Software Programming For Music and Audio
Audio Playlist Windows Form Application

By Dinis Tavares

Design

The application consists of two GUI Windows forms, where the main form (Figure 1) is used to playback audio and the second form (Figure 2) is used to edit the track details of each song as well as add new songs to the playlist. A simple design has been implemented to each form which provides an obvious indication of what should be done with the application (Galitz, 2007). Once the application is opened the 'Audio Player' form is open all of the buttons are disabled, limiting the user to the menu strip items, where the user will have the option to create a new playlist, open an existing playlist or exit the application.

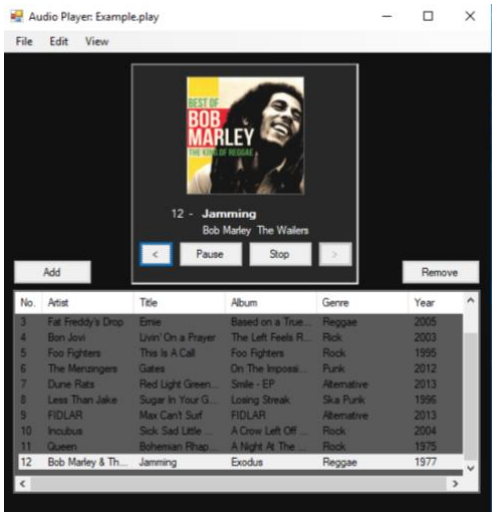


Figure 1Player Form, this is form 1 in the code

If the user selects 'New', the second form will be presented with all the buttons and fields disabled, where the user will need to select 'New' once again, but this time in the menu strip of the second form. As it was not possible to send variables from one form to another, where it is further discussed in the analysis, the 'New' menu strip button in form1 is unable to command the second form to open a new playlist. When the new playlist is then open the user will have to manually enter each track to the playlist by; selecting an audio file and image file by clicking the 'Find Audio' or 'Find Image' buttons and filling out the track details. Once all the details have been filled and the files have been selected, the user will need to select 'Add To Playlist' to update the record, where a new track will be added. However, if the new track does not contain an audio file, the artist name and song title an information message will inform the user to fill those fields and not update the record, if the other text fields are not filled the form will update them to 'Unknown; automatically. Subsequently, the 'Add Track' button will be changed to 'New Track', the 'Next' and 'Delete' buttons will be enabled and the 'Track number' will be updated. If the 'Track Details' form is then close, the user will be prompted to save the new playlist before closing the window.

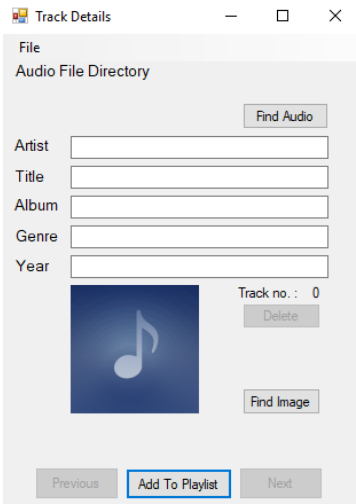


Figure 2 Edit Tracks form, named EditTracks in code

Returning to the main form, ideally the playlist saved in the edit tacks form would open in the main player form however, as the forms were unable to communicate, the user will have to open the saved playlist once again. Once opened in both forms, the user can select 'Edit Track Details' to change the details of each track or the 'Add' button to add new tracks. The remove button allows the user to delete any selected track from the playlist and when there are no tracks selected, the button will be disabled. Using the list view, the user is able to double click a song to start the playback of a desired track from the playlist. Furthermore, if last track of the playlist is selected when the audio is stopped, the next button will be disabled. When the audio is playing the album art and song title will match the playing track and when the audio is stopped the album art will match the selected item, but the song title underneath will be disabled. In addition, if no track is selected or playing, a default image will appear. Finally, once the application is closed, the user will be prompted to save the playlist.

