

```

using System;
using System.Collections;
using System.ComponentModel;
using System.Drawing;
using System.Data;
using System.Windows.Forms;
using SB54_CSAI.DatabaseAccess;
using SB54_CSAI.Genres;
using System.IO;
using System.Threading;
using System.Security;
using System.Net;
using SB54_CSAI.MP3Lib;
using System.Text.RegularExpressions;

namespace SB54_CSAI.MediaLibrary
{
    #region FolderTree CLASS
    /// <summary>
    /// FolderTree is a sub class of a <see cref="System.Windows.Forms.TreeView">TreeView </see>
    /// that provides a a file system structure of <see cref="SB54_CSAI.DatabaseAccess.DBAccess">database </see>
    /// stored directories.The files for each FolderTree node are shown to the user for
    /// each selected FolderTree node. This control is used as part of the
    /// <see cref="SB54_CSAI.MediaLibrary.uctMediaLibrary">media library GUI</see>
    /// </summary>
    public class FolderTree : System.Windows.Forms.TreeView
    {
        #region Instance Fields
        /// <summary>
        /// ScanningFolderhandler delegate for the Scan event
        /// </summary>
        public delegate void ScanningFolderhandler(object sender, ScanningEventArgs e);
        /// <summary>
        /// Raised during the scanning of files. Each time a new directory is scanned
        /// a new event is raised with the directory name available via the
        /// ScanningEventArgs</see>
        /// </summary>
        public event ScanningFolderhandler Scan;
        private MP3TreeNode leafNode;
        private ArrayList topLevelNodes = new ArrayList();
        private TreeNode tvnRoot;
        private System.Threading.Thread TvThread;
        private DBAccess db;
        private bool isScanning = false;
        private MP3 workingMP3;
        private ArrayList arGenres;
        private int currentRootFolderID = -1;
        #endregion
        #region Public Constructor
        /// <summary>
        /// Simply creates a new FolderTree object
        /// </summary>
        public FolderTree()
        {
        }
        #endregion
        #region Public Methods/Properties
        /// <summary>
        /// Takes an int value for the MP3 genre and returns the string equivalent
        /// For example 0 would return "Blues"
        /// </summary>
        /// <param name="id3Genre">The integer value of the Genre for the current MP3</param>
        /// <returns>A string representing the Genre for the current MP3</returns>
        public string getGenreName(int id3Genre)
        {
            return (string)arGenres[id3Genre];
        }
    }
}

```

```

    }

    /// <summary>
    /// Initialises the FolderTree TreeView with a root node
    /// </summary>
    public void InitTree()
    {
        ShowRootLines = true;
        ShowLines = false;
        ShowPlusMinus = true;
        //create DataBase access object
        db = new DBAccess();
        //get the Genres
        arGenres = MThreadSingleton_genres.Instance.getGenres();

        try
        {
            //add the Genres to the database, as they might not exist yet, and may be
            required later by
            //other controls, like the Media Library.
            db.createGenres(arGenres);
            //add the root node
            tvnRoot = new TreeNode(Dns.GetHostName().ToUpper() + " Logged Music",0,0);
            Nodes.Add(tvnRoot);
        }
        catch(Exception ex)
        {
            MessageBox.Show("Problem with DB access-\n\n connection: "
                + "\r\n\r\n query: "
                + "\r\n\r\n\r\n" + ex.ToString(),
                "Database Access ERROR", MessageBoxButtons.OK,
                MessageBoxIcon.Error);
        }
    }

    /// <summary>
    /// Will abort the Process Thread associated with the FolderTree scanning
    /// </summary>
    /// <param name="hasActiveThreads">True if the FolderTree is currently scanning
    folders</param>
    public void CancelFolderScanning(bool hasActiveThreads)
    {
        isScanning = false;
        db.hasActiveThreads = isScanning;
        //Abort the Thread associated with the FolderTree scanning if its active
        if (TvThread != null)
        {
            if (TvThread.ThreadState == ThreadState.Running && TvThread.IsAlive)
            {
                try
                {
                    TvThread.Abort();
                }
                catch(SecurityException se)
                {
                    //should not occur as this thread should always be owned by the
                    launching form,
                    //but catch it any way for safety
                    Console.WriteLine("Error with thread " + se.Message);
                }
            }
        }
        this.ExpandAll();
        this.Sorted = true;
    }

    /// <summary>
    /// Recursively looks at folders contained within the fldrShares collection and
    /// adds the details of these folders and files to the
    /// <see cref="SB54_CSAI.DatabaseAccess.DBAccess"> database stored information</see>
    /// </summary>

```

```

in
    /// <param name="fldrShares">A collection of top level folders to recusively store
    /// database</param>
    public void ScanFolders(ArrayList fldrShares)
    {
        //Add all top level folders
        topLevelNodes.Clear();
        foreach (string s in fldrShares)
        {
            topLevelNodes.Add(s);
        }
        //Create a new process thread to deal with the scanning of all folders.
