Read Songs from iTunes

Have you ever wanted to retrieve the list of songs from your iTunes library? Getting songs from iTunes is not easy. In fact, since Apple stopped supplying their COM component for reading from their iTunes library, about the only way to get song data is to export the library into an XML file, then parse the XML. In this blog post you are going to learn to parse the XML using the classes contained in the System.Xml.Linq namespace.

Get Songs from iTunes

To generate an XML file, open iTunes and export the complete library by choosing File > Library > Export Library… from the menu as shown in Figure 1. When prompted, type in the name **Library.xml** and store this file somewhere on your hard drive.

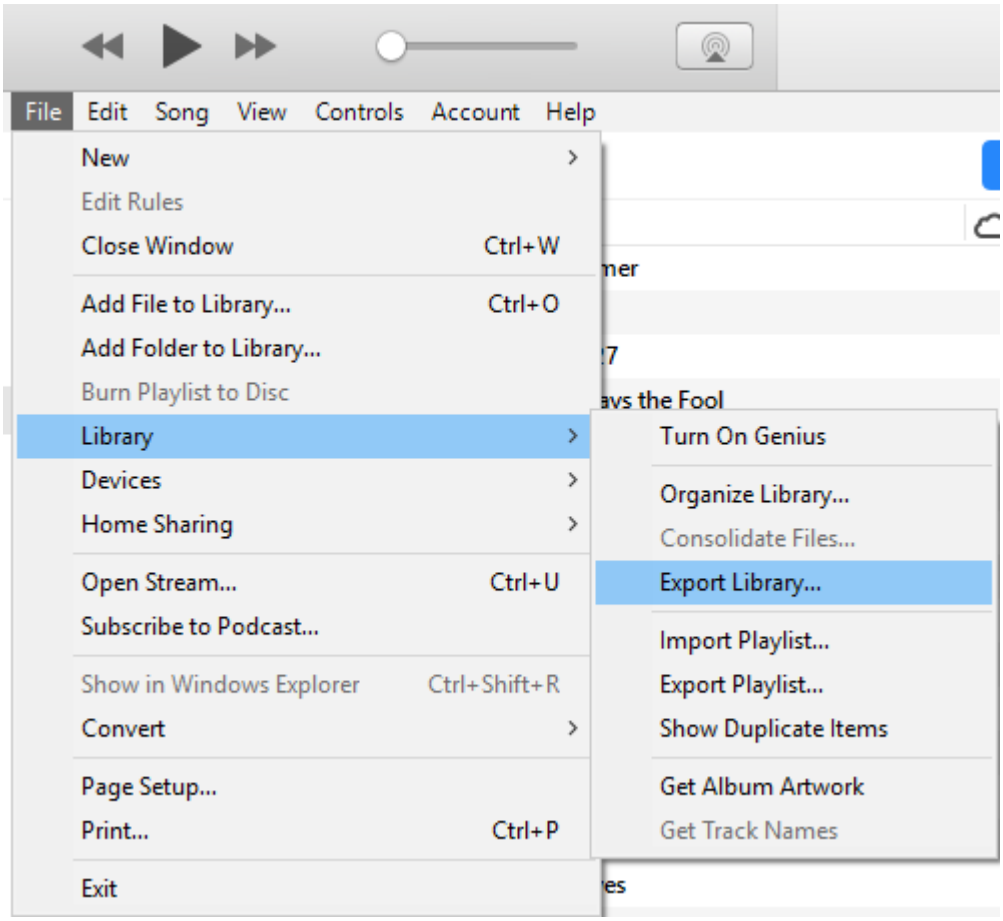

Figure 1: Export your iTunes to an XML file.

