Synoptic Project

Media Manager

Alex Thomson

2020

Contents

[Project Plan 3](#_Toc57641549)

[Design (1 Day) 3](#_Toc57641550)

[Documentation 3](#_Toc57641551)

[User Interface 3](#_Toc57641552)

[Construction & Testing (3 Days) 3](#_Toc57641553)

[Visual Studio 3](#_Toc57641554)

[Version Control 3](#_Toc57641555)

[Programming 3](#_Toc57641556)

[Coding Practices 3](#_Toc57641557)

[Testing 3](#_Toc57641558)

[Documentation 4](#_Toc57641559)

[Final Day (1 Day) 4](#_Toc57641560)

[Documentation 4](#_Toc57641561)

[Summary 4](#_Toc57641562)

[Overview & Objectives 4](#_Toc57641563)

[Critical Activities 4](#_Toc57641564)

[Background Information 4](#_Toc57641565)

[Conceptual Model 5](#_Toc57641566)

[Use Case Model 5](#_Toc57641567)

[Scope 6](#_Toc57641568)

[Load & Save State 6](#_Toc57641569)

[Specify Categories 6](#_Toc57641570)

[Manage Media Files 6](#_Toc57641571)

[Make Playlists 7](#_Toc57641572)

[Requirements 7](#_Toc57641573)

[Functional 7](#_Toc57641574)

[Non-Functional 7](#_Toc57641575)

[Requirements Assumptions 7](#_Toc57641576)

[Play / pause 7](#_Toc57641577)

[Media progress 7](#_Toc57641578)

[Recursive scope scanning 7](#_Toc57641579)

[Displaying images 8](#_Toc57641580)

[Unsupported file types 8](#_Toc57641581)

[Metadata 8](#_Toc57641582)

[Design 8](#_Toc57641583)

[Saving Data 8](#_Toc57641584)

[Initial User Interface Design 9](#_Toc57641585)

[Context Menus 10](#_Toc57641586)

[Input Box UI 10](#_Toc57641587)

[How Will the Application Work? 11](#_Toc57641588)

[Construction 11](#_Toc57641589)

[Context Menus 11](#_Toc57641590)

[Debugging & Testing 16](#_Toc57641591)

[Project Limitations 17](#_Toc57641592)

[Playing Audio 17](#_Toc57641593)

[Time 17](#_Toc57641594)

[Selected File, Directory, and Playlist 17](#_Toc57641595)

[Sorting Directories and Playlists 18](#_Toc57641596)

[Improvements 18](#_Toc57641597)

[Mute Button 18](#_Toc57641598)

[Playlist Sorting 18](#_Toc57641599)

[Visual Indicator for Selected Items 18](#_Toc57641600)

[Audio Graphics 18](#_Toc57641601)

[Application Icon 18](#_Toc57641602)

[Persistent Volume 18](#_Toc57641603)

[Project Conclusion 18](#_Toc57641604)

# Project Plan

The project will be completed over 5 days (8 hours of work per day).

## Design (1 Day)

### Documentation

* Summary
* Overview & Objectives
* Background Information
* Conceptual Model
* Use Case
* Requirements
* Design

### User Interface

* Design wireframe of main window
* Design wireframe of context menus
* Design wireframe of input box

## Construction & Testing (3 Days)

### Visual Studio

* Create main XAML project
* Create test project for testing main XAML project (using NUnit)

### Version Control

* Initialise project root with git and set remote location
* Create readme.md containing basic project information
* Create gitignore file to exclude unnecessary files
* Make initial commit and push to repo

### Programming

* Create XAML document for main window and corresponding C# file for the window
* Implement all UI features in XAML
* Implement all call-backs required by XAML

### Coding Practices

* Ensure I am following Gamma programming standards and leaving comments in my code where necessary.
* Ensure program is logically structured

### Testing

* Create testing document with test plan for unit tests and manual tests (in an Excel spreadsheet)
* Create unit tests based off testing document and test plan
* Populate testing document with results from unit tests
* Perform manual tests and populate testing document with results
* If any tests fail during the testing process, start debugging and fix the issue before trying the test again until it succeeds.

### Documentation

* Update construction section of project documentation with important snippets of code and explanations of what the code does.
* Update testing section of project documentation with information about how testing was performed.