        //We now have a mult-threaded GUI, so user can explore the rest of the GUI
        //while the FolderTree is scanning
        TvThread = new System.Threading.Thread(new
            System.Threading.ThreadStart(populateTree));
        TvThread.Start();
    }

    /// <summary>
    /// Raises the OnScanning event for the FolderTree. Any users of the
    /// FolderTree may now subscribe to this event using a ScanningFolderhandler
    /// delegate.
    /// </summary>
    /// <param name="e">The <see cref="SB54_CSAI.MediaLibrary.ScanningEventArgs">
    scanning event arguments</see></param>
    public virtual void OnScanning(ScanningEventArgs e)
    {
        if (Scan != null)
        {
            // Invokes the delegates.
            Scan(this, e);
        }
    }
#endregion
#region Private Methods

    /// <summary>
    /// Populates the FolderTree with all required folders that were previously stored
    /// within the <see cref="SB54_CSAI.DatabaseAccess.DBAccess">database</see> and then
    /// scans the filesystem for new folders that may not have been logged in the
    database
    /// the last time the application ran
    /// </summary>
    private void populateTree()
    {
        DateTime dl = DateTime.Now;
        isScanning = true;
        ScanningEventArgs ea;

        //loop through all folders
        foreach (string tvnRootFolder in topLevelNodes)
        {
            //create a new Scan event for event subscribers, START OF SCAN
            ea = new ScanningEventArgs(true,"Scanning folder " + tvnRootFolder);
            OnScanning(ea);

            if (! NodeExists(tvnRoot,tvnRootFolder))
            {
                try
                {
                    //make a new TreeView node for each root directory seen
                    DirectoryInfo d = new DirectoryInfo(tvnRootFolder);
                    leafNode = new MP3TreeNode(d.FullName,d.FullName,1,1);
                    addNodeToTree(tvnRoot,leafNode);
                    //As this is a tvnRoot folder, there may have been new files added
                    since
                    //the last file system scan, so simply delete all the database
                    stored

                    //files for this tvnRoot folder, then re-add them.
