Title	ReMp3 – The Remote MP3 Player

Document	Final Report

Author	Sacha Barber [sb54]

CSAI Final Year Project : Final Report	
Author:	Sacha Barber
Issue Date:	20/04/06

STATEMENT OF ORIGINALITY

This report is submitted as part requirement for the degree of Computer Science and Artificial Intelligence at the University of Sussex. It is the product of my own labour except where indicated in the text. The report may be freely copied and distributed provided the source is acknowledged.

Sacha Barber
Thursday, 20 April 2006

ACKNOWLEDGEMENTS

I would like to specifically thank the following people :

Paul Newbury (Sussex University) : For being a brilliant project supervisor, that has provided constant support and reassurance throughout the entire project.

Phil Whatton (Sussex University) : For the assistance with some of the more exotic .NET programming issues that have arisen. These issues, and meetings are included within the project Log.

Nick Cross : Who is a friend that is employed as a C++ programmer, that helped me to understand the Windows NetAPI documentation.

Idael Cardoso : Who is someone I met through a .NET chat group, who is my opinion, is some what of a genius developer, who provided the CD Ripper and MP3 Converter libraries.

Foood's icons [1], which are free icons for non-commercial usage.

All the knowledgeable contributors : At the codeproject [2], where I found some of the free libraries that the ReMP3 project makes use of. The usage of these libraries and the individual acknowledgements, is included both within the individual class comments, and also later within this report, at section 7.

SUMMARY

In days of old many people would have amassed vast collections of vinyl and tape music, which are all analogue based formats. With the invention of digital storage came the ability to store music digitally.

Further development of computer networking and a whore series of peer-to-peer (P2P) networks, now allow users to download and to share their music collections digitally.

Digital format music collections, are effectively files that may be stored on computer hard drives, and may even be transferred to portable media players like Apple's iPod, or a plethora of other digital media players.

There are currently many software based media players available within the market place, offering various features such as

- Peer to Peer music player and search software
- Web streamed media
- Stand alone media players

On examining all of the available media players and software, it has become clear that there seemed to be a gap in the market. There did not seem to be any music software that offered remote control facilities, where one computer can arrange a list of music to be listened to, and have another computer (that may be attached to the main stereo) play the music.

This was the basis for the ReMP3 project, which this report describes.

The aim at the outset was to produce a free tool useful for users whom wish to control the audio playback of a master media computer using a remote control application, that was easy and intuitive to use.

Overall, the project has been successful, and all of the primary requirements have been met, as well as most of the extended requirements. There have been a few limitations within the chosen language (C# .NET), which have forced certain decisions, which are all discussed in the subsequent sections.

The GUI has a very usable feel, which has been achieved by applying sound HCI principles to the design of the application.

On a more personal level, two of the main goals of this project were to become familiar with the .NET Remoting features, and to create a GUI that looked as good as some of the better designed GUIs, such as Microsoft Windows XP, where there has been considerable time and expense spent to make the system look and act in a reasonable manner. On this level, the project has been a tremendous success.

TABLE OF CONTENTS

TABLE OF FIGURES

1. INTRODUCTION

In the past people would have accumulated music on various formats such as tape, vinyl. All of these formats were analogue based, these days the focus has shifted from analogue music formats to digital formats. Where music may be downloaded using a series of peer-to-peer (P2P) networks.

What has also become common place, is home networking. Which in turn allows users to share portions of certain hard drives, or complete hard drives. It is clear that these shared resources could hold digital music collections, which could effectively be played across a network.

Digital format music collections, are effectively files that may be stored on computer hard drives, and may even be transferred to portable media players like Apple's iPod, or a plethora of other digital media players.

There are currently many software based media players available within the market place, offering various features such as

- Peer to Peer music player and search software
- Web streamed media
- Stand alone media players

On examining all of the available media players and software, it has become clear that there seems to be a gap in the market. At present there does not seem to be any music software that offers remote control facilities, where one computer can arrange a list of music to be listened to, and have another computer (that may be attached to the main stereo) play the music.

This project will attempt to provide a solution to this problem area.

1.1 Aims and Objectives

The project consists of 2 separate applications, comprising a ReMP3 (Server application), and a ReMP3 (Client application).

The server application will be used to play and organise MP3 files, where as the client application can be thought of as a remote play list provider. The idea being that the server is a fully contained Mp3 player and organizer, but upon request, can also play remote play lists that have been provided by the client application.

The client application will make use of distributed computing, such that the client can be located on a completely different machine to the server application.

The main motivation behind wanting to create this application, is the author's interest in the distributed computing elements that this project has to offer, a side benefit of this project will be, that the technology behind the client application would be easily transportable to a .NET web service (WSDL), or even an ASP .NET (ASPX) web page, which in turn make it possible to control the re- ReMP3 server application on a global scale.

However, for this project, a standard GUI (Windows.Forms) environment will be provided for both the ReMP3 Server and Client applications. This approach allows the control of a main Multi-Media centre, that is connected to a main stereo (ReMP3 Server), to be controlled remotely by a ReMP3 client.

1.2 Methodology

The program was designed and implemented using sound object-oriented software engineering methods. The general process was iterative, in the sense that previous stages were revisited, in the light of new insights found during a subsequent stage, and user evaluations. UML is used where appropriate, to clarify and specify the workings of the system. For the user interface, HCI principles were followed as much as possible, and user interview questionnaires were completed by selected users, prior to any code implementation.

As an aid in documenting the progress of the project, a devoted web site has been developed at http://www.vibrant-web-design.com/sacha/sb54_CSAI_Proj.htm. All project documentation is available on the site, and a log of work done has been kept.

1.3 Constraints

As with all projects of this type, the major constraint is time. There is only a limited amount of time to finish the program and therefore, only a limited amount of functionality can be implemented. This is something that the author has tried to take into account, by dividing the requirements in the specification in Section 4, into core features and extended features. The core features are to be seen as the minimal specification, whereas extended features are extras that will be implemented if time permits.

2. PROFESSIONAL CONSIDERATIONS

The ethical standards governing the computing profession in Britain are defined by the Code of Conduct and Code of Practice published by the British Computer Society [3]. The application of these two standards to this project is discussed in the following sections.

2.1 BCS : CODE OF CONDUCT

Defines the professional standards required by the BCS. It applies to all of its members including students, and affiliates. Currently I am not a member of the BCS, and as my degree BSc in Computer Science and Artificial Intelligence, is not recognized by the BCS as a qualification for membership, it is unlikely that I will obtain BCS membership on graduation. However the code of code of conduct is still relevant to all software projects.

The code of conduct is concerned with personal conduct of BCS members and is split down into four areas :

2.1.1 Public Interest

The main point of relevance to this project in this section is the requirement to carry out work or study in accordance with the relevant authority's requirements, the relevant authority in this case being the University of Sussex whose requirements are defined in the Handbook for Undergraduate Candidates [35]

The remainder of this section specifies how a BCS member should behave, and highlights legal areas, that the individual member has particular responsibility for, such as :

- Public health, safety and environment.
- Protect legitimate rights of third parties.
- Knowledge and understanding of relevant legislation such as the – Public Disclosure Act, Data Protection, Computer Misuse law, Legislation.
- Non discrimination against clients or colleagues on grounds of race, colour, ethnic origin, sexual orientation.
- Reject any offers of bribery or inducement.

It is envisaged that none of the above will be breached by the ReMP3 project.

2.1.2 Duty to Relevant Authority

This section details conflict of interest between BCS member and the relevant authority – the university, disclosure of confidential information and misrepresentation or withholding information on product performance.

It is envisaged that there be no conflicting interests between the student and the university. This project does not require any access to confidential material, as such this point is not

relevant, but is noted. The 3rd point does have direct relevance to this project, as the projects overall performance shall be tested. The testing will adopt an open policy such that any failures or successful testing shall be included.

One issue that should be noted is that all music required for use with the ReMP3 project should be legally purchased music. The ReMP3 project is not a peer-to-peer system, and such does not allow violate any copyrights that may be imposed by record companies or artist on music items.

Also where code is being used by a third party, Intellectual property rights have to be considered. To this end, any time a third party section of code is used the code contains references to the original author and the original source location. The original author is also credited within the body of this report.

2.1.3 Duty to The Profession

Is aimed at BCS members and addresses how those members should uphold the reputation and good standing of the BCS, and also encourage fellow members in their professional dealings and encourages mutual assistance between professionals.

As previously stated I am not a BCS member, although I do believe in sharing knowledge, so can see the direct benefit of this point. This however does raise an interesting point of collusion, as such for the duration of this project, this section of the code of conduct has been deemed to be inappropriate, and shall not be followed.

2.1.4 Professional Competence and Integrity

This section specifies that you should seek to upgrade your professional knowledge and skills, keeping up to date with new developments, and that one should work within ones own limits, and that the member be familiar with technology involved.

This is of particular interest to this project, as the chosen technology that this project uses, namely C# .NET, is not something that has been taught as part of the syllabus during my degree. As such this choice for language could be considered unfamiliar. However, I have had a lot of prior experience with C# .NET, so expect this to be within my own technical limitations.

2.2 BCS : CODE OF PRACTICE

This document describes standards of practice relating to the information technology (IT) industry, for members within the IT area, it is not compulsory and may be used in parallel with the Code of Conduct [3].

The document is designed as a reference guide that may have some relevancy to certain areas of a project. For this project the document will be referred to for guidance on :

- Project Planning
- Requirements Specification
- Software Development

3. DOMAIN AND REQUIREMENTS ANALYSIS

The first phase of the project involved carrying out research into the project ideas, and tools needed to carry out the project function points. The outcome of the initial research is presented in this section.

3.1 Existing Solutions

There are a number of Media players out there that will manage collections of digital music, such as :

- Shareaza [31] : A P2P download application , that also organizes digital music collections using Genre, Artist, Album meta data.
- Microsoft Windows Media Player [7] : Which is a Windows application that allows users to create play lists, rip music, and organize their digital music collections using Genre, Artist, Album meta data.
- Nullsoft Winamp [32] : Which is a digital music player, that allows users to create play lists, rip music, and organize their digital music collections using Genre, Artist, Album meta data.

When the ReMP3 project proposal was first written, there did not seem to be any solutions out in the market place, that offered remote control capabilities. It is only now several months on, that there seems to be one similar project.

The existing software is entitled “Remote Amp” by Terminal Zero [4].

The Remote Amp is intended to work with WinAmp and iTunes. Upon examining the list of features that Remote Amp claims to provide, cross referenced with the main function points of the ReMP3 solution, it can be seen that about 60% are the same. Table 3-1 RemoteAmp vs ReMP3 overleaf demonstrates this.

Finding the Remote Amp product, has assured me in the following areas

- There is obviously a need for products which are similar to the ReMP3 project.
- There were certainly no such projects in existence before Remote Amp, which shows that the ReMP3 project idea was an original idea.
- Terminal Zero seems to have published many different software products. So it is pleasing that the ReMP3 application, offers similar services to a company that has obviously been around a while. Similarly Terminal Zero, must be a collective of people that have spent some time, creating the Remote Amp product. The ReMP3 project was created by a single individual, so it was good to note that the ReMP3 project idea was a sound and worthy one.

To compare the features that Remote Amp and the ReMP3 project have, consider the results shown in Table 3-1 RemoteAmp vs ReMP3 below.

Feature Name	Description	Remote Amp	ReMP3
Support	Supported Media Players	WinAmp iTunes	ReMP3
Dynamic Drag and Drop Playlist Editing	Allow drag and drop to file system to form play lists	ü	ü
Smart media library management	Allow media library to be viewed by directory or by Artist, Album Genre	ü	ü
Searching	Allow search through media items	ü	
EQ control	Complete EQ	ü	
Free upgrades for registered users	Free upgrades for registered users	ü	
Common controls	Fast forward	ü	ü
	Skip Back	ü	ü
	Stop	ü	ü
	Clear	ü	ü
Audio extraction	CD ripping		ü
Audio Encoding	WAV – MP3 compression		ü
Meta data editing	ID3 tag editing		ü

Table 3-1 RemoteAmp vs ReMP3

3.2 The ReMP3 Project Pre- Requisites

3.2.1 Programming Language Chosen

When the ReMP3 project was first conceived, a research phase was carried out to establish the following :

- What solutions exist already
- What certain languages had to offer
- What 3rd party libraries were available, and which language they were available for

It became fairly obvious during this initial research phase, that there seemed to be many more libraries written for the C# .NET language, when compared with the likes of Java [33] and Visual Basic [34]. The existing solutions provided by Microsoft actually make use of known API's, and were also regarded as being stable technologies, that could be trusted.

I have also used C# .NET prior to commencing this project, so was fairly happy with the language capabilities, to this end the C# .NET language, is the chosen language for the ReMP3 project.

3.2.2 Supported Operating Systems

Table 3-2 shows a compatibility matrix of all the different Operating Systems that Microsoft have released since Windows 98, also shown are the components that the ReMP3 project is using.

	Direct X 9.0c	Media Player 10	Media Player 10 SDK	.NET Framework 1.1
	See note 1	See note 2	See note 3	See note 4
Windows 98	ü			
Windows 98 Second Edition	ü			
Windows ME	ü			
Windows 2000	ü			ü
Windows 2000 Advanced Server	ü			ü
Windows 2000 Professional Edition	ü			ü
Windows 2000 Server	ü			ü
Windows 2000 Service Pack 2	ü			ü
Windows 2000 Service Pack 3	ü			ü
Windows 2000 Service Pack 4	ü			ü
Windows XP	ü	ü	ü	ü
Windows XP Home Edition	ü	ü	ü	ü
Windows XP Media Center Edition	ü	ü	ü	ü
Windows XP Professional Edition	ü	ü	ü	ü
Windows XP Service Pack 1	ü	ü	ü	ü
Windows XP Service Pack 2	ü	ü	ü	ü
Windows Server 2003	ü			ü

Table 3-2 Supported Operating Systems

Table 3-2 shown above has been designed using information shown in Table 3-3 below.

Note1	Microsoft : DirectX 9.0c End-User Runtime [5]
Note2	Microsoft : Windows Media SDK Components [6]
Note3	Microsoft : Windows Media Player 10 [7]
Note4	Microsoft : Get the .NET Framework 1.1 [8]

Table 3-3 Compatibility Matrix

As shown from the Table 3-2 above Windows XP is the only operating system that supports all the software that the ReMP3 project requires, as such Windows XP will be the operating system required for the ReMP3 project.

3.2.3 Server Application

The server will require the following software to be installed and correctly configured

3.2.3.1 Microsoft .NET v1.1 Framework

The Microsoft .NET framework [9], is a collection of classes and objects that may be used for application development. Indeed this project will make use of a large number of these objects and classes. The .NET framework has gone through several different revisions, at the time of writing this project v1.1 of the framework was the latest version available. The .NET framework is equivalent to the Java v1.x SDK. The framework simply allows software developers to re-use common classes and objects, and where applicable to inherit and extend these classes to be more problem specific.

3.2.3.2 Microsoft Access

The media catalogue of the ReMP3 project will need to make use of MP3 ID3 tag information in order to store media data. This data will be stored within a centralized database, from which it can be retrieved to populate media tree controls on the main GUI.

In order to accomplish the storing of information a choice had to be made as to which database system would be best suited.

There were several different options available, which were as follows

- **MySQL [10]**, which is free, but not a stand alone, database that can be distributed with the ReMP3 application. MySQL is really marketed as a free alternative for software like Microsoft SQL Server and Oracle, where all clients make requests to a central database system. The ReMP3 Project needs a single standalone database per application install, to store users data, as such MySQL is not suitable for the task.

  So this really only left 2 options.

- **OPTION 1 FireBird Embedded Database [11]**, Which is an embedded database that can be embedded within an application, and also provides what appears to be a reasonable .NET Data Provider API, which is very similar to the native .NET Data Provider API. There are however several problems here, in the fact that some of the SQL syntax is non standard, namely auto number generated primary keys, which are constructed as database generators (across the entire database). These generators do not adhere to the ANSI (92/99) SQL standards [30], which is something that was of concern. The SQL used MUST be standard, in case the underlying database is swapped for another different database in the future. So this problem prohibited the use of the FireBird Embedded Database option.

- **OPTION 2 Microsoft Access**, Which is really made up of 2 different parts. The development environment, which costs money, and the underlying database JET engine which is free to use.

Providing .NET can communicate with the database JET engine, the development environment is not required by clients of the ReMP3 application. The .NET Framework provides all the required classes and objects to communicate with an Access database, so this was not a problem.

The database is still a standard Microsoft Access file, but the user will not be able to edit the database unless they already have a full copy of Microsoft Access installed on the client computer. In any case, the distributed Access database file, is password protected, to prevent it from being opened by client users.

The ReMP3 project makes use of Microsoft Access, for the following reasons

- It is something, that the author is more familiar with
- It carries out all the requirements that the ReMP3 project has
- The SQL Microsoft Access uses is standard (ANSI 92/99) SQL
- Several connections can be made to the same database at the same time, which is a fundamental requirement of my project.
- The user of the ReMP3 application does not need to have Microsoft Access installed

Access v10.0 was used, which has a COM (Component Object Model) Object model 10.0

3.2.3.3 Microsoft Media Player SDK

The ReMP3 project makes use of multimedia objects, music to be precise. As such, a suitable software development kit (SDK) had to be found, that could be used with .NET, that would also allow the use of media objects, MP3's in particular.

There were several different options available, which were as follows

- **Microsoft Direct X SDK [12]**, which is free. However when considering the quite basic audio requirements of the ReMP3 project, seemed to be a bit too advanced. Direct X is more concerned with 3D graphics and advanced video / audio handling.

- **Microsoft Media Player SDK [13]**, which is free, and has all the classes and objects that were needed to achieve the ReMP3 project requirements. This is the choice that was made for the ReMP3 project.

The Microsoft Media Player SDK is a collection of classes and objects that may be used to develop media applications. The ReMP3 project is a media type project, as such it has made use of some of the functionality provided by the Media Player SDK.

The Microsoft Media Player SDK has gone through several different revisions, at the time of writing this project v10 of the SDK was the latest version available, as such v10 is the one used by the ReMP3 project.

It should be noted that the media player SDK relies on the Windows Media player [14] actually being installed.

3.2.4 Client Application

The client requires the following software to be installed and correctly configured

3.2.4.1 Microsoft .NET v1.1 Framework

The client also requires the .NET v1.1 framework, see section 3.2.3.1 for details of how to obtain and install this framework.

3.2.5 Additional Code Libraries

To carry out some of the more exotic function points (CD Ripping, MP3 Encoding), there was a need to make use of several add on code libraries that may be used by the .NET framework. These libraries are free to use and provide additional capabilities to the .NET programming API. They are effectively additional classes and interfaces, that when installed may be used within user .NET code.

Where 3rd party libraries have been used within the ReMP3 classes, they shall be acknowledged within section 7 of this document.

4. REQUIREMENTS SPECIFICATION

4.1 Functional Requirements

The following list of requirements was produced by careful consideration of the intended purpose and scope of the software. This section is divided into core requirements that should be implemented, and extended requirements that will only be implemented if time permits.

4.1.1 Core Features

The server features will be as follows :

100 Provide a categorized media library (based on database stored MP3 data)
100.1 Using ArtistName
100.2 Using AlbumName
100.3 Using Genre
100.4 Using Directory Structure
100.5 Allow customisation of catalogue track visualisation
100.6 Allow more details to be shown for catalogue information
101 Provide a media player
101.1 Provide a play list to use with the media player
101.2 Support drag and drop into the play list from the PC's file system
101.3 Allow the media player appearance to be changed by user
102 Provide an MP3 file ID3 tag editor, that supports saving and loading
103 Allow the client application to create a new play list and trigger the playing of the remotely compiled play list, this will be based on XML files and .NET Remoting
104 Provide easy to use navigation
105 Provide access to help for users

The client features will be as follows :

200 Provide a categorized media library (based on database stored MP3 data)
200.1 Using ArtistName
200.2 Using AlbumName
200.3 Using Genre
200.4 Using Directory Structure
200.5 Allow customisation of catalogue track visualisation
200.6 Allow more details to be shown for catalogue information
201 Provide a remote control to control the ReMP3 server media player
201.1 Provide a play list to use with the ReMP3 server media player
201.2 Show a remotely compiled play list that represents the play list selections that the user made
201.3 Allow the remote play list to be sent to the server for playing
201.4 Allow the control over common tasks such as FWD 1 TRACK, STOP, REVERSE 1 TRACK which will be sent to the ReMP3 server application
202 Provide easy to use navigation
203 Provide access to help for users

4.1.2 Extended Features

Possible Server features may be as follows :

300 Provide a CD file ripper
301 Support recording into WAV format from soundcard
301.1 Provide a list of sound devices, such that one can be selected for recording
301.2 Provide a list of different quality levels to use for recording
302 Conversion of WAV to MP3
303 CDDB audio data information gathering
304 Provide a standard web based ReMP3 client, that will simply allow a new track list to be created for the ReMP3 server from a web browser
305 Provide the facility to save and recall track lists within the ReMP3 server/client
306 Provide some sort of intelligent agent that knows what music has been the most played music, and will display these albums to users, as possible future track lists

4.2 Data Requirements

Table 4-1 below shows, the minimum pieces of data about each song that is stored in the database. The mp3 ID3 tag will be queried to provide this logged data. So only the data that is available will be logged, if a mp3 does not have an ID3 tag, it will not be logged.

Description	Domain Type
Folder Name	String
Folder Is Root	Boolean
File Name	String
Artist Name	String
Album Name	String
Year	Date
Comment	String
Genre	String

Table 4-1 Data Requirements

This data will be spread across several different database tables. This is discussed in more detail in section 7, Media Library.

