Streaming MP3 player in WP7

This article explains how to create a streaming MP3 music player in Windows Phone 7.

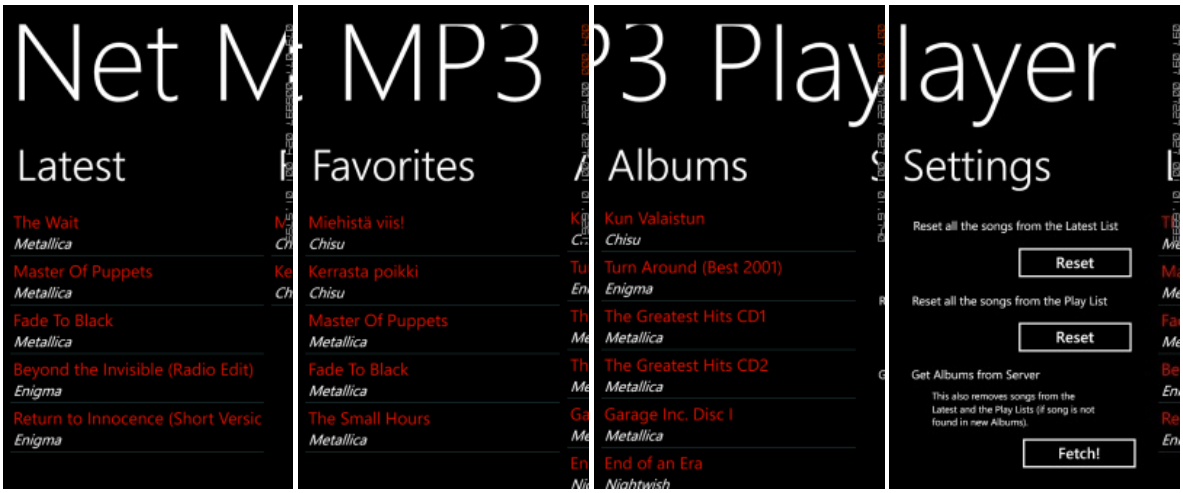
20 May 2012

Introduction

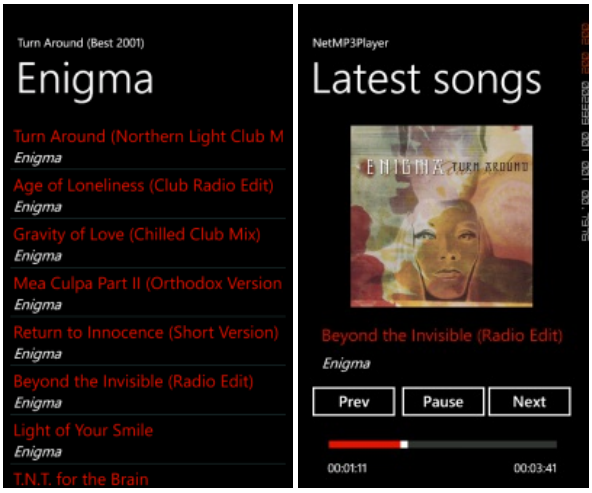
This article show how to create a streaming MP3 music player in Windows Phone 7. All the music files are stored in the server and are streamed to Windows Phone. Application first loads all the albums data from server with JSON data. In server side there are PHP script which loads ID3 tags from all the MP3 files and generates JSON string, which is returned to Windows Phone. After that application deserializes JSON string to dynamic objects and the albums data is displayed in the Panorama page (image 3). Selected album songs will be displayed in the new Songs Page (image 5).

User can add favorite songs to Favorite list (image 2) from Latest or Album songs. Selected song is played in Player Page (image 6) when user selects song from song lists. All the selected songs will be added to Latest played songs (image 1). In Player page user can see album cover art, title of the song, album name and playing time of the song. User can skip to previous or next song in this selected list which music is now playing and seek the position of the song. All the list can be reset from the Setting section (image 4) of the Panorama page.

Panorama Page:



Individual Pages:



All the album data is saved to Isolated Storage and loaded from there when the application is launched again. Of course, a new album data can be loaded from the server side when the new albums are added or removed in server side. When a new album data is fetched from the server again - Latest and Favorite song list are checked and songs that are not available anymore will be removed from the lists. This application uses Windows Phone Background Audio Agent to play MP3 music, so the music can be played in background like the original music player does in Windows Phone.

NOTE: This is only a **code example** and it **doesn't include any MP3 or album cover art files**, so you have to use your own files and PHP server.

Here is a small demo video, which shows how the application works (sorry about my assistant shaking hands with capturing this): The media player is loading...

Back end of this application

(C) Copyright Nokia 2013. All rights reserved.

Folder structure

This example uses PHP server as a back end of this application. Create a directory StreamingDemo to your server and create there folder named Music and upload your own MP3 albums with MP3 files and JPG cover art files. Store album cover art in to your album folder. Cover album file name should be cover.jpg, all the other file names will be ignored. This code example reads ID3 tags from the MP3 files with getID3(). Download latest version of the getID3() here and copy getid3 folder to your StreamingDemo folder. Your folder structure should look like this:

```
StreamingDemo/getid3
StreamingDemo/Music/Album1
StreamingDemo/Music/Album1/Some nice music.mp3
StreamingDemo/Music/Album1/Some other nice music.mp3
StreamingDemo/Music/Album1/cover.jpg
StreamingDemo/Music/Album2/
StreamingDemo/Music/Album2/Another music file here too.mp3
...
```

Get albums data from the server

PHP script (name it to getAlbums.php) will be called from the Windows Phone application. First application check if there are a "secret code" passed from the application, if not PHP script will return empty JSON string. (Note: you can change this secret code to any code you want in the Windows Phone application and here in PHP script.) After that all the Music folder album directories are fetched. Music files are scanned in alphabetical order and needed information are parsed to JSON string: album name and directory, mp3 file name, artist, title, album name, url to album art and playing time of the song.

getAlbums.php file in server:

```
<?php
// check request info
if ($_POST['request'] != "secretRequest") {
  echo "{}";
  return;
}

// include getID3() library
require_once('getid3/getid3.php');

// initialize getID3 engine
$getID3 = new getID3;

// get all album directories
$albumDirectories = array_filter(glob("Music/*"), "is_dir");

// start JSON string
echo '{"Albums":[';
$albums = 0;

// go through all the albums
foreach ($albumDirectories as &$DirectoryToScan) {
  // scan directory files to get music in alphabetical order
  $dir = scandir($DirectoryToScan);
  // no songs yet, used to generate JSON data
  $songs = 0;
  // is there cover.jpg in this album directory
  $filename = $DirectoryToScan."/cover.jpg";
  $cover = "";
  if (is_file($filename)) $cover = $filename;
  // go through all files in directory (in album)
  foreach ($dir as $file) {
    // get full path to music file
    $FullFileName = realpath($DirectoryToScan.'/'.$file);
    // get file extension
    $ext = substr($file, strrpos($file, '.') + 1);
    if ($ext == "jpg") continue;
    // if file is music file
    if ((substr($FullFileName, 0, 1) != '.') && is_file($FullFileName) && ($ext == "mp3" || $ext == "MP3")) {
      // if there are albums all ready listed to JSON, add comma
      if ($albums > 0 && $songs == 0) echo ',';
      // time limit to get id3 tags
      set_time_limit(30);
      // analyze file
      $ThisFileInfo = $getID3->analyze($FullFileName);
      getid3_lib::CopyTagsToComments($ThisFileInfo);
      // set JSON data
      if ($songs == 0) {
        echo '{"albumname":"'.implode('',$ThisFileInfo['comments_html']['album']).'",';
        echo '"albumdirectory":"'.$DirectoryToScan.'/",';
        echo '"songs":[';
      } else {
        echo ",";
      }
      echo '{';
      echo '"albumdirectory":"'.$DirectoryToScan.'/",';
      echo '"filename":"'.utf8_encode(html_entity_decode($ThisFileInfo['filename'])).'",';
      echo '"artist":"'.utf8_encode(html_entity_decode(implode('', $ThisFileInfo['comments_html']['artist']))).'",';
      echo '"title":"'.utf8_encode(html_entity_decode(implode('',$ThisFileInfo['comments_html']['title']))).'",';
      echo '"album":"'.utf8_encode(html_entity_decode(implode('',$ThisFileInfo['comments_html']['album']))).'",';
      echo '"albumart":"'.utf8_encode(html_entity_decode($cover)).'",';
      echo '"playtime":"'.$ThisFileInfo['playtime_string'].'"';
      echo '}';
      // one song added
      $songs++;
    }
  }
  // add JSON data, one album added
  echo ']}';
  $albums++;
}
// end JSON string (albums)
echo ']}';
?>
```

