

```

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 SB54_CSAI.MP3Lib;

namespace SB54_CSAI.MediaLibrary
{
    #region GenreTree CLASS
    /// <summary>
    /// GenreTree is a sub class of a <see cref="System.Windows.Forms.TreeView">TreeView </
    see>
    /// that provides a new GenreTree TreeView, which can be viewed using "ByAlbum",
    "ByArtist", "ByGenre"
    /// <see cref="SB54_CSAI.DatabaseAccess.DBAccess">database stored information</see>.
    Where the files
    /// that are stored for each view are retrieved from the database and shown to the user
    for
    /// each selected GenreTree node. This control is used as part of the
    /// <see cref="SB54_CSAI.MediaLibrary.uctMediaLibrary">media library GUI</see>
    /// </summary>
    public class GenreTree : System.Windows.Forms.TreeView
    {
        #region Instance Fields
        //Instance fields
        private DBAccess db;
        private ArrayList arGenres;
        private TreeNode tvnRoot;
        private TreeNode tvnByAlbum;
        private TreeNode tvnByArtist;
        private TreeNode tvnByGenre;
        #endregion
        #region Public Constructor
        /// <summary>
        /// Simply creates a new GenreTree object
        /// </summary>
        public GenreTree()
        {
        }
        #endregion
        #region Public Methods/Properties
        /// <summary>
        /// Initialised the GenreTree look and feel, then retrieves a list of all
        /// <see cref="SB54_CSAI.DatabaseAccess.DBAccess">
        /// database stored information</see> for Album, Artist and Genre. All retrieved
        /// data is then added to one of these 3 categories within the GenreTree
        /// </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();
            //add the root
            tvnRoot = new TreeNode("Your Music",0,0);
            Nodes.Add(tvnRoot);
            //add all Album, Artist and Genre data
            SetUpAlbumNodes();
            SetUpArtistNodes();
            SetUpGenreNodes();
        }
        #endregion
        #region Private Methods

```

```

/// <summary>
/// Retrieves all By Album meta data from the
/// <see cref="SB54_CSAI.DatabaseAccess.DBAccess">database</see>
/// </summary>
private void SetUpAlbumNodes()
{
    //create a new By Album tree node
    tvnByAlbum = new TreeNode("By Album",1,1);
    tvnRoot.Nodes.Add(tvnByAlbum);
    tvnByAlbum.EnsureVisible();
    //Get all the Albums and create ClassifiedTreeNode tree nodes
    //for each DISTINCT Album. The ClassifiedTreeNode will get
    //its own files from the database that match this each Album
    DataSet ds = db.getClassifiedData("ByAlbum");
    foreach (DataRow dr in ds.Tables["ClassifiedData"].Rows)
    {
        string albumName = dr["albumName"].ToString();
        if (! albumName.Equals(string.Empty))
        {
            if (!NodeExists(tvnByAlbum,albumName))
            {
                tvnByAlbum.Nodes.Add(new ClassifiedTreeNode(albumName, "ByAlbum", 2,
2));
            }
        }
    }
}

/// <summary>
/// Retrieves all By Artist meta data from the
/// <see cref="SB54_CSAI.DatabaseAccess.DBAccess">database</see>
/// </summary>
private void SetUpArtistNodes()
{
    //create a new By Album tree node
    tvnByArtist = new TreeNode("By Artist",1,1);
    tvnRoot.Nodes.Add(tvnByArtist);
    tvnByArtist.EnsureVisible();
    //Get all the Artists and create ClassifiedTreeNode tree nodes
    //for each DISTINCT Artist. The ClassifiedTreeNode will get
    //its own files from the database that match this each Artist
    DataSet ds = db.getClassifiedData("ByArtist");
    foreach (DataRow dr in ds.Tables["ClassifiedData"].Rows)
    {
        string artist = dr["artistName"].ToString();
        if (! artist.Equals(string.Empty))
        {
            if (!NodeExists(tvnByArtist,artist))
            {
                tvnByArtist.Nodes.Add(new ClassifiedTreeNode(artist, "ByArtist", 3,3));
            }
        }
    }
}

/// <summary>
/// Retrieves all By Genre meta data from the
/// <see cref="SB54_CSAI.DatabaseAccess.DBAccess">database</see>
/// </summary>
private void SetUpGenreNodes()
{
    //create a new By Genre tree node
    tvnByGenre = new TreeNode("By Genre",1,1);
    tvnRoot.Nodes.Add(tvnByGenre);
    tvnByGenre.EnsureVisible();
    //Get all the Genres and create ClassifiedTreeNode tree nodes
    //for each DISTINCT Genre. The ClassifiedTreeNode will get
    //its own files from the database that match this each Genre
    DataSet ds = db.getClassifiedData("ByGenre");
    foreach (DataRow dr in ds.Tables["ClassifiedData"].Rows)
    {
        string genre = dr["genreName"].ToString();