iTunes XML

If you open the Library.xml file using Notepad++ or some other editor that reads large files, you should see a structure that looks like the following.

```
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE plist PUBLIC "-//Apple Computer//DTD PLIST 1.0//EN"
"http://www.apple.com/DTDs/PropertyList-1.0.dtd">
<plist version="1.0">
  <dict>
    <key>Major Version</key>
    <integer>1</integer>
    <key>Minor Version</key>
    <integer>1</integer>
    <key>Date</key>
    <date>2018-08-07T13:36:15Z</date>
    <key>Application Version</key>
    <string>12.8.0.150</string>
    <key>Features</key>
    <integer>5</integer>
    <key>Show Content Ratings</key>
    <true/>
    <key>Music Folder</key>
    <string>file://localhost/P:/Music/</string>
    <key>Library Persistent ID</key>
    <string>0472F8AA57FD57AE</string>
    <key>Tracks</key>
    <dict>
      <key>3063</key>
      <dict>
        <key>Track ID</key>
        <integer>16445</integer>
        <key>Name</key>
        <string>When I'm Gone</string>
        <key>Artist</key>
        <string>3 Doors Down</string>
        <key>Album Artist</key>
        <string>3 Doors Down</string>
        <key>Composer</key>
        <string>Brad Arnold/C. "DJ Smooth" Henderson/Matt
Roberts/Todd Harrell</string>
        <key>Album</key>
        <string>Away From The Sun</string>
        <key>Genre</key>
        <string>Rock</string>
        <key>Kind</key>
        <string>AAC audio file</string>
        <key>Size</key>
        <integer>4298973</integer>
        <key>Total Time</key>
        <integer>261919</integer>
        <key>Disc Number</key>
        <integer>1</integer>
        <key>Disc Count</key>
        <integer>1</integer>
        <key>Track Number</key>
        <integer>1</integer>
        <key>Year</key>
        <integer>2002</integer>
        <key>Date Modified</key>
        <date>2013-06-11T21:24:14Z</date>
```

```
        <key>Date Added</key>
        <date>2017-08-30T01:53:27Z</date>
        <key>Bit Rate</key>
        <integer>128</integer>
        <key>Sample Rate</key>
        <integer>44100</integer>
        <key>Play Count</key>
        <integer>54</integer>
        <key>Play Date</key>
        <integer>3572238134</integer>
        <key>Play Date UTC</key>
        <date>2017-03-13T13:22:14Z</date>
        <key>Skip Count</key>
        <integer>2</integer>
        <key>Skip Date</key>
        <date>2014-06-18T12:51:59Z</date>
        <key>Rating</key>
        <integer>80</integer>
        <key>Album Rating</key>
        <integer>80</integer>
        <key>Album Rating Computed</key>
        <true/>
        <key>Normalization</key>
        <integer>5347</integer>
        <key>Artwork Count</key>
        <integer>1</integer>
        <key>Persistent ID</key>
        <string>AACFF77027CEFA7F</string>
        <key>Track Type</key>
        <string>File</string>
        <key>Location</key>
        <string>
          file://localhost/P:/Music/3%20Doors%20Down/
          Away%20From%20The%20Sun/01%20When%20I'm%20Gone.m4a
        </string>
        <key>File Folder Count</key>
        <integer>4</integer>
        <key>Library Folder Count</key>
        <integer>1</integer>
      </dict>

      // Repeat <key> and <DICT> elements for each song
      <key>3064</key>
      <dict>
        // Song information here
      </dict>

    </dict>
  </dict>
</plist>
```

The XML that is generated is not very well structured. All song information is stored as sibling elements under a <dict> element. To retrieve the song name for instance, you first need to locate the <key>Name</key> element, then go to the next node to retrieve the song name.

When you look at the XML data you sometimes see that some of the fields have percent signs (%), followed by some numbers, in them. In some fields, Apple converts spaces to a "%20", but they don't in others. They also convert backslashes to forward slashes. Ampersands (&) are converted to "&" and so on. Since these characters could show up a song name, artist, album, and especially in the physical path and file name you are going to need a method to convert these characters into human-readable characters. You are going to build a method to perform this conversion.

Apple iTunes Class Library

Now that you understand the format of the XML file produced by iTunes, it is time to create a class library to read the song data from this file. Create a new C# class library in Visual Studio named **AppleTunesLibrary**. Into this library add two C# classes named **Song** and **AppleSongReader**. The Song class contains properties to hold the song data you wish to retrieve from the XML file. The AppleSongReader class contains properties and methods used to read the XML file. You can see the overall structure of these classes in Figure 2.