                    db.hasActiveThreads = isScanning;
                    db.updateFolder(Dns.GetHostName(),tvnRootFolder,1,-1);
                }
            }
        }
    }

```

```

        db.connectDB();
        currentRootFolderID = db.getFolderNum(tvnRootFolder);
        db.disconnectDB();
        db.RemoveFiles();
        logAudioFiles(leafNode);
        //get all the database stored sub folders that were previously
        //stored for this root directory and then add them to the FolderTree
        //treeview. This keeps FolderTree and GenreTree data synchronized
        DataSet ds = db.getAllLoggedSubFolders(currentRootFolderID);
        foreach (DataRow dr in ds.Tables["subFolders"].Rows)
        {
            string folderName = dr["folderName"].ToString();
            //add folder that dont already exist in FolderTree treeview
            if (! NodeExists(leafNode, folderName))
            {
                d = new DirectoryInfo(folderName);
                MP3TreeNode nd = new MP3TreeNode(d.Name, folderName, 1, 1);
                addNodeToTree(leafNode, nd);
            }
        }
        //Get this root directories sub directories
        String[] allFolders = Directory.GetDirectories(tvnRootFolder);
        if (allFolders.Length != 0 )
        {
            //if there are some, call scanFolder for all sub directories
            scanFolder(tvnRootFolder, leafNode);
        }
    }
    catch(Exception ex)
    {
        //simply catch exception
    }
}
//FINISHED SCANNING NOW
isScanning = false;
db.hasActiveThreads = isScanning;
DateTime d2 = DateTime.Now;
TimeSpan ts = d2 - d1;
//create a new Scan event for event subscribers, END OF SCAN
ea = new ScanningEventArgs(false, "Finished scanning folders, Duration was " + ts
.Seconds.ToString("###,###,###") + " Seconds....You may now use the Media Library");
OnScanning(ea);
this.Sorted = true;
}

/// <summary>
/// Takes a specific MP3TreeNode node and gets all the files associated with it and
then
/// logs these MP3 file details to the database
/// </summary>
/// <param name="srcNode">The source MP3TreeNode to retrieve the files from and log
/// to the <see cref="SB54_CSAI.DatabaseAccess.DBAccess">database</see></param>
private void logAudioFiles(MP3TreeNode srcNode)
{
    //for all the files for this MP3TreeNode, log the files MP3 details to the
database
    string[] MP3s = srcNode.PlayableFiles;
    foreach(string s in MP3s)
    {
        try
        {
            //check the file is an MP3 file
            FileInfo fFileInfo = new FileInfo(s);
            if (fFileInfo.Extension.ToLower().Equals(".mp3"))
            {
                //Is MP3 so read its TAG
                workingMP3 = new MP3(fFileInfo.DirectoryName, fFileInfo.Name);
                MP3Tag.readMP3Tag (ref workingMP3);
                //Simply ensure that all resident nasty characters (Database
unfriendly ones,
                //that cause the embedded SQL updates/selects to fail) are stripped

```

```

from MP3Tag
    //prior to attempting to store the MP3 details within the database
    MP3Tag.updateMP3Tag(ref workingMP3);
    db.insertMP3(ref workingMP3);
}
}
catch(Exception ex)
{
    //simply catch exception
}
}
}

/// <summary>
/// Searches the srcNode nodes for the text specified by
/// nodeName and returns true if the text is found
/// </summary>
/// <param name="srcNode">The source node to search</param>
/// <param name="nodeName">The text to search for</param>
/// <returns>True if the srcNode or any of children contain
/// the text specified by nodeName</returns>
private bool NodeExists(TreeNode srcNode, string nodeName)
{
    //return true is a node already exists with the given nodeName
    foreach (TreeNode n in srcNode.Nodes)
    {
        if (n is MP3TreeNode)
        {
            if (((MP3TreeNode)n).FullPath.Equals(nodeName))
            {
                return true;
            }
        }
    }
    return false;
}

/// <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.Invokee
/// 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);
}

    /// <summary>
    /// Scans the file system (which may be recusively, if the folderSpec contains sub-
directories)
    /// using the folderSpec parameter as a source directory for scanning. When a new
folder is
    /// found that was not previously stored in the
    /// <see cref="SB54_CSAI.DatabaseAccess.DBAccess">database</see> the new folder
will be
    /// added and the new files for this folder will have their MP3 details logged
within the
    /// <see cref="SB54_CSAI.DatabaseAccess.DBAccess">database.</see> This attempts to
act something
    /// like a realtime FileSystemWatcher, without the FileSystemWatchers need for the
application
    /// to be constantly running. Basically on startup get all logged folders, then scan
using the
    /// root folders as starting point, then if there are new folders found, log them.
