Project 2: **hTunes**
GUI Programming
100 Points

Overview

You are to create a WPF application that allows a user to organize and play songs similar to iTunes. Audio files can be added to and removed from the app and organized into playlists. The last.fm web API is used to show extra information about each song. Music and playlist data is stored in an XML file.

Requirements

The following specifications must be met in order to receive **95 points**:

1. A list box should list "All Music" and all playlists. When All Music is selected, all the songs should be displayed in the data grid. When a specific playlist is selected, only songs from that playlist should be displayed in the data grid.

2. Songs should be displayed in a data grid control that shows each song's title, artist, album, and genre.

3. When a song is selected, a detail pane should display details about the song including a picture of the album cover which is obtained from last.fm.

4. A song can be played by selecting the song in the data grid pressing the Play button or by right-clicking the song and selecting "Play" from the context menu. The Stop button should stop a playing song. Other buttons that pause, change the volume, or advance to the next song would be helpful but are not required. When a song is finished playing, ideally your app should advance to the next song in the list, but this is not a requirement.

5. A toolbar should be used instead of a menu. It should have a button for adding a song, adding a new playlist, and showing the About dialog box.

6. The program should allow songs to be added to the listing of songs by launching an open dialog box and allowing the user to select a song. Provide a filter that by default shows .mp3, .m4a, .wma, and .wav files in the open file dialog box. After selecting a song, the program should read the metadata stored in the music file (if available), retrieve the album cover image from last.fm, add the song to the data grid, and select/highlight the song (so the user can see it among the list of potentially hundreds of songs).

7. The program should allow the user to create playlists in a manner similar to iTunes. The user should be able to create a new playlist by selecting a New Playlist button from the toolbar. Songs can be added to a playlist by dragging them from the grid control and dropping them onto the desired playlist in the list box. Each song that is dropped onto a playlist should be added to the end of the playlist. When viewing a playlist, the songs should be presented in order of position.

8. Songs can be removed from the list of All Music by right-clicking the song and selecting "Remove" from the context menu. A dialog box should confirm the removal. A song can be removed from a playlist by viewing the songs in the playlist, right-clicking the song, and selecting "Remove from Playlist" from the context menu. No dialog box is necessary. Note that the ordering of the songs in a playlist should be updated to reflect the removal of a song so if song at position 2 was removed, the 3rd song is now position 2, the 4th is position 3, and so on.

9. The list of all song files and playlists should be stored in a file called music.xml. This file should reside in the same folder as the app's .exe. The file should be loaded when the app is first executed and saved when the app terminates. (Ideally the file should be saved whenever a change is made to a song or playlist, but you should only save it once when the app terminates.)

10. Song information can be modified directly in the data grid when viewing All Music. Song info should *not* be editable when viewing a playlist.

11. The main window should be resizable, and the contents should stretch/shrink to fit the size of the window.

12. There should be an About dialog box (*not* a simple MessageBox) that identifies the programmers.

13. You must use the MusicLib library which will be supplied to you for separating the UI code from the business logic.

Additional Requirements

Each additional requirement is worth 5 points:

1. Add the ability to rename and delete a playlist. The user should be able to right-click a playlist and select "Rename" from the context menu. Then a dialog box should display which allows the user to rename the playlist. Make sure you only change the playlist name if the user supplies a valid playlist (no blanks or identically named playlists). A "Delete" option should also be available from the context menu that deletes the playlist.

2. Use a Control Template to alter the look and behavior of the Play and Stop buttons.

3. Allow the user to enter a search string. As the string is being entered, the song list should be populated with any song whose title, artist, album, or genre match any part of the search string. An empty search string should yield all songs.

Teams

Everyone has been assigned to a two-person team:

1. Nicholas Terrill and Adam Campbell
2. Cameron Crouch and Bryan Cuneo
3. Christian Ellison and John Michael Eaton
4. Weicheng (Matthew) Gan and Jon-Michael Fields
5. Alaina Bridges and Jeffrey Gasvoda
6. James Goodpasture and Garrett Holmes
7. C.J. Wilson and Noah Kinslow
8. Caleb Krise and Anna Crowder
9. Levi Mason and Jerred Shepherd
10. Mario Racancoj and Stephen Hoffmann
11. Ricardo Perez and Alec Watson
12. Matthew Sorrell and Lauren Ragland

Both teammates should aim to contribute equally to the project. Teams may use pair programming if desired. Ideally, the project should be implemented in parallel with each teammate responsible for a part of the application.

Each team should create a private repo on github.com and add me (fmccown) as a contributor. Both teammates should consistently push their work to the repository so a history of the project's creation is accurately captured on GitHub.

The repo's README file should contain the following information:

1. URL to the repo on GitHub
2. Brief summary of what the program does
3. List of any known bugs that still remain (hopefully none)
4. List of any additional requirements that were implemented
5. List what each teammate contributed (e.g., code to transform images, populate the list view, etc.)
6. The percentage of work performed by each teammate

Ideally the work performed by each teammate will be 50/50, but if one teammate does more than the other, the percentages should be adjusted accordingly. The percentages will influence each teammate's final grade on the project. Note that the repo history on GitHub should be a fairly accurate indicator of how much work each teammate did. Pair programming is one technique that helps to equalize the amount of effort each teammate puts in, but it does not allow for parallel implementation.

