***JAudioTags***

JAudioTags.dll is a .Net class library written in C# 6.0. It implements a set of classes and methods for querying and modifying metadata in Flac and Mp3 (version 2.3) audio files. There is also a class for checking the information inside M3U playlist files.



***Classes***

The diagram opposite shoes the public classes in the dll.

AudioFile is an abstract base class that gathers together common properties of Flac files and mp3 files.

FLACFile and MP3File are the classes that you instantiate to represent each of your audio files. These are the classes you will use most often.

M3U is a class to represent an M3U playlist file.

TagChecker contains a method that will walk through your files checking the tags. (This methods makes many assumptions about how your files are organised – see below.)

Testing is a debugging class. You should not need to use this. It contains a method which you point at a test folder of audio files. It performs various pre-configured modifications and saves to each file, then checks that basic parameters of the file, such as the total number of music bytes, remain unaltered.

Helpers is a static class that contains a library of non-audio ‘helper’ methods used by the other classes.

LasySW is a class that wraps a StreamWriter in an attempt to make it ‘lazy’ – i.e. only creates a file if you try to write to it. Used for logging.

BadAudioFileException implements custom exceptions used by the other classes.

V23TextTags is a public enumeration – not a class. The ID3 v.3 tags have pre-designated names. This enumeration lists the names of all the text frames – the ones that this dll can read and modify.

***AudioFile Public Members***

Because AudioFile is the parent class of both FLACFile and MP3File these may be used with objects of both kinds.

*General Public properties:*

|  |  |  |
| --- | --- | --- |
| public const string \_Version | Read only | Returns the version of the code. |
| public string AudioPath | Read only | Hold the path to the current audio file. |
| public long NumberOfMusicBytes | Read only | Returns the number of bytes in the file storing the actual music (i.e. not metadata.) |
| public bool HasEmbeddedGraphic | Read only | Returns true if the file has a graphic embedded within. |
| static public bool IsCaseSensitive | Read / write | Set this to true if you want tag naming to be case sensitive. Defaults to false. |
| public int NumberOfTags | Read only | Returns the number of ‘primary’ tags in the file. For FLAC this is the number of tags, for MP3 it is the number of IDE 2.3 Tags. (i.e. NOT v1 tags.) |

*Public Tag Properties (All read/write)*

Flac files and mp3 have different tag sets with different tag names. These represent the 10 most commonly used, and map internally to the appropriate tag name for the chosen file type.

|  |  |
| --- | --- |
| public string ALBUM | Read and writes the album name tag. |
| public string ALBUMARTIST | Read and writes the album artists tag. |
| public string ARTIST | Read and writes the artist tag. |
| public string COMPOSER | Read and writes the composer tag. |
| public string DISCNUMBER | Read and writes the disc number tag. |
| public string GENRE | Read and writes the genre tag. |
| public string TITLE | Read and writes the track title tag. |
| public string TRACKNUMBER | Read and writes the track number tag. |
| public string DATE | Read and writes the date tag. |
| public string COMMENT | Read and writes the comment tag. |

*Public methods:*

|  |  |
| --- | --- |
| public void Save(bool MakeBackup) | Re-writes a modified file to disc. This is only possible if the file was created with the constructor argument ‘ReadOnly’ set to false. If MakeBackip is set to true, a backup copy of the file prior to modification is made. |
| static public int FileWalk(  string Root,  Func<string, LazySW, int> ProcessFile,  List<string> FileExtensions,  string LogPath,  string ErrorLogPath) | A static method. Does a recursive ‘tree walk’ starting at ‘Root’. The second parameter must be a method taking two arguments: the string is the path to a file. The other is a ‘Lazy’ StreamWriter used to log results. It must return an int. Your code goes into this method and every file in the path is processed by the method. See example code below. The third argument is a list of extensions for files to be processed. If this is null, all files are processed.  The last two are paths/filenames for result and error logs. |
| static public int FileWalk(  string Root,  Func<string, LazySW, int> ProcessFile,  List<string> FileExtensions) | This is a ‘cut down’ version of the method above. Same functionality but does not use log files. |
| static public int DirWalk(  string Root,  Func<string, List<string>, LazySW, int>  ProcessDirectory,  List<string> FileExtensions,  string ResultsPath,  string ErrorLogPath) | Another static tree-walking method. This one visits each directory in turn – not each file. This allows you to process a directory at a time, for example to verify all tracks have the same ALBUM tag. The method passed in the second parameter takes three arguments: a string which is the path/name of the current directory; a list of file extensions to be processed within that directory; and a ‘lazy’ StreamWriter to log results. See example code below. |
| public IEnumerator GetEnumerator() | Makes the class enumerable so that you can use foreach() to iterate through the tags. |
| static public int Dump(  string Root,  string Results,  string Errors) | Walks a directory tree, starting at ‘Root’, and dumps basic audio file attributes (FLAC and MP3) to the file pointed to by ‘Results’. Errors are written to ‘Errors’. |