4.3 Operational Requirements

- Run on Windows XP
- Run in a minimum resolution of 1024 x 768, which is a fairly common resolution setting, that most monitors are capable of achieving. It also looks less gimmicky than some lower screen resolutions

4.4 Usability Requirements

The project shall:

- Provide contextual help in the form of tool tips and compiled HTML files
- Use consistent terminology throughout
- Provide consistent methods of interaction and navigation

4.5 Look-And-Feel Requirements

The project shall:

- Look modern and sleek, In fact, the ReMP3 project shall attempt to match some of the better looking GUI's available, such as Microsoft Windows XP, where there has been considerable time and expense spent to make the system look and act in a reasonable manner.
- Be aesthetically pleasing and enjoyable to use
- Have a native look-and-feel
- Apply the same theme as that currently in use by Windows XP, Desktop Theme. This is to say the ReMP3 application WILL fully support bespoke themes and skins that are supported by the Windows XP uxtheme.dll. There are however **NO** guarantees made for skin/theme designs that do not provide the correct details. This is down to the skin/theme designer

4.6 User Evaluations

When the ReMP3 project was first conceived, some HCI principles were applied to produce an initial prototype of the system. This prototype was a lo-fidelity paper prototype that was shown to a selection of users, along with some scenarios of use for the prototype. The selected users were also given questionnaires that they filled out in response to the tasks of carrying out the scenarios described.

After this initial evaluation stage the results of the questionnaires was analysed and the resulting changes were examined. If the changes where acceptable and not too time consuming, and actually contributed real benefits to the system, they were incorporated into the end system.

Appendix A of the previously submitted Interim Report, contains a GUI Prototype document that states several scenarios of use that were used to gather user feedback regarding the initial GUI design.

5. SYSTEM OVERVIEW

The system is made up of 2 applications ReMP3 Client and ReMP3 Server. These 2 applications are intended to run on separate computers and must have access to each other using a TCP/IP network.

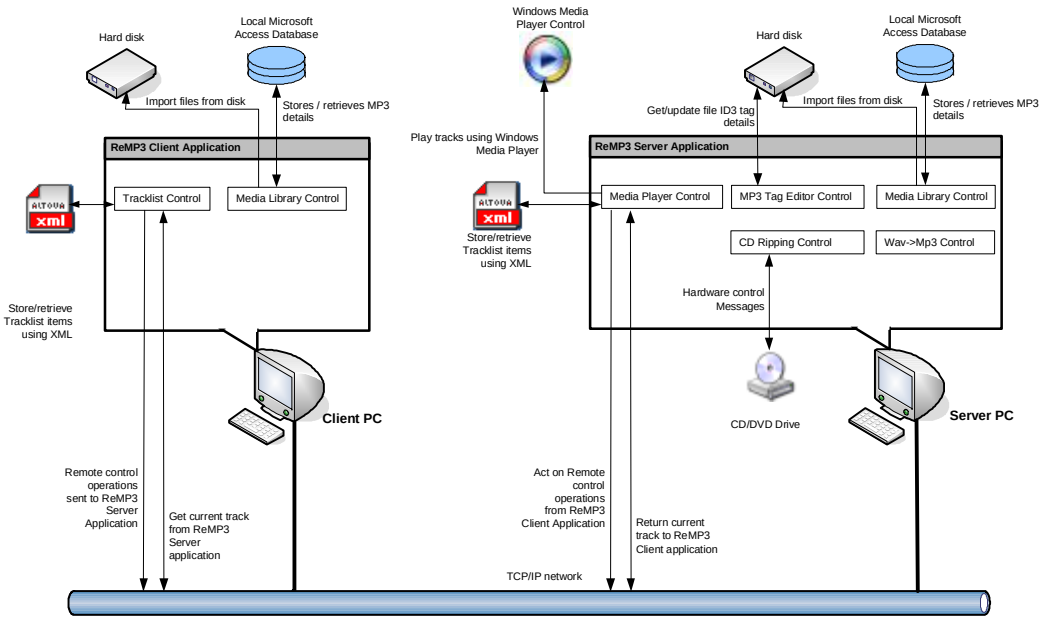

Figure 5-1 System Overview

As shown in Figure 5-1 above, in order for both the ReMP3 Client and ReMP3 Server applications to run correctly, there is a lot of interaction required to other system objects such as :

- Microsoft Access Database : For storing / retrieving logged music data
- FileSystem calls : For gathering media item meta data
- XML : For storing tracklist information
- Networking : To enable the use of remote control operations, and current track retrieval
- CD Hardware : To communicate directly with the CD/DVD hardware, to open close the bay, and also to retrieve a table of contents
- Windows Media Player Control : To enable the playback of media items

6. SYSTEM ARCHITECTURE

The primary role of requirements analysis is to establish the following elements

- Key domain abstractions, Classes, Usually presented in the form of a class diagram
- Common tactical policies that shall outline how certain codes issues should be dealt with in a clear and consistent manner.

The rest of this section of the document outlines the points above in more detail.

6.1 High Level Structure

When dealing with GUI applications, the model-view-controller (MVC) design pattern could be considered.

The MVC pattern enables the division of the system into 3 distinct parts: the model, which contains the actual application logic and data, the view, which is the windows and widgets on the screen, and the controller, which takes care of the mapping between user actions and model functions.

Prior to starting any coding of the ReMP3 project, the MVC design pattern was explored. However as the design progressed, it became apparent that the .NET widgets, were not particularly suited to the use of the MVC design pattern. The .NET implementation, tends to combine the view. Each GUI widget, like a button, is a subclass of component which provides internal events that should be subscribed to by all interested parties.

This means that each widget has its own events, essentially acting as the controller for the functions that can be invoked from it. It can also refrain from defining an event, in which case the events it generates get propagated back up the inheritance tree, to its parent.

The drawback of this is that it makes it harder to replace only the view, or only the controller, with something else. However, as it's highly unlikely that this will ever be necessary for this particular application, it was decided to abandon the MVC approach, in favour of the native component event handling model within .NET.

6.2 Packages

When conducting the initial design, one of the requirements was re-use. With this in mind most of the individual packages were written as Windows Controls, that could be used within any .NET windows forms application.

The packages were conceived by first examining the set of requirements, and trying to sub divide these requirements into a set of packages that could be treated as stand alone units. Taking this approach meant that each unit could be developed and tested in isolation, despite being part of the overall application.

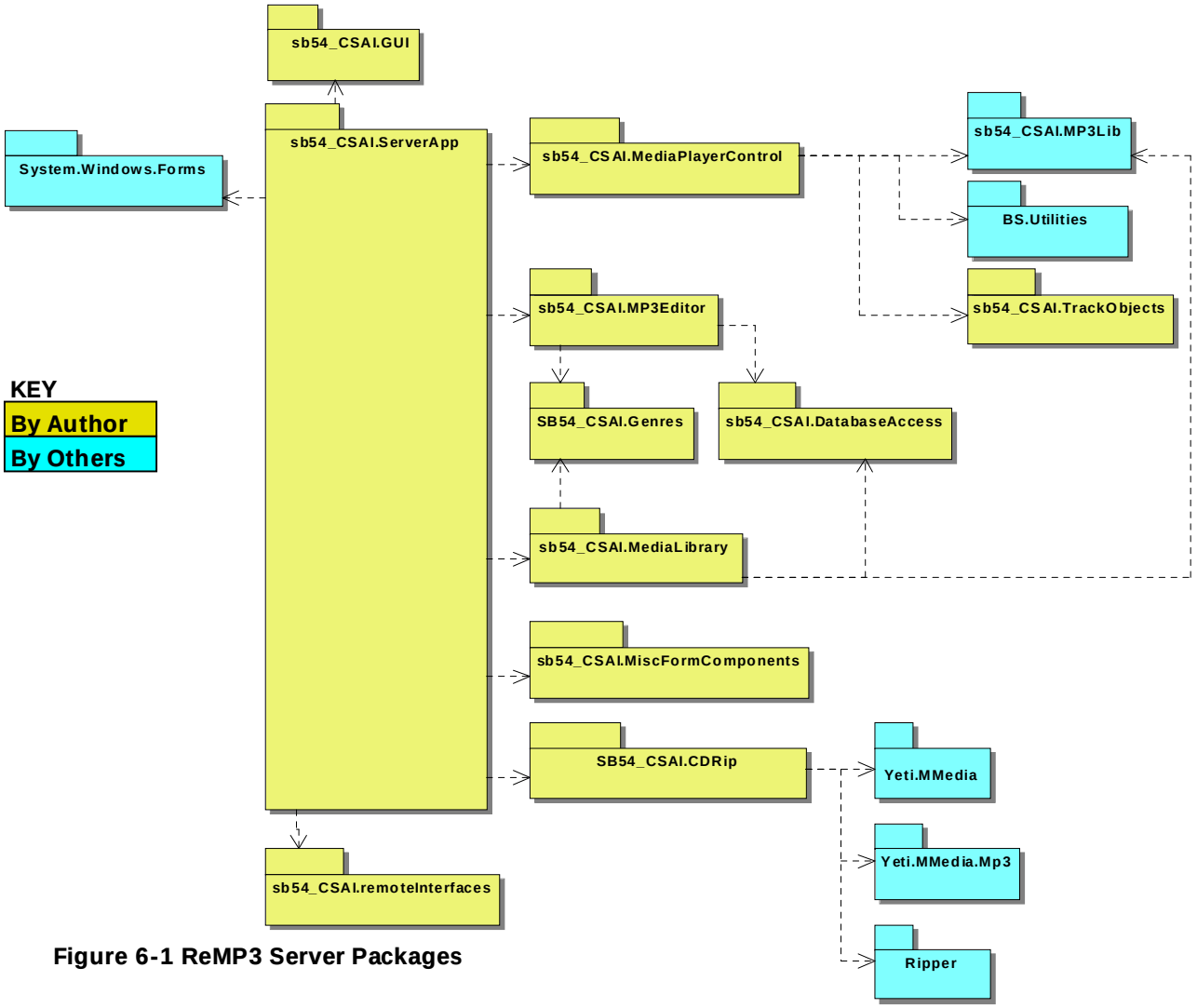

Figure 6-1 ReMP3 Server Packages

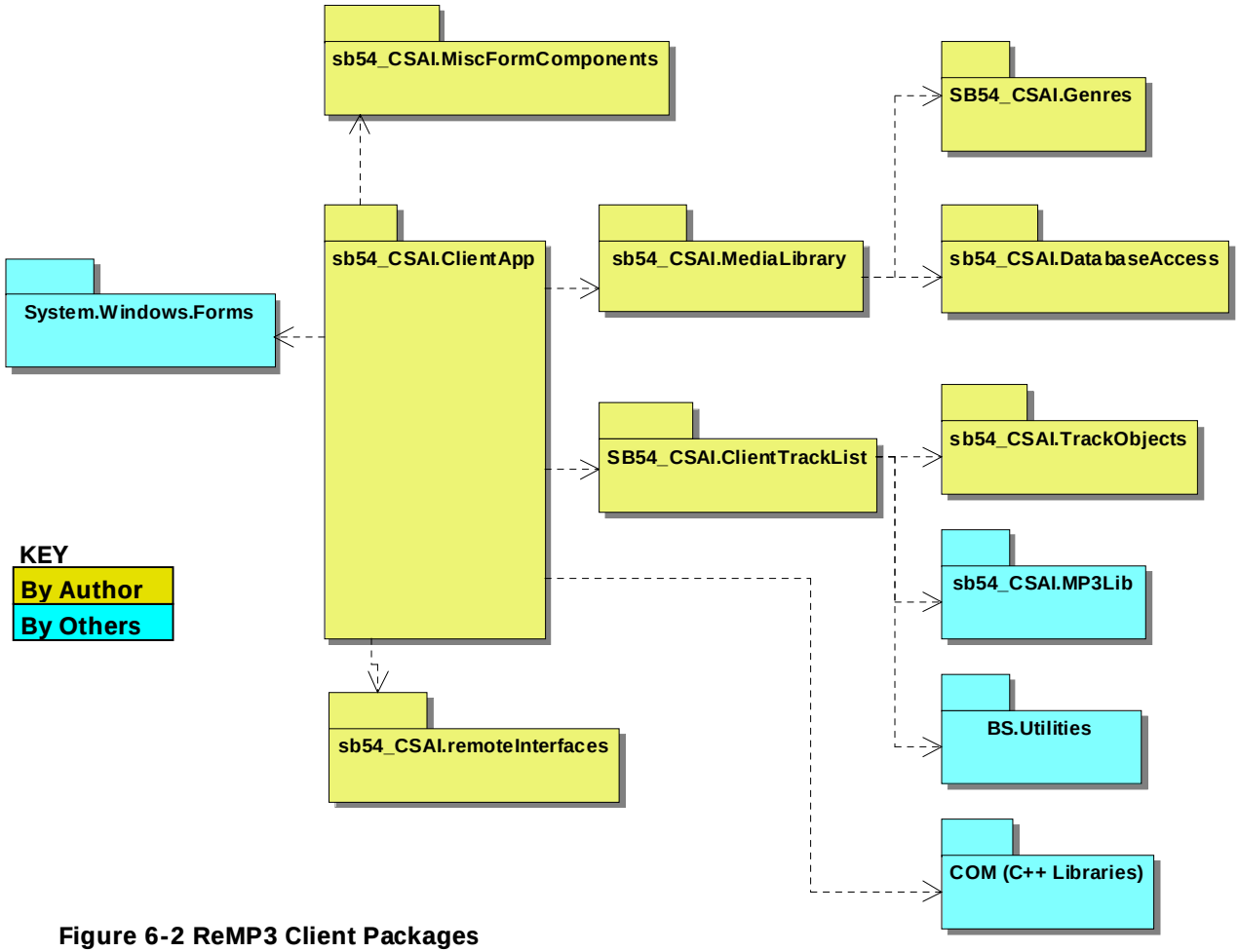

Figure 6-2 ReMP3 Client Packages

Figure 6-1 and Figure 6-2 show the top level package structure of the system. External libraries are also shown along with the dependencies on them.

Figure 6-1 shows the main packages that make up the ReMP3 Server application. The main packages being :

- SB54_CSAI.GUI : Provides some common user controls.
- SB54_CSAI.ServerApp : Provides the ReMP3 Server application form(s)
- SB54_CSAI.RemoteInterfaces : Provides the .NET remoting classes, that allow the remote control operations
- SB54_CSAI.MediaPlayerControl : Provides a media player control that is used within the ReMP3 Server application main form
- SB54_CSAI.MP3Editor : Provides an MP3 editor control that is used within the ReMP3 Server application main form
- SB54_CSAI.Genres : Provides a thread safe singleton object for storing and retrieving a list of genres to the database
- SB54_CSAI.DataBaseAcess : Provides a class for manipulating the Microsoft Access Database
- SB54_CSAI.MediaLibrary : Provides an media library control that is used within the ReMP3 Server application main form
- SB54_CSAI.MiscFormComponents : Provides some common user enhancement classes, such as adding icons to menu items.
- SB54_CSAI.CDRip : Provides an CDRip and Wav to Mp3 control that are used within the ReMP3 Server application main form
- SB54_CSAI.TrackObjects : Provides some common media control classes

Figure 6-2 shows the main packages that make up the ReMP3 Client application. The main packages being :

- SB54_CSAI.MiscFormComponents : As stated above
- SB54_CSAI.MediaLibrary : As stated above
- SB54_CSAI.RemoteInterfaces : As stated above
- SB54_CSAI.Genres : As stated above
- SB54_CSAI.DataBaseAcess : As stated above
- SB54_CSAI.TrackObjects : As stated above
- SB54_CSAI.ClientApp : Provides the ReMP3 Client application form(s)
- SB54_CSAI.ClientTrackList : Provides a track lit control that is used within the ReMP3 Client application main form

6.3 Common Tactical Policies

The following are common localised mechanisms which occur throughout the system and the policies which have been developed to handle them.

6.3.1 Disposing of Held Resources

All held resources associated with a given class shall be released, when no longer required, which will allow automatic garbage collection of unwanted resources.

Within the C# language nearly every object supports a Dispose() method. It is by using this Dispose() method that an object is marked for garbage collection.

The following rules should be used for Disposal of resources

6.3.1.1 GUI Components

A single overridden method must be provided of the format shown below

```
protected override void Dispose( bool disposing )
{
    if( disposing )
    {
        if (components != null)
        {
            components.Dispose();
        }
        if (oImgArtWork != null)
            oImgArtWork.Dispose();
    }
    base.Dispose( disposing );
}
```

Where Components is an instance field of type IContainer

Where specific objects must also be disposed by calling their Dispose() methods

NOTE : That if the Visual Studio Designer IDE is used, the method signature above will be auto generated for any Form that inherits from System.Windows.Forms.Form and also for any control that inherits from another Forms control object.

6.3.1.2 Classes

If there is the possibility that a class will maintain resources (For example a file access), a single Dispose() method must be provided in the class that holds the resource. This Dispose() method should close or free any held resources in the appropriate manner.

6.3.2 Object Naming

6.3.2.1 GUI Object Naming

To make it easier to use intellisense (if the Visual Studio Designer IDE is used) and for simplicity, an object naming convention will be adopted as follows.

o[ObjectName] : Where o indicates an actual object.

o[ObjectName] : Where [ObjectName] is the name of the object.

As this project will be highly graphical, and will make use of a lot of visual components, the standards shown in Table 6-1 apply for determining the actual [ObjectName] prefix that should be adopted for any given GUI component object.

Visual Component Type	ObjectName Prefix
System.Windows.Forms.Menu	Mnu
System.Windows.Forms.Menuitem	Mnu
System.Windows.Forms.Panel	Pnl
System.Windows.Forms.Label	Lbl
System.Windows.Forms.Textbox	Txt
System.Windows.Forms.Treeview	Tv
System.Windows.Forms.Listbox	Lst
System.Windows.Forms.Combobox	Cmb
System.Windows.Forms.Listview	Lv
System.Windows.Forms.Linklabel	Lnk
System.Windows.Forms.Imagelist	ImgList
System.Windows.Forms.Tabpane	Tab
System.Windows.Forms.Tabcontrol	Tab

Table 6-1 Object Naming Conventions

So some example GUI component object names might be as follows :

- oPnlHomeTop
- oTabTrackList
- oImgList_84
- oMnuExit
- oLnkTrackList

6.3.2.2 Code Variable Object Naming

For Non graphical objects (basically every other variable, in C# everything is an object even primitives such as Int and Bool are automatically boxed into object types) the field variable names will be named in a manner that makes the code reading as transparent as possible. The user should not have to wonder what a particular field does, the name of the field should be enough to make the function of the field clear.

6.3.3 Error Handling

Error handling must always be of the form shown below, where a try catch block is used and a System.Windows.Forms.MessageBox is used to show the error message to the user.

```
// Start the remoting services
try
{

}
catch(Exception ex)
{
    MessageBox.Show("Problem with Remoting in FrmMain : " +
        "\r\n\r\n\r\n" + ex.Message,
        "Remoting Services ERROR", MessageBoxButtons.OK,
        MessageBoxIcon.Error);
}
```

Part1 : Constructed message, must state which class has error
Part2 : the exception message
Part3 : Title
Part4 : Buttons to show
Part5 : Icon to show

6.3.4 IO Readers

The following method illustrate 2 possible methods of dealing with IO reader resources.

6.3.4.1 Option1

Where an IO reader is used it shall be wrapped in the C# using statement, which is defined as follows

Using (type variable = initialization) embedded statement

An example of which is shown below for a TextReader

```
using (TextReader reader = new StreamReader(filename))
{

}
```

This method ensures that as soon as the Using section of code is exited, the held resources by the Reader are immediately Disposed, so providing a safe way of using reader resources, without having to care about cleaning up resources.

6.3.4.2 Option2

Another method will be to use Try-Catch-Finally blocks, where the resource is always tidied up in the Finally block, as shown below.

```
XmlTextReader oXmlTextReader =null;
try
{
    //creating a text reader for the XML file already picked by the
    //overloaded constructor
    oXmlTextReader = new XmlTextReader(XMLFileName);
}
catch (Exception ex)
{
    MessageBox.Show("Problem with MediaPlayer Control clsXMLValidator " +
        "\r\n\r\n\r\n" + ex.Message,
        "XML Validation ERROR", MessageBoxButtons.OK,
        MessageBoxIcon.Error);
    return false;
}
finally
{
    // close the readers, no matter what.
    oXmlValidatingReader.Close();
    oXmlTextReader.Close();
}
```

Depending on the code requirements, either of these methods may be used, when using IO reader resources.

6.3.5 Commenting

Visual Studio .NET allows for XML comments to be applied to code and methods. This is similar to java's JavaDoc facility.

Table 6-2 below, provides commonly used functionality in user documentation :

Tag	Meaning
<exception>	lets you specify which exceptions can be thrown /// <exception cref="System.Exception">Thrown when... .</exception>
<list>	is used to define the heading row of either a table or definition list /// <list type="bullet"> /// <item> /// <description>Item 1.</description> /// </item>
<param>	should be used in the comment for a method declaration to describe one of the parameters for the method /// <param name="Int1">Used to indicate status.</param>
<remarks>	is used to add information about a type, supplementing the information specified with <summary>. /// <remarks> /// You may have some additional information about this class. /// </remarks>
<returns>	should be used in the comment for a method declaration to describe the return value /// <returns>Returns zero.</returns>
<see>	lets you specify a link from within text /// <see cref="MyClass.Main"/>
<summary>	should be used to describe a type or a type member /// <summary>MyMethod is a method in the MyClass class. /// <para>Here's how you could make a second paragraph in a description. <see cref="System.Console.WriteLine"/> for information about output statements.</para> /// <seealso cref="MyClass.Main"/> /// </summary>
<value>	lets you describe a property. /// <value>Name accesses the value of the name data member</value>

Table 6-2 Commenting Style

When XML comments are provided, a 3rd party tool can be used to provide a compiled help file. To this end, the NDoc documentation tool [15] has been used. Which produces documentation in many different formats. The ReMP3 project API documentation has been built in the Microsoft MSDN documentation style as shown in Figure 6-3**.**

NOTE : Section 7 CLASS DESIGN, discusses the classes in a broad manner, and does not go into a blow by blow account, of what each method of each class does. If this level of detail is required, the NDoc MSDN documentation should be referred to.