## Final Day (1 Day)

### Documentation

* Analyse limitations and any failures of the project.
* List possible improvements based off limitations and failures.
* Conclude and summarize project outcome.
* Create user guide

# Summary

I have been asked to create a media manager that displays media files and allows the organization and categorization of media in directories and playlists. I will be using the agile development methodology for making this project since it allows me to plan how I spend my time on the project and allows a high degree of flexibility.

# Overview & Objectives

You work for Whizzy Software, a software house specialising in serving the needs of clients in the media industry, such as TV and radio companies, music streaming services, etc. One of their clients’ needs a stand-alone component that supports the organisation of media files on a device such as a desktop computer, laptop, tablet, or smartphone. Your manager would like you to design, build, and test an initial version of this component – the “Media Organiser”.

## Critical Activities

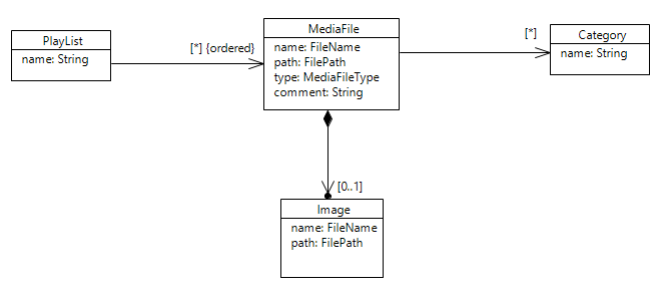
* Review all the key information and create a design for the media organiser;
* Construct the media organiser in accordance with the design;
* Test that the media organiser meets its requirements;
* Document what you built.

# Background Information

* Whizzy Software develops software components for clients in media industries: music, video, radio, TV etc.
* This project, for one of their clients, is to build a component intended to form part of larger systems which will need to be ported to other platforms.
* The component will be used as a fully functional prototype that the client can use to test requirements with their own customers.
* The component may run on any execution environment of your choice, on a desktop computer or a mobile computing device.
  + The execution environment should support a local filing system that may contain streamable media files (AAC, MP3, WAV, MP4, AVI, etc).
  + The component should launch as a stand-alone application in that environment.
  + The component does not need to access the network.
* The component should be designed and built to production standards.
* You are advised to get a basic version of the component working first, before adding richer features. Examples of richer features could include:
  + A choice of ways to browse the contents of the organiser.
  + The ability to sort Playlists according to various user-specified criteria.
  + Showing thumbnails of images in the user interface.
  + The ability to change the scope and search again during a session.

# Conceptual Model

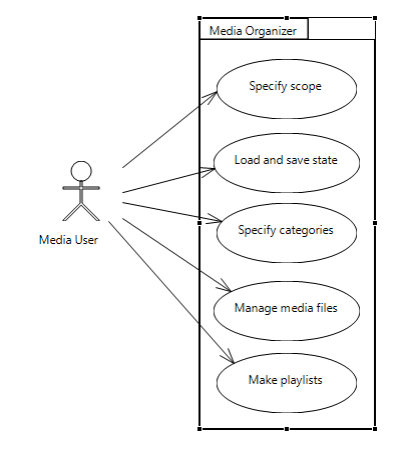
The UML model below shows the main concepts required by the application and their relationships. Data held in the memory of the organiser, and data loaded and saved by the organiser, should correspond to this conceptual model.



* Each MediaFile has a name which is a valid filename (such as “Moonlight Sonata”), a file path (such as “D:\Data\Music\”), a file type (such as “MP3”), and a comment.
* Each MediaFile may be associated with zero or more Categories, each of which is named by a string (such as “Classical”).
* Each MediaFile may be related to zero or one Image, where an Image has a filename and a file path.
* Each MediaFile may appear ordered in any number of Playlists

# Use Case Model

The UML use case model below shows the interactions that a user may have with the media organiser.



## Scope

* Specify file paths of interest, i.e. the locations in the filing system where the organiser will look for MediaFiles.
* Specify file types of interest, i.e. the types of streamable MediaFiles that the organiser will look for.
* Search for MediaFiles, i.e. within the specified locations, find all the Media Files of the specified types and create corresponding data structures within the organisers internal state.