```

```

        if (! genre.Equals(string.Empty))
        {
            if (!NodeExists(tvnByGenre,genre))
            {
                tvnByGenre.Nodes.Add(new ClassifiedTreeNode(genre,"ByGenre",4,4));
            }
        }
    }
}

/// <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.Text.Equals(nodeName))
        {
            return true;
        }
    }
    return false;
}
#endregion
}
#endregion
#region ClassifiedTreeNode CLASS
/// <summary>
/// Provides a new inherited TreeView TreeNode that accepts a strig which dictates
/// what sort of clasified TreeNode is being constructed. Depending on what type
/// of ClassifiedTreeNode is being created a different <see cref="SB54_CSAI.
DatabaseAccess.DBAccess">
/// database</see> query will be sued to create this ClassifiedTreeNode file data
/// for use within the <see cref="SB54_CSAI.MediaLibrary.uctMediaLibrary">media library
GUI</see>
/// </summary>
public class ClassifiedTreeNode : System.Windows.Forms.TreeNode
{
    #region Instance Fields
    //Instance fields
    private ArrayList MP3Files = new ArrayList();
    private string classifiedType="";
    private string classifiedID="";
    private bool filesAdded=false;
    #endregion
    #region Public Constructor

    /// <summary>
    /// Initializes a new instance of the ClassifiedTreeNode class with the specified
    label
    /// text and images to display when the ClassifiedTreeNode is in a selected and
    unselected state.
    /// On construction the ClassifiedTreeNode will call a database method to get its
    own list of
    /// stored database files.
    /// </summary>
    /// <param name="text">The label Text of the new MP3TreeNode.</param>
    /// <param name="classifiedType">The type of tree node this should be, "ByArtist",
    "ByAlbum", "ByGenre".
    /// Each ClassifiedTreeNode will call a different database method, to get it own
    internal list of files
    /// so when a user clicks on a ClassifiedTreeNode, it will have already queried the
    database
    /// using the classifiedType value and retrieved all its owned files</param>
    /// <param name="imageIndex">The index value of Image to display when the

```

```

MP3TreeNode is unselected.</param>
    /// <param name="selectedIndex">The index value of Image to display when the
MP3TreeNode is selected.</param>
    public ClassifiedTreeNode(string text,string classifiedType,int imageIndex,int
selectedIndex)
        : base(text,imageIndex,selectedIndex)
    {
        //store parameters
        this.classifiedType = classifiedType;
        classifiedID = text;
    }
#endregion
#region Public Methods/Properties

    /// <summary>
    /// Returns an array of strings that represent this ClassifiedTreeNode files,
    /// which are retrieved from the <see cref="SB54_CSAI.DatabaseAccess.DBAccess">
    /// database</see> using the information about the classifiedType that was provided
on
    /// construction. Where the database query will also use the Text associated with
    /// this ClassifiedTreeNode as part of the WHERE clause, to limit the DataSet
returned
    /// values to being just the ones associated with this particular ClassifiedTreeNode
    /// For example a ClassifiedTreeNode constructed as follows
    /// new ClassifiedTreeNode("Appetite for Desruction","ByAlbum",2,2) would be
created
    /// with the text "Appetite for Desruction" and would use Image index 2 and would
    /// then query the database and SELECT all track that are stored where the album
    /// name is "Appetite for Desruction"
    /// </summary>
    /// <returns>An array of strings representing this ClassifiedTreeNode files</
returns>
    public string[] PlayableFiles
    {
        get
        {
            //if the files have not already been added to this node
            if (!filesAdded)
            {
                //go get the ClassifiedTreeNode files, and return them to the caller
                getExpandedNodeData();
            }
            return MP3Files.ToArray(Type.GetType("System.String")) as string[];
        }
    }

    /// <summary>
    /// Returns an boolean that indicates whether this ClassifiedTreeNode has files
    /// </summary>
    /// <returns>True if this ClassifiedTreeNode has files</returns>
    public bool hasPlayableFiles()
    {
        //if the files have not already been added to this node
        if (!filesAdded)
        {
            //go get the ClassifiedTreeNode files, and return whether this node has
files
            //or not to the caller
            getExpandedNodeData();
        }
        return MP3Files.Count > 0;
    }
#endregion
#region Private Methods
    /// <summary>
    /// Performs a query against the <see cref="SB54_CSAI.DatabaseAccess.DBAccess">
    /// database</see> using the information about the classifiedType that was provided
on
    /// construction and adds the files associated with this node to the internal
    /// files collection
    /// </summary>
    private void getExpandedNodeData()

```

```

    {
        //new database
        DBAccess db = new DBAccess();

        //Get ByArtist DataSet
        if (classifiedType.Equals("ByArtist"))
        {
            DataSet ds = db.getClassifiedNodeData("ByArtist", classifiedID);
            addFiles(ds);
        }
        //Get ByAlbum DataSet
        else if (classifiedType.Equals("ByAlbum"))
        {
            DataSet ds = db.getClassifiedNodeData("ByAlbum",classifiedID);
            addFiles(ds);
        }
        //Get ByGenre DataSet
        else if (classifiedType.Equals("ByGenre"))
        {
            DataSet ds = db.getClassifiedNodeData("ByGenre", classifiedID);
            addFiles(ds);
        }
    }

    /// <summary>
    /// Uses the results of the database query and simply loops through the
    /// DataSet parameter rows, adding the files contained to the files
    /// internal collection
    /// </summary>
    /// <param name="ds">The source DataSet that contains the database query results</
param>
    private void addFiles(DataSet ds)
    {
        //Loop through DataSet adding all Files retrieved for this ClassifiedTreeNode
        //to the files internal collection
        foreach (DataRow dr in ds.Tables["ClassifiedNodeData"].Rows)
        {
            string filePath = dr["folderName"].ToString() + @"\" + dr["fileName"].
ToString();
            if (! filePath.Equals(string.Empty))
            {
                try
                {
                    FileInfo f = new FileInfo(filePath);
                    MP3Files.Add(filePath);
                }
                catch(Exception ex)
                {
                    //simply catch exception
                }
            }
        }
        //set filesAdded flag to prevent future calls attempting to add files again
        filesAdded=true;
    }
    #endregion
}
#endregion
}

```