

```

using System;
using System.IO;
using System.Text;
using System.Collections;
using System.Windows.Forms;
using System.Reflection;
using SB54_CSAI.Genres;
using System.Text.RegularExpressions;

namespace SB54_CSAI.MP3Lib
{
    #region MP3Tag CLASS

    /// <summary>
    /// Provides methods for reading and writing to an MP3 file
    /// ID3 v1 and v1.1 tag data. the MP3 ID3 tag information is stored within
    /// the last 128 bytes of the physical file.
    /// <br></br>
    /// <br></br>
    /// This class was written and using an article contained at the following URL
    /// http://www.c-sharpcorner.com/Tools/MP3TagEditorB2PL.asp, where the original
    /// author was Paul Lockwood
    /// <br></br>
    /// I have had to modify the source code to make the field handling more robust,
    /// in particular the handling of special ASCII characters (NUL, SOH, STX, ETX) was
    /// not taken into account within the original library.
    /// <br></br>
    /// The following web site illustrates how the tag information is stored
    /// http://www.absoluteastronomy.com/encyclopedia/I/ID/ID31.htm for Mp3 Id3 tags using
    /// v1 and v1.1 of the id3 tag convention
    /// </summary>
    public class MP3Tag
    {
        #region Public Methods
        /// <summary>
        /// Accepts an int that represents the current MP3 Genre, then
        /// requests the string value that matches this MP3 Genre from the
        /// <see cref="SB54_CSAI.Genres.MThreadSingleton_genres">genres singleton object </
see>
        /// </summary>
        /// <param name="GenreID">An int representing the GenreID of this MP3</param>
        /// <returns>string value that of the input int GenreID, which is retrieved from
        /// the <see cref="SB54_CSAI.Genres.MThreadSingleton_genres">genres singleton object
</see></returns>
        public static string getGenreFromID(int GenreID)
        {
            return MThreadSingleton_genres.Instance.getGenreFromID(GenreID);
        }

        /// <summary>
        /// Reads the MP3 ID3 tag information from disk into the
        /// <see cref="SB54_CSAI.MP3Lib.MP3">MP3 Structure </see> parameter
        /// provided
        /// </summary>
        /// <param name="paramMP3">MP3 Structure to read values into</param>
        public static void readMP3Tag (ref MP3 paramMP3)
        {
            byte[] bBuffer;

            try
            {
                //Read the last 128 bytes from file into byte array
                using (FileStream oFileStream = new FileStream(paramMP3.fileComplete ,
FileMode.Open))
                {
                    bBuffer = new byte[128];
                    oFileStream.Seek(-128, SeekOrigin.End);
                    oFileStream.Read(bBuffer,0, 128);
                    oFileStream.Close();
                }
            }
        }
    }
}

```

```

// Convert the Byte Array to a String, using ASCIIEncoding
Encoding tagEncoding = new ASCIIEncoding();
string id3Tag = tagEncoding.GetString(bBuffer);

//See if there is a tag
if (id3Tag.Substring(0,3) == "TAG")
{
    //yes there is so simply, get all the required fields by getting the
    //sub strings that matches the number of bytes for each field
    string temp = id3Tag.Substring( 3, 30).Trim();
    paramMP3.id3Title      = temp.Substring(0,ParseString(temp));

    temp = id3Tag.Substring( 33, 30).Trim();
    paramMP3.id3Artist     = temp.Substring(0,ParseString(temp));

    temp = id3Tag.Substring( 63, 30).Trim();
    paramMP3.id3Album      = temp.Substring(0,ParseString(temp));

    temp = id3Tag.Substring( 93, 4).Trim();
    paramMP3.id3Year       = temp.Substring(0,ParseString(temp));

    temp = id3Tag.Substring( 97, 28).Trim();
    paramMP3.id3Comment    = temp.Substring(0,ParseString(temp));

    //See if byte 125 is 0, if is is then, we have a track number seperator
    //so get the track number
    if (id3Tag[125]==0)
    {
        paramMP3.id3TrackNumber = bBuffer[126];
    }
    else
    {
        paramMP3.id3TrackNumber = 0;
    }
    //get the genre byte (number between 0-147)
    paramMP3.id3Genre = bBuffer[127];
    paramMP3.hasID3Tag = true;
}
else
{
    // ID3 Tag not found so create an empty TAG in case the user saves later
    paramMP3.id3Title      = "";
    paramMP3.id3Artist     = "";
    paramMP3.id3Album      = "";
    paramMP3.id3Year       = "";
    paramMP3.id3Comment    = "";
    paramMP3.id3TrackNumber = 0;
    paramMP3.id3Genre      = 0;
    paramMP3.hasID3Tag     = false;
}
}
catch (Exception ex)
{
    //As this class is used with operations that are occurring regularly
    //such as when scanning database files (could be 100's even 1000's of files)
    //do no really wanto to show a message. But must ensure the error is caught.