***FLACFile Public Members***

*Public Properties*

|  |  |  |
| --- | --- | --- |
| public new const string \_Version | Read only | Returns the version of this class. |
| static public string Version | Read only. | Returns the version of this class and all supporting classes. |

*Public Methods*

|  |  |
| --- | --- |
| public FLACFile(string FilePath, bool ReadOnly) | Constructor. First argument is the path/name of the file to be opened. Second argument should be true if you only plan to read the file. (This speeds up processing.) False if you plan to modify and re-write the file. |
| public override void DebugPrint(bool IncludeDetail, bool IncludeTags) | A debugging method. Prints the internals of an audio file. The first parameter includes more detail and the second lists the tags. |
| public override void SaveTestFile() | A debugging method. Instantiate a file from disc, then call this method. It lets you re-reorganise the internals of the file before saving it. This enables you to generate unusual internal configurations that you might not easily find in ‘real’ files. |
| static public int GetDumpDetails(  string FilePath, LazySW TheWriter) | A method that can be used with AudioFile.FileWalk() to print basic details of all FLAC files in the path. |
| public void AddTag(  string Name, string Value) | Adds a new tag to the file. FLAC file tags can have any name. |
| public int CountTags(string Name) | Returns how many tags with the given name. |
| public bool Exists(string Name) | Returns true if the file has a tag with the given name. |
| public string First(string Name) | Returns the value of the first tag with the given name. |
| public void RemoveAll(string Name) | Removes tall tags with the given name. |
| public void RemoveFirst(string Name) | Removes the first tag with given name. |
| public void RemoveExact(string Name, string Value) | Remove a tag with the specified name and value. |
| public void RemoveExact(string Name, string Value) | Removes all tags with the same name and adds a new tag with that name and the new value. |

***MP3File Public Members***

*Public Properties*

|  |  |  |
| --- | --- | --- |
| public new const string \_Version | Read only | Returns the version of this class. |
| static public string Version | Read only | Returns the version of this class and all supporting classes. |

*Public Methods*

|  |  |
| --- | --- |
| public MP3File(string FilePath, bool ReadOnly) | Constructor. First argument is the path/name of the file to be opened. Second argument should be true if you only plan to read the file. (This speeds up processing.) False if you plan to modify and re-write the file. |
| public override void DebugPrint(bool IncludeDetail, bool IncludeTags) | A debugging method. Prints the internals of an audio file. The first parameter includes more detail and the latter lists the tags. |
| public void AddTag(V23TextTags Name, string Value) | Adds a new text tag. The first argument must be a value from the V23TextTags enumeration. The second is the value of the new tag. |
| public int CountTags(V23TextTags Name) | Counts how many tags have the name specified by the argument – a value from the V23TextTags enumeration. |
| public bool Exists(V23TextTags Name) | Returns true if there is a tag with the name specified by the argument – a value from the V23TextTags enumeration. |
| public string First(V23TextTags Name) | Returns the value of the first tag with the name specified in the argument – a V32TextTags value. |
| public void RemoveAll(V23TextTags Name) | Removes all tags with the name specified – a V32TextTags value. |
| public void RemoveFirst(V23TextTags Name) | Removes the first tag with the name specified – a V23TextTags value. |
| public void RemoveExact(V23TextTags Name, string Value) | Remove a tag with the specified name and value. |
| public void ReplaceAll(V23TextTags Name, string Value) | Remove all tags with the given name, and add a new tag with that name and the new value. |
| public override string ToString() | Returns a string representation of the tags in the file. |
| public override void SaveTestFile() | A debugging method. Instantiate a file from disc, then call this method. It lets you re-reorganise the internals of the file before saving it. This enables you to generate unusual internal configurations that you might not easily find in ‘real files. |
| public void BinaryFrameTypeRemoveAll(string TargetName) | Used to remove non-text frames from an MP3 file – for example unwanted PRIV or COMM frames. |

***M3U Public Members***

*Public Properties*

|  |  |  |
| --- | --- | --- |
| public const string \_Version | Read only | Returns the current version number of this class. |