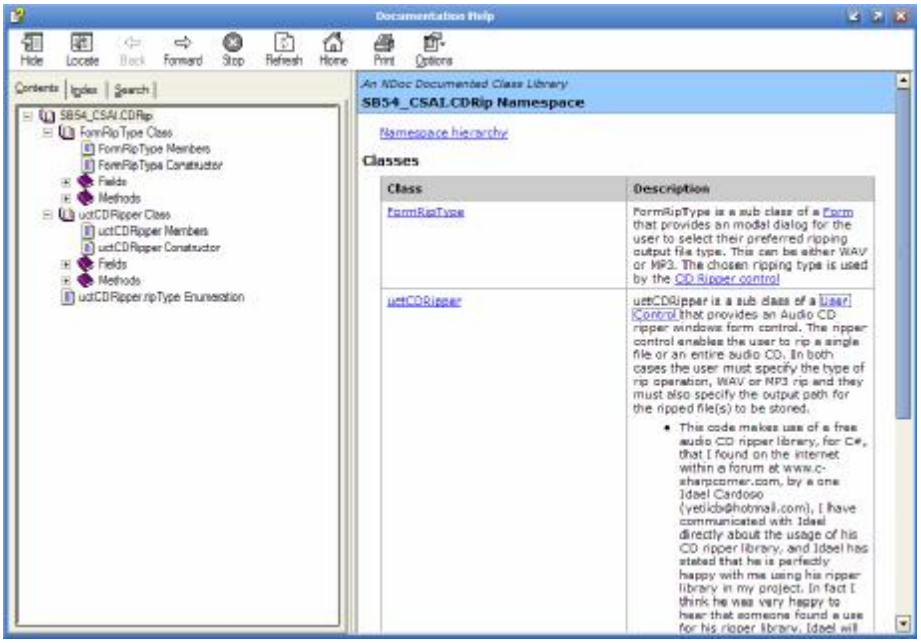

Figure 6-3 NDOC Documentation

6.3.6 Installation Process

Visual Studio .NET allows the building of installation Windows Installer Package MSI files. These are installation wizards that guide the user through the entire installation process, with status messages and prompts at each step. There are 2 Windows Installer Package MSI files, one for the Server application and another for the Client application.

It should be noted that the Windows Installers are actually being written and compiled in .NET themselves, so the .NET framework could not be included as part of the installer, as the installer actually relies on the .NET framework itself.

Another component that needs to be manually installed is the Windows Media Player SDK, as this SDK does not provide a redistributable executable that can be included as part of the installer.

7. CLASS DESIGN

7.1 .NET Framework Discussion

.NET is a relatively new language that has been developed by Microsoft, and has been written from scratch. It is quite similar to Java [33] in lots of ways, where automatic memory allocation / garbage collection is performed. It also exposes some software design ideas that are subtly different from other languages. It should also be noted that, as .NET is relatively new, certain features simply do not exist. The following sub sections shall try and explain these areas.

7.1.1 Events

"An event is a message sent by an object to signal the occurrence of an action. The action could be caused by user interaction, such as a mouse click, or it could be triggered by some other program logic. The object that raises (triggers) the event is called the event sender. The object that captures the event and responds to it is called the event receiver.

In event communication, the event sender class does not know which object or method will receive (handle) the events it raises. What is needed is an intermediary (or pointer-like mechanism) between the source and the receiver. The .NET Framework defines a special type (Delegate) that provides the functionality of a function pointer.

A delegate is a class that can hold a reference to a method. Unlike other classes, a delegate class has a signature, and it can hold references only to methods that match its signature. A delegate is thus equivalent to a type-safe function pointer or a callback."

.NET Framework Developer's Guide, Events And Delegates tutorial [16]

The ReMP3 project uses a lot of custom events to provide communication between different software packages.

For more information about how Event are raised and consumed the Events And Delegates tutorial [16] should be read, as this is something the ReMP3 project makes use of in a number of places.

7.1.2 Delegates

As well as being used with events, Delegates can also be used for asynchronous programming and multi threading GUI applications. The ReMP3 project makes use of multithreading within the Media Library control, which is discussed later at section 7.2.5.

7.1.3 Assemblies

"Before the .NET Platform was introduced we had to deal with the predecessors of assemblies: Normal DLLs exporting global functions, and COM DLLs exporting COM classes. Microsoft itself introduced the phrase "DLL-HELL" to describe traditional problems with DLLs.

The .NET platform's answer to DLL Hell and all its problems is ***assemblies****. Assemblies are self describing installation units, consisting of one or more files. One assembly could be a single DLL or EXE that contains metadata, or it can be made of different files, for example, resource files, metadata, DLLs and an EXE."*

Simon Robinson [0]

7.1.3.1 Assembly Structure

An assembly consists of assembly metadata describing the complete assembly, type metadata describing the exported types and methods, Microsoft Intermediate Language (MSIL) code, and resources (which may include images, strings, files etc etc). All these parts can be in one file or spread across several files.

An example of this is as shown in Figure 7-1 Assembly Structure below:

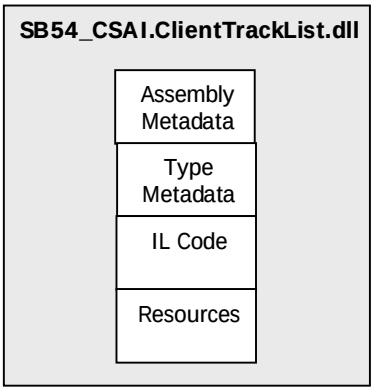

Figure 7-1 Assembly Structure

7.1.4 .NET Shortcomings

As mentioned earlier, .NET and the v1.1 framework is a relatively new technology. As such, there are certain areas that have limited, or no support, as yet. The following illustrate some of the problem areas that the ReMP3 project faced during development:

1. Networking : At the moment of writing this report, there was not really any support available for dealing with network collections/items. The solution was to write some managed code that deals with un-managed C++ com compliant code, which are really COM dll's that must be consumed by the use of the **DllImport** C# compiler predirective. It is by using this technique that the ReMP3 Client application, is able to browse a list of LAN computers

2. Full UNC Path names are not supported by the FileInfo and DirectoryInfo classes of the System.IO namespace of the v1.1 framework. Although the Path class does support a full UNC path, it does not contain any methods that are much use. All the really handy methods that were required, reside in the FileInfo and DirectoryInfo classes. To rectify this shortcoming the following approach needed to be taken.

    - Where a full UNC path like \\computer1\c:\directory1\file2.mp3 was expected, this had to be changed to use the default administrative share of \\computer1\c$\directory1\file2.mp3. This seems to rectify the .NET IO class shortcomings.

7.2 ReMP3 Server Package Descriptions

The following sub sections describe the functionality of the ReMP3 Server application.

7.2.1 ServerApp

The class diagram for the ServerApp is shown below in Figure 7-2. At its most basic level the ServerApp is simply acting as a container for other user controls, and capable of receiving remote control instructions. There is also some work that has been carried out to make the ServerApp look as good as possible, which is possible by the use of the classes contained within the GUI package (See section 7.2.2), and the use of some very nice icons (Foood's icons [1])

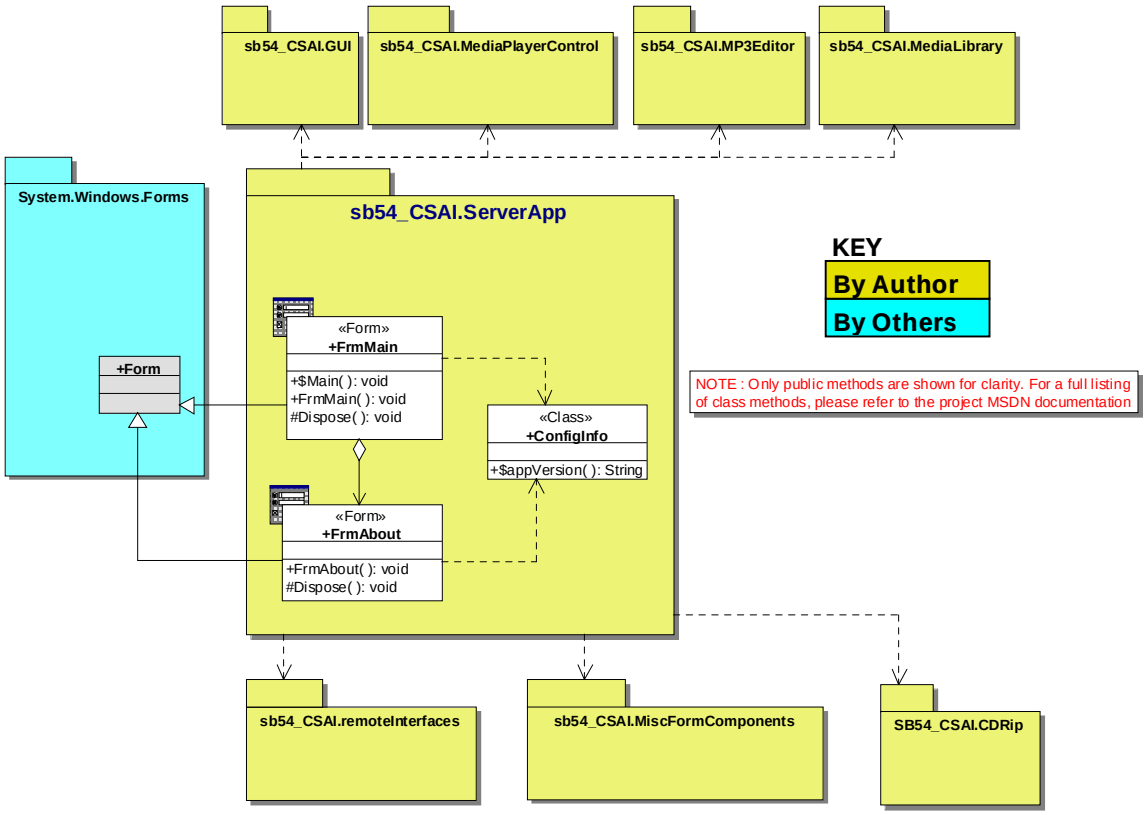

Figure 7-2 ServerApp Class Diagram

The ServerApp form hosts the following user controls

- ShrinkPanel : See section 7.2.2
- MP3 Editor Control : See section 7.2.4
- Media Library Control : See section 7.2.5
- Media Player Control : See section 0
- CD Ripper Control : See section 7.2.7
- WAV to MP3 Converter Control : See section 7.2.7

The ServerApp main GUI form is as shown below in Figure 7-3.

Figure 7-3 ReMP3 Server main GUI form

It should also be noted that contextual help is provided by using the F1 key.

7.2.2 GUI

The GUI Package simply contains a number of classes that provide an XP Style collapsible panel. This group of classes were obtained from the codeproject [2]. The author of the original classes [17] is Derik Lakin.

Shown in Figure 7-4 below is the class diagram for Derik Lakins [17] original classes. As developing GUI components is an area of personnel interest to the author, Derik Lakin's [17] original classes were dismantled and rebuilt during the development phase of the ReMP3 project, such that a better understanding of the inner workings of Derik's codeproject [2] article [17] could be formed.

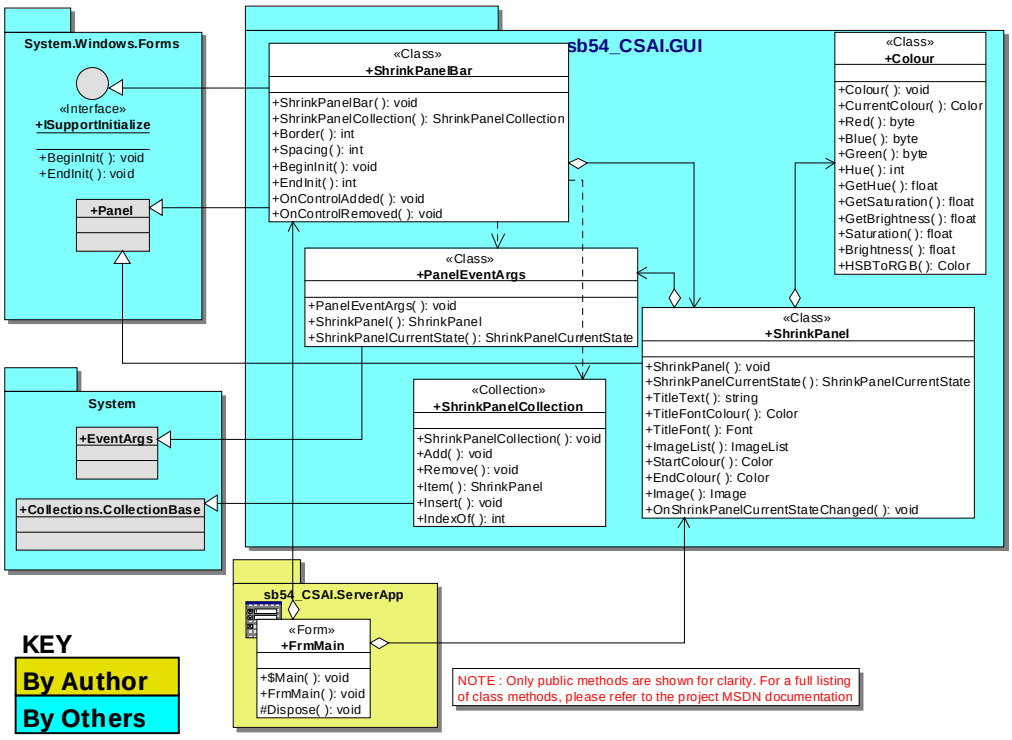

Figure 7-4 GUI Class Diagram

This GUI package contains a fully reusable control, ShrinkPanelBar, that is a general control that provides a collection of ShrinkPanel objects that may be shrunk or grown by clicking on the title bar of each ShrinkPanel. An example of this is as shown below, where the ShrinkPanelBar contains 2 individual ShrinkPanels, one for Tools and one for Help & About.

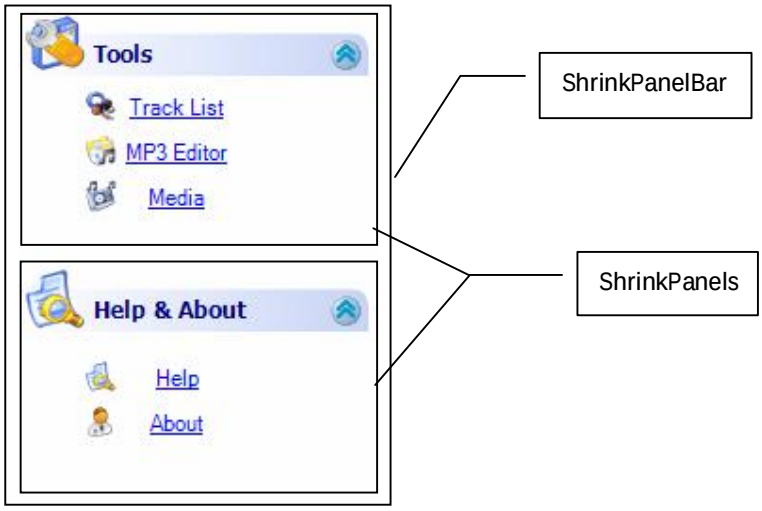

Figure 7-5 ShrinkPanelBar and ShrinkPanel

The ShrinkPanelBar (Figure 7-5) control extends the System.Windows.Forms.Panel and is simply used as a container for a collection of ShrinkPanels.

Properties are provided for the following

- Spacing : The required spacing between panels
- Border : The required border of a ShrinkPanel

7.2.2.1 ShrinkPanel

A single ShrinkPanel object extends the System.Windows.Forms.Panel class and as such provides all the protected (or above) level events/properties and methods of the inherited Panel class.

Each ShrinkPanel control is capable of holding sibling controls, like buttons textfields etc. The ShrinkPanel are placed inside a ShrinkPanelBar container control.

7.2.3 MiscFormComponents

The MiscFormComponents Package shown in Figure 7-6 simply contains a number of classes that provide some nice GUI controls and a skinning class for skinning a Windows XP application.

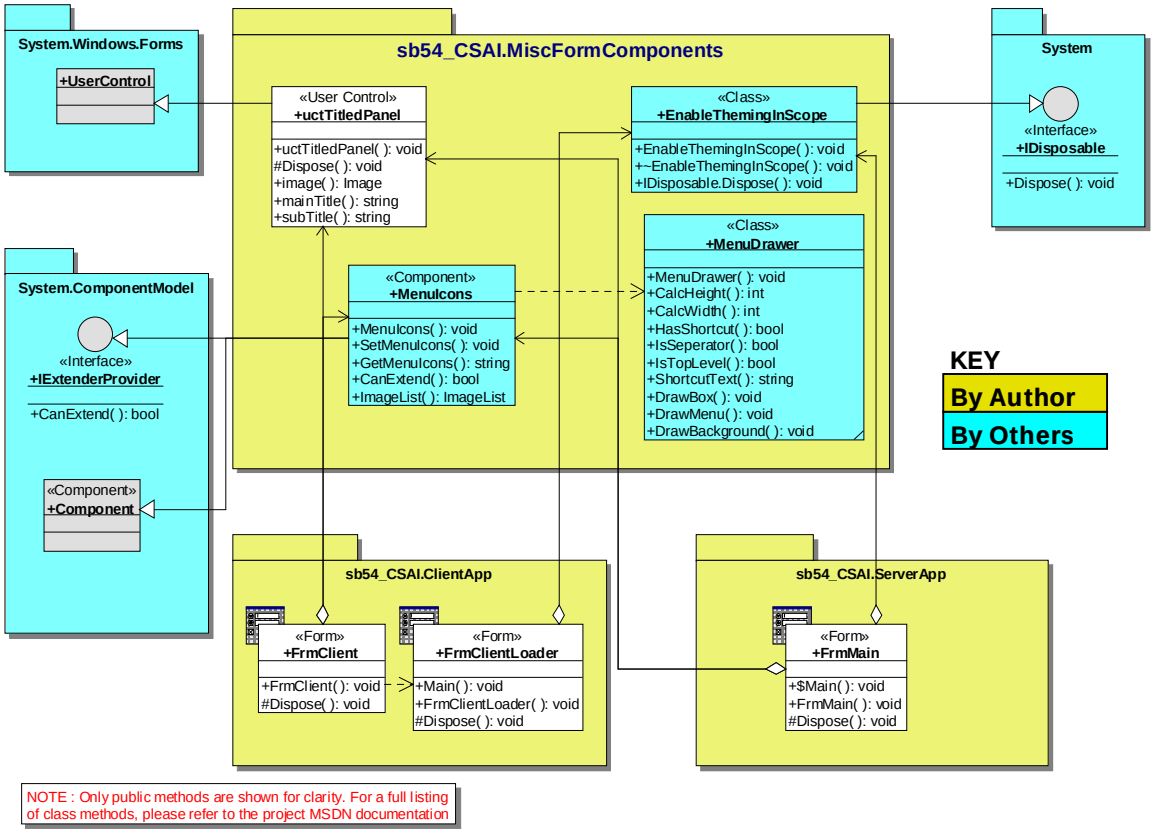

Figure 7-6 MiscFormComponents Class Diagram

Most of the classes contained within this package have been taken from other authors, and shall be explained individually below. The uctTitledPanel is however one control that was constructed entirely by the author.

The uctTitledPanel control extends the System.Windows.Forms.Panel and provides the following Properties

- Image : The image to be painted using a custom paint method call
- mainTitle : The main title text to be painted using a custom paint method call
- subTitle : The sub title text to be painted using a custom paint method call

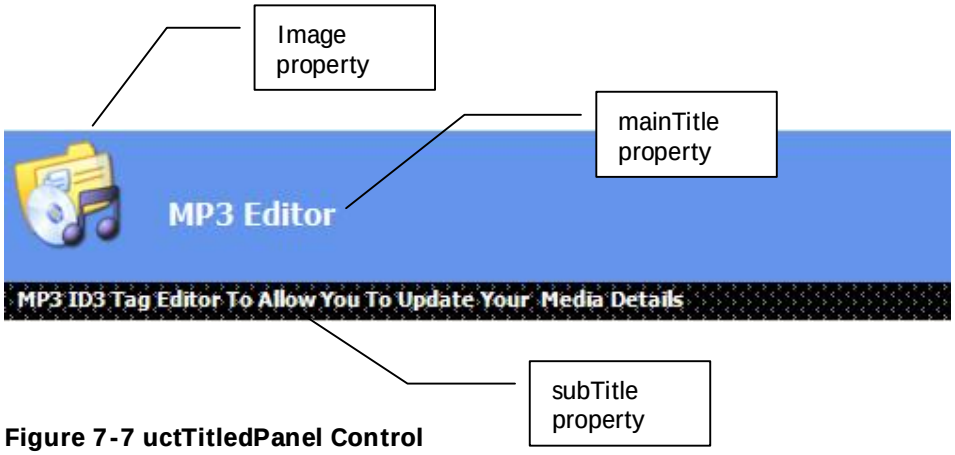

Figure 7-7 uctTitledPanel Control

The uctTitledPanel control shown in Figure 7-7, is used within both the ReMP3 Server and ReMP3 Client applications at the top of each tab page.

7.2.3.1 EnableThemingInScope Class

The EnableThemingInScope class provides a mechanism to apply a Windows XP theme to the controls, within the application the theming is being applied to.

This class was obtained from the Microsoft web site [18].

.NET actually should allow theming to be performed without this class, by issuing a Application.EnableVisualStyles() within the main method of the application, around the main message loop call : Application.Run(new frmMain());

This approach is discussed further at the Cool Client Stuff blog [19]

However this approach seemed to suffer with the following error : “InteropServices.SEHException: External component has thrown an exception”. So an alternative approach had to be implemented.

Using the EnableThemingInScope class, a call is made to some unmanaged code using a DllImport directive, which calls the following methods of the Kernel32.dll :

- CreateActCtx : Create active content
- ActivateActCtx : Activate active content
- DeactivateActCtx : Deactivate active content

When the CreateActCtx method is called, the current controls (which are v5.8, which by default are non themed) are swapped for themed controls (which are v6.0). Then when the DeactivateActCtx method is called the current controls are swapped back to being non themed controls.