Implementation

The coding of both forms used multiple functions to avoid repetition within the header files, where both forms have a few functions alike. The first step was to create a list using a structure called 'track' in the 'Tracks.h' as well as an index reference each track. When opening a playlist, a 'streamreader' reads a play file which is then further broken down into lines, where each line represents a track. The track (or line) is then further divided into an array of strings called 'Details' by separating each detail with existing tab spaces (Figure 3). Each 'Detail' is then added to the specified strings in the Tracks.h file and is then further added into a single line of the playlist, repeating the process until there are no more lines to read. Once the file is read the list view is then populated, where each track is added to

an item of the list by using the strings stored in the details array. A ‘for’ loop is used to cycle through each line of the playlist, adding each detail into sub-items of the list view (Figure 4).

```
//Using the tab to seperate between details for each track
char deliminator = '\t';

//String array containing details for a given track
array<String^>^ details;

//Dividing the line read into strings, with the use of the tab charater to determine the split points
details = line->Split(deliminator);

//Creating a new structure to hold the details of the track which was just read
Track^ track = gcnew Track();

//Declating each element of the 'details' array using the opened file
track->Artist = details[0];
track->Title = details[1];
track->Album = details[2];
track->Genre = details[3];
track->Year = details[4];
track->AudioDirectory = details[5];
track->ImgDirectory = details[6];

//Adding the track to the end of the list
playlist.Add(track);
```

Figure 3 Separating a string into substrings using tab spaces

Once the file is read and the list-view is populated, a function named user ‘UpdateUI’ is called, which is used to change the functionality of the GUI throughout most events. The UpdateUI function is vital to the functionality of the application, where the buttons of the GUI, track index and labels are changed according to specific attributes. When a selected item is changed, the function will check if any tracks are selected by using the list views selected items index and change whether or not the ‘Play’ or ‘Remove’ buttons are enabled. Furthermore, if the track is stopped the ‘UpdateUI’ function will disable the stop button, change the text of the Play button to pause and index of the track if changed to what is selected (Figure 5). Furthermore, the album art and the nowplaying ID is changed when a track is selected, however, if the track is playing, any highlighted tracks will not change the image and nowplaying ID and they will remain on what the current playing track is (Figure 6).

```
for (int i = 0; i < playlist.Count; i++)
{
    //creating a new list item and adding subitems to each list item
    listViewItem = gcnew System::Windows::Forms::ListViewItem(System::Convert::ToString(i + 1));
    listViewItem->SubItems->Add(this->playlist[i]->Artist);
    listViewItem->SubItems->Add(this->playlist[i]->Title);
    listViewItem->SubItems->Add(this->playlist[i]->Album);
    listViewItem->SubItems->Add(this->playlist[i]->Genre);
    listViewItem->SubItems->Add(this->playlist[i]->Year);
    listViewItem->SubItems->Add(this->playlist[i]->AudioDirectory);
    listViewItem->SubItems->Add(this->playlist[i]->ImgDirectory);
    this->lvTracks->Items->Add(this->listViewItem);
```

Figure 5 Adding track details into listview list

```
//checks if the audio is stopped, and disables buttons if true
if (outputDevice->PlaybackState == PlaybackState::Stopped)
{
    btnStop->Enabled = false;
    btnPlay->Text = "Play";
    if (lvTracks->SelectedItems->Count > 0)
    {
        idxTrack = lvTracks->SelectedItems[0]->Index;
    }
}
```

Figure 4 Changing GUI buttons and track index

```
//if there is no audio output and the number of items already  selected are more than 0...
if (outputDevice->PlaybackState == PlaybackState::Stopped && lvTracks->SelectedItems->Count > 0)
{
    //Calling function
    UpdateUI();

    //Setting the index to the selected item
    idxTrack = lvTracks->SelectedItems[0]->Index;
}
else
{
    txtImgDirectory->Text = "";
    txtFilePath->Text = "";
}
```

Figure 5 Only updates the User Interface if the track stopped and an item is selected