    MessageBox.Show("Problem with File Read Access in MP3Tag : " +
        "\r\n\r\n" + ex.Message,
        "MP3 Write ERROR", MessageBoxButtons.OK,
        MessageBoxIcon.Error);
}
}

/// <summary>
/// Update the MP3 ID3 tag information from the
/// <see cref="SB54_CS.AI.MP3Lib.MP3">MP3 Structure </see> parameter
/// provided to the disk stored file
/// </summary>
/// <param name="paramMP3">MP3 Structure to write values from</param>
public static void updateMP3Tag (ref MP3 paramMP3)
{

```

```

    //Make sure tag does not contain any nasty Database unfriendly characters, such
    //as NUL, STX, ETX, '%' etc etc
    EnsureTagIsDbFriendly(ref paramMP3);

    // Trim any whitespace
    paramMP3.id3Title   = (paramMP3.id3Title == null ? string.Empty : paramMP3.
id3Title.Trim() );
    paramMP3.id3Artist  = (paramMP3.id3Artist == null ? string.Empty : paramMP3.
id3Artist.Trim() );
    paramMP3.id3Album   = (paramMP3.id3Album == null ? string.Empty : paramMP3.
id3Album.Trim() );
    paramMP3.id3Year    = (paramMP3.id3Year == null ? string.Empty : paramMP3.
id3Year.Trim() );
    paramMP3.id3Comment = (paramMP3.id3Comment == null ? string.Empty : paramMP3.
id3Comment.Trim() );

    //Make sure that all fields are correct sizes
    if (paramMP3.id3Title.Length > 30) paramMP3.id3Title   = paramMP3.id3Title.
Substring(0,30);
    if (paramMP3.id3Artist.Length > 30) paramMP3.id3Artist = paramMP3.id3Artist.
Substring(0,30);
    if (paramMP3.id3Album.Length > 30) paramMP3.id3Album   = paramMP3.id3Album.
Substring(0,30);
    if (paramMP3.id3Year.Length > 4) paramMP3.id3Year      = paramMP3.id3Year.
Substring(0,4);
    if (paramMP3.id3Comment.Length > 28) paramMP3.id3Comment = paramMP3.id3Comment.