## Load & Save State

* Save the internal state of the organiser (i.e. the set of MediaFile data, Annotations, Image data and Playlists) in a text file.
* Load the state of the organiser from a previously saved file.
* You will need to design the format of these files so that your program can save and load them easily. You may consider basing it on XML, JSON or any other format of your choice.

## Specify Categories

* Allow the user to specify a set of Categories which may be associated with MediaFiles.
* The user may add a Category, delete a Category, or rename an existing Category.

## Manage Media Files

* The user may select a MediaFile.
* The user may add, edit, or delete a comment about the selected MediaFile.
* The user may add or change an Image related to the selected MediaFile.
* The user may choose or change the Categories associated with the selected MediaFile.

## Make Playlists

* A Playlist refers to a sequence of MediaFiles chosen by the user.
* The user may create and delete Playlists, change their names, and reorganise their content.

# Requirements

### Functional

* Play streamable media files such as ACC, MP3, WAV, MP4, AVI, etc. The media manager therefore needs to be able to play both music and video. The media player could also display images; however, this is not required.
* Launch as standalone application
* Add multiple scopes (scan locations for media files)
* Create renamable playlists that media can be added to
* Reload scopes, directories, and playlists via context menu
* Add categories to media files
* Save the current session to the system when the application is closed
* Browse contents of organizer (playlists & scopes)
* Bind a single image to a media file by absolute path
* Volume slider
* Media progress slider for controlling the current position in the currently playing media
* Play/Pause button
* Media file name and comment text on UI
* Previous/next media buttons
* Sorting for playlists (e.g. alphabetical ascending)

### Non-Functional

* Easy to use user interface
* Application must be scalable to cope with lots of different scopes and playlists existing at one time

## Requirements Assumptions

When writing the requirements of the application, I created some myself since they are required for a functional media organizer.

## Play / pause

It is not mentioned that the organizer needs to be able to pause the current content. I assumed since it needs to be able to play the media, it should also be able to pause it.

## Media progress

I assumed the user might want to change the position in the media they are in with a progress slider and be able to see how far they are through it so I’ve added a progress slider to my design.

## Recursive scope scanning

The specification did not say scopes needed to be recursively scanned but I have added that as a feature.

## Displaying images

While it was not mentioned as a type of file to support displaying, since the media manager needed the ability to display images associated with media files anyway it made sense to also let it display images as media.

## Unsupported file types

The specification does not say if unsupported file types should be displayed in the media organizer or not. I have decided they should not be shown.

## Metadata

Nowhere was it mentioned that the organizer should use the metadata attached to the media files to get the name, artist, thumbnail, etc for the media. I have chosen not to use metadata as the specification says users should be able to assign an image to a media file, implying not to use the thumbnail from the metadata.

# Design

The media manager needs to be built to production standards, this includes the source code and the UI design. Therefore, the application needs to be easy to use and navigate and should work on different sized displays so it can later be ported to other devices.

I am targeting the Windows 10 (desktop) operating system for the project. I have decided to use XAML and C# to create the application since the language was designed by Microsoft and comes with plenty of built in features that directly interfaces with lots of Windows 10 operating system features such as the file explorer. XAML also has the capability to be used to create applications for a variety of different platforms including both mobile and desktop.

## Saving Data

The application needs to be able to save the current state. I have chosen to use an XML format to save the state of the organizer since the format is very useful for saving the states of objects and child objects. Since I am using an object-orientated approach it made the most sense to use XML. I considered another format, JSON, but decided against it since I need to be able to store arguments for each node which is something XML does better than JSON.

The root node will be called “MediaManager” (the name of the application). Then inside this node, everything about the current state of the application will be stored. Starting with the “Hierarchy” node. This node will contain “ExpandedItem” nodes which stores a GUID (**G**lobally **U**nique **Id**entifier) of each expanded tree view item. This is so when the application is closed and opened again, the tree view is expanded in the same way it was closed in.

The next child node of the root node is the “Scopes” node which stores “Scope” nodes that reference each scope the user has defined. It should store the absolute path of the scope as the value of the “Scope” node.