In order to use the EnableThemingInScope class the following code snippet is used within the main method of any application that required theming.

```
//So create the theming the using the correct approach, by using the
//EnableThemingInScope class
using( new EnableThemingInScope( true ) )
{
    FrmMain fMain = new FrmMain();
    fMain.CreateControl();
    Application.Run(fMain);
}
```

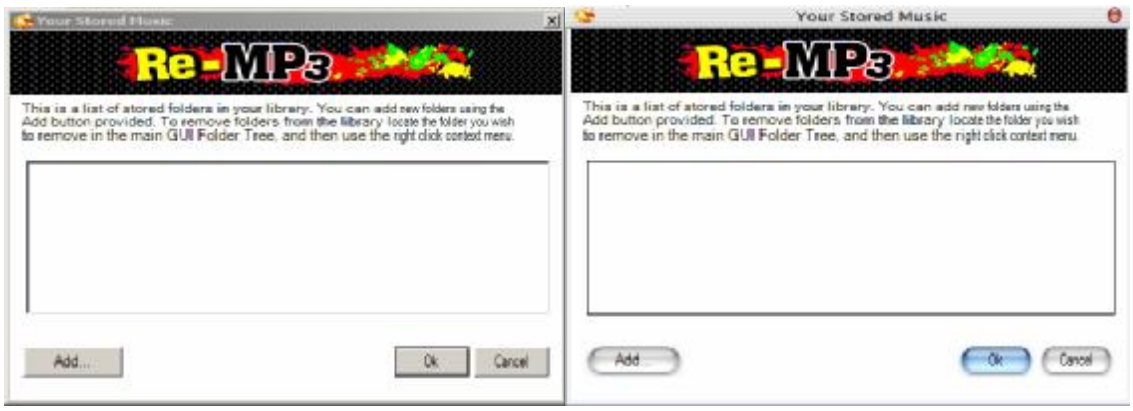

Figure 7-8 Skinning In Action

Figure 7-8 shown above shows the effect on skinning on one of the ReMP3 Server forms, where the left image is not skinned, and the image on the right is skinned using a Mac OS VIII style skin.

By allowing the application to be skinned, the ReMP3 project will feel like part of the operating system (Windows XP), rather than just a bolt on extra program. Microsoft have actually invested a lot of time and expense to ensure that their native skins adhere to strict HCI guidelines and exhibit sound usability principles. Most users will be familiar with this style as they would probably have used Microsoft products before.

If the user is not a Microsoft fan, then the ability to apply an Apple like skin, should be fulfilling and fun to use, which again follows some known HCI Heuristics.

Section 4.6 covers the concept of user evaluations in more detail.

7.2.3.2 MenuIcons / MenuDrawer Class

Menu images are not provided as part of the normal .NET menu development. It would of course be possible to provide owner drawn menus for each menu item, this is however quite a short sighted and non generic solution. So with this in mind an alternative approach was searched for. What was found were 2 free classes, MenuIcons / MenuDrawer.

The MenuIcons / MenuDrawer classes were obtained from the codeproject [2] web site. The author of the original classes [20] is Chris Beckett.

Shown in Figure 7-6 above is the class diagram for Chris Becketts [20] original classes. As developing GUI components is area of personnel interest to the author, Chris Becketts [20] original classes were dismantled and rebuilt with minor modifications (to draw items using different rendering when selected) during the development phase of the ReMP3 project, such that a better understanding of the inner workings of Chrises codeproject [2] article [20] could be formed.

The MenuIcons class extends both the System.ComponentModel.Component and System.ComponentModel.IExtenderProvider. By extending component the MenuIcons component may be used within the Visual Studio designer work surface. By also extending the IExtenderProvider interface, any control that meets the CanExtend method requirements (can be extended) of the MenuIcons class, will be able to have an additional extender property provided automatically within the designer properties for the control in question. The extender property appears as an additional property to the normal properties of the standard System.Windows.Forms controls.

The clever part is the implementation of the CanExtend method within the MenuIcons class. This has been written in a way such that ONLY MenuItem controls may use the extender property provided by the MenuIcons class.

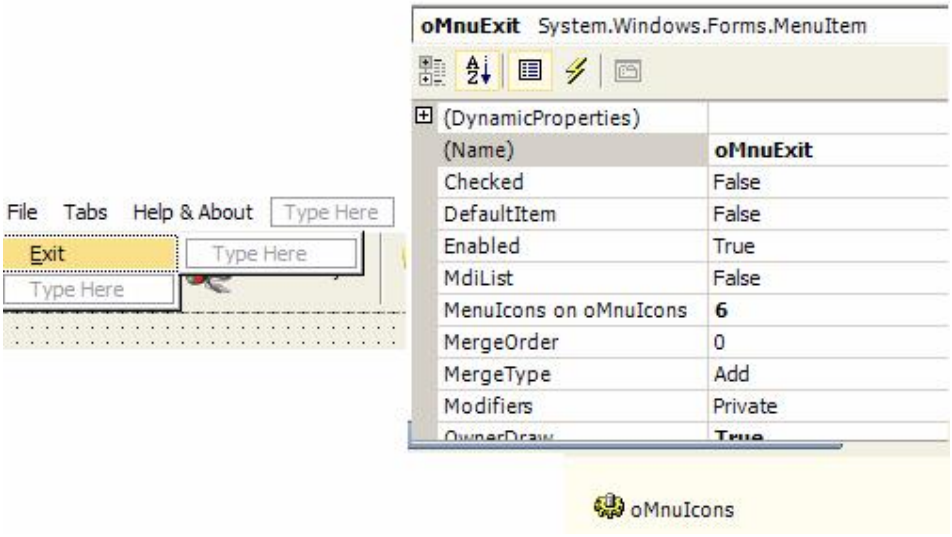

Figure 7-9 IExtender Properties

It can be seen in Figure 7-9 above that an instance of the MenuIcons class is created within the Visual Studio work surface, and also that when a MenuItem is examined, and extra property appears. This extra property is actually contained within the MenuIcons class. This Solution is totally generic and may be used to provide icons to any number of MenuItem objects.

Figure 7-10 below shows the MenuIcons class providing an icon to a number of menuitems at runtime.

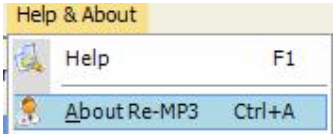

Figure 7-10 MenuIcons at runtime

7.2.4 MP3 Editor

The uctMP3Editor is used to edit mp3 files to either modify a ID3 tag that may already exist within the file, or to create a new ID3 tag for a mp3 file. The uctMP3Editor control shown in Figure 7-11, is one of the simpler user controls that is hosted within the Server application. The class diagram for the MP3Editor control is shown below.

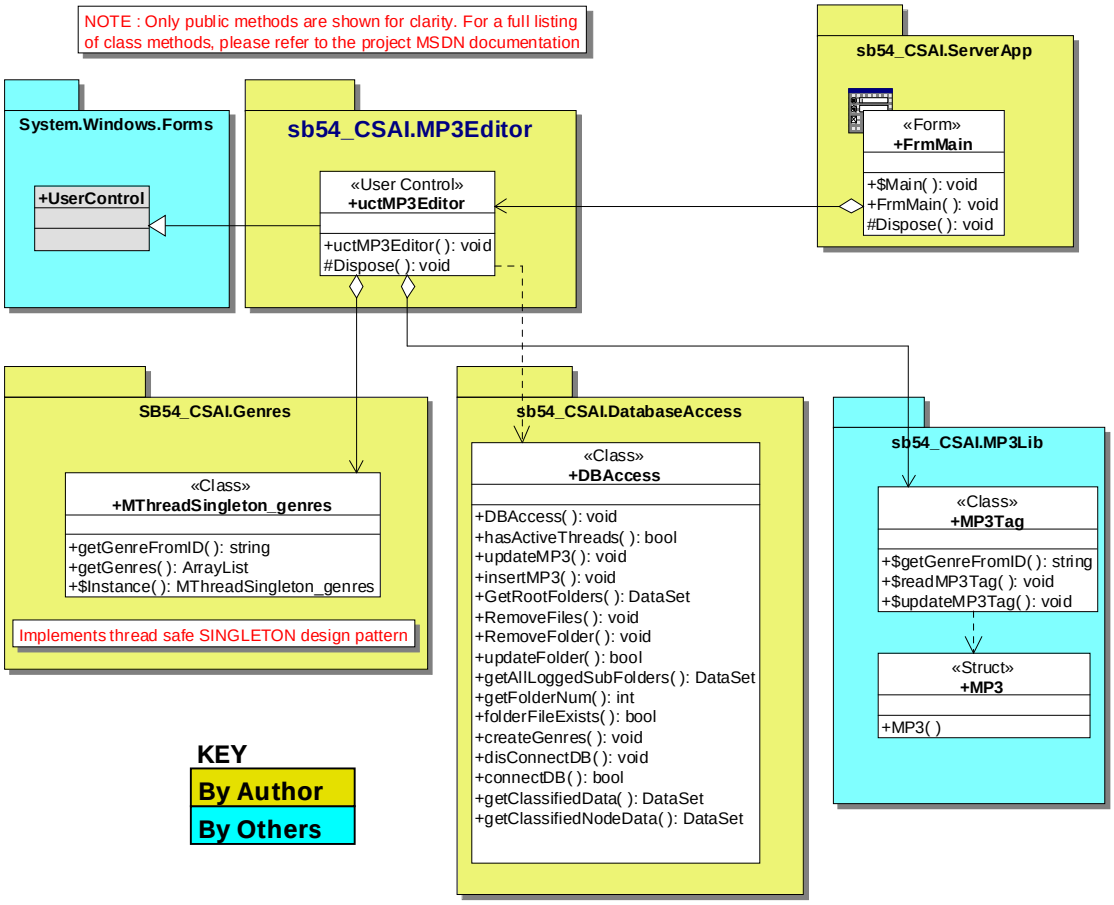

Figure 7-11 MP3Editor Class diagram

Figure 7-12 MP3Editor Control

It can be seen in Figure 7-12, that the control provides a hyper link to allow the user to select the current MP3, and fields for each of the MP3 ID3 values. The MP3 ID3 values are read by using a free MP3Lib package that was obtained from the c-sharpcorner [21]. The author of the original classes [22] is Paul Lockwood.

This MP3Lib is effectively a byte stripper, where the last 128 bytes of the MP3 file are used to construct the ID3 metadata fields. This MP3Lib works for v1 and v1.1 ID3 tag formats only, as version 2.0 is completely different and far more complex.

The format of a V1 ID3 [23] tag is as follows:

Metadata field	Byte Start	No of Bytes Used (last 128 bytes in file)
TAG	0	3
Song Title	3	30
Artist	33	30
Album	63	30
Year	93	4
Comment	97	30
Genre	127	1 (byte is set to a value of 0-147 which represents specific genre)

Figure 7-13 v1 ID3 Tag metadata

The format of a V1.1 ID3 [23] tag is as follows:

Metadata field	Byte Start	No of Bytes Used (last 128 bytes in file)
TAG	0	3
Song Title	3	30
Artist	33	30
Album	63	30
Year	93	4
Comment	97	28
Zero byte separator	125	1
Track byte	126	1
Genre	127	1 (byte is set to a value of 0-147 which represents specific genre)

Figure 7-14 v1.1 ID3 Tag metadata

Using the MP3Lib package [22] the metadata fields are read into the fields provided on the uctMP3Editor control. The user is then free to modify these fields, and then click the update button. Various checks are performed, to ensure that data entered is correct. These validation methods, range from not null validation checks, to more complicated validation, using regular expressions. If the user manages to update the MP3 with new details, the fields are cleared and disabled, awaiting a new MP3 file to be chosen for editing.

It should be noted that the MP3Lib package, is used throughout the ReMP3 project. During the coding phase, several limitations were found when using the original MP3Lib library, with the accompanying Access database. Namely the use of non database supported ASCII characters being part of the metadata field, that were byte extracted within the original MP3Tag class. This caused problems, when trying to insert values stored within the MP3 Structure, into the database tables. As such the MP3Tag class was heavily modified, to strip out all database unfriendly characters.

Some examples of these database unfriendly characters are as follows:

- NUL, SOH, STX, ETX

7.2.4.1 Genres Usage

The uctMP3Editor control, has a need to display a list of genres (values 0-147) for the Genre byte of the MP3 ID3 tag. The individual genres are also needed within the database, to use as foreign keys within the database tables that the Media Library uses (see section 7.2.5). The logging of these genre values into the database should only be performed once.

As stated the uctMP3Editor control and the Media Library both use the Genres package, and the ReMP3 Server application does not know which of these controls, will be interacted with first. So to deal with this issue, the ReMP3 project uses a Singleton design pattern, so there is only ever one instance of the MThreadSingleton_genres object in existence.

The MThreadSingleton_genres class has an embedded resource file called Genres.txt, that is part of the Genres assembly file.

When the Instance property of the MThreadSingleton_genres class is called, the Genres.txt file is read into an internal ArrayList (and subsequently used to populate the Access database with this Genre data), which is then, available to the MThreadSingleton_genres current instance object.

The MThreadSingleton_genres class, also provides synchronization between threads, as the Media Library is multi-threaded, and may be using the MThreadSingleton_genres instance object, when another call to the Instance property is made by the uctMP3Editor control. By providing synchronization locks, the MThreadSingleton_genres code is deemed thread safe.

7.2.4.2 Database Access

The uctMP3Editor control, also makes use of the accompanying database, in that once an update to an MP3 file on disk is performed, the uctMP3Editor control will also check to see if this MP3 file is logged in the database. If it is found, the values of its metadata fields are updated with the new values from the newly saved MP3 file on disk.

7.2.5 Media Library

The uctMediaLibrary control is quite a complicated user control that makes use of other constituent controls, and interacts with the underlying Access database, to catalogue the user audio files. The uctMediaLibrary control is a common control, which is used by the Server and Client applications.

The class diagram Figure 7-15, is shown overleaf.

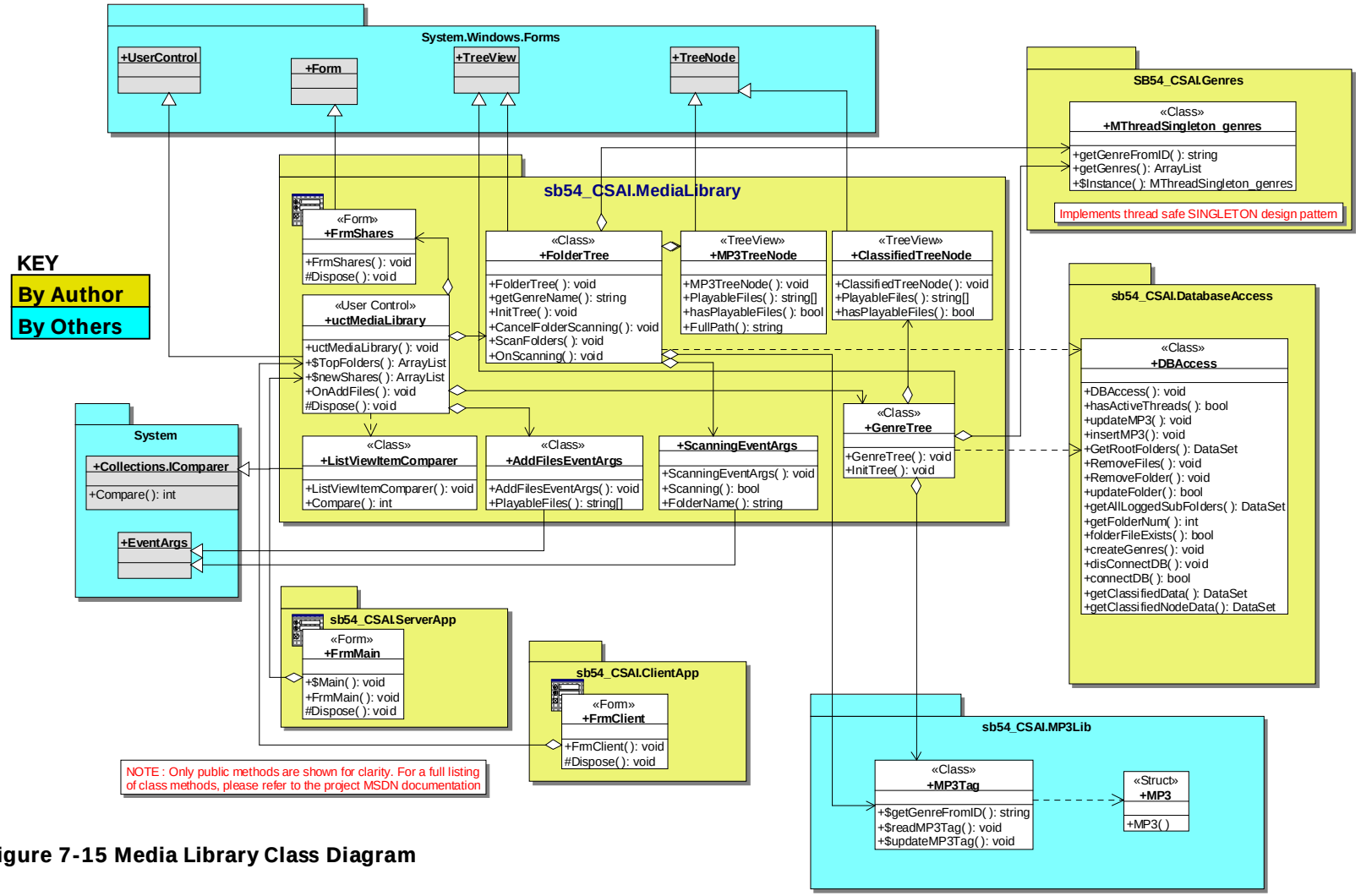

Figure 7-15 Media Library Class Diagram

When first shown (with no previously logged music) the uctMediaLibrary is shown as Figure 7-16 below.

The user may scan in new folders to the Library by using either of the “Share Files” or “Add Folder” buttons. Where the user will be prompted with the amount of hard disk space in Mbytes that is about to be scanned, prior to the scan commencing. The user may also ABORT the scan using the “Cancel Scan” button, that appears during a scanning operation.

It should be noted that the Media Library scanning is performed within a separate process thread, so the user may explore the rest of the application whilst a scan is being performed.

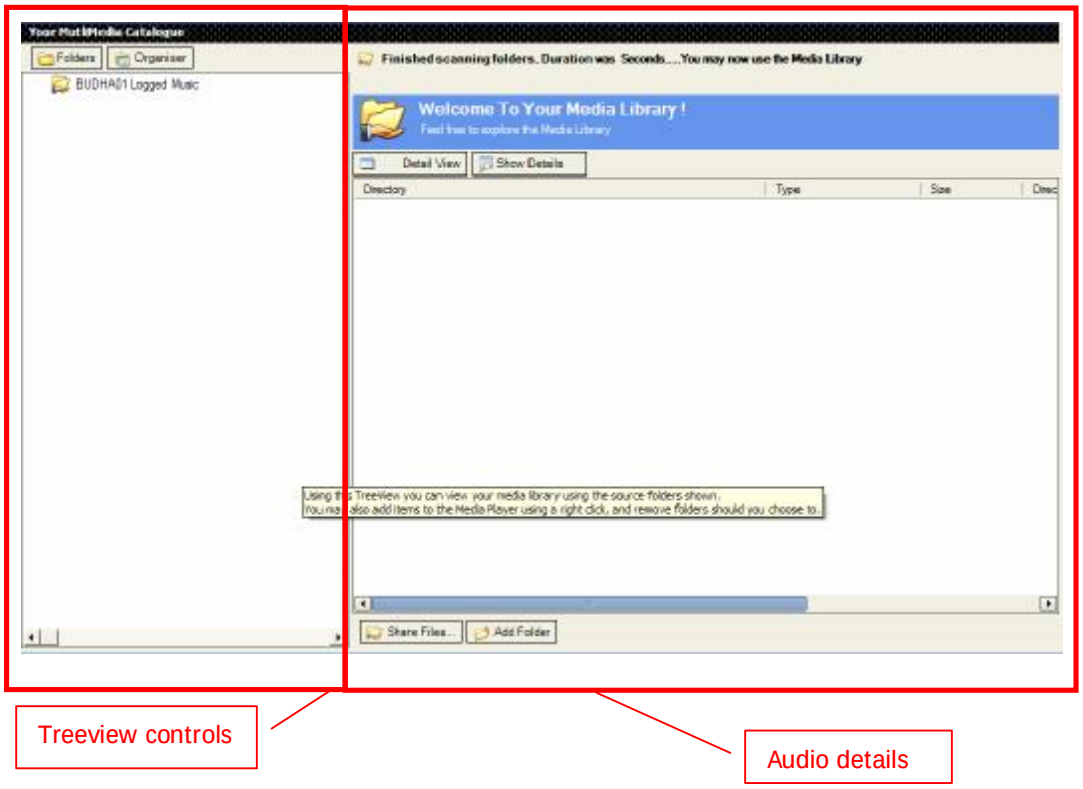

Figure 7-16 Media Library Control

7.2.5.1 Why Have A Media Library / What Does it Do

One of the ReMP3 project prerequisites is the Windows Media Player (WMP) SDK. The WMP SDK, when installed, already provides an object model of the entire list of interfaces that are realized by the WMP itself. The SDK is really meant to be used to query the already installed WMP, to create new play lists / retrieve track information and audio meta data.

For example it is possible to retrieve a list of all music under a particular Genre using the following SDK call.

```
WMPLib.IWMPPlaylist all;
all = oMMPlayer.mediaCollection.getByAttribute("Artist", "Avril Lavigne");
```

This will interrogate the installed WMP to see what items it has catalogued for the Genre selected. It may have none, as it relies on scanning folders to get this audio meta data, the same way as a dozen other audio applications do.

If the user has never forced the installed WMP to look at a particular folder, there will be no music catalogued at all. So the concept of retrieving track lists from a repository that has no items is just not that useful.

However this problem can be remedied, as the WMP allows new items to be added to the catalogue with the *mediaCollection.add* SDK method. There is however another more serious problem area that the SDK suffers from, in that there does not seem to be any way of retrieving a collection of attributes from the repository.

For instance, if a list of all artist names that are currently stored within the installed Media Player are required, there does not seem to be a way to get this information.

As demonstrated above, once the Artist is known, there is no problem retrieving a play list of that Artist's music, but if the Artist name is simply not known, up front, no such call can be made.

The design of any reasonable GUI would display all the audio meta data up front, and allow the user to make a selection of which Artist or Album they would like to listen to, from a group of Artist's or Albums. As this sort of feature does not seem to be part of the WMP SDK an alternative approach had to be considered.