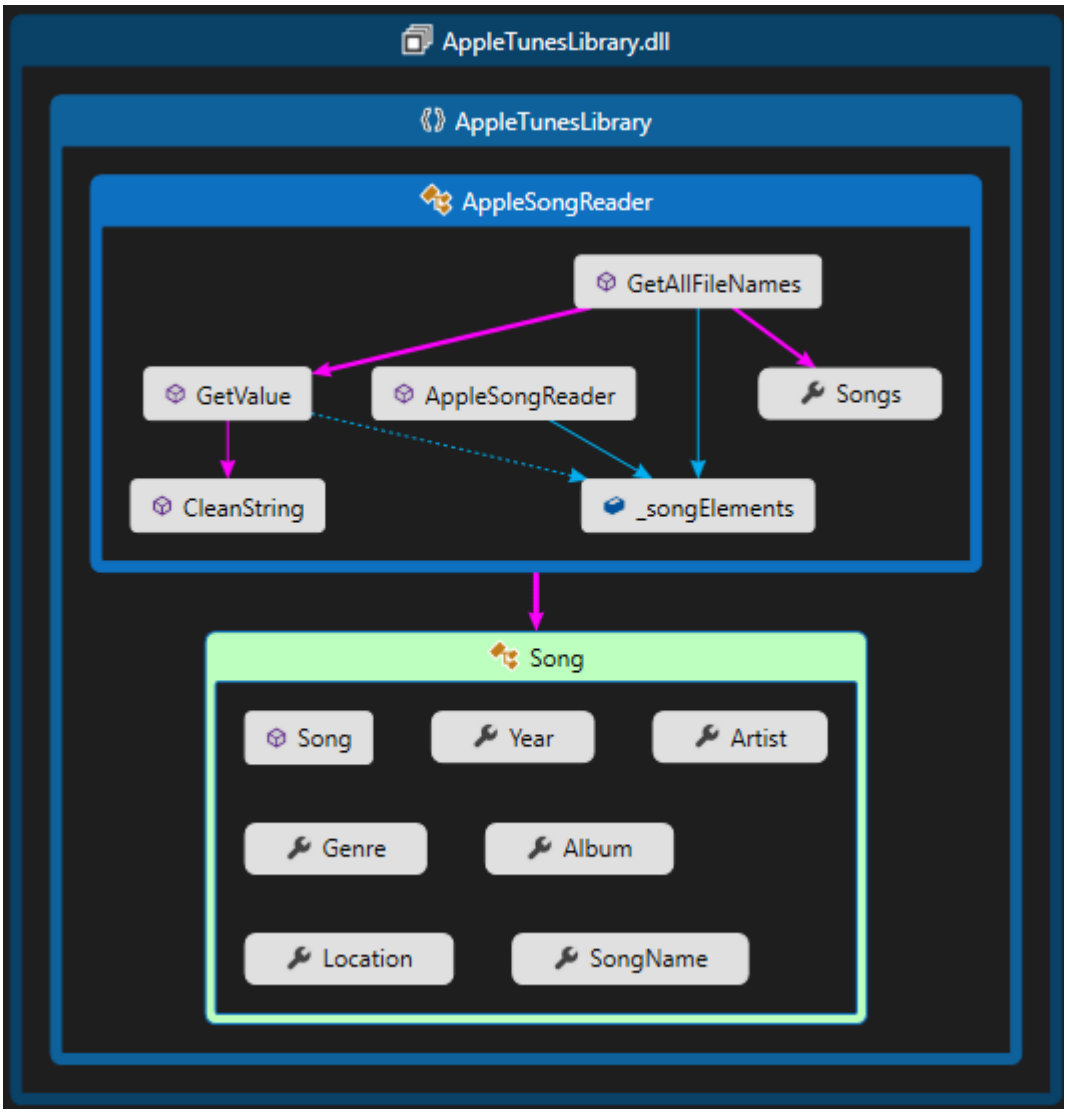

Figure 2: The overall structure of the AppleTunesLibrary.

Song Class

Add several properties to the Song class to represent many of the various elements in the XML file. Feel free to add more or less properties depending on what you wish to see from your iTunes library.

```
public class Song
{
  public string SongName { get; set; }
  public string Artist { get; set; }
  public string Album { get; set; }
  public string Genre { get; set; }
  public int Year { get; set; }
  public string Location { get; set; }
}
```

Apple Song Reader Class

The AppleSongReader class the following properties and methods within it.

Properties	Description
Songs	Holds a generic list of Song objects.
_songElements	A list of XElement objects that represent all elements of one song.
Methods	**Description**
GetAllSongs	Call this method by passing in a full path and file name to the library.xml file exported from iTunes. This method fills in the Songs property.
GetValue<T>	Pass in a key to locate, and this method returns the value for that key.
CleanString	Pass in the value from the XML and this method converts all special characters to human-readable characters.