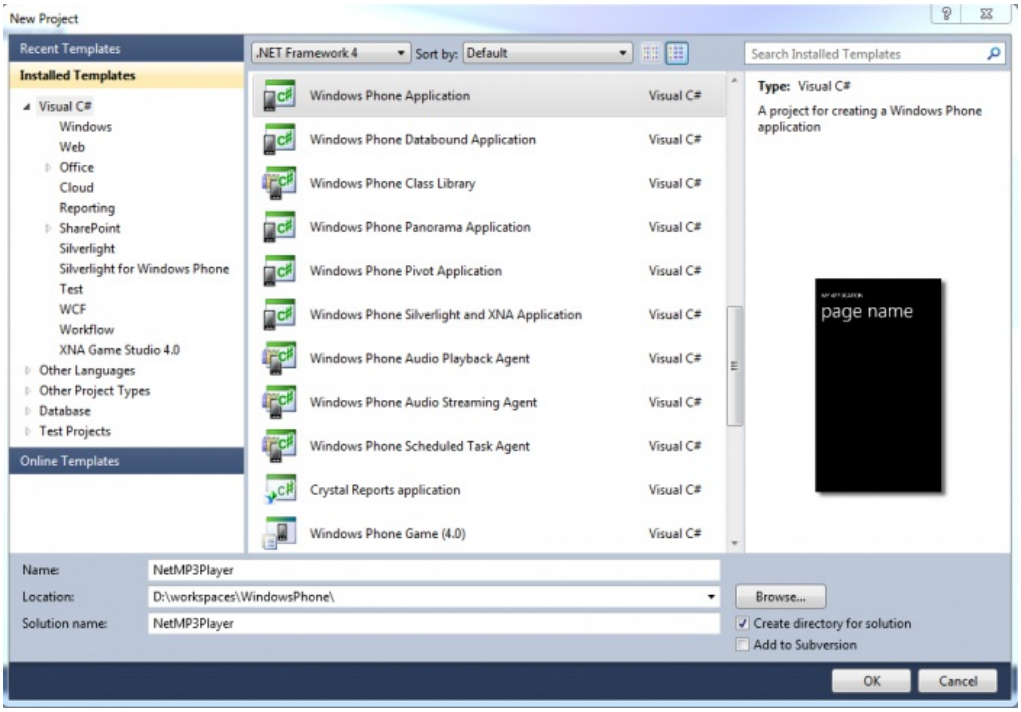
Windows Phone 7.1 SDK

To Develop application for Windows Phone 7 devices, you need to install Windows Phone 7.1 SDK. You can download latest SDK for Windows Phone here.

(C) Copyright Nokia 2013. All rights reserved.

Windows Phone Application

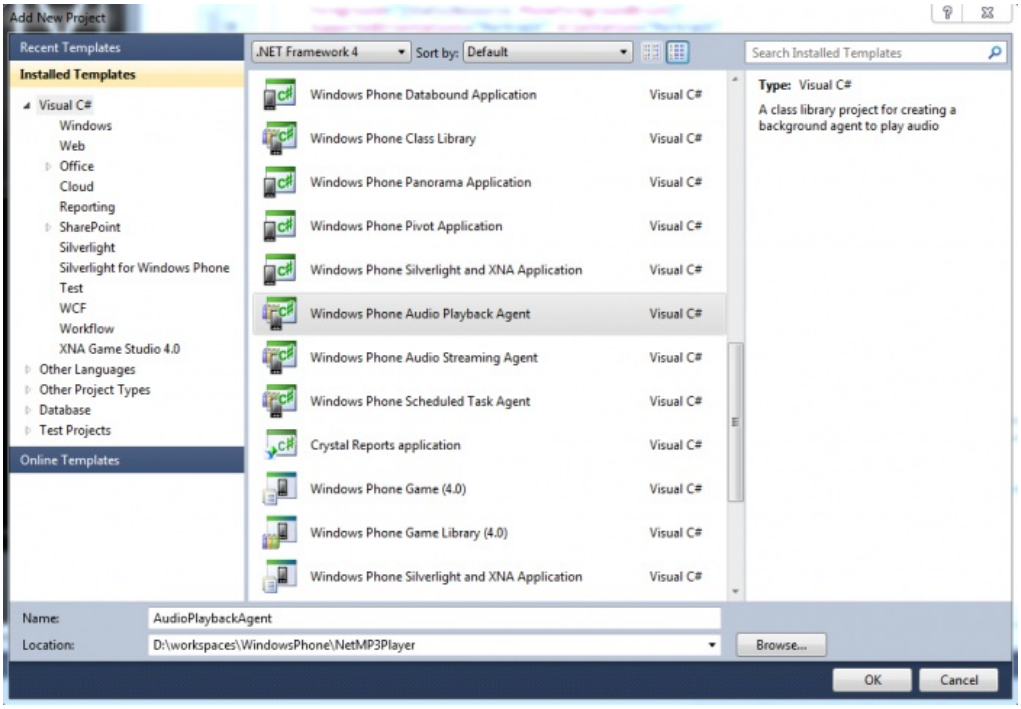
To start creating a new Windows Phone application, start **Microsoft Visual Studio** then **create a new Project** and select **Windows Phone Application** Template. Panorama Items will be created in the code.



This example uses C# as code behind language.

Audio Playback Agent

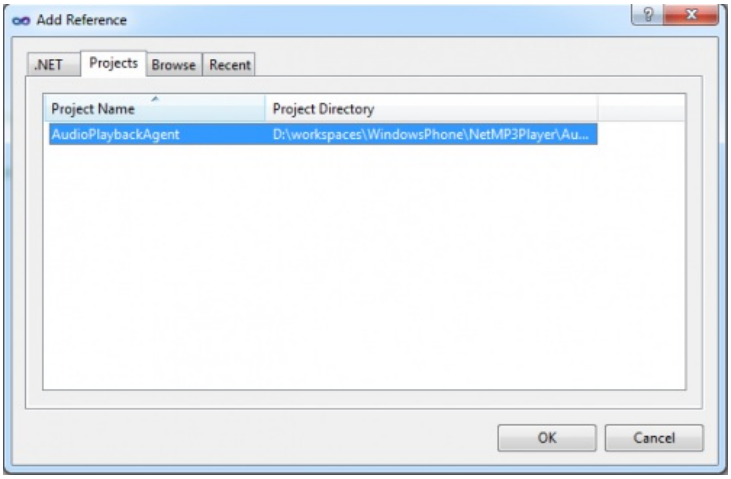
This code example uses Audio Playback Agent to play MP3 music in Windows Phone. You have to add Audio Playback Agent to your Solution. Select Solution from your project and Add a new Project to your solutions and name to AudioPlaybackAgent. This adds Audio Playback Agent to your project which is instantiated by the operating system to handle actions requested by this application.



You can open AudioPlayer.cs class from AudioPlaybackAgent project to see generated code to handle music playing states. Code will be modified later in this code example. Read more information about Background Audio in Windows Phone 7 here: Background Audio Overview for Windows Phone.

Audio Playback Agent reference should be added to Windows Phone project. Select your project and add a new reference to your AudioPlaybackAgent.

(C) Copyright Nokia 2013. All rights reserved.



Project extensions

Microsoft Visual Studio extensions can be installed in many different ways. This article uses Nuget which makes extension installation easier. Find and install Microsoft Silverlight Toolkit and Simple JSON extensions. Look more information about these extensions and installation instruction from Picasa Image Gallery with JSON in WP7 article.

Install SilverLight Toolkit only for you main project and Simple JSON for both projects.

Class behind the solution : Album and Song

Album and song data will be saved to **Album.cs** and **Song.cs** classes.

Album.cs

Album class stores album `Name`, `Artist` and `Directory` data to string variables. All the album songs are stored to `Songs` list collection.

```
namespace NetMP3Player
{
    public class Album
    {
        public string Name { get; set; }
        public string Artist { get; set; }
        public string Directory { get; set; }
        public List<Song> Songs { get; set; }
    }
}
```

Song.cs

Song class stores one song data to different string based variables.

```
namespace NetMP3Player
{
    public class Song
    {
        public string Directory { get; set; }
        public string Filename { get; set; }
        public string Artist { get; set; }
        public string Title { get; set; }
        public string Album { get; set; }
        public string Playtime { get; set; }
        public string AlbumArt { get; set; }
    }
}
```

Handling albums data between different Pages

This code example has a few different pages: Main Panorama Page (with different Lists), Songs Page (displays one album songs) and Player Page. One of the easiest way to share data between these pages in Windows Phone is to store all data to `App.xaml.cs` class. One other advantage to store all data to same place is when you have to handle Tombstoning mode in Windows Phone.