Next is the “Metadata” node. This node stores “Media” nodes which have a GUID argument that reference a media file. The GUID tells the application which scope the media file comes from and what media file in the scope it is referencing. The “Media” node then contains any meta-data the user has created associated with the described media file such as any comment or categories associated with the media file. These are stored in “Comment” and “Categories” nodes as a child of the “Media” node.

Finally, is the “Playlists” node which contains “Playlist” nodes. Each playlist has Name, SortMode, and Reverse arguments. The Name argument contains the name the user has specified for the playlist. The SortMode argument contains the sorting algorithm to use for the playlist contents. The Reverse argument is a Boolean that when true will reverse the output of the sorting algorithm used on the playlist. Inside the “Playlist” node are “Media” nodes what each have a GUID argument containing the GUID of the media file in the playlist.

## Initial User Interface Design



Above is the initial design for the user interface of the application. I always plan to have everything in one place yet also having minimal interfaces for users to interact with to keep the application simple to use.

I have split the application into 3 main parts. The top left area will have a tree view that allows the user to browse their scopes and playlists. This is where the user will manage both of those features. The user will be able to right click on any tree view item and be presented with options for the item in a context menu. For example, a user may right click on a playlist and be greeted with the option to rename the playlist.

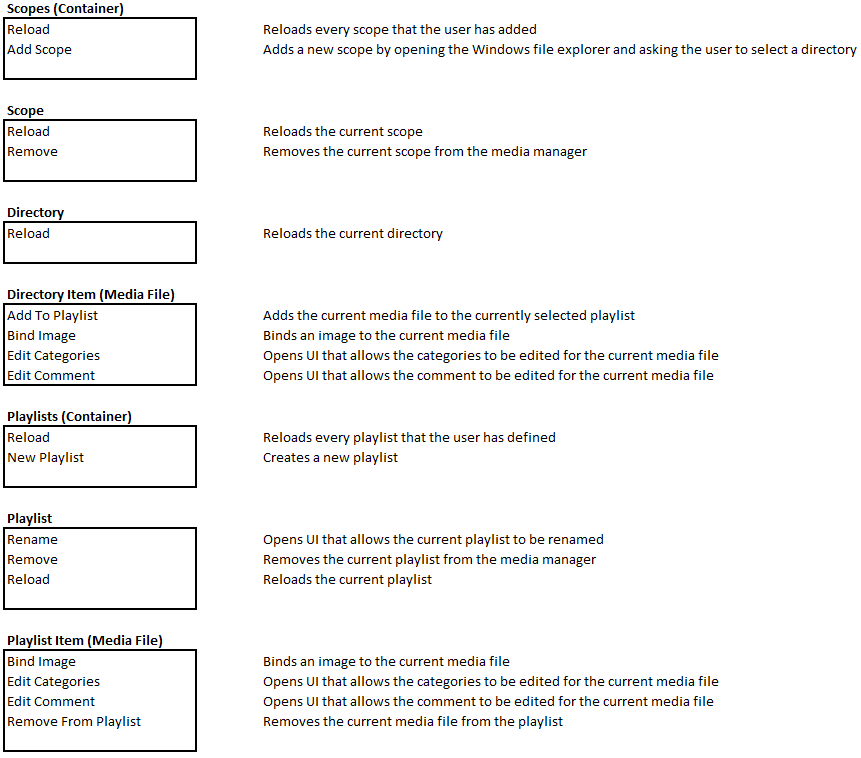
The next section is the control bar. This is the bar that spans the bottom of the application. This is where all the controls and information for the current media will live. Starting from the left, you have the name of the media file and the comment associated with the media. To the right of that you have the preview image associated with the media. This image must be linked to the media file by the user. I would like to get the preview image from the meta-data of the media file, but the requirements clearly state it must be linked where the image has a filename and a file path.

The next section of the control bar manages the media itself. The bar has a previous media button and a next media button. This lets the user move to the previous or next media in the current selected directory of playlist. This section is also home to a play/pause button. This button will look like a play button when no media is playing, or the current media is paused. If media is playing, the button will look like a pause button. I chose to combine these two functions (play and pause) into one button since they cannot be triggered at the same time and it reduces the total number of buttons available to the user at once (making the application easier to use). Finally, is a progress bar that lets the user skip to a certain time in the currently playing media. This bar will also have the current time on the left of the bar and the total duration of the media to the right of the bar.