Let's build this class little by little. First, build the overall structure for this class so it looks like the following.

```
public class AppleSongReader
{
  public List<Song> Songs { get; set; }
  protected List<XElement> _songElements = null;

  public virtual List<Song> GetAllSongs(string libraryFile)
  {

    return this.Songs;
  }

  protected virtual T GetValue<T>(string keyName, T defaultValue)
  {
    T ret;

    return ret;
  }

  protected virtual string CleanString(string value)
  {
    return value;
  }
}
```

The CleanString() Method

Next, fill in the CleanString() method. This method performs several Replace() methods on the string value passed in. Each Replace() method looks for the special characters such as "%20" and converts them to a human readable character. Feel free to add additional conversions for any special characters you find in your song list.

```
protected virtual string CleanString(string value)
{
  return value.Replace("file://localhost/", "")
              .Replace("%20", " ")
              .Replace("/", @"\")
              .Replace("&amp;", "&")
              .Replace("&#38;", "&")
              .Replace("&lt;", "<")
              .Replace("&gt;", ">")
              .Replace("%23", "#")
              .Replace("%25", "%")
              .Replace("%5B", "[")
              .Replace("%5D", "]")
              .Replace("%C2%AD", "----")
              .Replace("%C2%A1", "¡")
              .Replace("%C3%A0", "à")
              .Replace("%C3%A1", "á")
              .Replace("%C3%A4", "ä")
              .Replace("%C3%A8", "è")
              .Replace("%C3%A9", "é")
              .Replace("%C3%AD", "í")
              .Replace("%C3%AF", "ï")
              .Replace("%C3%B3", "ó")
              .Replace("%C3%B6", "ö")
              .Replace("%C3%BC", "ü")
              .Replace("%C3%9F", "ß")
              .Replace("%C5%91", "ő")
              .Replace("%E5%B9%BD", "幽")
              .Replace("%E5%A5%B3", "女");
}
```

The GetValue<T>() Method

The GetValue() method is responsible for locating the <key> element, and returning the value found in the next element. For example, if you have following XML:

```
<key>Name</key>
<string>When I'm Gone</string>
```

To retrieve the song name you call the GetValue() method using the following code:

```
song.SongName = GetValue<string>("Name", "Unknown Name");
```

In the above code snippet, what is placed into the SongName property is the value "When I'm Gone". The second parameter passed into this method is a value to return if the key is not found. Below is the code you should write in the GetValue() method.

```
protected virtual T GetValue<T>(string keyName, T defaultValue)
{
  T ret;
  string value = null;
  XElement elem;

  // Attempt to locate key
  elem = _songElements.Find(k => k.Value == keyName);
  if (elem != null) {
    // Get value from next sibling node and clean it up
    value = CleanString(((XElement)elem.NextNode).Value);
  }

  // Convert value into return type
  if (value != null) {
    try {
      ret = (T)Convert.ChangeType(value, typeof(T));
    }
    catch {
      ret = (T)defaultValue;
    }
  }
  else {
    ret = defaultValue;
  }

  return ret;
}
```

The GetAllSongs() Method

Now that you have the supporting methods created, it is time to build the GetAllSongs() method. Add the code shown below to the GetAllSongs() method. An explanation for this method follows the code.

```
public virtual List<Song> GetAllSongs(string libraryFile)
{
  Song song = null;

  // Load iTunes XML library
  XElement doc = XElement.Load(libraryFile);

  // Create song collection
  Songs = new List<Song>();

  // Get all songs
  IEnumerable<XElement> songs =
          from dict in doc.Elements("dict")
                          .Elements("dict")
                          .Elements("dict")
          select dict;

  foreach (XElement songNode in songs) {
    // Get all children elements for song
    _songElements = songNode.Elements().ToList();

    // Get song information
    song = new Song();
    song.SongName = GetValue<string>("Name", "Unknown Name");
    song.Artist = GetValue<string>("Artist", "Unknown Artist");
    song.Album = GetValue<string>("Album", "Unknown Album");
    song.Genre = GetValue<string>("Genre", "Unknown Genre");
    song.Year = GetValue<int>("Year", 1900);
    song.Location = GetValue<string>("Location", "Unknown
Location");

    // Add song to collection
    this.Songs.Add(song);
  }

  // Sort songs by artist
  Songs = Songs.OrderBy("Artist").ToList();

  return this.Songs;
}
```

The first thing this method does it to load the library XML file using the XElement.Load() method.

```
// Load iTunes XML library
XElement doc = XElement.Load(libraryFile);
```

Next, you locate all songs by reading all the third <dict> elements in the XML document object. You can use LINQ to XML to select all elements and place them into a variable named *songs*.

```
// Get all songs
IEnumerable<XElement> songs =
        from dict in doc.Elements("dict")
                        .Elements("dict")
                        .Elements("dict")
        select dict;
```