App.xaml.cs class

In `App.xaml.cs` class all the album data is stored to different List collections. All the albums are stored to `Albums` List, all the latest songs are stored to `LatestSongs` list and all the selected favorite songs are stored to `FavoriteSongs` List. There is also one string variable `ToPlayer`, which describes which list is used when the latest song is selected (Latest, Favorite or Album).

```
// List of Albums
public List<Album> Albums = new List<Album>();
```


(C) Copyright Nokia 2013. All rights reserved.

```
// List of Latest Songs
public List<Song> LatestSongs = new List<Song>();
// List of Favorite Songs
public List<Song> FavoriteSongs = new List<Song>();
// To Player (from which Page and list : Main - Latest, Favorite, Album - to Player Page)
public string ToPlayer = "";
```

Application load all of the those lists data from Isolated Storage when the application is launched. `Application_Launching` method will be executed in Windows Phone when the application is launched.

```
private void Application_Launching(object sender, LaunchingEventArgs e)
{
// connect to isolated storage
IsolatedStorageFile isoStore = IsolatedStorageFile.GetUserStoreForApplication();

// Albums.dat
using (IsolatedStorageFileStream stream = new IsolatedStorageFileStream("AlbumsData.dat", FileMode.OpenOrCreate, isoStore))
{
 if (stream.Length > 0)
 {
  try {
   DataContractSerializer serializer = new DataContractSerializer(typeof(List<Album>));
   Albums = serializer.ReadObject(stream) as List<Album>;
  } catch (SerializationException) {
   MessageBox.Show("Error loading Albums data from Isolated Storage!");
  }
 }
}
// Latest.dat
using (IsolatedStorageFileStream stream = new IsolatedStorageFileStream("LatestData.dat", FileMode.OpenOrCreate, isoStore))
{
 if (stream.Length > 0)
 {
  try {
   DataContractSerializer serializer = new DataContractSerializer(typeof(List<Song>));
   LatestSongs = serializer.ReadObject(stream) as List<Song>;
  } catch (SerializationException) {
   MessageBox.Show("Error loading Latest data from Isolated Storage!");
  }

 }
}
// PlayList.dat
using (IsolatedStorageFileStream stream = new IsolatedStorageFileStream("FavoriteData.dat", FileMode.OpenOrCreate, isoStore))
{
 if (stream.Length > 0)
 {
  try
  {
   DataContractSerializer serializer = new DataContractSerializer(typeof(List<Song>));
   FavoriteSongs = serializer.ReadObject(stream) as List<Song>;
  } catch (SerializationException) {
   MessageBox.Show("Error loading Favorite data from Isolated Storage!");
  }
 }
}
}
```

Application save lists data to Isolated Storage when the lists data is changed in the application. These saving methods are described here in App.xaml.cs class.

Save Favorites to isolated storage (this is called from Application Pages)

```
public void SaveFavoritesToIsolatedStorage()
{
 IsolatedStorageFile isoStore = IsolatedStorageFile.GetUserStoreForApplication();
 // FavoriteList.dat
 using (IsolatedStorageFileStream stream = new IsolatedStorageFileStream("FavoriteData.dat", FileMode.Create, isoStore))
 {
  DataContractSerializer serializer = new DataContractSerializer(typeof(List<Song>));
  serializer.WriteObject(stream, FavoriteSongs);
 }
}
```

Save Albums to isolated storage (this is called from Application Pages)

```
public void SaveAlbumsToIsolatedStorage()
{
 IsolatedStorageFile isoStore = IsolatedStorageFile.GetUserStoreForApplication();
 // Albums.dat
 using (IsolatedStorageFileStream stream = new IsolatedStorageFileStream("AlbumsData.dat", FileMode.Create, isoStore))
 {
  DataContractSerializer serializer = new DataContractSerializer(typeof(List<Album>));
  serializer.WriteObject(stream, Albums);
 }
}
```

Save Latest to isolated storage (this is called from Application Pages)

```
public void SaveLatestToIsolatedStorage()
{
 IsolatedStorageFile isoStore = IsolatedStorageFile.GetUserStoreForApplication();
 // Latest.dat
 using (IsolatedStorageFileStream stream = new IsolatedStorageFileStream("LatestData.dat", FileMode.Create, isoStore))
 {
  DataContractSerializer serializer = new DataContractSerializer(typeof(List<Song>));
  serializer.WriteObject(stream, LatestSongs);
 }
}
```

Thombstoning mode is handled here in App.xaml.cs class. So all of the lists are saved with PhoneApplicationServices current state object. And loaded with same way if Thombstoning mode is detected in application activating or deactivating.

`Application_Activated` method will be executed when the application is activated (brought to foreground). Here all the data is loaded from the

(C) Copyright Nokia 2013. All rights reserved.

```
private void Application_Activated(object sender, ActivatedEventArgs e)
        {
            if (e.IsApplicationInstancePreserved)
            {
                // DORMANT
                //System.Diagnostics.Debug.WriteLine("EVENT: Application Activated");
            }
            else
            {
                // THOMBSTONED - mode, read Lists from State
                // get Albums
                if (PhoneApplicationService.Current.State.ContainsKey("Albums"))
                {
                    Albums = (List<Album>)PhoneApplicationService.Current.State["Albums"];
                    PhoneApplicationService.Current.State.Remove("Albums");
                }
                // get LatestSongs
                if (PhoneApplicationService.Current.State.ContainsKey("LatestSongs"))
                {
                    LatestSongs = (List<Song>)PhoneApplicationService.Current.State["LatestSongs"];
                    PhoneApplicationService.Current.State.Remove("LatestSongs");
                }
                // get FavoriteSongs
                if (PhoneApplicationService.Current.State.ContainsKey("FavoriteSongs"))
                {
                    FavoriteSongs = (List<Song>)PhoneApplicationService.Current.State["FavoriteSongs"];
                    PhoneApplicationService.Current.State.Remove("FavoriteSongs");
                }
                // get ToPlayer
                if (PhoneApplicationService.Current.State.ContainsKey("ToPlayer"))
                {
                    ToPlayer = (string)PhoneApplicationService.Current.State["ToPlayer"];
                    PhoneApplicationService.Current.State.Remove("ToPlayer");
                }
            }
        }
```

Same way the `Application_Deactivated` method will be executed when the application is deactivated (sent to background). Here all the data is saved to the PhoneApplicationService State.

```
private void Application_Deactivated(object sender, DeactivatedEventArgs e)
        {
            // Store lists to States (and read in Application_Activated if Thobstoned)
            if (!PhoneApplicationService.Current.State.ContainsKey("Albums")) PhoneApplicationService.Current.State.Add("Albums", Al
            if (!PhoneApplicationService.Current.State.ContainsKey("LatestSongs")) PhoneApplicationService.Current.State.Add("Latest
            if (!PhoneApplicationService.Current.State.ContainsKey("FavoriteSongs")) PhoneApplicationService.Current.State.Add("Favo
            if (!PhoneApplicationService.Current.State.ContainsKey("ToPlayer")) PhoneApplicationService.Current.State.Add("ToPlayer"
        }
```

This is all in the App class.

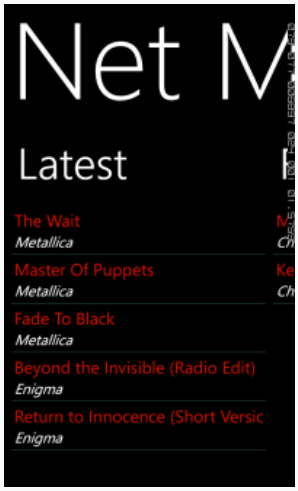
Panorama - Main Page of the application

This application uses Panorama as a main page of the application. Here in Panorama Page user can select song from Latest, Favorite and Album song lists. All the lists data can be removed from Settings in Panorama Page.

Design (MainPage.xaml)

This Panorama Page uses three different lists as a `PanoramaItems`. These lists data are binded from the Album and Song classes.