    /// </summary>
    /// <param name="folderSpec">The source folder to scan</param>
    /// <param name="currentNode">The current FolderTree node to add newly created
sub folders to</param>
    private void scanFolder(String folderSpec, TreeNode currentNode )
    {
        //create a new Scan event for event subscribers, START OF SCAN
        ScanningEventArgs ea = new ScanningEventArgs(true,"Scanning folder " +
folderSpec);
        OnScanning(ea);
        //get all the sub-directories for the input folder
        String[] allFolders = Directory.GetDirectories(folderSpec);
        //if there are some folders to scan
        if (allFolders.Length != 0 )
        {
            //scan them
            foreach (String thisFolder in allFolders)
            {
                try
                {
                    //make a new TreeView node for each new directory seen
                    DirectoryInfo d = new DirectoryInfo(thisFolder);
                    MP3TreeNode nd = new MP3TreeNode(d.Name,thisFolder,2,2);
                    if (! NodeExists(currentNode,thisFolder))
                    {
                        addNodeToTree(currentNode,nd);
                    }
                    //As this is not a tvnRoot folder, simply need to check whether
there are already
                    //some files stored within the database against this folder. If
there are some
                    //files, simple skip to next folder This is a time saving method,
only add what
                    //needs to be added. If it stored in database already, dont need to
do anything.
                    //Just add the TreeNode to the Tree, to keep GUI Correct
                    db.hasActiveThreads = isScanning;
                    db.updateFolder(Dns.GetHostName(),thisFolder,0,currentRootFolderID);
                    if (! db.folderFileExists())
                    {
                        logAudioFiles(nd);
                    }
                    //do a RECURSIVE call to make sure we get all the sub-directories
and files
                    scanFolder(thisFolder,nd);
                }
            }
        }
    }
}

```

```

        catch(Exception ex)
        {
            //Simply catch Exception
        }
    }
}
#endregion
}
#endregion
#region MP3TreeNode CLASS

/// <summary>
/// Provides a new inherited TreeView TreeNode, MP3TreeNode, that accepts a string which
dictates
/// which directory this MP3TreeNode represents. As this MP3TreeNode holds a pointer to
/// a directory location, it is a simple matter to get all files for this directory.
/// As such each MP3TreeNode, also contains a list of its own files. This list will be
/// interigated by the <see cref="SB54_CSAI.MediaLibrary.FolderTree">FolderTree </see>
/// to use to add the files to the <see cref="SB54_CSAI.DatabaseAccess.DBAccess">
/// database</see>
/// </summary>
public class MP3TreeNode : System.Windows.Forms.TreeNode
{
    #region Instance Fields
    //Instance Fields
    private string fullPath="";
    private ArrayList MP3Files = new ArrayList();
    private ArrayList supportedFiles = new ArrayList();
    #endregion
    #region Public Constructor
    /// <summary>
    /// Initializes a new instance of the MP3TreeNode class with the specified label
    /// text and images to display when the MP3TreeNode is in a selected and unselected
state.
    /// On construction the MP3TreeNode will fetch a list of all its own supported files
    /// </summary>
    /// <param name="text">The label Text of the new MP3TreeNode.</param>
    /// <param name="fullPath">The full path of the folder this MP3TreeNode represents.
</param>
    /// <param name="imageIndex">The index value of Image to display when the
MP3TreeNode is unselected.</param>
    /// <param name="selectedImageIndex">The index value of Image to display when the
MP3TreeNode is selected.</param>
    public MP3TreeNode(string text,string fullPath,int imageIndex,int
selectedImageIndex)
        : base(text,imageIndex,selectedImageIndex)
    {

        this.fullPath = fullPath;
        //get list of supported file types
        setUpSupportedFiles();
        //scan the fullPath for MP3's
        String[] files = Directory.GetFiles(fullPath);
        foreach (string fileStr in files)
        {
            try
            {
                FileInfo f = new FileInfo(fileStr);
                if (f.Extension.Length !=0 && isSupportedFiled(f.Extension))
                {
                    MP3Files.Add(fileStr);
                }
            }
            catch(Exception ex)
            {
                //Simply catch exception
            }
        }
    }
}
#endregion
#region Public Methods/Properties
/// <summary>

```

```

    /// Returns an array of strings that represent this MP3TreeNode files,
    /// which will be used to store in the <see cref="SB54_CSAI.DatabaseAccess.DBAccess" >
>
    /// database</see>
    /// </summary>
    /// <returns>Returns an array of strings that represent this MP3TreeNode files</
returns>
    public string[] PlayableFiles
    {
        get
        {
            return MP3Files.ToArray(Type.GetType("System.String")) as string[];
        }
    }

    /// <summary>
    /// Returns an boolean that indicates whether this MP3TreeNode has files
    /// </summary>
    /// <returns>True if this MP3TreeNode has files</returns>
    public bool hasPlayableFiles()
    {
        return MP3Files.Count > 0;
    }

    /// <summary>
    /// Returns a string which is the full path of this MP3TreeNode
    /// For example D:\Mp3's\Appetite For Destruction
    /// </summary>
    /// <returns>Returns a string which is the full path of this MP3TreeNode</returns>
    new public string FullPath
    //new FullPath method, to allow inherited <see cref="System.Windows.Forms.TreeNode">
TreeNode</see>
    //to still be seen. This is to allow a cast to a TreeNode, FullPath method to still
work.