The final section of the control bar includes volume controls. A simple volume slider that allows the user to set the volume of the media player on a linear scale (from 0.0 to 1.0). There also exists a toggle mute button that will allow the user to toggle if the media player is muted or not in one button.

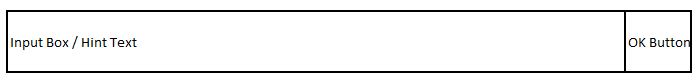
## Context Menus

For my media player to work with the current design, it requires context menus for each different type of object that can be displayed in the tree view. Context menus will allow me to keep everything on one single screen at the same time.



## Input Box UI

The context menus require a way to request input from the user. I designed a simple text box that the user can enter text into and press a button to submit the text to the application.



The Input Box / Hint Text Section of the box will contain a hint if there is no text in the box that tells the user what they should enter. For example, if you were adding a comment to a media file the hint might be Media File Comment. The user can click this area and start typing, when they have typed something, the hint text will disappear and be replaced by what they have typed. The OK Button is a button that when pressed submits the contents of the input box to the application.

## How Will the Application Work?

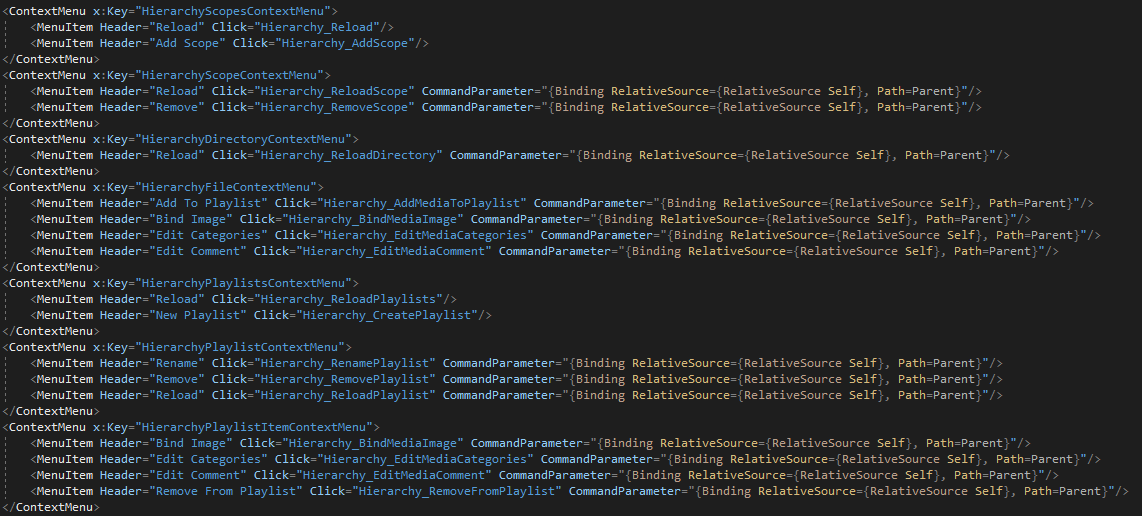
The application relies on the user selecting scopes that it can find media files in. These scopes will be added to a tree view where the user can navigate them. Each directory and media file can have a GUID associated with it. This is essentially a hash of the relative location of the media file (relative to the directory it is in) or of the absolute path of a directory. The important thing about the hash is that a hash value must be the same no matter what machine you are running the organizer on and must be the same every time you launch the application.

The GUID is the main way the application organizes and links things together. These GUIDs are used to add media to playlists. Playlists are also displayed in the directory tree. The application uses this to figure out how to structure the tree view and therefore how media should be organized.