Grading

Your application should work similar to my hTunes app located at http://cs.harding.edu/fmccown/hTunes/

If you fulfill all the requirements, the most points you can receive is 95 points. You will need to fulfill at least one of the additional requirements to get a higher score. You could receive 110 out of 100 points on this project.

These are the things I will be looking for when evaluating your programs:

1. Does your program implement all the requirements and work properly?
2. Does your program make any GUI bloopers?
3. Is your program well written? Did you use proper C# naming conventions for variables, methods, and classes? Did you use constants and functions when appropriate?
4. Did you clearly cite any code that you may have found online?
5. Did you use git adequately when developing your solution? There should be at least **20 commits** in your repo, and each person must have committed at least twice. Your .gitignore should keep out any unnecessary files.

Your source code on GitHub should not be modified after the deadline (10% off each day any code is modified after the due date). I will clone your repository on my machine to grade it. One teammate should submit the team's README file to Easel before the deadline; there is no need to submit anything else to Easel.

Web API

Many native apps make use of data obtained from a data file, a database, or even a remove server. This application will use a web API to obtain additional information about each song. You will need to obtain free API key from last.fm in order to use their API. Once you have an API key, you can use the track.getInfo method to obtain more information about the song. Take a look at the documentation here: http://www.last.fm/api/show/track.getInfo

Here is an example call using my API key which asks for information about the song "Wrong" by Depeche Mode:
http://ws.audioscrobbler.com/2.0/?method=track.getInfo&api_key=e1d7cac3d825c39c69e1e0f2a73ca7f8&artist=depeche+mode&track=wrong

The URL returns the following XML:

```
<?xml version="1.0" encoding="UTF-8" ?>
<lfm status="ok">
   <track>
      <name>Wrong</name>
      <mbid>1788ad67-4f70-4abe-beb6-4e512f2034d2</mbid>
      <url>http://www.last.fm/music/Depeche+Mode/_/Wrong</url>
      <duration>193000</duration>
      <streamable fulltrack="0">0</streamable>
      <listeners>277412</listeners>
      <playcount>2341846</playcount>
      <artist>
         <name>Depeche Mode</name>
         <mbid>8538e728-ca0b-4321-b7e5-cff6565dd4c0</mbid>
         <url>http://www.last.fm/music/Depeche+Mode</url>
      </artist>
      <album position="3">
         <artist>Depeche Mode</artist>
         <title>Sounds Of The Universe</title>
         <mbid>6df83755-4fc6-4e2f-b65c-dbf043bbc407</mbid>
         <url>http://www.last.fm/music/Depeche+Mode/Sounds+Of+The+Universe</url>
         <image size="small">http://img2-ak.lst.fm/i/u/34s/ebfd381f672a4d25acfe3914d030b2b9.png</image>
         <image size="medium">http://img2-ak.lst.fm/i/u/64s/ebfd381f672a4d25acfe3914d030b2b9.png</image>
         <image size="large">http://img2-ak.lst.fm/i/u/174s/ebfd381f672a4d25acfe3914d030b2b9.png</image>
         <image size="extralarge">http://img2-ak.lst.fm/i/u/300x300/ebfd381f672a4d25acfe3914d030b2b9.png</image>
      </album>
      <toptags>
         <tag>
            <name>electronic</name>
            <url>http://www.last.fm/tag/electronic</url>
         </tag>
         <tag>
            <name>synthpop</name>
            <url>http://www.last.fm/tag/synthpop</url>
         </tag>
      </toptags>
      <wiki>
          <published>22 Feb 2009, 15:09</published>
          <summary>summary...</summary>
          <content>Wrong...</content>
      </wiki>
   </track>
</lfm>
```

You should at a minimum display one of the images from the album in the detail pane of the data grid. You may want to provide other information or a link that when clicked would lead to the last.fm web page for the song.

Note: The last.fm library is pretty extensive, but they do not have every song in the world. And not every song is going to have <album> tags or have <image> tags inside the album. If there is no image given, you can display a default image or no image at all.

Since the music data is not likely to change, you should use the web API to obtain song data when a song is first added to the library. You can store the obtained metadata in your music.xml file. This will require you to add any necessary column information into music.xsd and to make changes to your Song class.

You should make the API calls in your MusicLib class. It is not necessary to use a separate thread to make the API calls although that would be an ideal implementation.

Here is some C# code that shows how to make an HTTP request to the API and parse the XML that is returned:

```csharp
String url = "http://ws.audioscrobbler.com/2.0/?method=track.getInfo&api_key=" + API_KEY + "&" +
                "artist=" + WebUtility.UrlEncode(artist) + "&track=" + WebUtility.UrlEncode(title);
try
{
   HttpWebRequest request = WebRequest.Create(url) as HttpWebRequest;
   using (WebResponse response = await request.GetResponseAsync())
   {
      Stream strm = response.GetResponseStream();
      using (XmlTextReader reader = new XmlTextReader(strm))
      {
         while (reader.Read())
         {
            if (reader.NodeType == XmlNodeType.Element)
            {
               if (reader.Name == "image")
               {
                  if (reader.GetAttribute("size") == "medium")
                     Console.WriteLine("Image URL = " + reader.ReadString());
               }
            }
         }
      }
   }
}
catch (WebException e)
{
   // A 400 response is returned when the song is not in their library
   Console.WriteLine("Error: " + e.Message);
}
```