With the list of elements in the *songs* variable, iterate over those elements using a foreach statement.

```
foreach (XElement songNode in songs) {
  // Get all children elements for song
  _songElements = songNode.Elements().ToList();

  // Get song information
  song = new Song();
  song.SongName = GetValue<string>("Name", "Unknown Name");
  song.Artist = GetValue<string>("Artist", "Unknown Artist");
  song.Album = GetValue<string>("Album", "Unknown Album");
  song.Genre = GetValue<string>("Genre", "Unknown Genre");
  song.Year = GetValue<int>("Year", 1900);
  song.Location = GetValue<string>("Location", "Unknown Location");

  // Add song to collection
  this.Songs.Add(song);
}
```

Within the foreach loop, retrieve all child XElement objects from the current song node. Assign this list of elements to the field named *_songElements*. This field is used in the GetValue() method to locate the key and value from the sibling XML elements. A new Song object is created and the data from each element is filled into the corresponding properties. The newly created Song object is added to the Songs list, and this logic is repeated for each song in the XML.

After all songs have been loaded into the Songs list, sort the list by the Artist property. Sorting is performed using the OrderBy() method. This method comes from the System.Linq.Dynamic.dll. Add this DLL to your class library project using the NuGet Package Manager.

```
// Sort songs by artist
Songs = Songs.OrderBy("Artist").ToList();
```

WPF Application

To try out this new class library you just created, add a new WPF project to your solution. Create a new WPF window to display all your song data like that shown in Figure 3.

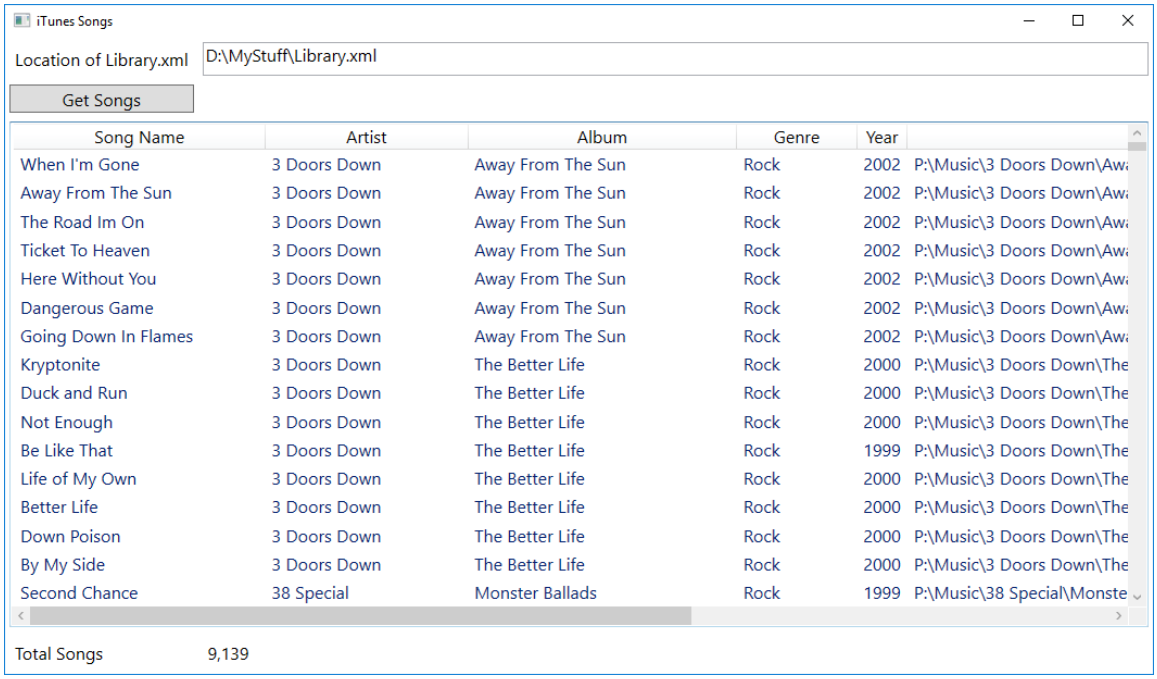

Figure 3: Create a WPF window to display all your songs from iTunes.