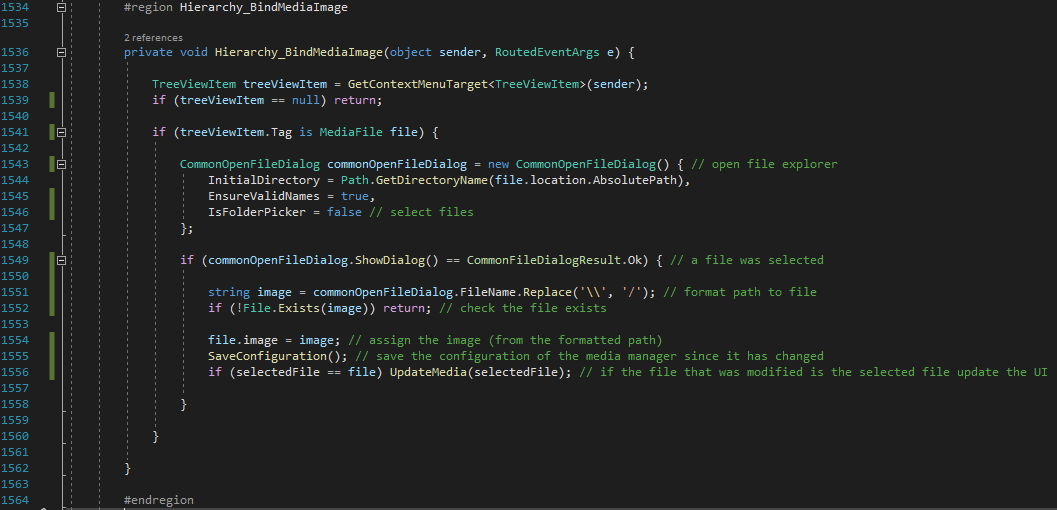
# Construction

I have included some snippets in this section of some important code in the application.

## Context Menus

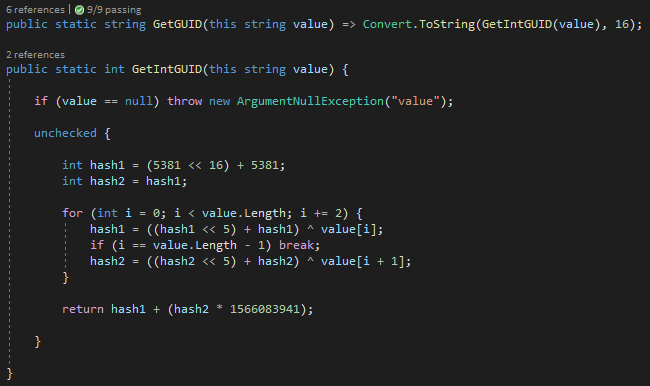


The context menus for my application are one of the most important features. They are where most of the management will take place. Above are the XAML implementations of each of the context menus. Each item has a corresponding call-back in the C# file associated with the main window.



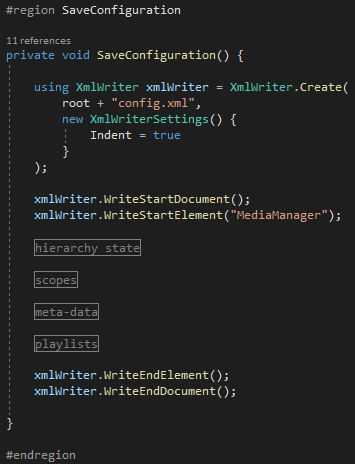
The above snippet of code is responsible for binding an image to a media file. It does the by first getting the TreeViewItem from the context menu (which is the sender of the event). Next it checks if the tag associated with the TreeViewItem is a MediaFile. If it is the application will open the Windows file explorer and the user can select an image file. Then if a file was selected the path to the file will be formatted and the file will be checked that is exists on the system. Finally, the image path will be set, the configuration of the media manager will be saved, and the UI will be updated if necessary.

I chose this snippet as an example since it has an issue. The user can select any file (including a non-image file) with this approach. In the future I would add two ways to prevent this from happening. Firstly, I would give the Windows file explorer a list of supported extensions to allow the user to pick from. Secondly, I would verify the selected file has one of the supported extensions when I get the path back from the Windows explorer as a final check.

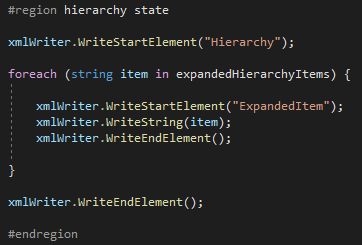


Above is a very important set of methods in my code. These methods convert a string into a GUID by creating a 4-byte hash from the string. The GetIntGUID method takes in a string value and first checks it is not null. Then I have created an unchecked block because I do not want exceptions to be thrown when an integer is overflown since this is vital to the hashing algorithm.