Latest list shows the latest song title and artist as in a one row in the `LatestListBox` list. This latest played song can be also added to Favorite songs by pressing and holding this item with finger. SilverLight Toolkit has ContextMenu which can be added for example to the list item. `LatestListBox_SelectionChanged` method will be called when list item will be selected and Player Page is displayed to play a new song. TiltEffect is also enabled in the list item, so it moves a little bit down when it is pressed.




```
<controls:Panorama Title="Net MP3 Player">
```

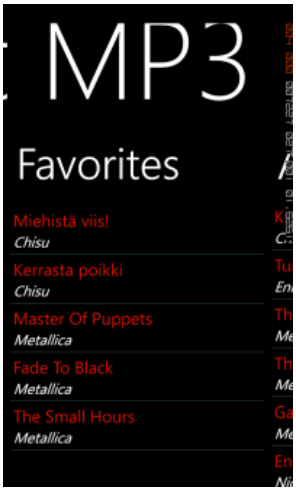

(C) Copyright Nokia 2013. All rights reserved.

```
<controls:PanoramaItem Header="Latest">
 <ListBox x:Name="LatestListBox"
    SelectionChanged="LatestListBox_SelectionChanged"
    toolkit:TiltEffect.IsTiltEnabled="True">
  <ListBox.ItemTemplate>
   <DataTemplate>
    <StackPanel>
     <Grid>
      <StackPanel Grid.Column="0" Margin="5" Width="450">
       <toolkit:ContextMenuService.ContextMenu>
        <toolkit:ContextMenu>
         <toolkit:MenuItem Header="Add to PlayList" Click="AddToFavoritesMenuItem_Click"/>
        </toolkit:ContextMenu>
       </toolkit:ContextMenuService.ContextMenu>
       <TextBlock Text="{Binding Title}" FontSize="28" Foreground="{StaticResource PhoneAccentBrush}"/>
       <TextBlock Text="{Binding Artist}" FontSize="24" FontStyle="Italic" Foreground="White"/>
      </StackPanel>
     </Grid>
     <Rectangle Fill="DarkSlateGray" Height="1" HorizontalAlignment="Stretch" />
    </StackPanel>
   </DataTemplate>
  </ListBox.ItemTemplate>
 </ListBox>
</controls:PanoramaItem>
```

Remember add reference to Microsoft.Phone.Controls to use Panorama and following namespaces to use Panorama Controls and Toolkit in Main.xaml.

```
xmlns:controls="clr-namespace:Microsoft.Phone.Controls;assembly=Microsoft.Phone.Controls"
xmlns:toolkit="clr-namespace:Microsoft.Phone.Controls;assembly=Microsoft.Phone.Controls.Toolkit"
```

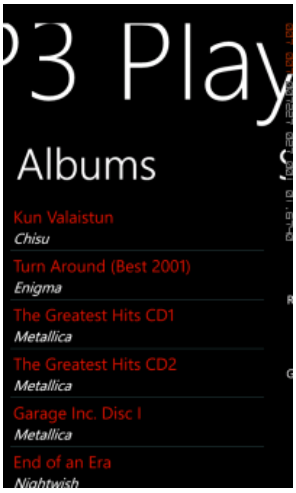
Favorite list shows the favorite song title and artist as in a one row in the `FavoriteListBox` list. This is almost the same as the latest song list except that favorite song can be removed from the list by pressing and holding finger on the selected item. This will call the `RemoveFromMenuItem_Click` to remove song from the list. `FavoriteListBox_SelectionChanged` method will be called when list item will be selected and Player Page is displayed to play a new song.




```
<controls:PanoramaItem Header="Favorites">
 <ListBox x:Name="FavoriteListBox"
    SelectionChanged="FavoriteListBox_SelectionChanged"
    toolkit:TiltEffect.IsTiltEnabled="True">
  <ListBox.ItemTemplate>
   <DataTemplate>
    <StackPanel>
     <Grid>
      <StackPanel Grid.Column="0" Margin="5" Width="450">
       <toolkit:ContextMenuService.ContextMenu>
        <toolkit:ContextMenu>
         <toolkit:MenuItem Header="Remove from Favorites" Click="RemoveFromMenuItem_Click"/>
        </toolkit:ContextMenu>
       </toolkit:ContextMenuService.ContextMenu>
       <TextBlock Text="{Binding Title}" FontSize="28" Foreground="{StaticResource PhoneAccentBrush}"/>
       <TextBlock Text="{Binding Artist}" FontSize="24" FontStyle="Italic" Foreground="White"/>
      </StackPanel>
     </Grid>
     <Rectangle Fill="DarkSlateGray" Height="1" HorizontalAlignment="Stretch" />
    </StackPanel>
   </DataTemplate>
  </ListBox.ItemTemplate>
 </ListBox>
</controls:PanoramaItem>
```

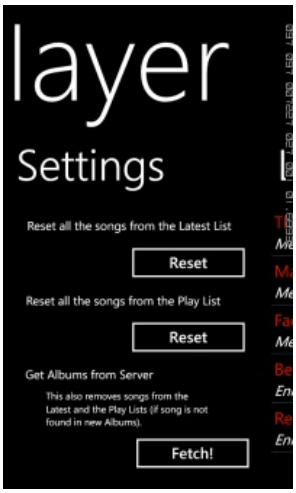
Albums list shows the albums song name and artist as in a one row in the `AlbumsListBox` list. `AlbumsListBox_SelectionChanged` method will be called when list item will be selected and Songs Page will be opened to display the songs of the album.

(C) Copyright Nokia 2013. All rights reserved.




```
<controls:PanoramaItem Header="Albums">
 <ListBox x:Name="AlbumsListBox"
    SelectionChanged="AlbumsListBox_SelectionChanged"
    toolkit:TiltEffect.IsTiltEnabled="True">
  <ListBox.ItemTemplate>
   <DataTemplate>
    <StackPanel>
     <Grid>
      <StackPanel Grid.Column="0" Margin="5"  Width="450">
       <TextBlock Text="{Binding Name}" FontSize="28" Foreground="{StaticResource PhoneAccentBrush}"/>
       <TextBlock Text="{Binding Artist}" FontSize="24" FontStyle="Italic" Foreground="White"/>
      </StackPanel>
     </Grid>
     <Rectangle Fill="DarkSlateGray" Height="1" HorizontalAlignment="Stretch" />
    </StackPanel>
   </DataTemplate>
  </ListBox.ItemTemplate>
 </ListBox>
 </Grid>
</controls:PanoramaItem>
```

Final `PanoramaItem` in Panorama Page is Settings. This section only displays some info texts and buttons to reset data list of this applications.




```
<controls:PanoramaItem Header="Settings">
 <Grid>
  <Button Content="Reset" Height="72" HorizontalAlignment="Left" Margin="190,56,0,0"
    Name="ResetFavoriteListButton" VerticalAlignment="Top" Width="210"
    Click="ResetFavoriteListButton_Click" />
  <Button Content="Reset" Height="72" HorizontalAlignment="Left" Margin="190,179,0,0"
    Name="ResetPlayListButton" VerticalAlignment="Top" Width="210"
    Click="ResetPlayListButton_Click" />
  <Button Content="Fetch!" Height="72" Margin="190,371,7,0"
    Name="FetchAlbumsButton" VerticalAlignment="Top" Width="210"
    Click="FetchAlbumsButton_Click" />
  <TextBlock Height="30" HorizontalAlignment="Left" Margin="28,16,0,0" Name="textBlock1"
       Text="Reset all the songs from the Latest List" VerticalAlignment="Top" Width="372" />
  <TextBlock Height="30" HorizontalAlignment="Left" Margin="26,140,0,0" Name="textBlock2"
       Text="Reset all the songs from the Play List" VerticalAlignment="Top" Width="352" />
  <TextBlock Height="30" HorizontalAlignment="Left" Margin="25,262,0,0" Name="textBlock3"
       Text="Get Albums from Server" VerticalAlignment="Top" />
  <TextBlock Height="77" HorizontalAlignment="Left" Margin="59,301,0,0" x:Name="InfoTextBlock" FontSize="16"
       Text="This also removes songs from the&#13;Latest and the Play Lists (if song is not&#13;found in new Albums)." VerticalAlign
 </Grid>
</controls:PanoramaItem>

</controls:Panorama>
```

Programming (MainPage.xaml.cs)

A few variables are described here in the Main page. `ServerURL` holds the URL of the server and your StreamingDemo folder where the music and other needed files are stored. `App` is a reference to App class where data lists are stored. Albums data can be reset and load again and `albumsLoadedAgain` is

(C) Copyright Nokia 2013. All rights reserved.

used in this purpose.

```
private string ServerURL = "http://YOUR OWN SERVER HERE/StreamingDemo/";
private App app = App.Current as App;
private bool albumsLoadedAgain = false;
```

All the data loading happends here in `MainPage` class. If the Album count is zero here in the `OnNavigatedTo` method, then data is loaded from the PHP server. This Main Page is also starting point of this application, so if music is already playing, Player Page will be loaded and displayed.

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

 // Back Key is pressed - do nothing
 if (e.NavigationMode == System.Windows.Navigation.NavigationMode.Back) return;

 // If music is playing, navigate to PlayerPage
 if (BackgroundAudioPlayer.Instance.PlayerState == PlayState.Playing)
 {
  this.NavigationService.Navigate(new Uri("/PlayerPage.xaml?main=true", UriKind.Relative));
  return;
 }

 // If there are no albums, try load from Server
 if (app.Albums.Count() == 0)
 {
  // load albums data from Server
  LoadJSONData();
 }

}
```

Data will be loaded from PHP server with JSON string.