This is in essence the Media Library control, that has been developed. The rest of this section is devoted to the workings of the Media Library control.

7.2.5.2 Database Design

The Media Library control actually contains 2 custom treeview controls :

- FolderTree
- GenreTree

In order to construct the GenreTree and FolderTree tree view controls, there needed to be an underlying database.

The database allows music catalogue folders and files to be stored. This database also stores MP3 meta data, which enables the GenreTree media tree view to function correctly.

The database schema will be as shown in Figure 7-17 below.

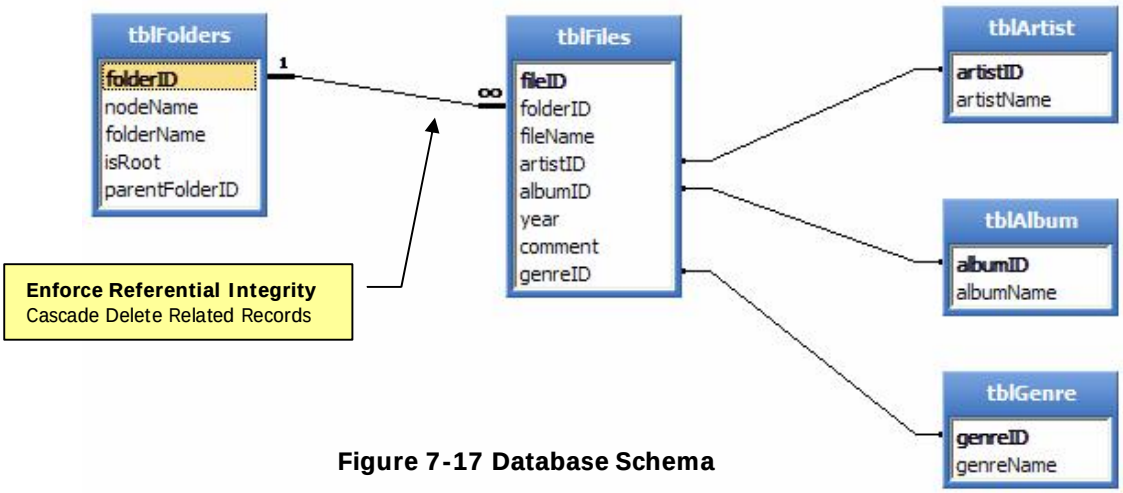

Figure 7-17 Database Schema

When interacting with the database, the ReMP3 project uses a mixture of classic Microsoft Access Data Objects (MDAC 2.7 [28]) and also the new totally overhauled .NET MDAC approach, which is MDAC 2.8 [29].

The difference being that in MDAC 2.7, the underlying database will be manipulated using standard SQL command strings like

- SELECT * FROM tblFolders
- INSERT INTO tblFolder VALUES (value1,value2.....etc etc)
- DELETE FROM tblFolders WHERE folderID = 32

Where as in MDAC 2.8, there is now the ability to create an, in memory, virtual copy of the underlying database, using classes like DataRow, DataSet, DataView, DataRelation, DataAdaptor. These classes allow the programmer to manipulate the database in a more intuitive manner.

Looking as some examples, where there are already several tables within a DataSet :

- To get a row from the DataSet :
  DataRow dr = DataSet.tables[“tblFolders”].row[1];
- To create a new within the DataSet :
  DataRow newRow=dsFolder.Tables["Folders"].NewRow();
- To update a DataSet with a DataAdaptor
  daFolder.Update(dsFolder.Tables["Folders"]);

Both the GenreTree and FolderTree rely heavily on the database. The following sub sections shall outline the database interactions required by these 2 custom controls.

7.2.5.3 GenreTree Treeview

The GenreTree is a subclass of System.Windows.Forms.Treeview and uses both normal TreeNode and a specialized ClassifiedTreeNode which is a subclass of System.Windows.Forms. TreeNode, which holds a list of its own media items, that may be added to the Media Player control (see section 7.2.6) by using the right click context menu. The format of the GenreTree is as shown in Figure 7-18 below.

The GenreTree uses the database for 2 purposes

1. To gather the catalogue sub items for By Album / By Artist and By Genre. This involves doing 3 separate queries into the database to retrieve the relevant datasets.
2. Once a user selects a sub node from one of the By Album / By Artist or By Genre nodes of the tree, a separate query will be run against the database to match the users currently selected sub node information.

In order to carry out step 1 above, the following queries will be used.

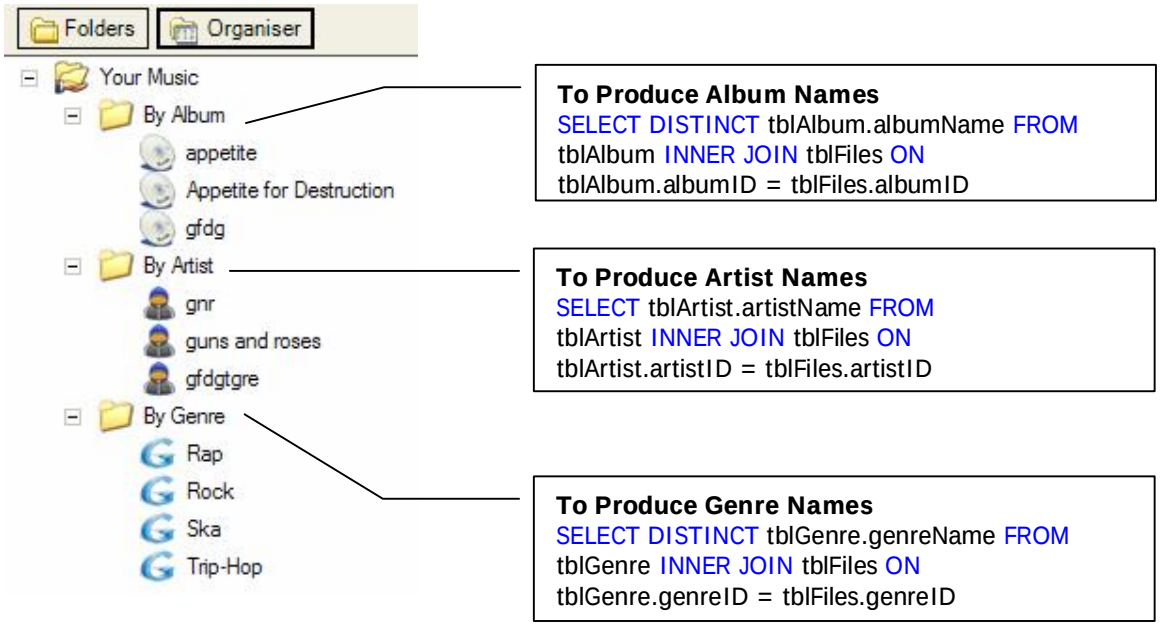

Figure 7-18 Media Library embedded GenreTree control

In order to carry out step 2 above, the following style of query will be used.

Get all tracks with Genre type = "Rock"

```
SELECT tblFolders.folderName, tblFiles.fileName FROM tblFolders
        INNER JOIN (tblGenre
        INNER JOIN tblFiles ON tblGenre.genreID = tblFiles.genreID)
        ON tblFolders.folderID = tblFiles.folderID
WHERE (((tblGenre.genreName)='Rock'))
```

Running the query shown above would produce a Dataset similar to that shown in Figure 7-19 shown below.

Figure 7-19 GenreQuery database query

7.2.5.4 FolderTree Treeview

The folderTree is a subclass of System.Windows.Forms.Treeview and uses both normal TreeNode and a specialized MP3TreeNode which is a subclass of System.Windows.Forms. TreeNode, which holds a list of its own media items, that may be added to the Media Player control (see section 7.2.6) by using the right click context menu. The format of the GenreTree is as shown in Figure 7-20 below.

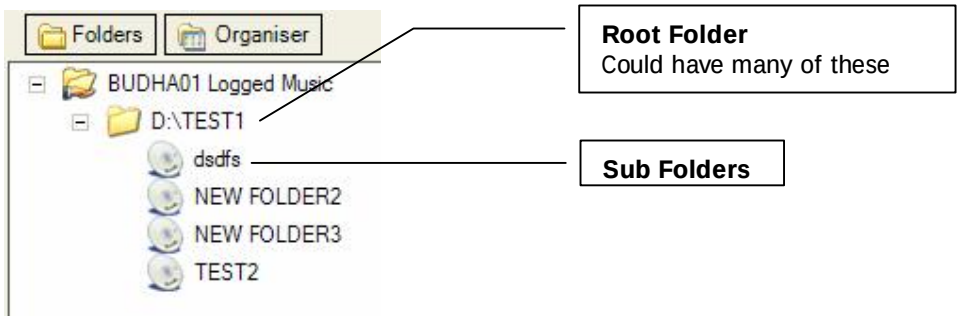

Figure 7-20 Media Library embedded FolderTree control

The folderTree control also makes heavy use of the database. The basic operation will be as follows

1st Time Run

1. The database tblFolders will be queried the 1st time the control is loaded (shown) to produce tree view root nodes. Initially there will be no folders to show in the folderTree as the user has not selected any folders to catalogue. So this query to the database for root folder should produce 0 rows.
2. The user will pick a folder to catalogue, where they will be prompted with the total size of this folder in Mbytes. Where they must confirm that they wish the scan to continue.
3. If the scan is confirmed, the application will log the folder from step 2 in the database and mark this folder as a root folder and shall store any MP3 details for files contained in this folder. The computer name where the folder is local shall also be stored within the database, so that the correct folders may be retrieved from the database the next time the FolderTree is initialized.
4. The folder will be explored recursively to establish all sub folders, each sub folder will be examined, and any MP3 details for files contained in these folders will be stored within the database.

Future Runs

Once the initial folders have been selected and logged within the database, the next time around the root folder names from the database (for the current PC name) will be used to populate the folderTree. Providing the user has actually selected 1 folder to catalogue there will be at least 1 root folder to add to the folderTree.

This is a nice feature as this time around, any sub folders that were not present the 1st time the root folder was scanned into the database, shall now be added to the database.

So it is like a FileSystemWatcher without the need for the application to be running constantly in memory.

7.2.5.5 File System Operations

In order to populate the database and the 2 Treeview controls (folderTree / GenreTree) the file system will have to be accessed in order to carry out the following tasks :

- Recursively build folderTree folder hierarchy.
- Create a MP3Tag class per mp3 file found, and store the attributes associated with the file within the database.
- It is anticipated that the scanning of files and database / Treeview population will be a time consuming operation. Therefore this scanning is done within a separate process thread. This did however lead to one complication, where the folderTree treeview control, is created using one thread, and the operations that populate the folderTree are on another thread, which caused control handle Invoke issues. To solve this, some asynchronous GUI code had to be used. This is illustrated in Figure 7-21 below.

```
/// <summary>
/// Asynchronously delegate to handle the Multithreaded call back. This delegate
///is a pointer to the address of the method that will be run when the Begin.Invooke
///is called for the FolderTree
/// </summary>
private delegate void addNodeToTreeDelegate(TreeNode srcNode, TreeNode newNode);

/// <summary>
/// Adds a new TreeNode to a source TreeNode as specified by the input parameters.
/// However as this class is multithreaded a call to InvokeRequired is required.
/// Controls in Windows Forms are bound to a specific thread and are not thread safe.
/// Therefore, if you are calling a control's method from a different thread,
/// you must use one of the control's invoke methods to marshal the call to the
/// proper thread. This property can be used to determine if you must call an invoke
/// method, which can be useful if you do not know what thread owns a control.
/// </summary>
/// <param name="srcNode">The source node to add new nodes to</param>
/// <param name="newNode">The new node to be added to the source node</param>
private void addNodeToTree(TreeNode srcNode, TreeNode newNode)
{

    //true if the control's Handle was created on a different thread than the calling thread
    //(indicating that calls to the control, must be made through an invoke method)
    if (this.InvokeRequired)
    {
        //Pass the same function to BeginInvoke, but the call would come on
        //the correct thread and InvokeRequired will be false.

        //BeginInvoke Executes a delegate asynchronously on the thread
        //that the control's underlying handle was created on.
        // Perform a BeginInvoke call to the FolderTree in order to marshal to the
        //correct thread.
        this.BeginInvoke(new addNodeToTreeDelegate(addNodeToTree),
            new object[] {srcNode,newNode});

        return;
    }

    //add the node, but now its coming from the same thread as the FolderTree control
    srcNode.Nodes.Add(newNode);
}
```

Figure 7-21 GUI Multithreading with .NET and Delegates

The Media Library also allows the user to customize the layout of their logged media, and also to show, and navigate through the details of the logged media,
Figure 7-22 shows this. This example has details shown from where the user may navigate using the hyperlinks, and the view type is set to IconView. Other choices are DetailsView, ListView and ThumbnailView.

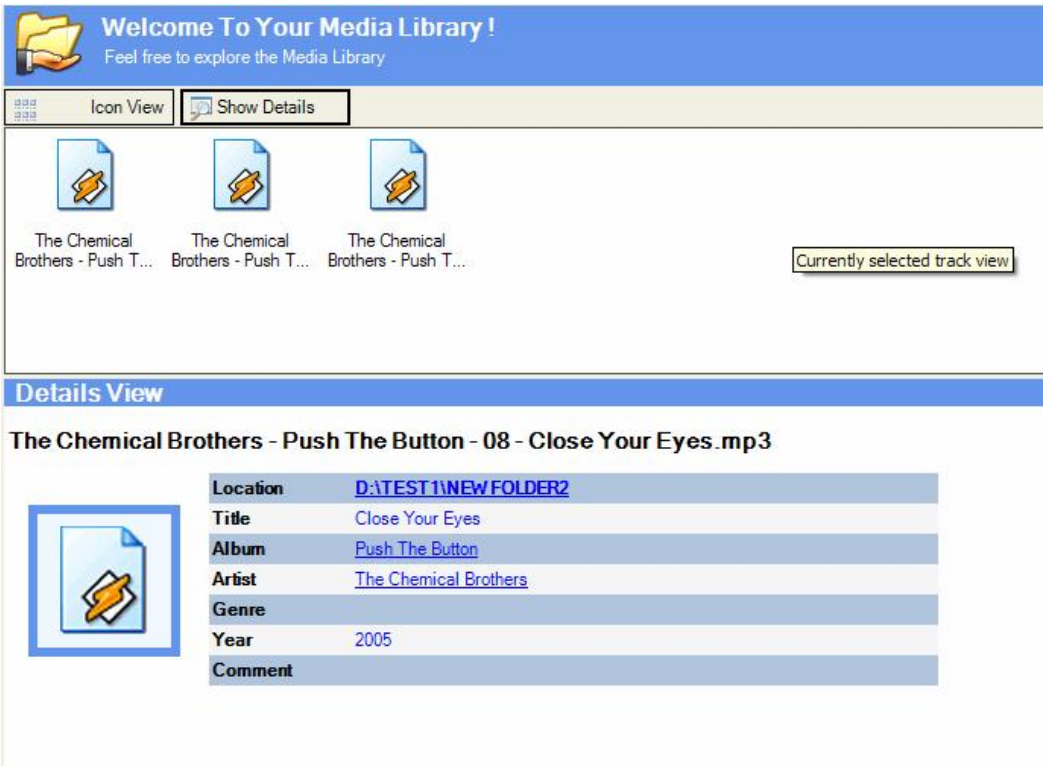

Figure 7-22 Media Library media item views

The user may also use either of the treeviews to add files to the Media Player (Server, see section 7.2.6) / Clienttracklist (Client, see section 7.3.2) controls for playing, by use of a right click context menu as shown in Figure 7-23 below.

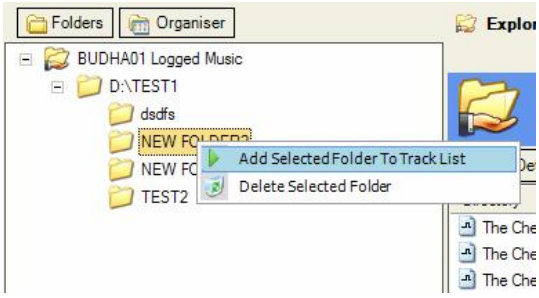

Figure 7-23 Adding Items To Playlist From Media Library

7.2.6 Media Player Control

The uctMediaPlayer, shown in Figure 7-24 below, control extends the System.Windows.Forms.UserControl and is hosted within the Server application.

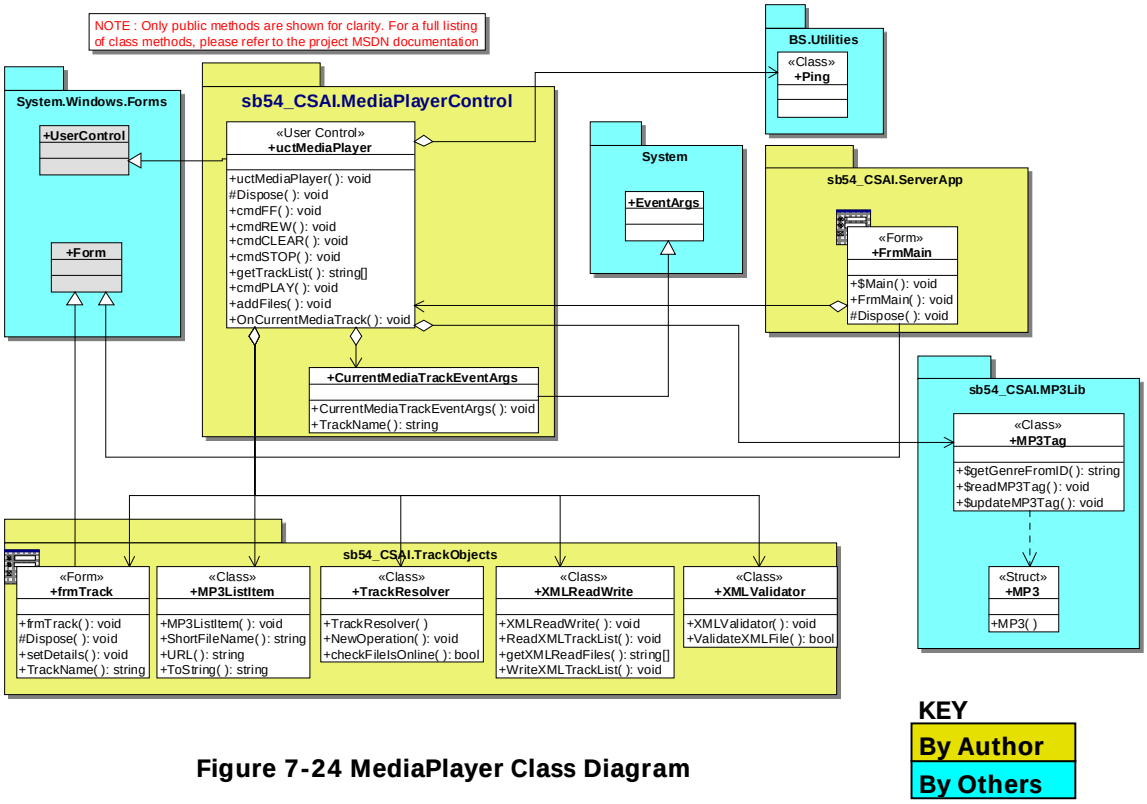

Figure 7-24 MediaPlayer Class Diagram

The main class is the uctMediaPlayer class which, is itself, hosting another user control. Namely the Windows Media Player (WMP) control. The uctMediaPlayer is effectively a wrapper around the embedded, code created WMP control, which is available within the WMP SDK. The WMP control provides an object model of the entire list of interfaces that are realized by the WMP itself.

The code created WMP control will have most of its FF/Reverse/Stop/Clear/load song functions controlled through code, for use by the ReMP3 Client. There are however, facilities available for the user, to manually stop the player, or adjust the volume of it, using its own internal GUI interface operations.

It should be noted that the Windows Media Player SDK relies on the Windows Media Player actually being installed, that is why the Windows Media Player is a pre-requisite.

In order to construct the code that is needed to play back music using the Windows Media Player SDK, the following class and events are catered for.

- **AxWindowsMediaPlayer**
- **WMPOCXEvents_PlayStateChangeEvent**
- **WMPOCXEvents_OpenStateChangeEvent**
- **WMPOCXEvents_MediaErrorEvent**

The uctMediaPlayer control is as shown in Figure 7-25 below.

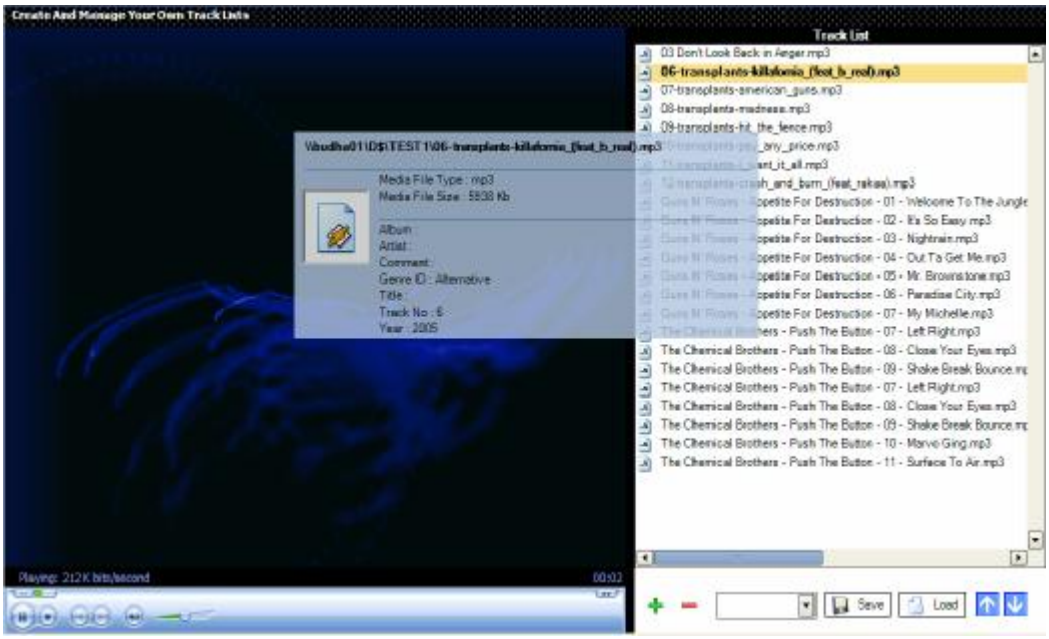

Figure 7-25 Media Player Control

The ReMP3 project can use any of the following methods to dictate what music is played on the embedded WMP control.