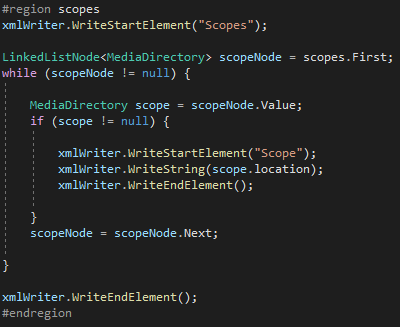
The GUID system is vital to the application since it needs to index files and directories in the configuration file. This is the most efficient way to do that since the hash takes up less space on the system than the full path so is a more effective way of indexing.



Above is the SaveConfiguration method. This is the method responsible for saving the state of the media manager to the system. It saves into an XML format. I have used regions to split up this method into smaller regions of code that are easier to manage when separated.



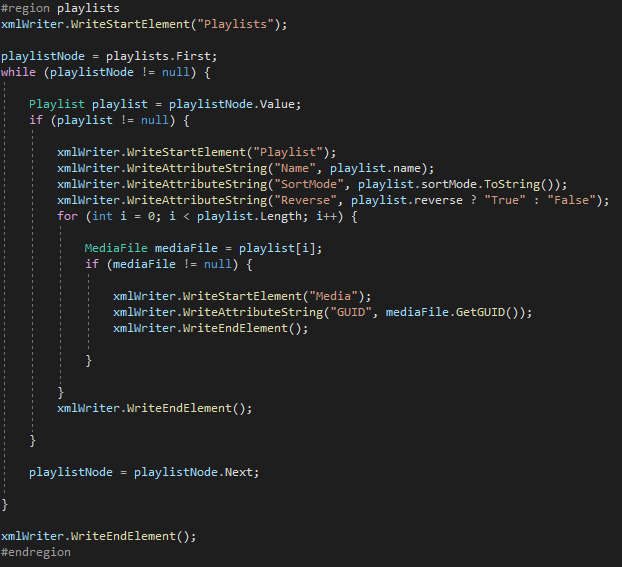
The first thing added to the XML document is the hierarchy state. This stores the GUID of all expanded elements in the hierarchy so when the program is next opened, it looks the same as when it was closed.



Next is the scopes, which are the user defined locations where media files will be searched for. These are stored as absolute paths so if the application is moved to a different part of the system it will still reference the correct scope. This would not happen if you used a relative scope since it would be relative to the location of the application.



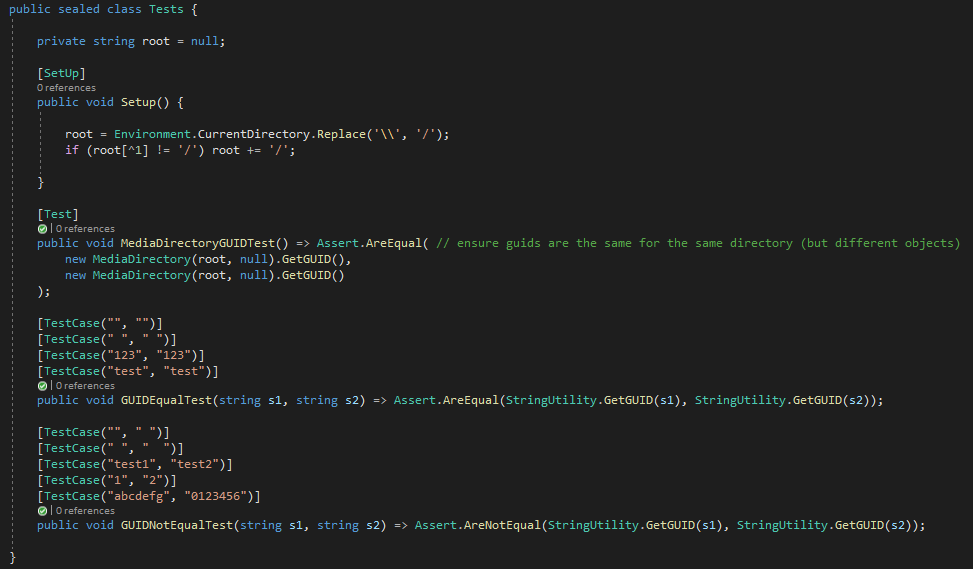
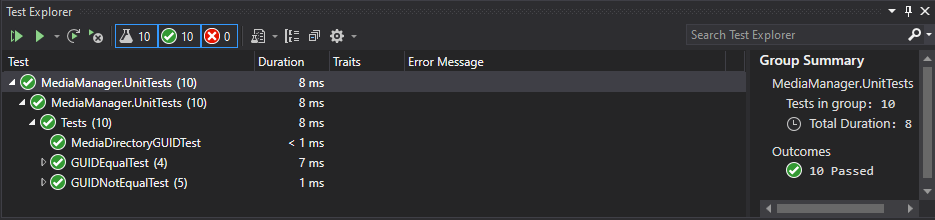
Next you have the meta-data defined by the user for each of the media files. This is where the comment, categories, and image associated with the media file is stored in a media node. The media node uses a GUID to reference the media file this applies to.