```
private void LoadJSONData()
{
 // albums data are now loading from net
 WebClient webClient = new WebClient();
        // server url and php file
 Uri uri = new Uri(ServerURL + "getAlbums.php";
 webClient.UploadStringCompleted += new UploadStringCompletedEventHandler(JSONDataDownloaded);
        // send a secret requst string to php
 webClient.UploadStringAsync(uri, "request=secretRequest");
}
```

`JSONDataDownloaded` method will be called when data is returned from the PHP server. Here all the JSON data is deserialized to dynamic objects and album data is stored to album and song objects.

```
private void JSONDataDownloaded(object sender, UploadStringCompletedEventArgs e)
{
 if (e.Result == null || e.Error != null)
 {
  MessageBox.Show("Cannot get albums data from server!");
  return;
 }
 // clear Albums List, new data is coming
 app.Albums.Clear();
 try
 {
  // Deserialize JSON string to dynamic object
  IDictionary<string, object> json = (IDictionary<string, object>)SimpleJson.DeserializeObject(e.Result);
  // Albums List
  IList albumsData = (IList)json["Albums"];
  // Find album details
  for (int i = 0; i < albumsData.Count; i++)
  {
   // Create a new Album
   Album album = new Album();
   album.Songs = new List<Song>();
   // Album object
   IDictionary<string, object> albumData = (IDictionary<string, object>)albumsData[i];
   // Get albumname
   album.Name = (string)albumData["albumname"];
   // album artist (artist or Various Artists)
   album.Artist = "";
   // Get album directory
   album.Directory = (string)albumData["albumdirectory"];
   // Get songs
   IList songsData = (IList)albumData["songs"];
   // Get song data from JSON
   for (int k = 0; k < songsData.Count; k++)
   {
    // Create a new Song
    Song song = new Song();
    // Get Song object
    IDictionary<string, object> songData = (IDictionary<string, object>)songsData[k];
    // Get directory
    song.Filename = (string)songData["filename"];
    // Get filename
    song.Directory = (string)songData["albumdirectory"];
    // Set cover
    song.AlbumArt = (string)songData["albumart"];
    // Get artists
    song.Artist = (string)songData["artist"];
    if (k == 0) album.Artist = (string)songData["artist"];
    else if (album.Artist != song.Artist) album.Artist = "Various Artists";
    // Get title
    song.Title = (string)songData["title"];
    song.Title = HttpUtility.HtmlEncode(song.Title);
    // Get album
    song.Album = (string)songData["album"];
    // Get playtime
    song.Playtime = (string)songData["playtime"];
    // Add song to album
    album.Songs.Add(song);
   }
   // Add album to Albums (in app)
   app.Albums.Add(album);
```


(C) Copyright Nokia 2013. All rights reserved.

```
  }
  // save albums data to Isolated Storage
  app.SaveAlbumsToIsolatedStorage();
 }
 catch (SerializationException)
 {
  MessageBox.Show("Cannot load Albums data from Server - JSON serialization exception happened!");
 }
 catch (WebException)
 {
  MessageBox.Show("Cannot get albums data from server!");
 }
 catch (KeyNotFoundException)
 {
  MessageBox.Show("Cannot load Albums data from Server - JSON parsing error happened!");
 }
 // sort albums by Album.Artist
 app.Albums.Sort(SortByAlphabet);
 // show albums data in list
 ShowDataInAlbumList();
 // albums are fetched again, check Latest and PlayList (remove songs that arent in albums anymore)
 if (albumsLoadedAgain) CheckLists();
}
```

Albums data is stored to Isolated Storage and sorted with album artist name. This sorting can be done with own sorting method:

```
// Sort albums by album name
private int SortByAlphabet(Album a1, Album a2)
{
      return a1.Artist.CompareTo(a2.Artist);
}
```

If a new albums are fetched from the server, then `CheckLists()` method will be called. If there are songs in Latest and Favorite lists that are not anymore in new Albums data, then those songs will be removed from the lists. See more information about this from the source codes. This is basic job with compare objects in a few different lists.

Albums data will be binded to UI with following method. Same kind of method is used with Latest and Favorite list in this application. First list `ItemSource` will be sent to empty, then a new data is binded to list and finally nothing is yet selected in the list.

```
private void ShowDataInAlbumList()
{
      AlbumsListBox.ItemsSource = null;
      AlbumsListBox.ItemsSource = app.Albums;
      AlbumsListBox.SelectedIndex = -1;
}
```

From Albums list user can select a album to display album songs. When a new album is selected from `AlbumsListBox`, a new Songs Page is displayed. Selected album index is send to Song Page.

```
private void AlbumsListBox_SelectionChanged(object sender, SelectionChangedEventArgs e)
{
 if (AlbumsListBox.SelectedIndex == -1) return;
 int selectedIndex = (sender as ListBox).SelectedIndex;
 this.NavigationService.Navigate(new Uri("/SongsPage.xaml?selectedIndex=" + selectedIndex, UriKind.Relative));
}
```

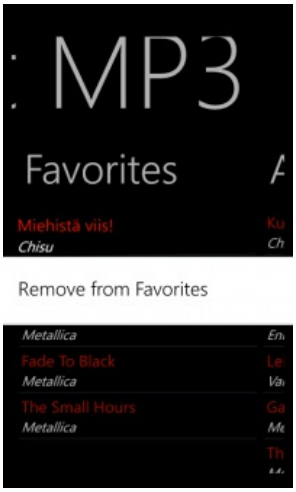
Player Page is displayed when user has selected a new song from Latest or Favorite lists. Selected song index is send to Player Page.

```
private void LatestListBox_SelectionChanged(object sender, SelectionChangedEventArgs e)
{
 if (LatestListBox.SelectedIndex == -1) return;
 int selectedIndex = (sender as ListBox).SelectedIndex;
 this.NavigationService.Navigate(new Uri("/PlayerPage.xaml?selectedIndex=" + selectedIndex + "&latest=true", UriKind.Relative));
}

private void FavoriteListBox_SelectionChanged(object sender, SelectionChangedEventArgs e)
{
 if (FavoriteListBox.SelectedIndex == -1) return;
 int selectedIndex = (sender as ListBox).SelectedIndex;
 this.NavigationService.Navigate(new Uri("/PlayerPage.xaml?selectedIndex=" + selectedIndex + "&favorite=true", UriKind.Relative));
}
```

Here in Main Page user can remove songs from Favorite list. Context menu will be pop-up when user press and hold finger in list item. There is a little trick to find selected song index from `FavoriteListBox`. First selected item will be get and then it's position in the `FavoriteListBox` and finally song will be removed from favorite list. A new data will be shown in UI and modified list will be saved to Isolated Storage.

(C) Copyright Nokia 2013. All rights reserved.




```
private void RemoveFromMenuItem_Click(object sender, RoutedEventArgs e)
{
 // Get selected ListBoxItem
 ListBoxItem selectedListBoxItem = FavoriteListBox.ItemContainerGenerator.ContainerFromItem((sender as MenuItem).DataContext) as Lis

 // Get selected ListBoxItem index in FavoriteListBox
 ListBox view = ItemsControl.ItemsControlFromItemContainer(selectedListBoxItem) as ListBox;
 int index = view.ItemContainerGenerator.IndexFromContainer(selectedListBoxItem);

 // remove song from Favorite songs
 app.FavoriteSongs.RemoveAt(index);

 // show data in Favorite List
 ShowDataInFavoriteList();

 // save Favorite data to Isolated Storage
 app.SaveFavoritesToIsolatedStorage();
}
```

Here in Main Page user can also add songs to Favorite list from Latest list. Added song index will be get in same way as remove song in above. A new data will be shown in UI and saved to Isolated Storage.

```
private void AddToFavoritesMenuItem_Click(object sender, RoutedEventArgs e)
{
 // get selected ListBoxItem
 ListBoxItem selectedListBoxItem = LatestListBox.ItemContainerGenerator.ContainerFromItem((sender as MenuItem).DataContext) as ListB

 // get selected ListBoxItem index in LatestListBox
 ListBox view = ItemsControl.ItemsControlFromItemContainer(selectedListBoxItem) as ListBox;
 int index = view.ItemContainerGenerator.IndexFromContainer(selectedListBoxItem);

 // add song to Favorites Songs
 app.FavoriteSongs.Add(app.LatestSongs[index]);

 // save Favorites List data to Isolated Storage
 app.SaveFavoritesToIsolatedStorage();
}
```

Album data can be reset from the Setting in Panorama Page. A new albums data will be fetched when `FetchAlbumsButton` is clicked. This will empty the album list and data is loaded from PHP server again.

