
  4. Open the **AudioPlayer.cs** file created as part of the project and delete the entire contents of the **AudioPlayer** class.
  5. Add the following using statements to the file:
     1. C#
     2. using System.IO.IsolatedStorage;
     3. using System.Xml.Serialization;
     4. using System.IO;
     5. using Models;
  6. Add the following fields to the **AudioPlayer** class. These will be used to store the current playlist and keep track of the currently playing track:
     1. C#
     2. static int currentTrackNumber = 0;
     3. static Playlist playlist;
  7. Add the following constructor, which will access the playlist XML file we took care to place in the application’s isolated storage:
     1. C#
     2. public AudioPlayer()
     3. : base()
     4. {
     5. //Load from IsoStore & deserialize
     6. using ( IsolatedStorageFile isoStorage = IsolatedStorageFile.GetUserStoreForApplication() )
     7. {
     8. using ( IsolatedStorageFileStream file = isoStorage.OpenFile("playlist.xml", FileMode.Open) )
     9. {
     10. XmlSerializer serializer = new XmlSerializer(typeof(Playlist));
     11. var reader = new StreamReader(file);
     12. playlist = (Playlist)serializer.Deserialize(reader);
     13. }
     14. }
     15. }
  8. Add the following **OnPlayStateChanged** override, which handles changes in the agent’s playback status:
     1. C#
     2. protected override void OnPlayStateChanged(BackgroundAudioPlayer player, AudioTrack track, PlayState playState)
     3. {
     4. switch ( playState )
     5. {
     6. case PlayState.TrackEnded:
     7. PlayNext(player);
     8. break;
     9. case PlayState.TrackReady:
     10. player.Play();
     11. break;
     12. default:
     13. break;
     14. }
     15. NotifyComplete();
     16. }
     17. You can see that once a track ends we simply move on to the next track (using a method we will soon introduce) and that once a track is ready we simply use the background audio player to play it. The call to **NotifyComplete** is required to let the agent know that the state change was handled.
  9. Add the following helper methods, which initiate playback of the next/previous track:
     1. C#
     2. private void PlayNext(BackgroundAudioPlayer player)
     3. {
     4. var songsCount = playlist.Tracks.Count;
     5. if ( ++currentTrackNumber >= songsCount )
     6. {
     7. currentTrackNumber = 0;
     8. }
     9. Play(player);
     10. }
     11. private void PlayPrev(BackgroundAudioPlayer player)
     12. {
     13. var songsCount = playlist.Tracks.Count;
     14. if ( --currentTrackNumber < 0 )
     15. {
     16. currentTrackNumber = songsCount - 1;
     17. }
     18. Play(player);
     19. }
  10. Add the following override for **OnUserAction**, to handle the various actions a user may perform when interacting with the audio playback agent:
      1. C#
      2. protected override void OnUserAction(BackgroundAudioPlayer player, AudioTrack track, UserAction action, object param)
      3. {
      4. switch ( action )
      5. {
      6. case UserAction.FastForward:
      7. player.FastForward();
      8. break;
      9. case UserAction.Pause:
      10. player.Pause();
      11. break;
      12. case UserAction.Play:
      13. if ( player.PlayerState == PlayState.Paused )
      14. {
      15. player.Play();
      16. }
      17. else
      18. {
      19. Play(player);
      20. }
      21. break;
      22. case UserAction.Rewind:
      23. player.Rewind();
      24. break;
      25. case UserAction.Seek:
      26. player.Position = (TimeSpan)param;
      27. break;
      28. case UserAction.SkipNext:
      29. PlayNext(player);
      30. break;
      31. case UserAction.SkipPrevious:
      32. PlayPrev(player);
      33. break;
      34. case UserAction.Stop:
      35. player.Stop();
      36. break;
      37. default:
      38. break;
      39. }
      40. NotifyComplete();
      41. }
  11. Add an additional helper method for initiating playback of the current track, as dictated by the **currentTrackNumber** field we added previously:
      1. C#
      2. private void Play(BackgroundAudioPlayer player)
      3. {
      4. var currentTrack = playlist.Tracks[currentTrackNumber];
      5. Uri tileUri = ( currentTrack.Tile == null ? new Uri("Shared/Media/no-art.jpg", UriKind.Relative) :
      6. ( currentTrack.Tile.IsAbsoluteUri ? new Uri("Shared/Media/no-art.jpg", UriKind.Relative) :
      7. new Uri(currentTrack.TileString.Replace("/Images", "Shared/Media"), UriKind.Relative) ) );
      8. var audioTrack = new AudioTrack(currentTrack.Source,
      9. currentTrack.Title,
      10. currentTrack.Artist,
      11. currentTrack.Album,
      12. tileUri,
      13. currentTrackNumber.ToString(),
      14. EnabledPlayerControls.All);
      15. player.Track = audioTrack;
      16. }
  12. Add a pair of additional overrides for handling errors or cancellations, though we do not have any specific logic for such cases in this lab:
      1. C#
      2. protected override void OnCancel()
      3. {
      4. NotifyComplete();
      5. }
      6. protected override void OnError(BackgroundAudioPlayer player, AudioTrack track, Exception error, bool isFatal)
      7. {
      8. NotifyComplete();
      9. }
  13. The last step is to add a reference to this new project from the **MusicPlayer** project.
  14. Compile the application and make sure you do not get any errors.
  15. Run the application. At this stage, the main page should be fully functional.
  16. Now it is time to play some music. Click on the play () button and be sure to turn the volume up (press F9 to increase the emulator’s own volume).
      + 1. 
        2. Figure 4
        3. The fully functional application
  17. Now you can navigate away from the application by pressing the Start Key () and see that the music continues to play.
  18. Pressing F9 to change the volume will reveal the current playback state, and you can control the current playing track just as you would through the application itself.
      1. 
      2. Figure 5
      3. Controlling playback while the application is inactive
  19. Change the currently track and launch the application again from the list of installed applications. The application should display the currently playing track correctly.
  20. Stop the debugger.
  21. This step concludes the current task and the entire lab.

Summary

* 1. This lab has shown you how to use the new Windows® Phone Mango APIs to play music. You have also seen how to use this new API to develop an application that can play audio even while not in the foreground.