- Selection of music comes from user selection, or from catalogued music that is stored within the Media Library (see section 7.2.5)
- From music that has been dragged and dropped to the track list.
- From the Client, as a remote tracklist

The tracklist also allows the following features :

- Drag and drop tracks from the OS file system
- Load tracks by browsing
- Load and save tracks to XML files
- Use CTRL+SHIFT + DEL to delete tracks
- Reorder tracks using up/down arrows

7.2.6.1 TrackObjects

Since the uctMediaPlayer Control shares some features with the Client ClientTrackList control (see section 7.3.2), it seemed to make sense to make these common classes into a separate, generic, reusable package. This package is called TrackObjects, and carries out various tasks.

FrmTrack : Is a class that simply provides a popup transparent form, with the currently hovered MP3 file details shown. An example of this as shown in Figure 7-26 below.

Figure 7-26 frmTrack popup details

MP3ListItem : Extends System.Windows.Form.ListItem and simply provides a more specific listitem, that holds the MP3 short name, and full URL and can be used to add items to the Tracklist listview.

TrackResolver : Uses a Ping library, that was obtained from the codeproject [2]. The author of the original classes [24] is Wesley Brown..

The Ping library is used to validate whether a certain track is available. This is used when loading tracks from XML files, as the XML files may contain references to tracks that are not local, and the machine the tracks actually reside on, may not be available when trying to load the tracks from the XML file.

XMLValidator : As stated the ReMP3 project allows tracklists to be saved/loaded using XML files. In order to allow this, a XMLSchema needed to be developed that could be used to validate input XML files against.

The XMLSchema that is used is shown in Figure 7-27 below :

```
<?xml version="1.0" ?>
<xs:schema id="ReMP3TrackList" targetNamespace="http://tempuri.org/ReMP3TrackListSchema.xsd"
    xmlns:mstns="http://tempuri.org/ReMP3TrackListSchema.xsd"
    xmlns="http://tempuri.org/XMLFile1.xsd" xmlns:xs="http://www.w3.org/2001/XMLSchema"
    xmlns:msdata="urn:schemas-microsoft-com:xml-msdata"
    attributeFormDefault="qualified" elementFormDefault="qualified">
    <xs:element name="TrackList">
        <xs:complexType>
            <xs:sequence>
                <xs:element name="Track" nillable="true" minOccurs="0" maxOccurs="unbounded">
                    <xs:complexType>
                        <xs:simpleContent msdata:ColumnName="Track_Text" msdata:Ordinal="0">
                            <xs:extension base="xs:string"></xs:extension>
                        </xs:simpleContent>
                    </xs:complexType>
                </xs:element>
            </xs:sequence>
        </xs:complexType>
    </xs:element>
</xs:schema>
```

Figure 7-27 ReMP3TrackListSchema

XMLReadWrite : In order to carry out the XML storing/retrieval the following need to be provided :

- **XML Writing** : A XMLWriter method to construct the XML file which contains the track list data. The writing of XML will conform to a predefined XML schema.
- **XML Reading**: Will need a XMLReader method to read the XML file which the user created. The XML input file will firstly be validated against an existing XML schema (XSD) document. Only if the XML input file conforms to the schema will it be parsed to form new track list information.

The only element type that will be parsed (All others will be read, but shall not be parsed) is <xs:element name="Track"> this ensures that even if a bogus XML file gets through the validation process, it must still contain the correct type of elements.

A further check is also performed such that, the element data must be found to be valid audio files that exist, before the element data is added to the tracklist. With these checks in place the recreation of a tracklist from an XML file should be quite safe.

7.2.7 CDRip

The CDRip Package contains 2 user controls (as shown in Figure 7-28), that are hosted within the Server application.

- uctCDRipper : See section 7.2.7.1
- uctWAVtoMP3Convertor : See section 7.2.7.2

These 2 user controls owe much of their functionality to a group of packages that I obtained from a C# chat group at the c-sharpcorner [21]. These packages are :

- Yeti.MMedia
- Yeti.MMedia.MP3
- Ripper

The author of the original classes is Idael Cardosa.

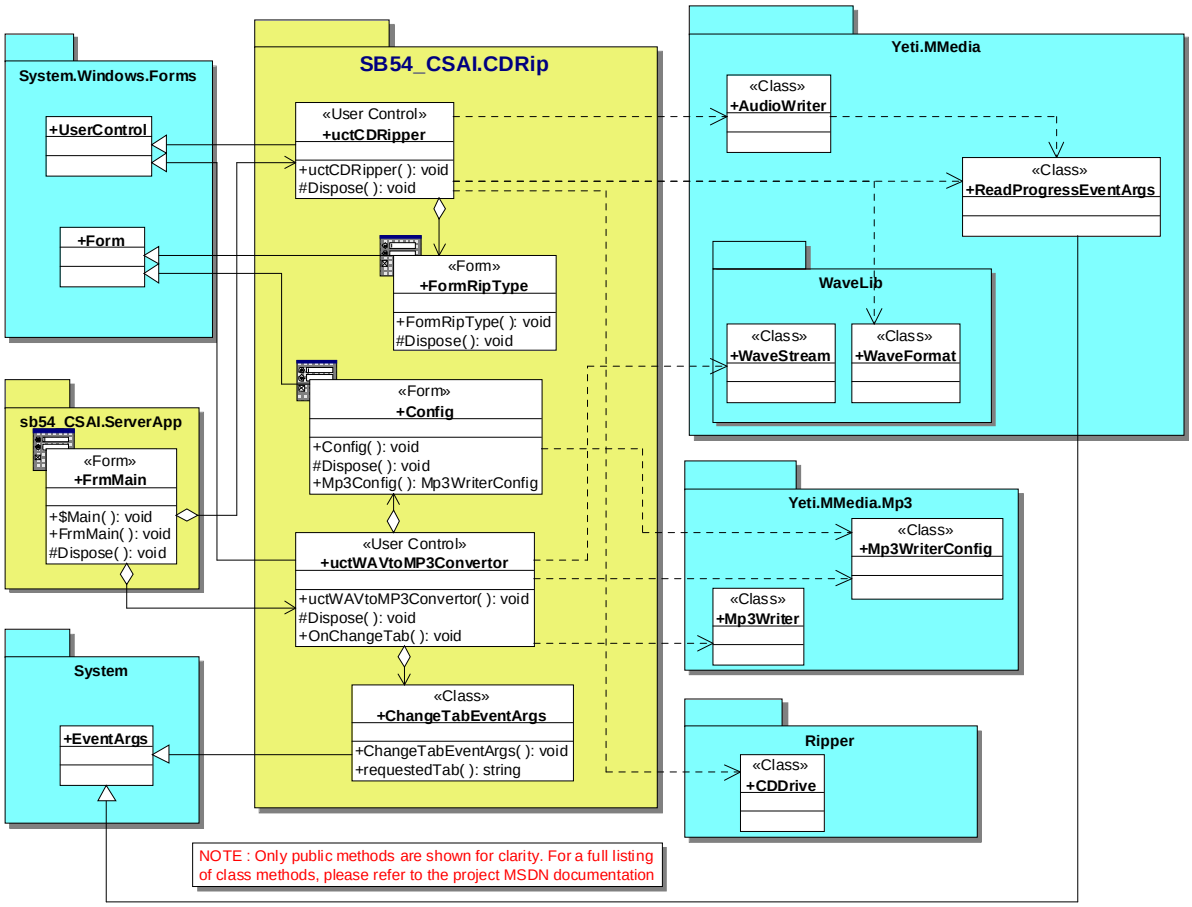

Figure 7-28 CDRip Class Diagram

7.2.7.1 uctCDRipper

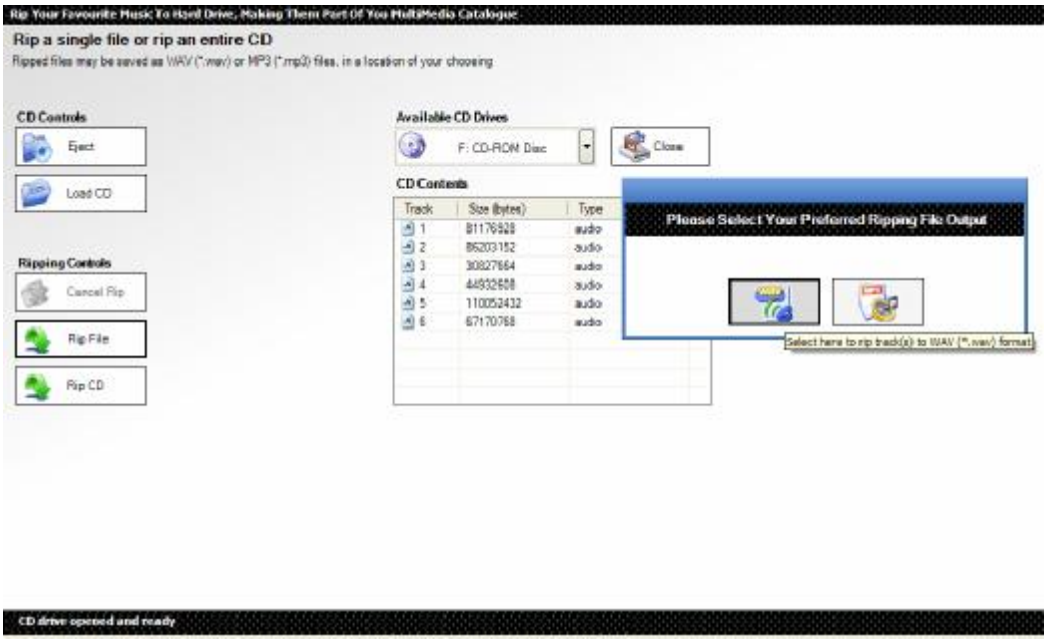

Figure 7-29 uctRipper Control

The uctCDRipper control extends the System.Windows.Forms.UserControl and simply provides a means of ripping CD audio to either WAV or MP3. The uctRipper control is shown in Figure 7-29 above.

What Idaels Yeti.MMedia and Ripper packages actually do, are to provide a number of stream and configuration classes that enable the ripping from a CD drive (CDDRive class) to either WAV or MP3. The Yeti.MMedia.MP3 package also depends on a number of other Dll's namely the following :

- Kernel32.dll : Which is a core Windows OS dll
- winmm.dll : Which is the Windows multi media dll

Although the uctWAVtoMP3Convertor control makes use of Idaels Yeti.MMedia and Ripper packages, to carry out the ripping to MP3, there is still quite a large amount of wrapper code, that needed to be provided, in order to use the classes contained within the Yeti.MMedia and Ripper packages.

Such as providing different stream objects that interact with the Yeti.MMedia library. These streams also have to have events connected, to alert the GUI of the ripping progress. There is also a lot of error handling code concerned with the correct operation of these stream objects, that had to be implemented by the ReMP3 project by the ReMP3 author.

Namely the following :

- Creating a list of local CD drives, which actually uses a rather nice object called a ManagementObjectSearcher. Which works with what look like SQL Queries, and yields the system resources requested by the query string. The following code is used to get a list of local CD drives for the GUI combobox :

```
//CD type drive = 5
const int CD   = 5;
// get a list of all drives on users system, then filter out CD Drives and add
//them to the CD drive combobox (cmbDrives)
ManagementObjectSearcher query = new ManagementObjectSearcher("SELECT * From Win32_LogicalDisk ");
ManagementObjectCollection queryCollection = query.Get();
foreach ( ManagementObject mo in queryCollection)
{
    int diskType = int.Parse(mo["DriveType"].ToString());
    if (diskType == CD)
    {
        cmbDrives.Items.Add (mo["Name"] + " " + mo["Description"]);

    }
}
```

- Controlling the CD tray operations via the GUI, this is by interacting with the CDDrive class.
- Getting the table of contents for the CD, this is by interacting with the CDDrive class.
- Controlling the ripping involves, managing a WaveStream and a either a WaveWriter / Mp3Writer steam object (dependant on current RipType). The GUI must also be updated with the status of the amount of data ripped.

7.2.7.2 uctWAVtoMP3Convertor

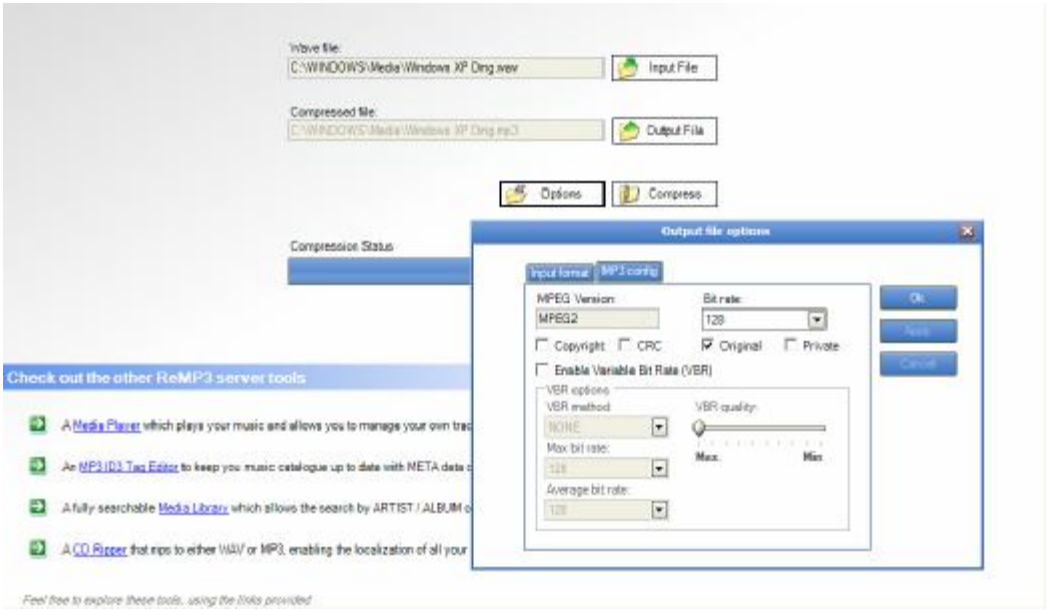

Figure 7-30 UctWAVtoMP3Convertor Control

The uctWAVtoMP3Convertor control extends the System.Windows.Forms.UserControl and simply provides a means of converting a WAV file to an MP3 file. The control is shown in Figure 7-30 above.

What Idaels Yeti.MMedia.MP3 package actually does, is to provide a number of stream and configuration classes that enable the compression to MP3, from WAV. The Yeti.MMedia.MP3 package also depends on a number of other Dll's namely the following :

- Lame_enc.dll : Which is a open source MP3 encoder [25]
- winmm.dll : Which is the Windows multi media dll

The Lame_enc.dll, is required to be located on the host system, as such this Dll forms part of the ReMP3 project installer.

Although the uctWAVtoMP3Convertor control makes use of Idaels Yeti.MMedia.MP3 package, to carry out the conversion to MP3, there is still quite a large amount of wrapper code, that needed to be provided, in order to use the classes contained within the MP3 package.

Namely the following :

- The Input button : Needs to create a WaveStream and use this to create a new Mp3WriterConfig, which is a class within the Yeti.MMedia.MP3 package.
- The options button allows the user to alter the default Mp3WriterConfig to suit their own needs.
- The Compress button must create and manage a WaveStream and a Mp3Writer steam object. It must also update the status of the amount of data compressed.

7.2.8 RemoteInterfaces

As the remote control operations are more to do with the Client than the Server, this is described in full at section 7.3.3.

7.3 ReMP3 Client Package Descriptions

The following sub sections describe the functionality of the ReMP3 Client application.

7.3.1 ClientApp

The class diagram for the ClientApp is shown in Figure 7-31 below. At its most basic level the ClientApp is simply acting as a container for other user controls, and issuing remote control instructions to the ReMP3 Server application.

However prior to the main ClientApp GUI form (frmClient) being shown the user must interact with a preloader (frmClientLoader) which shows the user a list of network computers. From where, the user must select a computer which has the ReMP3 Server application running. If the ClientApp can connect to the user supplied ReMP3 Server, the main ClientApp GUI form is shown, from where the user may interact with the 2 embedded user controls :

- ClientTrackList control : See section 7.3.2
- Media Library control : See section 7.3.5

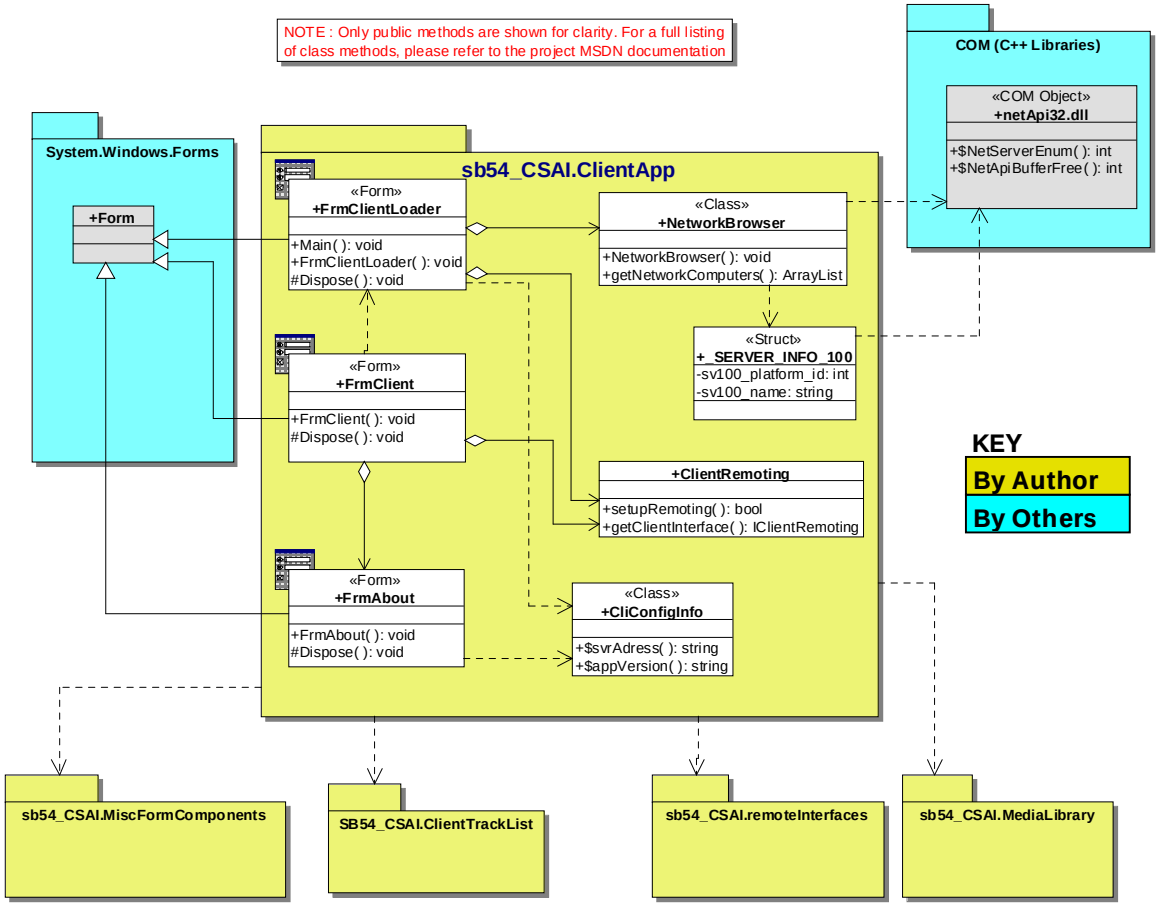

Figure 7-31 ClientApp Class Diagram

To use the ReMP3 Client remote controls features, the following steps must be taken.

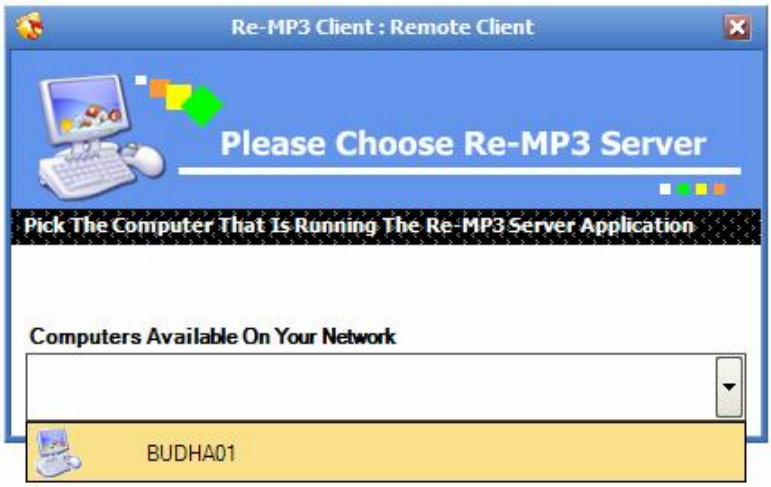

Figure 7-32 ReMP3 Client Loader Screen

The ClientApp.Exe is launched, this causes the preloader (frmClientLoader) screen to be shown (Figure 7-32, above) with a list of available LAN computers shown. The user must then select from the list presented to proceed to connect to the ReMP3 Server application. If a successful connection to the ReMP3 Server occurs, the ReMP3 Client main GUI form (frmClient) is shown. As shown in Figure 7-33 below. It should also be noted that contextual help is provided by using the F1 key.