```
// Get new albums data from Server
private void FetchAlbumsButton_Click(object sender, RoutedEventArgs e)
{
 // remove albums data
 ResetAlbumsList();
 // albums are loding again
 albumsLoadedAgain = true;
 InfoTextBlock.Text = "Loading data from server..";
 // start loading data from Server
```


(C) Copyright Nokia 2013. All rights reserved.

```
 LoadJSONData();
}
```

There is also possibility to empty the Latest and Favorite lists from the Settings. Empty data will be shown and stored to Isolated Storage.

```
private void ResetLatestListButton_Click(object sender, RoutedEventArgs e)
{
 // Remove all
 app.LatestSongs.Clear();
 // Show data in Latest List
 ShowDataInLatestList();
 // Save Latest data to Isolated Storage
 app.SaveLatestToIsolatedStorage();
}

private void ResetPlayListButton_Click(object sender, RoutedEventArgs e)
{
 // Remove all
 app.FavoriteSongs.Clear();
 // Show data in Favorite List
 ShowDataInFavoriteList();
 // Save Favorites data to Isolated Storage
 app.SaveFavoritesToIsolatedStorage();
}
```

Songs Page

This page shows the selected album songs in a list.

Design (SongsPage.xaml)

This page shows selected song album name in the Page Title and songs data in a list with title and artist as a list item. Selected song can be added to favorite songs (same way as it is described earlier in Main page) and `songsListBox_SelectionChanged` method will be called when selection is made from `SongsList` (and a new song is played).

```
<ListBox x:Name="SongsListBox"
   SelectionChanged="songsListBox_SelectionChanged"
   toolkit:TiltEffect.IsTiltEnabled="True">
 <ListBox.ItemTemplate>
  <DataTemplate>
   <StackPanel>
    <Grid>
     <StackPanel Grid.Column="0" Margin="5" Width="450">
      <toolkit:ContextMenuService.ContextMenu>
       <toolkit:ContextMenu>
        <toolkit:MenuItem Header="Add to Favorites" Click="AddToFavoritesMenuItem_Click"/>
       </toolkit:ContextMenu>
      </toolkit:ContextMenuService.ContextMenu>
      <TextBlock Text="{Binding Title}" FontSize="28" Foreground="{StaticResource PhoneAccentBrush}"/>
      <TextBlock Text="{Binding Artist}" FontSize="24" FontStyle="Italic" Foreground="White"/>
     </StackPanel>
    </Grid>
    <Rectangle Fill="DarkSlateGray" Height="1" HorizontalAlignment="Stretch" />
   </StackPanel>
  </DataTemplate>
 </ListBox.ItemTemplate>
</ListBox>
```

Coding (SongsPage.xaml)

Songs page is easy to program. It only has a few jobs to do: it load's selected albums data from App class and display it to UI, starts a new song (navigates to Player Page) or add's a song to Favorites.

`OnNavigatedTo` method will be called when this page is displayed. First selected album index is read from previous page and albums data is displayed in `SongsListBox`.

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


(C) Copyright Nokia 2013. All rights reserved.

```
 // nothing is now selected in SongsListBox
 SongsListBox.SelectedIndex = -1;

 // read data from previous Page (selected song index in album data)
 IDictionary<string, string> parameters = this.NavigationContext.QueryString;
 if (parameters.ContainsKey("selectedIndex"))
 {
  // album index in albums
  selectedAlbumIndex = Int32.Parse(parameters["selectedIndex"]);
  // set application and page titles
  ApplicationTitle.Text = app.Albums[selectedAlbumIndex].Name;
  PageTitle.Text = app.Albums[selectedAlbumIndex].Artist;
  // display songs in list
  SongsListBox.ItemsSource = app.Albums[selectedAlbumIndex].Songs;
 }
}
```

Player Page will be called when a new song is selected from the Songs list. Selected album and album song index is send to Player Page.

```
private void songsListBox_SelectionChanged(object sender, SelectionChangedEventArgs e)
{
 // get selected song index
 if (SongsListBox.SelectedIndex == -1) return;
 int selectedIndex = (sender as ListBox).SelectedIndex;
 // navigate to Player Page
 this.NavigationService.Navigate(new Uri("/PlayerPage.xaml?selectedAlbumIndex=" + selectedAlbumIndex + "&selectedSongIndex=" + selec
}
```

User can add songs to Favorites in same way as in Main Page. Only difference is to handle `SongsListBox` and select song from the Album data.

```
private void AddToFavoritesMenuItem_Click(object sender, RoutedEventArgs e)
{
 // get selected ListBoxItem
 ListBoxItem selectedListBoxItem = SongsListBox.ItemContainerGenerator.ContainerFromItem((sender as MenuItem).DataContext) as ListBo

 // get selected ListBoxItem index in SongsListBox
 ListBox view = ItemsControl.ItemsControlFromItemContainer(selectedListBoxItem) as ListBox;
 int index = view.ItemContainerGenerator.IndexFromContainer(selectedListBoxItem);

 // add song to Favorites Songs
 app.FavoriteSongs.Add(app.Albums[selectedAlbumIndex].Songs[index]);
 // save Favorites List data to Isolated Storage
 app.SaveFavoritesToIsolatedStorage();
}
```

Player Page

This is the last page described in this code example. Here user can play and pause the playing song, skip to next or previous song in that list which is playing and seek the position of the song. This page displays the album art if it is available from the PHP server (it must me named with cover.jpg in that folder where the album musics are too) and the playing song title and the artist name.

Design (PlayerPage.xaml)

The page is basic design with different controls. Grid contains: Buttons, TextBlocks, Slider and Image controls. Application title and page name is binded from the selected song data. Play, Next and Prev Buttons will call described `Click-` functions to do their job and when user has moved the slider `SoundSlider_ManipulationCompleted` method will be called.

```
<!--TitlePanel contains the name of the application and page title-->
<StackPanel x:Name="TitlePanel" Grid.Row="0" Margin="12,17,0,28">
 <TextBlock x:Name="ApplicationTitle" Text="NetMP3Player" Style="{StaticResource PhoneTextNormalStyle}"/>
 <TextBlock x:Name="PageTitle" Text="" Margin="9,-7,0,0" Style="{StaticResource PhoneTextTitle1Style}"/>
</StackPanel>

<!--ContentPanel - place additional content here-->
<Grid x:Name="ContentPanel" Grid.Row="1" Margin="12,0,12,0">
 <Button Content="Play" Height="72" HorizontalAlignment="Left" Margin="148,428,0,0" x:Name="PlayButton" VerticalAlignment="Top" Widt
 <Button Content="Next" Height="72" HorizontalAlignment="Left" Margin="290,428,0,0" x:Name="NextButton" VerticalAlignment="Top" Widt
 <Button Content="Prev" Height="72" HorizontalAlignment="Left" Margin="0,428,0,0" x:Name="PrevButton" VerticalAlignment="Top" Width=
 <TextBlock Height="51" HorizontalAlignment="Left" Margin="27,334,0,0" x:Name="TitleTextBlock" Text="Title:" VerticalAlignment="Top"
```


(C) Copyright Nokia 2013. All rights reserved.

```
 <TextBlock Height="39" HorizontalAlignment="Left" Margin="27,383,0,0" x:Name="ArtistTextBlock" Text="Artist:" VerticalAlignment="Top"
 <Slider Height="84" HorizontalAlignment="Left" Margin="27,507,0,0" x:Name="SongSlider" VerticalAlignment="Top" Width="400" Manipula
 <TextBlock Height="30" HorizontalAlignment="Left" Margin="37,559,0,0" x:Name="StartTextBlock" Text="00:00:00" VerticalAlignment="To
 <TextBlock Height="30" HorizontalAlignment="Left" TextAlignment="Right" Margin="327,559,0,0" x:Name="EndTextBlock" Text="00:00:00"
 <Image Height="300" HorizontalAlignment="Left" Margin="75,8,0,0" x:Name="AlbumArtImage" Stretch="Fill" VerticalAlignment="Top" Widt
</Grid>
```

Coding (PlayerPage.xaml)

There is a quite a lot programming here in Player Page. A few variables are used to get reference the App class, handle sound adding to latest songs and update the slider when the sound is playing.

```
private App app = App.Current as App;
private const int LATESTS_MAX = 20;
// check Songs playing time -> update TextBlocks and Slider
private DispatcherTimer dispatcherTimer = new DispatcherTimer();
```

Handle `PlayState` from the AudioPlayback Agent is controlled in constructor. So every time when a something new is happened in Audio Playback Agent, it will notify UI here in Player Page in `Instance_PlayStateChanged` method.

```
public PlayerPage()
{
 // Initialize
 InitializeComponent();
 // handle PlayState from AudioPlayback Agent
 BackgroundAudioPlayer.Instance.PlayStateChanged += new EventHandler(Instance_PlayStateChanged);
}
```

`OnNavigatedTo` method will be called when this page is displayed. This will be described here in parts.

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

First playing state is checked and pause/play button text is handled. If back key is pressed (so application is activated again), start timer again to move slider and show album data.