Finally, the playlists section. Playlists have a few arguments such as their name, sort mode, and if the output is reversed or not. Playlists contain media nodes which reference the media in the playlist once again by GUID.

# Debugging & Testing

While writing the code for the project I made sure to test as I was writing. I used visual studio to create the application. I used the *System.Diagnostics.Trace.WriteLine* to log information to the application output console while its running to make it easier to see when something has gone wrong with the application. Once I have identified a bug, I will write a unit test to test that part of the application for the expected behaviour and then work on fixing the bug.

Above are a few unit tests I wrote to test the GUIDs were behaving as expected. I’ve also constructed some manual tests which I detail in a different document.

# Project Limitations

## Playing Audio

When the application is playing audio, there is just a black area of wasted space where video would be displayed. I am using MediaElement to play all the media which does not support the ability to show an image while playing audio. In the future I would like to add graphic there that makes it clear audio is being played. This could be done by having some other XAML elements that are collapsed unless audio is playing and then they will make up the graphics.

TimeI was limited on time for the project so there were many features I wanted to add but simply could not in the 3 days I had to construct the application. Time also had an impact on the design since I designed the application to display everything at once, so I did not have to design multiple different sections of UI. Each of those sections would also have C# and XAML code associated with them which would take much more time to develop.

## Selected File, Directory, and Playlist

In the current version of the application there is no way to know what the currently selected directory, file, or playlist are unless they were the last thing you selected. In the future I would like to add highlighting to indicate which elements are selected.

## Sorting Directories and Playlists

I did not have enough time to implement the ability to sort directories and playlists to the application. Some of the logic exists but it was never implemented with the XAML.

# Improvements

## Mute Button

I added a mute toggle to the design of the window but did not ever implement it since it was not a required feature. It would be a quality of life improvement since it allows the user to quickly mute all audio instead of using the slider.

## Playlist Sorting

I wrote a good chunk of the logic for this and it was a feature I planned to implement into the project, but I ran out of time. I would have added it but designing the XAML for it would have taken too long so I made the decision not to include it in the application.

## Visual Indicator for Selected Items

The user can select a file, directory, and playlist. I want to add an indicator for each of those to show the currently selected one of each. This would be purely C# and would likely involve changing the font colour of the tree view item to a brighter colour for selected items.

## Audio Graphics

I want to add graphics for when audio files are being played so its clearer something is being played.

## Application Icon

Majority of applications have their own icon but my one does not. This helps identify the application in your list of programs or taskbar and makes it more user-friendly to use.

## Persistent Volume

The volume that the media play is at is not saved in the configuration file. It would be nice if the volume you left the player at was saved when you closed it.

# Project Conclusion

I had fun making this project and to a large extent it was a success. I added every required feature (required by user requirements) and added majority of the additional features I wanted to add to the software that were not explicitly defined in the user requirements. There are some things I would change about the software such as how selected tree view items are displayed. Currently you can select an item by left clicking it but there are no visual indications to which items are selected. This can be annoying when adding items to a playlist since you may not know which playlist you have selected.