*Public Methods*

|  |  |
| --- | --- |
| public M3UFile(string AudioPath, LazySW ResultsFile) | Constructor. First argument is the path to the file. The second is a ‘lazy’ streamwriter to record results.. |
| public M3UFile(string AudioPath, string ResultsFile) | Constructor. First argument is the path to the file. Second is a path/filename at which a results log will be written. |
| public bool CheckFile() | Call this to check the file for errors. |

***TagChecker Public Members***

This class was created to tie together all of the other code and check the tags in my music collection for omissions and inconsistencies. Its checks depend heavily on my filing system:

* One album per folder.
* Only flac OR mp3 files in a folder – never both. (Or M3U plus associated JPN or PNG graphics.)
* Folder.jpg graphic file in each music folder – nothing else.

If you use a different scheme you will need to write your own version.

*Public Properties*

|  |  |  |
| --- | --- | --- |
| public const string Version | Read only | The version of this class. |

*Public Methods*

|  |  |
| --- | --- |
| public TagChecker(  List<string> IList = null,  bool CheckTrackDigits = false,  bool CheckInvariantGenres = false,  bool CheckInvariantDates = false) | Constructor.  - An optional list of file extensions to ignore.  - Should we check how many digits in track numbers?  - Should we check all file in a folder have the same genre?  - Should we check that all files in a folder have the same sate? |
| public void CheckFiles(  string Root,  string ResultsFile,  string ErrorFile) | This wraps a tree walker and checks the files in every directory beneath ‘Root’  Root is the directory to start in.  ResultsFile is a path/.name for a file to hold output data.  ErrorFile is a file to log any erros. |

***Testing Public Members***

Testing is a debugging class. You should not need to use it.

*Public Properties*

|  |  |  |
| --- | --- | --- |
| public const string \_Version | Read only | Returns the version of this class. |

*Public Methods*

|  |  |
| --- | --- |
| static public void Test(  string RootPath,  string ResultsPath,  string ErrorLogPath) | A tree walker. Point it at a test folder with test copies of audio files, and it automatically performs numerous modifications and saves of those files. At the end it verify that basic internal parameters of the files have not changed – thus increasing confidence that the code works as expected.  RootPath – the out directory holding test files.  ResultsPath - Path/name of results file.  ErrorLogPath - Path/Name of errors log |

***How to use the Library***

Three ways:

1. Download the DLL here and copy it to a folder on your PC. In your client program in Visual Studio, right click on the project name in Solution Explorer, choose Add, then Reference. Navigate to the DLL, select it and click OK. Add ‘Using JAudioTags’ to the top of your code. You now have access to the classes and methods described above.

2. Download the source code here. Study it until you are happy that it is benign. In Visual Studio start a new ‘Class Library’ project. Delete the code file added by VS and drag my files into Solution Explorer. Click on build to compile your own DLL. Use it as in 1) above.

3. Down load the source code here. Drag all of the .cs files onto your project in Solution Explorer. Add ‘using JAudioTags’ to the top of your code. The classes will now be incorporated directly into your exe when you compile. No DLL needed.

***Example Code – Change the Artist Name Across a Whole Library***

using System;

using System.Collections.Generic;

using JAudioTags;

namespace chck02

{

class Program

{

static string Root = @"o:\testdata\testfolder2";

static string Log = @"D:\results.txt";

static string Errors = @"d:\Errors.txt";

static string OldName = "Prince";

static string NewName = "The artist formerly known as Prince";

static int Count = 1;

// This method matches the signature required by FileWalk.

// Its name is passed to the FileWalk constructor.

// It is called for every FLAC or MP3 file encountered

static int ChangeArtistName(string Filename, LazySW SW)

{

AudioFile AFile = null;

// Open with ReadOnly as false - we are changing the files

if (Helpers.JGetExtension(Filename) == "FLAC")

AFile = new FLACFile(Filename, false);

if (Helpers.JGetExtension(Filename) == "MP3")

AFile = new MP3File(Filename, false);

if (AFile.ARTIST == OldName)

{

string Msg = String.Format("{0,3} - Changing {1}", Count++, AFile.AudioPath);

Console.WriteLine(Msg);

SW.WriteLine(Msg);

AFile.ARTIST = NewName;

AFile.Save(true);

return 1;

}

return 0;

}

static void Main(string[] args)

{

//C all the FileWalk method to visit all files in tree

AudioFile.FileWalk(Root, ChangeArtistName, new List<string>() { "MP3", "FLAC" },

Log, Errors);

Helpers.PressAnyKeyToContinue();

}

}

}