```
// Playing - set Pause Button text
if (BackgroundAudioPlayer.Instance.PlayerState == PlayState.Playing)
{
 PlayButton.Content = "Pause";
}
// Paused - set Play Button text
else if (BackgroundAudioPlayer.Instance.PlayerState == PlayState.Paused)
{
 PlayButton.Content = "Play";
}

// Back Key is pressed - don't load songs - only update UI
if (e.NavigationMode == System.Windows.Navigation.NavigationMode.Back)
{
 // start timer to check position of the song
 startTimer();
 // set song info
 TitleTextBlock.Text = BackgroundAudioPlayer.Instance.Track.Title;
 ArtistTextBlock.Text = BackgroundAudioPlayer.Instance.Track.Artist;
 // set Page title
 PageTitle.Text = app.ToPlayer;
 // load album art
 LoadAlbumArt();
 return;
}
```

If this page is navigated from Main Page and specially from Album list, then start a playing a new song. First selected album and song index is detected from previous page url and song is added to Latest songs. `GeneratePlayListAndPlay` method will be called to generate song list to Audio Playback agent. This method will be described later in this article.

```
// We are coming from Main Page (Album -> Songs List) - start playing music (always)
IDictionary<string, string> parameters = this.NavigationContext.QueryString;
if (parameters.ContainsKey("selectedAlbumIndex") && parameters.ContainsKey("selectedAlbumIndex"))
{
 // album index in albums
 int selectedAlbumIndex = Int32.Parse(parameters["selectedAlbumIndex"]);
 // album music index
 int selectedSongIndex = Int32.Parse(parameters["selectedSongIndex"]);
 // add Song to Latest Songs
 AddToLatestSongs(app.Albums[selectedAlbumIndex].Songs[selectedSongIndex]);
 // generate play list
 GeneratePlayListAndPlay(selectedSongIndex, app.Albums[selectedAlbumIndex].Songs);
 // where we got here
 PageTitle.Text = app.ToPlayer = "Album songs";
}
```

If this page is navigated from Main page and specially from Latest song then if the same song is already playing - don't start a music again. If there are no sound playing then generate play list from the Latest song.

```
// We are coming from MainPage and (Latest Songs)
if (parameters.ContainsKey("selectedIndex") && parameters.ContainsKey("latest"))
{
 // get select Song title and artist from LatestSongs
 int selectedIndex = Int32.Parse(parameters["selectedIndex"]);
 var title = app.LatestSongs[selectedIndex].Title;
 var artist = app.LatestSongs[selectedIndex].Artist;

 // are we playing this song already and started from Latest Songs List (Main Page)
```


(C) Copyright Nokia 2013. All rights reserved.

```
 if (app.ToPlayer == "LatestSongs" &&
  title == BackgroundAudioPlayer.Instance.Track.Title &&
  artist == BackgroundAudioPlayer.Instance.Track.Artist)
 {
  // do nothing - this song is already playing
  System.Diagnostics.Debug.WriteLine("This song is already playing!");
 }
 else
 {
  // generate play list
  GeneratePlayListAndPlay(selectedIndex, app.LatestSongs);
 }
 // where we got here
 PageTitle.Text = app.ToPlayer = "Latest songs";
}
```

If this page is navigated from Main page and specially from Favorite song, then use Favorite songs to generate play list.

```
// We are coming from Main Page (Favorite Songs)
if (parameters.ContainsKey("selectedIndex") && parameters.ContainsKey("favorite"))
{
 // get select Song title and artist from Favorite Songs
 int selectedIndex = Int32.Parse(parameters["selectedIndex"]);
 var title = app.FavoriteSongs[selectedIndex].Title;
 var artist = app.FavoriteSongs[selectedIndex].Artist;

 // add Song to Latest Songs
 AddToLatestSongs(app.FavoriteSongs[selectedIndex]);
 // generate play list
 GeneratePlayListAndPlay(selectedIndex, app.FavoriteSongs);
 // where we got here
 PageTitle.Text = app.ToPlayer = "Favorite songs";
}
```

And finally, if this page is navigated from Main page and specially not from any list (then application is started again), just show playing album data in UI. `StartTimer`-method will start updating the slider. This will be explained soon in this article.

```
// We are coming from main (song is playing)
if (parameters.ContainsKey("main"))
{
 // display album art
 LoadAlbumArt();
 // show soung info
 TitleTextBlock.Text = BackgroundAudioPlayer.Instance.Track.Title;
 ArtistTextBlock.Text = BackgroundAudioPlayer.Instance.Track.Artist;
}
// start timer to check position of the song
startTimer();
```

End of the `OnNavigatedTo()`-method

```
}
```

When user leaves from this Player Page, `OnNavigatedFrom` method will be called. Here stop the timer to check position of the slider and remove event handler for the playing state changing update.

```
protected override void OnNavigatedFrom(System.Windows.Navigation.NavigationEventArgs e)
{
 dispatcherTimer.Stop();
 dispatcherTimer.Tick -= new EventHandler(dispatcherTimer_Tick);
 BackgroundAudioPlayer.Instance.PlayStateChanged -= new EventHandler(Instance_PlayStateChanged);
}
```

Above `OnNavigatedTo` method calls `AddToLatestSongs` to add started song to Latest song list. In this coding example, latest songs list are limited to 20 songs. First, this new song is compared to songs that are already in latest list and if the song is there, it will be removed from the list. Finally a new song is added to the top of the latest song list and latest songs are saved to Isolated Storage.

```
private void AddToLatestSongs(Song song)
{
 // add selected song to LatestSongs
 int index = 0;
 // remove if already in Latest and set it to first
 foreach (Song latestSong in app.LatestSongs) {
  if (song.Artist == latestSong.Artist && song.Title == latestSong.Title)
  {
   app.LatestSongs.RemoveAt(index);
   break;
  }
  index++;
 }
 app.LatestSongs.Insert(0, song);
 if (app.LatestSongs.Count() > LATESTS_MAX) app.LatestSongs.RemoveAt(app.LatestSongs.Count() - 1);
 // Save Latest data to Isolated Storage
 app.SaveLatestToIsolatedStorage();
}
```

Above `OnNavigatedTo` method calls also `GeneratePlayListAndPlay` method to generate play list, which can be send and played in Audio Playback Agent. In this example songs data is serialized to JSON object and this object is saved to Isolated Storage. This same object is loaded in Audio Playback Agent and deserialized to song objects and finally AudioTracks are generated for the Audio Playback Agent. This is one way to communicate with own UI and Audio Playback Agent.

```
private void GeneratePlayListAndPlay(int selectedSongIndex, List<Song> list)
{
 // JSON string - start playing index and songs data
 string jsonString = "{\"index\":" + selectedSongIndex + ",\"Songs\":" + SimpleJson.SerializeObject(list) + "}";

 // store list to isolated storage
 IsolatedStorageFile isoStore = IsolatedStorageFile.GetUserStoreForApplication();
 isoStore.DeleteFile("playlist.dat");
```


(C) Copyright Nokia 2013. All rights reserved.

```
 using (IsolatedStorageFileStream stream = new IsolatedStorageFileStream("playlist.dat", FileMode.Create, isoStore))
 {
  DataContractSerializer serializer = new DataContractSerializer(typeof(string));
  serializer.WriteObject(stream, jsonString);
 }
 // Stop playing if there are song playing
 if (PlayState.Playing == BackgroundAudioPlayer.Instance.PlayerState ||
  PlayState.Paused == BackgroundAudioPlayer.Instance.PlayerState) BackgroundAudioPlayer.Instance.Stop();
 // start playing a song (Call Play-method from AudioPlayback Agent)
 BackgroundAudioPlayer.Instance.Play();
 PlayButton.Content = "Pause";
}
```

Audio Playback Agent will notify Player page via `Instance_PlayStateChanged` method. If there are song playing, update data in screen (texts, album art and slider).

```
void Instance_PlayStateChanged(object sender, EventArgs e)
{
 // if something is playing (a new song)
 if (BackgroundAudioPlayer.Instance.Track != null)
 {
  // show soung info
  TitleTextBlock.Text = BackgroundAudioPlayer.Instance.Track.Title;
  ArtistTextBlock.Text = BackgroundAudioPlayer.Instance.Track.Artist;
  // handle slider and texts
  SongSlider.Minimum = 0;
  SongSlider.Maximum = BackgroundAudioPlayer.Instance.Track.Duration.TotalMilliseconds;
  string text = BackgroundAudioPlayer.Instance.Track.Duration.ToString();
  EndTextBlock.Text = text.Substring(0, 8);
  // album art
  LoadAlbumArt();
 }
}
```

Album art is loaded to `BitmapImage` and set to `AlbumArtImage` source. Album art URL is stored to playing track, if it is null then default album art is used (which is stored to project contents).