When an item in the list view is double clicked, the even checks if any audio is currently playing or paused, if so a stop function is called to stop the audio, the event then checks if the audio is stopped and then changes the image directory and audio file paths, calls the play function and updates the album art (Figure 7).

```
970  private: System::Void lvTracks_MouseDoubleClick(System::Object^  sender, System::Windows::Forms::MouseEventArgs^  e) {
971
972      //if the audio is playing or paused, stop track
973      if (outputDevice->PlaybackState == PlaybackState::Playing || outputDevice->PlaybackState == PlaybackState::Paused)
974      {
975          StopAP();
976      }
977
978      //Checks if the audio has stopped
979      if (outputDevice->PlaybackState == PlaybackState::Stopped)
980      {
981          //...change the image text to the selected item
982          txtImgDirectory->Text = lvTracks->SelectedItems[0]->SubItems[7]->Text;
983          txtFilePath->Text = lvTracks->SelectedItems[0]->SubItems[6]->Text;
984
985          //calling play function
986          StartAP();
987      }                void Assignment_02_003::Form1::StartAP()
988      //calling u
989      UpdateUI();
990  }
```

Figure 7 Double click event

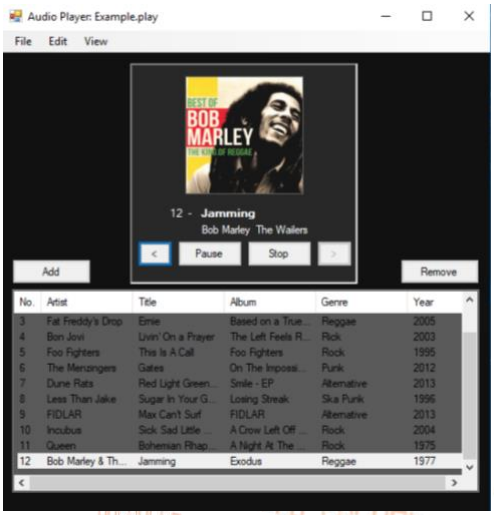
User Testing & Analysis

The application had to be tested multiple times because whilst testing, the goal was to 'break' the application in order to find bugs and fix them. Due to using a list view and allowing the user to select any track from the playlist, it created playing the audio and updating the user interface more complicated. An example of an encountered issue was; if the user was playing the last track of the playlist, and then highlighted a track in the middle of the playlist, the next button would then enable and if the user pressed it, the application would crash. This was caused as the index of the track would when a track was highlighted. To fix that error, an if statement was implemented (figure 5) where it would check if the audio was stopped and only change the track index when stopped. The biggest downfall of the application is the fact that the forms are unable to send values to one another, resulting in the requirement to open a playlist twice and the 'Add' button not working as intended. Using multiple online resources, attempts to send values to separate forms within the solution were made but unachieved as most recourses online are for the language C#. Finally, the next additions to the player would be including a seek position bar, a volume bar as well as reading audio file tags to include track information automatically when adding new songs.

Comparison of Initial Prototype and Finished product

Prototype:

Finished product:



Bibliography & References

- Galitz, Wilbert O., 2007. Title: The Essential Guide to User Interface Design : An Introduction to GUI Design Principles and Techniques. 3rd ed. Indianapolis: Wiley Publishing Inc..
- Stackoverflow.com. (2018). Stack Overflow - Where Developers Learn, Share, & Build Careers. [online] Available at: https://stackoverflow.com/ [Accessed 18 Jan. 2018].
- C-sharpcorner.com. (2018). C# Corner - A Social Community of Developers and Programmers. [online] Available at: http://www.c-sharpcorner.com/ [Accessed 12 Jan. 2018].
- Codeproject.com. (2018). CodeProject - For those who code. [online] Available at: https://www.codeproject.com [Accessed 21 Jan. 2018].
- Msdn.microsoft.com. (2018). Learn to Develop with Microsoft Developer Network | MSDN. [online] Available at: https://msdn.microsoft.com/ [Accessed 20 Jan. 2018].