Substring(0,28);

    //Construct the new id3 data, 128 bytes
    byte[] tagByteArray = new byte[128];
    for ( int i = 0; i < tagByteArray.Length; i++ ) tagByteArray[i] = 0; //
Initialise array to nulls

    // Convert the Byte Array to a String
    Encoding tagEncoding = new ASCIIEncoding();
    // Copy "TAG" to Array
    byte[] workingByteArray = tagEncoding.GetBytes("TAG");
    Array.Copy(workingByteArray, 0, tagByteArray, 0, workingByteArray.Length);
    // Copy Title to Array
    workingByteArray = tagEncoding.GetBytes(paramMP3.id3Title);
    Array.Copy(workingByteArray, 0, tagByteArray, 3, workingByteArray.Length);
    // Copy Artist to Array
    workingByteArray = tagEncoding.GetBytes(paramMP3.id3Artist);
    Array.Copy(workingByteArray, 0, tagByteArray, 33, workingByteArray.Length);
    // Copy Album to Array
    workingByteArray = tagEncoding.GetBytes(paramMP3.id3Album);
    Array.Copy(workingByteArray, 0, tagByteArray, 63, workingByteArray.Length);
    // Copy Year to Array
    workingByteArray = tagEncoding.GetBytes(paramMP3.id3Year);
    Array.Copy(workingByteArray, 0, tagByteArray, 93, workingByteArray.Length);
    // Copy Comment to Array
    workingByteArray = tagEncoding.GetBytes(paramMP3.id3Comment);
    Array.Copy(workingByteArray, 0, tagByteArray, 97, workingByteArray.Length);
    // Copy Track and Genre to Array
    tagByteArray[126] = paramMP3.id3TrackNumber;
    tagByteArray[127] = paramMP3.id3Genre;

    try
    {
        //write data back to disk
        using (FileStream oFileStream = new FileStream(paramMP3.fileComplete ,
FileMode.Open))
        {
            //Save the TAG data to disk, this will replace the last 128 bytes of the
file
            //with these new fields, which means this file now has a ID3 tag even if
all
            //the fields are empty
            if (paramMP3.hasID3Tag)
            {
                oFileStream.Seek(-128, SeekOrigin.End);
            }
            else

```

```

        {
            oFileStream.Seek(0, SeekOrigin.End);
        }
        oFileStream.Write(tagByteArray,0, 128);
        paramMP3.hasID3Tag = true;
        oFileStream.Close();
    }
}
catch (Exception ex)
{
    MessageBox.Show("Problem with File Update Access in MP3Tag : " +
        "\r\n\r\n\r\n" + ex.Message,
        "MP3 Write ERROR", MessageBoxButtons.OK,
        MessageBoxIcon.Error);
}
}

#endregion
#region Private Methods
/// <summary>
/// Ensures that the <see cref="SB54_CSAI.MP3Lib.MP3">MP3 Structure </see> parameter
/// provided does not contain any data within its fields that are special characters
/// that could cause the database updates/retrievals to fail.
/// </summary>
/// <param name="paramMP3">MP3 Structure to read values into</param>
private static void EnsureTagIsDbFriendly(ref MP3 paramMP3)
{
    //fix the year to current year if an incorrect year has been added
    if (!checkNumeric(paramMP3.id3Year))
    {
        paramMP3.id3Year = DateTime.Now.Year.ToString();
    }

    //fix the track no to 0 if it has been given an out of range track no
    int trackNo = (int)paramMP3.id3TrackNumber;
    if (trackNo > 255 || trackNo < 0)
    {
        paramMP3.id3TrackNumber = new Byte();
    }

    //fix the track no to 0 if it has been given an out of range genre no
    int genreID = (int)paramMP3.id3Genre;
    if (genreID > 147 || genreID < 0)
    {
        paramMP3.id3Genre = new Byte();
    }

    //strip out all database unfriendly characters that could cause the UPDATE sql
    //statement to fail
    stripDBReservedChars("'",ref paramMP3.id3Title);
    stripDBReservedChars('"',ref paramMP3.id3Title);

    stripDBReservedChars("'",ref paramMP3.id3Artist);
    stripDBReservedChars('"',ref paramMP3.id3Artist);

    stripDBReservedChars("'",ref paramMP3.id3Album);
    stripDBReservedChars('"',ref paramMP3.id3Album);

    stripDBReservedChars("'",ref paramMP3.id3Comment);
    stripDBReservedChars('"',ref paramMP3.id3Comment);
}

/// <summary>
/// Takes a input string parameter (srcString) and removes all occurences, of ALL of
the
/// characters contained within the input string parameter (toStrip) from it.
/// The database updates/retreivals could fail if these special characters are not
stripped
/// </summary>
/// <param name="toStrip">string containing all characters to strip from srcString
parameter</param>
/// <param name="srcString">The source string that will have all characters in
toStrip, removed

```

```

/// from it</param>
private static void stripDBReservedChars(string toStrip,ref string srcString)
{
    if (srcString != null)
    {
        //strip each char in toStrip from srcString
        foreach(char c in toStrip)
        {
            srcString=srcString.Replace(c.ToString(),"");
        }
    }
}

/// <summary>
/// Takes a input string parameter (srcString) and removes all occurences of the
/// input character parameter (toStrip) from it.