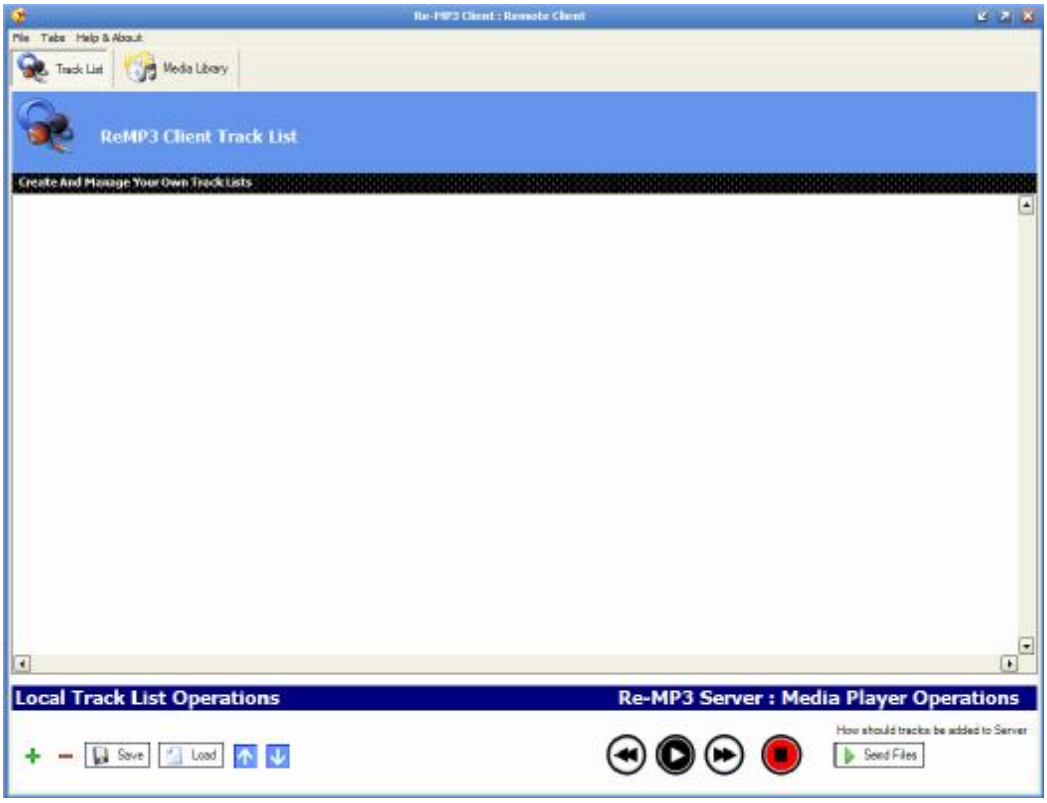

Figure 7-33 ReMP3 Client Main GUI Form

To fully understand how the ReMP3 client application works, consider the sequence diagram, Figure 7-34. Which shows the various forms and objects that are required to correctly launch the ReMP3 client application.

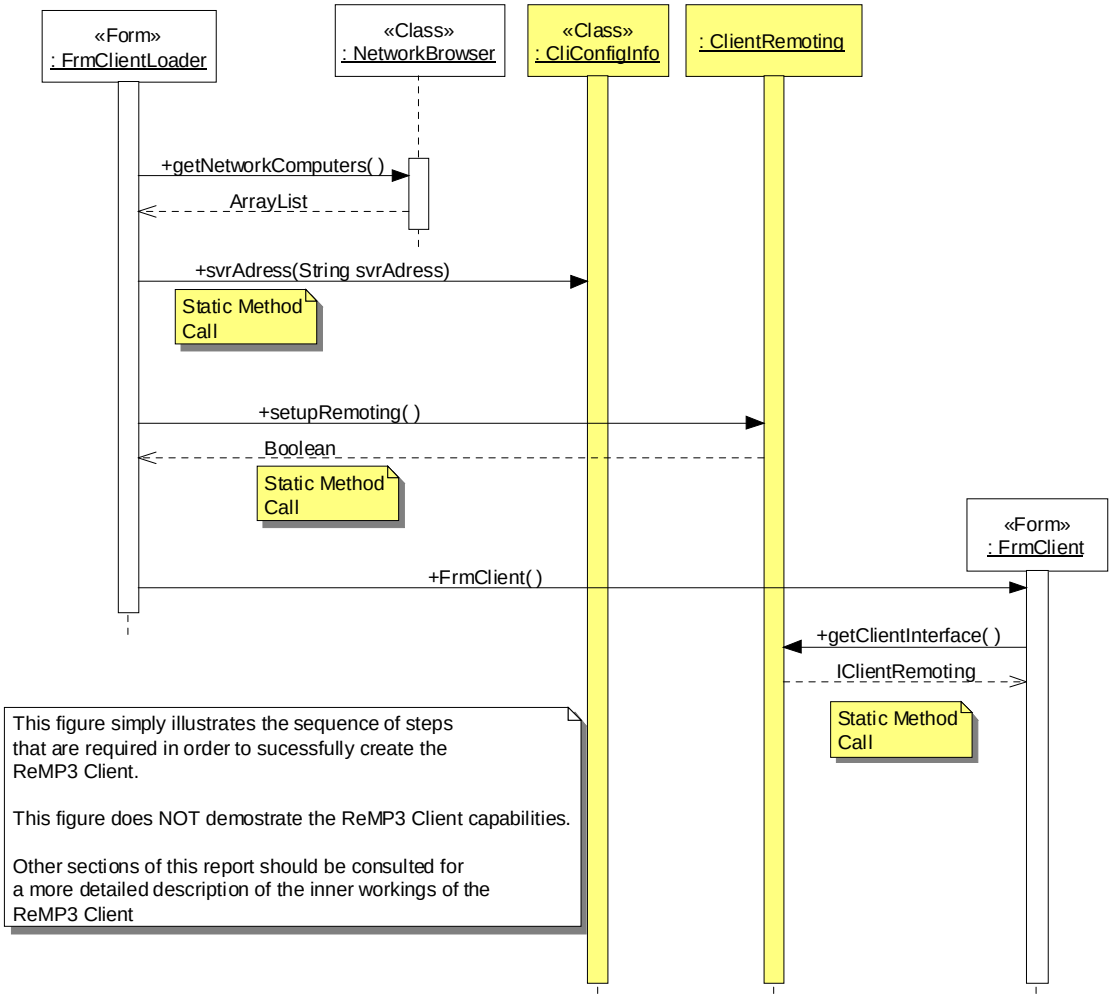

Figure 7-34 ReMP3 Client Application Startup

7.3.3.1 Network Browsing

As previously stated the .NET framework does not support network functions like that required by the ReMP3 Client loader screen. So how is this functionality achieved ?

Luckily Microsoft incorporated into the .NET language the ability to use 3rd party non .NET Dlls. These Dll's must still be COM compliant.

"To consume exported DLL functions

1. *Identify functions in DLLs.*
   Minimally, you must specify the name of the function and name of the DLL that contains it.
2. *Create a class to hold DLL functions.*
   You can use an existing class, create an individual class for each unmanaged function, or create one class that contains a set of related unmanaged functions.
3. *Create prototypes in managed code.*
   *[Visual Basic] Use the **Declare** statement with the **Function** and **Lib** keywords. In some rare cases, you can use the **DllImportAttribute** with the **Shared Function** keywords. These cases are explained later in this section.*
   *[C#] Use the **DllImportAttribute** to identify the DLL and function. Mark the method with the **static** and **extern** modifiers.*
   *[C++] Use the **DllImportAttribute** to identify the DLL and function. Mark the wrapper method or function with **extern "C"**.*
4. *Call a DLL function.*
   Call the method on your managed class as you would any other managed method. Passing structures and implementing callback functions are special cases. "

Taken from .NET Framework Developer's Guide, Consuming Unmanaged DLL Functions [26]

So this is exactly what is done within the ReMP3 ClientApp NetworkBrowser class. Where the following Functions are used from the NETAPI32.dll.

These methods are described at the Microsoft Platform SDK : Network Development [27], as shown in Figure 7-35 below.

```
Declare Function NetServerEnum Lib "NETAPI32.dll" ( _
        ByRef servername As LMCSTR, _
        ByVal level As Long, _
        ByVal bufptr As String, _
        ByVal prefmaxlen As Long, _
        ByRef entriesread As Long, _
        ByRef totalentries As Long, _
        ByVal servertype As Long, _
        ByRef domain As LMCSTR, _
        ByRef resume_handle As Long) As Long

Declare Function NetApiBufferFree Lib "NETAPI32.dll" ( _
        ByRef Buffer As Any) As Long
```

Figure 7-35 NetAPI32.Dll Functions

The method definitions within Figure 7-35, are the native definitions within the NETAPI32.dll. This is not how the methods will look in .NET. Some work needed to be done in order to create the correct types and structures within the managed .NET code, so that the

unmanaged structures can be presented as managed structures, and then used when calling the imported methods.

The .NET code for these imported DLL Methods, is as shown in Figure 7-36 below :

```
//declare the Netapi32 : NetServerEnum method import
[DllImport("Netapi32", CharSet=CharSet.Auto, SetLastError=true),
SuppressUnmanagedCodeSecurityAttribute]

/// <summary>
/// Netapi32.dll : The NetServerEnum function lists all servers
/// of the specified type that are visible in a domain. For example, an application can call
/// NetServerEnum to list all domain controllers only or all SQL servers only.
/// You can combine bit masks to list several types. For example, a value of 0x00000003
/// combines the bit masks for SV_TYPE_WORKSTATION (0x00000001) and SV_TYPE_SERVER (0x00000002)
/// </summary>
public static extern int NetServerEnum(
    string ServerNane, // must be null
    int dwLevel,
    ref IntPtr pBuf,
    int dwPrefMaxLen,
    out int dwEntriesRead,
    out int dwTotalEntries,
    int dwServerType,
    string domain, // null for login domain
    out int dwResumeHandle
    );

//declare the Netapi32 : NetApiBufferFree method import
[DllImport("Netapi32", SetLastError=true),
SuppressUnmanagedCodeSecurityAttribute]

/// <summary>
/// Netapi32.dll : The NetApiBufferFree function frees
/// the memory that the NetApiBufferAllocate function allocates. Call NetApiBufferFree
/// to free the memory that other network management functions return.
/// </summary>
public static extern int NetApiBufferFree(
    IntPtr pBuf);

//create a _SERVER_INFO_100 STRUCTURE
[StructLayout(LayoutKind.Sequential)]
public struct _SERVER_INFO_100
{
    internal int sv100_platform_id;
    [MarshalAs(UnmanagedType.LPWStr)]
    internal string sv100_name;
}
```

Figure 7-36 .NET Dll NETAPI Methods

This class owes much to the assistance offered, by a friend Nick Cross, a C++ programmer, who actually explained the NETAPI documentation, in the first place. The translation into correct .NET structures and types was simply a very long winded exploration of the MSDN documentation, the result of which is the code that is contained within the NetworkBrowser class.

7.3.2 ClientTrackList

The uctClientTrackList extends the .NET System.Forms.UserControl, and is hosted within the ReMP3 Client App main GUI form (FrmClient). The ClientTrackList is very similar to the

ReMP3 Server MediaPlayer control, in many ways. There is in fact, a common set of tasks that these 2 controls need to perform. As such these tasks have been placed within a separate package called TrackObjects, which was previously explained at section 7.2.6.1.

The common elements that are the same across these 2 user controls are as follows :

- The track list itself
- The adding and clearing of tracks
- The saving and loading of tracks using XML files
- The re-ordering of tracks

The class diagram for the ClientTrackList is shown in Figure 7-37 below.

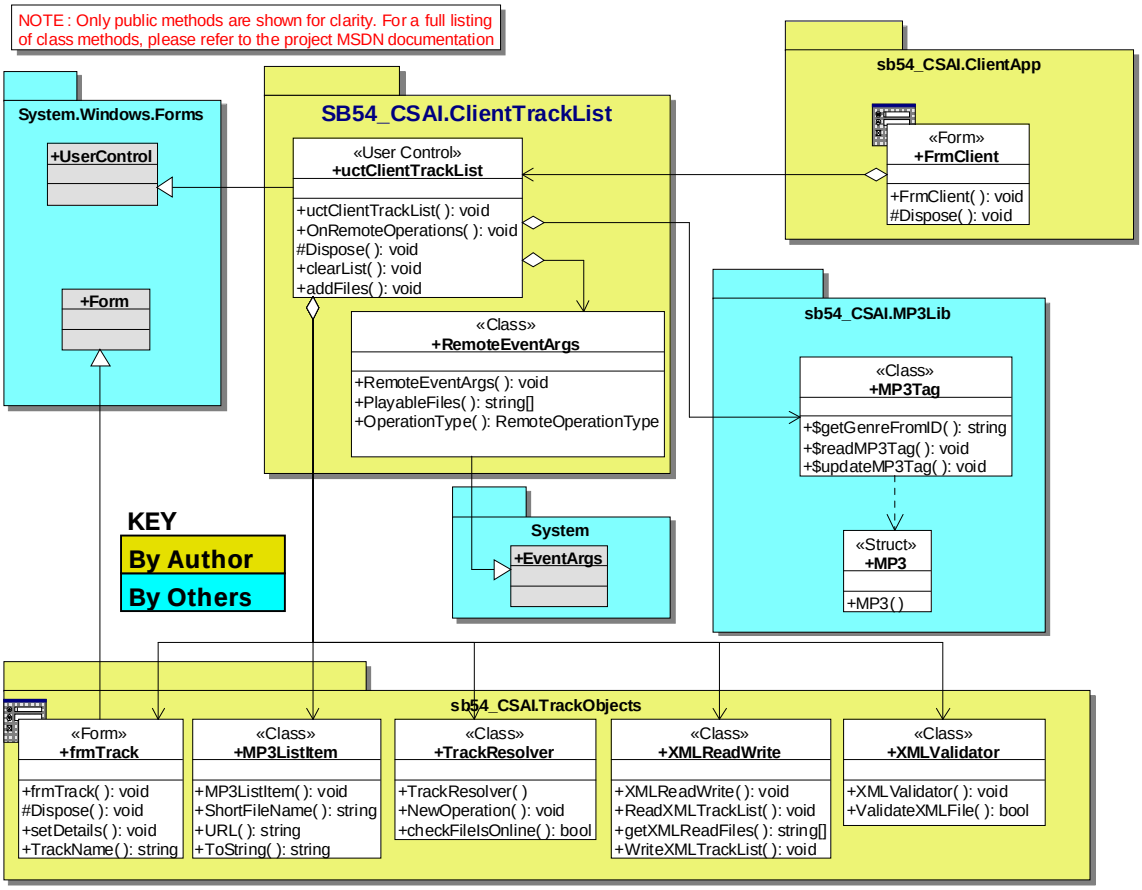

Figure 7-37 ClientTrackList Class Diagram

As stated the ClientTrackList, is a control that is hosted within the ReMP3 Client form (FrmClient). The format of the ClientTrackList is as shown in Figure 7-38 below.

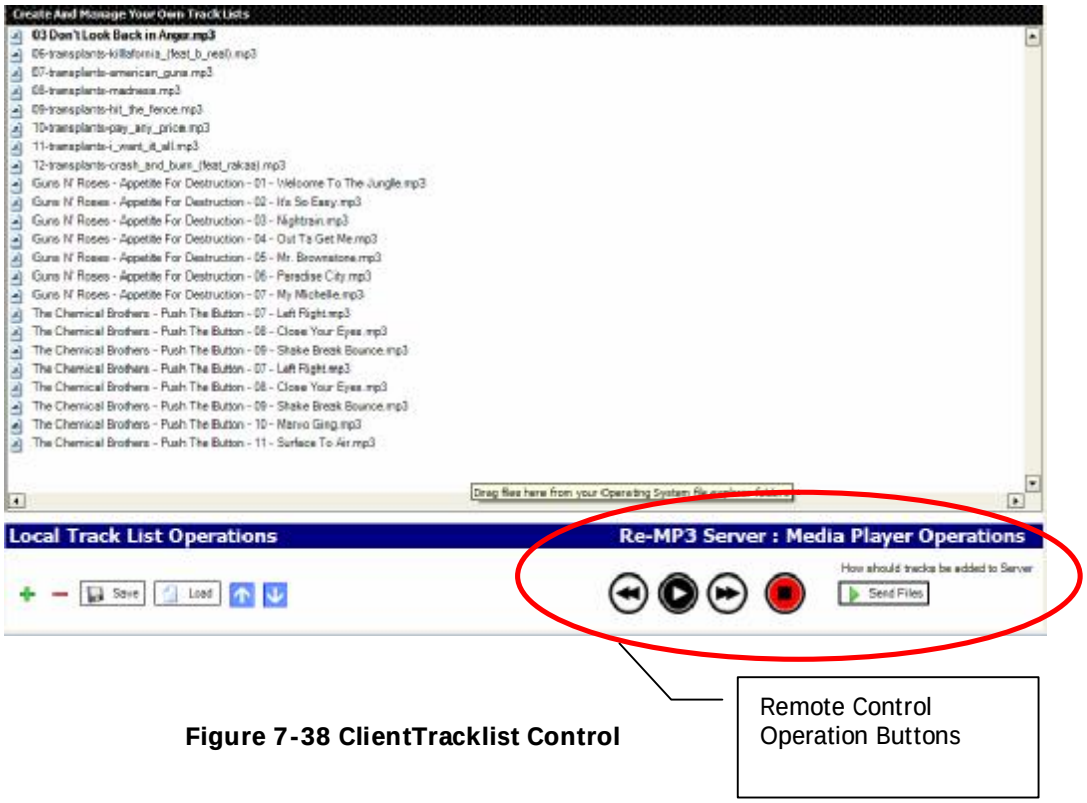

Figure 7-38 ClientTracklist Control

The ClientTrackList also provides remote control operations. The remote control operations are carried out by the use of the RemoteInterfaces package, which is explained below.

7.3.3 RemoteInterfaces

The RemoteInterfaces package is the core package, that allows the Remote Control functionality of the ReMP3 Server / Client applications. Before delving into how the remote control functionality is achieved, it is important to understand how .NET Remoting actually works.

7.3.3.1 .NET Remoting

.NET Remoting can be used for accessing objects in another application domain, for example on another server. Remoting can be offered from any .NET application type, for example a windows service or a windows application (This is the case for the ReMP3 project).

.NET Remoting offers great flexibility for the use of different network protocols and control of the formatting of any data that is sent across application domains. The programming model is the same whether objects are used on the server or on the client.

It is by using .NET Remoting, that the client informs the running server of new play lists, and how remote control operations such as FWD 1 TRACK, STOP, REVERSE 1 TRACK are achieved.

When an object is remoted using .NET Remoting, the user of the remote object is able to use the object as if it were a local object. This is to say the object methods may be called, properties may be set etc.

7.3.3.2 Using .NET Remoting

In order to use .NET remoting there are several key operations that must be performed. A simplified listing of these operations is shown below :

- A remotable class must be constructed that inherits from the System.MarshalByRefObject**,** Which enables access to objects across application domain boundaries in applications that support .NET remoting.
- **At the Server endpoint**, a new TCPChannel or HttpChannel must be created to host the remotable object. This must be given a port number.
- **At the Client endpoint**, a new TCPChannel or HttpChannel must be created to register the host remotable object (proxy object). This must be given a port number.

7.3.3.3 .NET Remoting to Perform Remote Control Functions

Figure 7-39 below, illustrates the .NET Remoting configuration that the ReMP3 project implements. Figure 7-39 shows the entire Remoting set-up for both the ReMP3 Server and the ReMP3 Client.

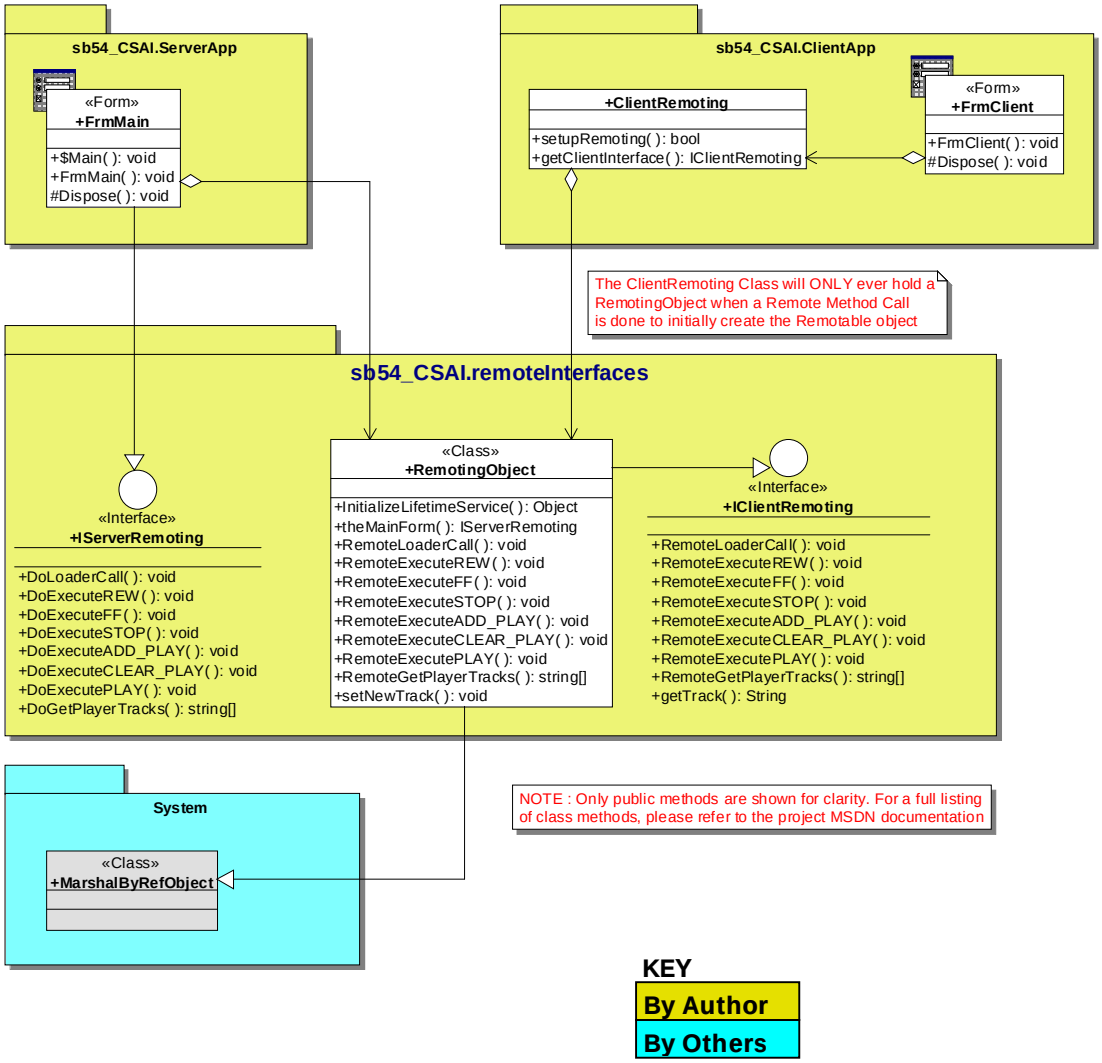

Figure 7-39 RemoteInterfaces Class Diagram

.NET Remoting set-up from the ReMP3 Server End Point

The ReMP3 server is created first. The ReMP3 server when constructed actually creates, hosts, and initialises the object lease time, of a new RemotingObject. The initial object lease time parameters are set at the following:

- InitialLeaseTime = 12 Hours
- SponsorshipTimeout = 12 Hours
- RenewOnCallTime = 24 Hours

This RemotingObject inherits from System.MarshalByRefObject, which is a prerequisite of creating a successful remoting application.