    //Without this new FullPath method the TreeNode.FullPath() method would be hidden,
and not accessible
    {
        get
        {
            return fullPath;
        }
    }
#endregion
#region Private Methods
    /// <summary>
    /// Check to see if the input string parameter is contained within the
    /// internal supportedFiles collection
    /// </summary>
    /// <param name="fileExtension">The current file extension</param>
    /// <returns>True if the current file extension is one that is contained within
    /// the internal supportedFiles collection, False otherwise</returns>
    private bool isSupportedFile(string fileExtension)
    {
        return supportedFiles.Contains(fileExtension.ToLower());
    }

    /// <summary>
    /// Add all the allowable extensions to the internal supportedFiles collection
    /// </summary>
    private void setUpSupportedFiles()
    {
        //add Window Audio Files
        supportedFiles.Add(".wav");
        supportedFiles.Add(".snd");
        supportedFiles.Add(".au");
        supportedFiles.Add(".aif");
        supportedFiles.Add(".aifc");
        supportedFiles.Add(".aiff");
        supportedFiles.Add(".wma");
        supportedFiles.Add(".mp3");

        //add Window Media Files
        supportedFiles.Add(".asf");

```

```

        supportedFiles.Add(".wm");

    }
    #endregion
}
#endregion
#region ScanningEventArgs CLASS
/// <summary>
/// Provides the ScanningEventArgs used with the <see cref="SB54_CSAI.MediaLibrary.
FolderTree">
/// FolderTree</see> Scan event. The Scan event is used by the
/// <see cref="SB54_CSAI.MediaLibrary.uctMediaLibrary">media library GUI</see> to enable
and
/// disable certain buttons based on whether a folder Scan is in operation or not.
/// </summary>
public class ScanningEventArgs : EventArgs
{
    #region Instance Fields
    //Instance fields
    private readonly bool hasActiveThreads=false;
    private readonly string foldername="";
    #endregion
    #region Public Constructor
    /// <summary>
    /// Constructs a new ScanningEventArgs object using the parameters provided
    /// </summary>
    /// <param name="hasActiveThreads">Should be set true if the folderTree is currently
scanning
    /// folders, false otherwise</param>
    /// <param name="foldername">The currently scanned folder</param>
    public ScanningEventArgs(bool hasActiveThreads, string foldername)
    {
        this.hasActiveThreads = hasActiveThreads;
        this.foldername = foldername;
    }
    #endregion
    #region Public Methods/Properties

    /// <summary>
    /// Returns true if the is currently scanning folders, false otherwise
    /// </summary>
    /// <returns>Returns true if the is currently scanning folders, false otherwise</
returns>
    public bool Scanning
    {
        get { return hasActiveThreads;}
    }

    /// <summary>
    /// Returns the currently scanned folder
    /// </summary>
    /// <returns>Returns the currently scanned folder</returns>
    public string FolderName
    {
        get { return foldername;}
    }
    #endregion
}
#endregion
}

```