/// The database updates/retrievals could fail if these special characters are not
stripped
/// </summary>
/// <param name="toStrip">Character to strip from srcString parameter</param>
/// <param name="srcString">The source string that will have occurences of the
characters
/// toStrip, removed from it</param>
private static void stripDBReservedChars(char toStrip,ref string srcString)
{
    if (srcString != null)
    {
        char chSpace = ' ';
        srcString=srcString.Replace(toStrip,chSpace);
        srcString=srcString.Trim();
    }
}

/// <summary>
/// Checks that a string contains numeric data ONLY
/// </summary>
/// <param name="s">The string to check for numeric data</param>
/// <returns>true if the string contains ONLY numeric data</returns>
private static bool checkNumeric(string s)
{
    if (s != null)
    {
        //check for empty string
        if (! s.Equals(String.Empty) && s != null)
        {
            //^ = Matches any single character not in the specified set of
characters.
            //\d = Matches any decimal digit.
            //+ = Specifies one or more matches
            //$ = Specifies that the match must occur at the end of the string
            // before \n at the end of the string, or at the end of the line.
            Regex isNumber = new Regex(@"^\d+$");
            //represents results from a single regular expression
            Match m = isNumber.Match(s);
            if (! m.Success)
            {
                //Not Numeric
                return false;
            }
            else
            {
                //Is Numeric OK
                return true;
            }
        }
        else
        {
            //Empty String, so Not Numeric
            return false;
        }
    }
    return false;
}

```

```

    /// <summary>
    /// Finds the last position within the input string of the last ASCII character in
the
    /// range 33-127, and then returns this index. This is required as the readMP3Tag
method
    /// uses an ASCII encoding stream of bytes to read MP3 tag data, and if the MP3 tag
    /// does not use its full allocation of bytes for a particular field, the fields
could
    /// contain some strange ASCII characters such as NUL, SOH, STX, ETX. These special
characters
    /// cause the <see cref="SB54_CSAI.DatabaseAccess.DBAccess">database </see> updates
to fail.
    /// So these special ASCII characters need to be removed prior to any database
string/retrieving
    /// is attempted.
    /// </summary>
    /// <param name="text">The text to parse for special ASCII characters, NUL, SOH, STX
, ETX
    /// etc etc</param>
    /// <returns>An integer representing the last position in the input text that was in
the
    /// range 33-127 Decimal</returns>
private static int ParseString (string text)
{
    int i=0;
    //loop each char
    foreach (char c in text)
    {
        //if < 32, must be some sort of control char, which cant be stored in
database
        //so exit loop
        if (c < 32)
        {
            break;
        }
        //if > 127, must be some sort of special character, which cant be stored in
database
        //so exit loop
        else if (c > 127)
        {
            break;
        }
        //good characters found, so proceed to next index
        else
        {
            i++;
        }
    }
    //return last known good character position
    return i;
}
#endregion
}
#endregion
#region MP3 STRUCT

/// <summary>
/// MP3 Structure that represents a single MP3 ID3 TAG
/// </summary>
public struct MP3
{
    #region Instance fields
    //instance fields
    public string filePath;
    public string fileFileName;
    public string fileComplete;
    public bool hasID3Tag;
    public string id3Title;
    public string id3Artist;
    public string id3Album;
    public string id3Year;
    public string id3Comment;

```

```
public byte    id3TrackNumber;
public byte    id3Genre;
#endregion
#region Public Constructor

/// <summary>
/// Constructor, simply creates a new MP3 structure using the parameter
/// information provided
/// </summary>
/// <param name="path">The path to the MP3</param>
/// <param name="name">The file name of the MP3</param>
public MP3(string path, string name)
{
    //Initialise an DEFAULT MP3 using parameter details
    this.filePath      = path;
    this.fileFileName  = name;
    this.fileComplete  = path + "\\\" + name;
    this.hasID3Tag     = false;
    this.id3Title      = null;
    this.id3Artist     = null;
    this.id3Album      = null;
    this.id3Year       = null;
    this.id3Comment    = null;
    this.id3TrackNumber = 0;
    this.id3Genre      = 0;
}
#endregion
}
#endregion
}
```