XAML for the WPF Window

Below is the XAML needed to build the WPF window shown in Figure 3.

```
<Window x:Class="DecodeiTunes.MainWindow"

xmlns="http://schemas.microsoft.com/winfx/2006/xaml/presentation"
        xmlns:x="http://schemas.microsoft.com/winfx/2006/xaml"
        xmlns:d="http://schemas.microsoft.com/expression/blend/2008"
        xmlns:mc="http://schemas.openxmlformats.org/markup-
compatibility/2006"
        xmlns:local="clr-namespace:DecodeiTunes"
        mc:Ignorable="d"
        FontSize="16"
        Title="iTunes Songs"
        WindowStartupLocation="CenterScreen">
  <Window.Resources>
    <Style TargetType="Label">
      <Setter Property="Margin"
              Value="4" />
    </Style>
    <Style TargetType="TextBox">
      <Setter Property="Margin"
              Value="4" />
    </Style>
    <Style TargetType="Button">
      <Setter Property="Margin"
              Value="4" />
    </Style>
    <Style TargetType="ListView">
      <Setter Property="Margin"
              Value="4" />
    </Style>
  </Window.Resources>
  <Grid>
    <Grid.ColumnDefinitions>
      <ColumnDefinition Width="Auto" />
      <ColumnDefinition Width="*" />
    </Grid.ColumnDefinitions>
    <Grid.RowDefinitions>
      <RowDefinition Height="Auto" />
      <RowDefinition Height="Auto" />
      <RowDefinition Height="*" />
      <RowDefinition Height="Auto" />
    </Grid.RowDefinitions>
    <Label Grid.Column="0">Location of Library.xml</Label>
    <TextBox x:Name="txtFileName"
             Grid.Column="1"></TextBox>
    <Button x:Name="btnGetSongs"
            Grid.Row="1"
            Content="Get Songs"
            Click="btnGetSongs_Click" />
    <ListView x:Name="lstSongs"
              ItemsSource="{Binding}"
              ScrollViewer.HorizontalScrollBarVisibility="Visible"
              ScrollViewer.VerticalScrollBarVisibility="Visible"
              Grid.Row="2"
              Grid.ColumnSpan="2">
      <ListView.View>
        <GridView>
```

```
          <GridViewColumn Header="Song Name"
                          Width="Auto"
                          DisplayMemberBinding="{Binding
Path=SongName}" />
          <GridViewColumn Header="Artist"
                          Width="Auto"
                          DisplayMemberBinding="{Binding
Path=Artist}" />
          <GridViewColumn Header="Album"
                          Width="Auto"
                          DisplayMemberBinding="{Binding
Path=Album}" />
          <GridViewColumn Header="Genre"
                          Width="Auto"
                          DisplayMemberBinding="{Binding
Path=Genre}" />
          <GridViewColumn Header="Year"
                          Width="Auto"
                          DisplayMemberBinding="{Binding Path=Year}"
/>
          <GridViewColumn Header="File Path/Name"
                          Width="Auto"
                          DisplayMemberBinding="{Binding
Path=Location}" />
        </GridView>
      </ListView.View>
    </ListView>
    <Label x:Name="lblSongCount"
           Content="Total Songs"
           Grid.Row="3" />
    <Label x:Name="lblCount"
           Content="0"
           Grid.Column="1"
           Grid.Row="3" />
  </Grid>
</Window>
```

WPF Window Code

Notice there is a button on this window to call a Click event named btnGetSongs_Click(). In this click event is where you write the code to connect to the AppleTunesLibrary.AppleSongReader class. Be sure to add a reference from the WPF project to the Class Library project. Below is the code you enter for this click event.

```
private void btnGetSongs_Click(object sender, RoutedEventArgs e)
{
  AppleSongReader reader = new AppleSongReader();

  // Read all iTunes songs
  Mouse.OverrideCursor = Cursors.Wait;
  reader.GetAllSongs(txtFileName.Text);
  Mouse.OverrideCursor = null;

  // Place song list into ListView control
  lstSongs.DataContext = reader.Songs;

  // Report total songs read
  lblCount.Content = reader.Songs.Count.ToString("###,###");
}
```

Summary

In this blog post you learned to export your iTunes library to an XML file. You wrote code to parse the XML file and extract the data about each of your songs. A song object is created with your song data and each song object is placed into a list of song objects. This list of songs is then displayed on a WPF window.

Sample Code

You can download the complete sample code at my website. http://www.pdsa.com/downloads. Choose "PDSA/Fairway Blog", then "Read Songs from iTunes" from the drop-down.