```
private void LoadAlbumArt()
{
 // get album art Uri from Audio Playback Agent
 Uri albumArtURL = BackgroundAudioPlayer.Instance.Track.AlbumArt;
 // load album art from net
 if (albumArtURL != null && latestAlbumArtPath != albumArtURL.AbsolutePath)
 {
  latestAlbumArtPath = albumArtURL.AbsolutePath;
  AlbumArtImage.Source = new BitmapImage(albumArtURL);
 }
 // there is no album art in net, load album art from project resources
 else if (albumArtURL == null)
 {
  Uri uri = new Uri("/Images/NoCoverAvailable.jpg", UriKind.Relative);
  BitmapImage bitmapImage = new BitmapImage(uri);
  AlbumArtImage.Source = bitmapImage;
 }
 else
 {
  System.Diagnostics.Debug.WriteLine("Don't load same album art again");
 }
}
```

When music is playing then slider is controlled with `DispatcherTimer` class. `StartTimer` method initializes timer interval and starts the timer.

```
private void startTimer()
{
 // start timer to check position of the song
 dispatcherTimer.Interval = new TimeSpan(0, 0, 0, 0, 500);
 dispatcherTimer.Tick += new EventHandler(dispatcherTimer_Tick);
 dispatcherTimer.Start();
}
```

`dispatcherTimer_Tick` method will be called every half on second. Slider value and song position texts will be updated when the song is playing.

```
private void dispatcherTimer_Tick(object sender, EventArgs e)
{
 // song is playing
 if (PlayState.Playing == BackgroundAudioPlayer.Instance.PlayerState)
 {
  // handle slider
  SongSlider.Minimum = 0;
  SongSlider.Value = BackgroundAudioPlayer.Instance.Position.TotalMilliseconds;
  SongSlider.Maximum = BackgroundAudioPlayer.Instance.Track.Duration.TotalMilliseconds;
  // display text
  string text = BackgroundAudioPlayer.Instance.Position.ToString();
  StartTextBlock.Text = text.Substring(0,8);
  text = BackgroundAudioPlayer.Instance.Track.Duration.ToString();
  EndTextBlock.Text = text.Substring(0, 8);
 }
}
```

`SoundSlider_ManipulationCompleted` method will be called when the user drags the slider in the UI. Slider value will be first readed and a new position to `BackgroundAudioPlayer` instance will be set.

```
private void SoundSlider_ManipulationCompleted(object sender, ManipulationCompletedEventArgs e)
{
 // get slider value
 int sliderValue = (int)SongSlider.Value;
 // create timespan object with milliseconds (from slider value)
 TimeSpan timeSpan = new TimeSpan(0, 0, 0, 0, sliderValue);
 // set a new position of the song
 BackgroundAudioPlayer.Instance.Position = timeSpan;
}
```


(C) Copyright Nokia 2013. All rights reserved.

This Player page has a few Button controls: Prev, Play/Pause and Next which can be used to give method calls to Background Audio Player.

```
private void PrevButton_Click(object sender, RoutedEventArgs e)
{
 // skip to previous song
 BackgroundAudioPlayer.Instance.SkipPrevious();
 // handle text and slider
 PlayButton.Content = "Pause";
 SongSlider.Value = 0;
}

private void NextButton_Click(object sender, RoutedEventArgs e)
{
 // skip to next song
 BackgroundAudioPlayer.Instance.SkipNext();
 // handle text and slider
 PlayButton.Content = "Pause";
 SongSlider.Value = 0;
}

private void PlayButton_Click(object sender, RoutedEventArgs e)
{
 // get Play Button state (Play or Pause)
 string state = (string)PlayButton.Content;
 // if pause, then pause music
 if (state == "Pause")
 {
  BackgroundAudioPlayer.Instance.Pause();
  PlayButton.Content = "Play";
 }
 // if play, then play music
 else
 {
  BackgroundAudioPlayer.Instance.Play();
  PlayButton.Content = "Pause";
 }
}
```

Audio Playback Agent

When creating a Audio Playback Agent project to this solutions, it creates a AudioPlayer class. This class is used to play music in Audio Playback Agent. In this example, songs data is send from the UI (Player Page) to the Audio Playback Agent via JSON string and more specially with Song objects data.

Modify AudioPlayer class and include following variables to store play list, current play index in list and path to server URL where to load songs and album art.

```
private static List<AudioTrack> playList = new List<AudioTrack>();
private static int currentSongIndex = 0;
private static string ServerURL = "http://YOUR OWN SERVER HERE/StreamingDemo/"
```

Audio Playback Agent `OnUserAction` method (in AudioPlayer.cs class) will be called when the user requests an action using application UI. **Modify** `UserAction.Play` case to start playing a song if player is in pause mode, and loading play list from the Isolated Storage if player is not in playing mode.

```
case UserAction.Play:
 if (player.PlayerState == PlayState.Paused)
 {
  // start playing again
  player.Play();
 }
 else if (player.PlayerState != PlayState.Playing)
 {
  // load play list from isolated storage and start playing
  LoadPlayListFromIsolatedStorage(player);
 }
 break;
```

First previous playList is cleared in the `LoadPlayListFromIsolatedStorage` method and a new playList is generated from JSON data which is saved to Isolated Storage. Data contains a list of songs and index where to start playing a song.

```
private void LoadPlayListFromIsolatedStorage(BackgroundAudioPlayer player)
{
 // clear previous playlist
 playList.Clear();

 // access to isolated storage
 IsolatedStorageFile isoStore = IsolatedStorageFile.GetUserStoreForApplication();
 using (IsolatedStorageFileStream stream = new IsolatedStorageFileStream("playlist.dat", FileMode.OpenOrCreate, isoStore))
 {
  if (stream.Length > 0)
  {
   DataContractSerializer serializer = new DataContractSerializer(typeof(string));
   string jsonString = serializer.ReadObject(stream) as string;

   // Deserialize JSON string to dynamic object
   IDictionary<string, object> json = (IDictionary<string, object>)SimpleJson.DeserializeObject(jsonString);
   // start Index
   int index = Int32.Parse(json["index"].ToString());
   currentSongIndex = index;

   // Songs List
   IList songsData = (IList)json["Songs"];
   // Find songs details
   for (int i = 0; i < songsData.Count; i++)
   {
    // Get Song object
    IDictionary<string, object> songData = (IDictionary<string, object>)songsData[i];
    // Get directory
    string directory = (string)songData["Directory"];
    // Get filename
    string filename = (string)songData["Filename"];
    // Get artists
    string artist = (string)songData["Artist"];
    // Get title
```


(C) Copyright Nokia 2013. All rights reserved.

```
    string title = (string)songData["Title"];
    // Get album
    string album = (string)songData["Album"];
    // Get album art
    string albumart = (string)songData["AlbumArt"];
    Uri albumArtURL = null;
    if (albumart != "") albumArtURL = new Uri(ServerURL+albumart,UriKind.Absolute);
    // Get playtime
    string playtime = (string)songData["Playtime"];
    // Create a new AudioTrack object
    AudioTrack audioTrack = new AudioTrack(
     new Uri(ServerURL + directory + filename, UriKind.Absolute), // URL
     title,         // MP3 Music Title
     artist,        // MP3 Music Artist
     album,         // MP3 Music Album name
     albumArtURL    // MP3 Music Artwork URL
     );
    // add song to PlayList
    playList.Add(audioTrack);
   }
  }
  else
  {
   // no AudioTracks from Isolated Storage
  }
  // start playing
                player.Play();
 }
}
```

AudioPlayer class has `GetNextTrack` and `GetPreviousTrack` methods which are called from the UI (from the Player Page in this example). Modify those methods to control currently playing song in Audio Background Player. In this example, a first song is played if next is selected and last song is already playing (and opposite).

```
private AudioTrack GetNextTrack()
{
 AudioTrack track = null;
 currentSongIndex++;
 if (currentSongIndex > playList.Count-1) currentSongIndex = 0;
 track = playList[currentSongIndex];
 return track;
}
```

```
private AudioTrack GetPreviousTrack()
{
 AudioTrack track = null;
 currentSongIndex--;
 if (currentSongIndex < 0) currentSongIndex = playList.Count - 1;
 track = playList[currentSongIndex];
 return track;
}
```

Summary

This code example show how to stream audio from server to Windows Phone application and play it with Audio Playback Agent. Hope you find this article useful and it helps you work with streaming audio in WP7.

You can download source codes here: File:PTMNetMP3Player.zip.

(C) Copyright Nokia 2013. All rights reserved.

(C) Copyright Nokia 2013. All rights reserved.