The ReMP3 Server also implements an interface call IServerRemoting, this interface contains method signatures for all of the remote operations, that the Server must respond to.

Next the ReMP3 Server application sets a field within the RemotingObject, to be a reference to the instantiated ReMP3 Server object (basically “this”, within the ServerApp code). So that any call made remotely to the RemotingObject, can then be re-directed to the instantiated ReMP3 Server object, where it will effectively be then, calling methods on the instantiated ReMP3 Server object itself.

This may seem long winded, but is necessary, as within .NET and many other languages, there is only allowed to be a single layer of class inheritance, and as the ReMP3 Server form is already inheriting from the System.Forms.Form, it can not inherit from another base class (System.MarshalByRefObject as it would need to). So a new intermediate class had to be developed, enter the “RemotingObject” class. So in order for the Client to communicate with the Server, the RemotingObject needs to hold a reference to the Server object. The IServerRemoting and IClientRemoting interfaces have been developed to facilitate this communication.

Once the RemotingObject has been created, and hosted, the Server is effectively waiting for some Remote request to come from the client, on the hosting channel, so the Server is free to carry out any other tasks, that the Server GUI event loop currently holds that need immediate servicing.

.NET Remoting set-up from the ReMP3 Client End Point

When started, the Client will use the name of the computer provided by the loader screen, to try and get a connection to the Server hosted RemotingObject. If the Client provides a name to a machine that is hosting the RemotingObject, a http channel to that machine is opened to use for inter computer communications.

If the Client manages to gain a connection to the Server, the Client will then be able to get a reference to the RemotingObject at the Server. The Client is then free to call these RemotingObject methods as if they were local to the Client.

There are actually 2 types of RemoteOperations that the Client performs.

1. Operations that either expect no response, or will get an immediate response from the Server, which are as follows :

    - `void RemoteExecuteREW();`
    - `void RemoteExecuteFF();`
    - `void RemoteExecuteSTOP();`
    - `void RemoteExecuteADD_PLAY(string[] files);`
    - `void RemoteExecuteCLEAR_PLAY(string[] files);`
    - `void RemoteExecutePLAY();`
    - `string[] RemoteGetPlayerTracks();`

    These operation types are very easy to deal with, and are handled as follows:

    - The ClientTrackList Control raises the OnRemoteOperations event, every time a Remote Control Operation button is clicked. (See Figure 7-38)
    - The Client main GUI form (FrmClient) receives this event, and then decodes the event type using the RemoteEventArgs, and then calls the appropriate method on the RemotingObject, passing over any required values from the RemoteEventArgs.
    - The RemotingObject, then calls the corresponding method on its internal reference to the Server object, again passing over any required values.
    - The Server object carries out the remote operation request, and sends the return value, if one is required.

2. Operations that may have to WAIT for a period of time, UNTIL some event occurs at the Server, where the operation will get a response. There is only one of these which is when the Server Media Player current track changes (SetNewTrack() method on the RemotingObject), and the Client TrackList control is expected to show which is the new media item playing at the Server. This event MUST be generated by the Server, as this is where it is actually occurring. The Client knows nothing about the event, so MUST be told about it by the Server.

    This was actually quite a complicated issue, that took some time and several meetings with Dr Phil Whatton to implement. These meetings are discussed within the Project Log at section 10.

    Consider this scenario :

    The Server instantiates the RemotingObject, then the Client connects to the Server and also creates the host RemotingObject, where the Client is presented with a proxy to the real object on the Server. In the mean time the current media item within the Server hosted Media Player control changes, the Server can see this change by way of an event raised within the Media Player control, but how does the Server tell the Client about this.

    The normal Remoting operation described in type 1 RemoteOperations (and also in many text book examples) has been that the Client asks the Server to do something

via a method call on the RemotingObject, and the Server responds, by doing what has been asked for, and sending back a return value if one is required.

This mode of operation is clearly not sufficient to carry out the scenario previously outlined, where the Server has to inform the Client that something has happened, without the Client first having asked.

The main issue here is synchronization across a Remotable object. As the Server hosts the remotable object, it is free to get the RemotingObject to raise an event. However the inner workings of .NET Remoting, prohibits the Client from subscribing to this event (I initially explored this approach, which I expected to fail, it did fail, due to securityViolations across domain access.). So an alternative approach had to be found.

Threading seemed to provide the answer. Which was used successfully, as follows:

- When the main Client GUI form is constructed it gets an instance of the RemotingObject to allow communication to the Server
- It also creates a new Thread that will handle this one Media Change Event (GetTrack method on IClientRemoting interface), the normal Main thread of the Client will still continue to allow all other operations/interactions to be dealt with.
- The new thread calls the getTrack() method on the RemotingObject viewed as IClientRemoting interface.
    - If the Server has something to tell the Client on this new thread, the value is returned. And the Client Simply recursively calls this method.
    - If the Server has nothing to tell the Client on this new thread, the thread is BLOCKED and forced to WAIT, UNTIL the Server has something to tell the Client (via the setNewTrack() method on the RemotingObject), at which time the thread is UNBLOCKED and the client may use this new value to change the display of the current Server Media item on the Client TrackList control. The Client recursively calls this method.

  This is illustrated within Figure 7-40.

NOTE : This Threading solution is based on an initial concept that Dr Phil Watten and I developed during a meeting in Term1.

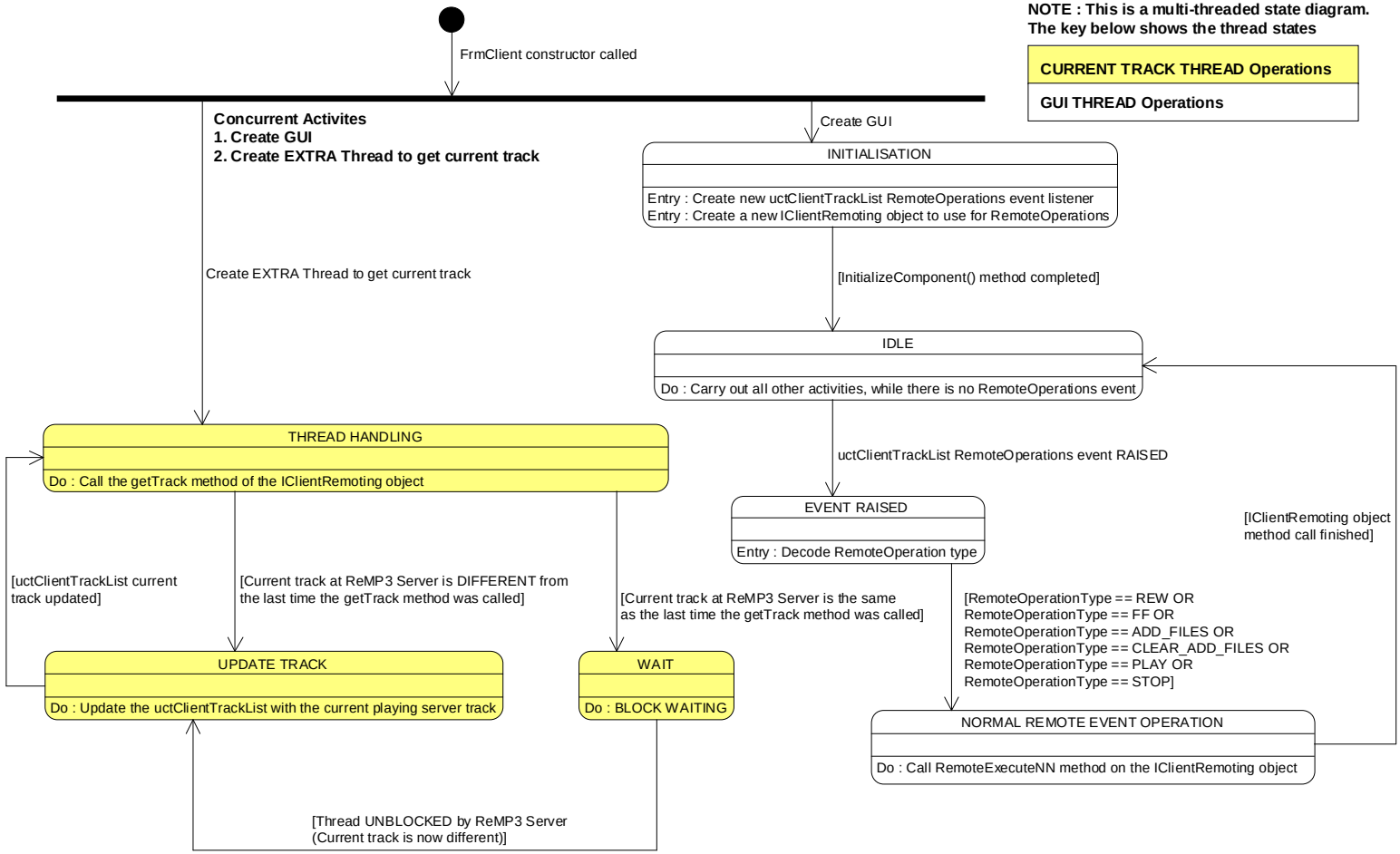

Figure 7-40 Remote Control Operations

7.3.4 MiscFormComponents

As described in section 7.2.3.

7.3.5 Media Library

As described in section 7.2.5, with the one exception that the catalogued music will be that of the current ReMP3 Client computer.

8. TESTING

Testing is obviously a large part of any software product. Testing is an iterative process that has been conducted throughout the entire life cycle of the project. Where changes have been required to move the project from one stage to the next, these changes have been made, in an informal non recorded manner.

As the final elements of the project were constructed to form the entire application, a test specification was developed that detailed how the system was to be tested as a whole.

This test specification is a separate document that is contained within Appendix A.

The testing is organised into small units concerned with the separate areas of the application. Each of these units of tests, has a number of tests that must be completed and signed of. If a test fails, a fault report is recorded against the test, and that test is not signed off.

Fault reports are logged, and revisited, and rectified (where possible), and the rectification is recorded such that when reviewed the tester can see what was changed.

9. CONCLUSION

The aim at the outset was to produce a free tool useful for users whom wish to control the audio playback of a master media computer using a remote control application, that was easy and intuitive to use.

Overall, the project has been successful. As there were very few faults recorded, and most of the functionality within the Requirements Specification works well. The only downfall is the lack of support for full UNC path names within the .NET framework. Such as a path like \\budh01\d:\directory1\file1.mp3 can not be used by the .NET IO classes. This issue forces the user into sharing Administrative share folders, like D$ which represents the D:\ drive that they may not actually like to share.

Considerable effort has been made to produce a good, usable GUI, by applying sound HCI principles to the design of the application. This is discussed in section 4.6 of this document, and also Appendix A of the previously submitted Interim Report.

For the most part, this goal has been achieved. The interface adheres to good principles of usability, and is of a sufficiently high standard to rival many commercial applications. The GUI is skinnable which is indeed quite a nice feature. I also spent a long time scouring the codeproject [2] for some nice GUI components. I Ended up using the MenuIcons, and ShrinkPanel libraries.

There are many, many opportunities for extensions and improvements of the application, such as recording of sound, and making the application internet enabled.

On a more personal level, two of the main goals of this project were to become familiar with the .NET Remoting features, and to create a GUI that looked as good as some of the better designed GUIs, such as Microsoft Windows XP, where there has been considerable time and expense spent to make the system look and act in a reasonable manner. On this level, the project has been a tremendous success, as I have learnt an incredible amount over the last year, both concerning .NET, and also what is involved with making a Photoshop generated design into reality in code.

The C# language itself was also a challenge at times. C# is similar to java in many ways, there is however the added complication of having to sometimes interact with old COM DLLs. This bridging to unmanaged code is anything but straight forward, and can at times be very frustrating, but also very rewarding once working.

9.1 Future Work

The ReMP3 employs technologies that could easily be ported to a web site or even transported to web services. To this end the system could be altered to provide ReMP3 Client / ReMP3 Server applications using ASP .NET or web services technology, such that a client machine would not have to load any software, but could instead simply use their web browser to initiate conversation with the ReMP3 server, which would also be hosted on a web site.

10. LOG

This section includes minutes of meetings with the project supervisor and material consulted so far.

APPENDIX A – TEST SPECIFICATION

APPENDIX B – CODE LISTINGS

Due to the sheer amount of code that would be have to be printed, the project supervisor (Paul Newbury) agreed the code listings could be hosted on the Project web site at the following URL http://www.vibrant-web-design.com/sacha/sb54_CSAI_Proj.htm

This will be the only source for the code listings. The individual class listings shall be hosted as Adobe Acrobat Reader files (*.pdf).

All the program code is presented in package order. Documentation about what the classes do can be found mainly in the <SUMMARY> xml comment tags, although there are also individual line comments. Each method has a comment preceding it explaining its purpose.

There is also separate MSDN style documentation that will be made available on CD, that simply contains the API style commenting of what the classes and methods do. This MSDN documentation is created using the XML code comments within the code, and the NDoc documentation tool [15], section 6.3.5 contains more information on this process.

APPENDIX C – PROJECT PLAN

APPENDIX D – REFERENCES

Publication References

[0] Simon Robinson (et Al) Professional C# 2nd edition (2002). Wrox

Web References		**Up On Date**
[1]	http://www.foood.net/index.htm Foood. Icons.	27/01/06
[2]	www.codeproject.com Communal Software Forum.	27/01/06
[3]	www.bcs.org British Computer Science.	27/01/06
[4]	http://www.sharewareorder.com/Remote-Amp-download-10542.htm Terminal Zero. Remote Amp - Control WinAmp directly over a network.	27/01/06
[5]	http://www.microsoft.com/downloads/details.aspx?FamilyId=0A9B6820-BFBB-4799-9908-D418CDEAC197&displaylang=en Microsoft Download Center.	27/01/06
[6]	http://www.microsoft.com/windows/windowsmedia/mp10/sdk.aspx Windows Media Home. Windows Media SDK Components.	27/01/06
[7]	http://www.microsoft.com/downloads/details.aspx?FamilyID=b446ae53-3759-40cf-80d5-cde4bbe07999&DisplayLang=en Microsoft Download Center.	27/01/06
[8]	http://msdn.microsoft.com/netframework/downloads/framework1_1/ Microsoft .NET v1.1 Framework Redistributable.	27/01/06
[9]	http://www.microsoft.com/downloads/details.aspx?FamilyId=262D25E3-F589-4842-8157-034D1E7CF3A3&displaylang=en Microsoft Download Center.	27/01/06
[10]	http://www.mysql.com/ MySQL Database	27/01/06
[11]	http://firebird.sourceforge.net/ Embedded database.	27/01/06
[12]	http://www.microsoft.com/windows/directx/default.aspx Windows. Microsoft Direct X	27/01/06
[13]	http://msdn.microsoft.com/library/default.asp?url=/library/en-us/wmplay10/mmp_sdk/windowsmediaplayer10sdk.asp MSDN. Windows Media Player 10 SDK	27/01/06
[14]	http://www.microsoft.com/downloads/details.aspx?FamilyID=b446	27/01/06

ae53-3759-40cf-80d5-cde4bbe07999&DisplayLang=en
Microsoft Download Center.

[15] http://ndoc.sourceforge.net/ 27/01/06
NDoc documentation tool.

[16] http://msdn.microsoft.com/library/default.asp?url=/library/en-us/cpguide/html/cpconeventsdelegates.asp 27/01/06
.NET Framework Developer's Guide. Events and Delegates

[17] http://www.codeproject.com/cs/miscctrl/collapsiblepanelbar.asp 27/01/06
Derik Lakin. Windows XP style Collapsible Panel Bar.

[18] http://support.microsoft.com/default.aspx?scid=kb;en-us;830033 27/01/06
Microsoft Help and Support. How to apply Windows XP themes to Office COM add-ins

[19] http://blogs.msdn.com/rprabhu/archive/2003/09/28/56540.aspx#62583 27/01/06
Raghavendra Prabhu. Cool Client Stuff, discussion thread

[20] http://www.codeproject.com/cs/menu/menuimage.asp 27/01/06
Chris Beckett. Menu Images using C# and IExtenderProvider - a better mousetrap

[21] http://www.c-sharpcorner.com/ 27/01/06
Communal Software Forum.

[22] http://www.c-sharpcorner.com/Tools/MP3TagEditorB2PL.asp 27/01/06
Paul Lockwood. ID3 Tag editor.

[23] http://www.absoluteastronomy.com/encyclopedia/I/ID/ID31.htm 14/12/05.

[24] http://www.codeproject.com/dotnet/CSharpPing.asp 27/01/06
Wesley Brown. C# Ping Component.

[25] http://lame.sourceforge.net/ 27/01/06
The lame project

[26] http://msdn2.microsoft.com/en-us/library/26thfadc.aspx 27/01/06
United States. MSDN. .NET Framework Developer's Guide.
Consuming Unmanaged DLL Functions

[27] http://msdn.microsoft.com/library/default.asp?url=/library/en-us/netmgmt/netmgmt/netserverenum.asp 27/01/06
MSDN. Platform SDK: Network Management. NetServerEnum

[28] http://www.microsoft.com/downloads/details.aspx?FamilyID=9ad000f2-cae7-493d-b0f3-ae36c570ade8&DisplayLang=en 27/01/06
Microsoft Download Center.

[29] http://www.microsoft.com/downloads/details.aspx?FamilyID=6C050FE3-C795-4B7D-B037-185D0506396C&displaylang=en 27/01/06
Microsoft Download Center. Microsoft Data Acess Components

(MDAC) 2.8

[30] http://sqlteam.com/item.asp?ItemID=15903 27/01/06
ANSI/ISO SQL 99 standard downloads

[31] http://shareaza.sourceforge.net/ 27/01/06
Peer to peer software.

[32] http://www.winamp.com/ 27/01/06
Media player application.

[33] http://java.sun.com 27/01/06
Java home page.

[34] http://www.microsoft.com 27/01/06
Microsoft home page

[35] http://www.sussex.ac.uk/Units/academic/academicoffice/ugmatters/ughand2004.pdf 27/01/06
Sussex University undergraduate handbook

APPENDIX E – BIBLIOGRAPHY

Publications

Alan Dix, Janet Finlay, Gregory D.Abowd, Russell Beale. Himan-Computer Interaction 3rd Edition (2004). Pearson, Prentice Hall

David Sceppa. Microsoft ADO.NET Core Reference (2002). Microsoft Press

John Sharp, Jon Jagger. Microsoft C# .NET Step By Step (2002). Microsoft Press

Simon Robinson (et Al) Professional C# 2nd edition (2002). Wrox

Web Publications

	Up On Date
http://www.microsoft.com/downloads/details.aspx?FamilyId=262D25E3-F589-4842-8157-034D1E7CF3A3&displaylang=en Microsoft Download Center.	27-06-05
http://msdn.microsoft.com/msdnmag/issues/03/02/Multithreading/default.aspx Ian Griffiths, MSDN Magazine	27-06-05
http://www.microsoft.com/downloads/details.aspx?FamilyID=e43cbe59-678a-458a-86a7-ff1716fad02f&DisplayLang=en Microsoft Download Center.	27-06-05
http://www.microsoft.com/downloads/details.aspx?FamilyID=b446ae53-3759-40cf-80d5-cde4bbe07999&DisplayLang=en Microsoft Download Center.	27-06-05
http://www.informatics.susx.ac.uk/research/nlp/carroll/se/ John Caroll, Guy McClusker. Software Engineering Course 2006.	27-06-05
http://www.id3.org/id3v1.html ID3 Made Easy	27-06-05
http://www.absoluteastronomy.com/encyclopedia/I/ID/ID31.htm	27-06-05
http://msdn.microsoft.com/ United Kingdom MSDN	27-06-05
www.pscode.com/vb/scripts/ShowCode.asp?txtCodeId=179&lngWId=10 Icons in VB .NET Menus!!!!!	27-06-05
msdn.microsoft.com/library/default.asp?url=/library/en-us/cpguide/html/cpconimplementingextenderprovider.asp United Kingdom MSDN, extender properties	27-06-05

http://support.microsoft.com/default.aspx?scid=kb;en-us;830033 27-06-05
Microsoft Help and Support. How to apply Windows XP themes to Office COM add-ins

http://blogs.msdn.com/rprabhu/archive/2003/09/28/56540.aspx#62583 27-06-05
Raghavendra Prabhu. Cool Client Stuff, discussion thread

http://www.codeproject.com/cs/menu/menuimage.asp 27-06-05
Chris Beckett. Menu Images using C# and IExtenderProvider - a better mousetrap! The code project.

http://www.codeproject.com/cs/miscctrl/CollapsiblePanelBar.asp?df=100&forumid=12638&exp=0&fr=51 27-06-05

http://msdn.microsoft.com/library/default.asp?url=/library/en-us/netmgmt/netmgmt/netserverenum.asp 27-06-05
MSDN. Platform SDK: Network Management. NetServerEnum