

```
using System;
using System.Data;
using System.IO;
using System.Data.OleDb;
using System.Windows.Forms;
using System.Collections;
using SB54_CSAI.MP3Lib;

namespace SB54_CSAI.DatabaseAccess
{
    #region DatabaseAccess CLASS
    /// <summary>
    /// This class provides a wrapper around all SQL calls to an underlying Access database
    file (*.MDB).
    /// There will be one Access database per ReMP3 application. 1 For the ReMP3 server
    application
    /// and another 1 for the ReMP3 client. It can be thought of as a auxiliary media
    storage
    /// file for the ReMP3 applications.
    /// <br></br>
    /// <br></br>
    /// This class makes use of the following technologies
    /// <br></br>
    /// <br></br>
    /// conventional ADO (Microsoft Data Access 2.6) where SQL strings/
    /// OLEDB commands will be used to INSERT and UPDATE underlying tables
    /// <br></br>
    /// <br></br>
    /// ADO .NET (Microsoft Data Access 2.7) where SQL strings will be uses to SELECT
    /// data but DataSets/DataAdaptor will be used to update the in memory tables, then
    these
    /// in memory DataSet tables will be sent back to the underlying database for remerging.
    /// <br></br>
    /// <br></br>
    /// Row updated events shall also be monitored when using ADO .NET, to ensure
    /// that the correct AutoNum primary key values can be retrieved for use with the
    DataSet
    /// tables prior to committing a database update using a DataAdaptor
    /// </summary>
    public class DBAccess
    {
        #region Instance Fields
        //Instance fields
        private OleDbConnection connection;
        private string connString = @"Provider=Microsoft.Jet.OLEDB.4.0;" +
            @"Data source=" + Application.StartupPath + @"\ReMp3Files.mdb";
        private static bool genresAdded=false;
        private MP3 workingMP3;
        private OleDbDataAdapter daArtist;
        private OleDbDataAdapter daAlbum;
        private OleDbDataAdapter daFolder;
        private int AutoArtistID;
        private int AutoAlbumID;
        private int AutoGenreID;
        private int AutoFolderID;
        private ArrayList arGenres= new ArrayList();
        private bool activeThreads=false;
        #endregion
        #region Public Constructor
        /// <summary>
        /// Simply creates a new DBAccess object
        /// </summary>
        public DBAccess()
        {
        }
        #endregion
        #region Public Methods/Properties

        /// <summary>
        /// Gets/Sets a field to show whether the database is being used by any process
        thread.
        /// This property is required to deal with the exceptions that occur for a given
```

```
process
    /// thread. This class should only shoe Exception messages if there is an active
thread.
    /// For example : If the user chooses to cancel the scanning of folders within the
    /// <see cref="SB54_CSAI.MediaLibrary.uctMediaLibrary">Media Library control </see>
    /// object, the database interaction process thread is stopped as it is on the same
thread
    /// as the scanning of folders, so under this circumstance, database Exception
messages
    /// should be inhbited to the user, As the database may be part way through
    /// an operation when the user chooses to cancel the scanning of folders, so the
database
    /// will raise an Exception (as the thread it was running on was stopped) that the
user no
    /// longer cares about.
    /// </summary>
public bool hasActiveThreads
{
    get
    {
        return this.activeThreads;
    }
    set
    {
        this.activeThreads= value;
    }
}

    /// <summary>
    /// Accepts a <see cref="SB54_CSAI.MP3Lib.MP3">MP3 file </see> as a parameter. The
MP3
    /// details for this MP3 file are then updated within the underlying database
    /// </summary>
    /// <param name="paramMP3">The workingMP3 that should be used when updating
    /// MP3 details within the underlying database</param>
public void updateMP3(ref MP3 paramMP3)
{
    try
    {
        workingMP3 = paramMP3;
        hasActiveThreads = true;

        //connect to database
        if (connectDB())
        {
            if (fileExists())
            {
                //FILE EXISTS - so need to update its ID3 info
                updateArtist();
                updateAlbum();
                getGenreNum();
                getFolderNum(workingMP3.filePath);
                updateMP3Details();
                disconnectDB();
                return;
            }
            else
            {
                //FILE DOESN'T EXIST - so exit
                disconnectDB();
                return;
            }
        }
    }
    catch(Exception ex)
    {
        if (hasActiveThreads)
        {
            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);
        }
    }
}
```

```

    }
    finally
    {
        disconnectDB();
        hasActiveThreads = false;
    }
}

/// <summary>
/// Accepts a <see cref="SB54_CSAI.MP3Lib.MP3">MP3 file </see> as a parameter. The
MP3
/// details for this MP3 file are then inserted within the underlying database
/// </summary>
/// <param name="paramMP3">The workingMP3 that should be used when updating
/// MP3 details within the underlying database</param>
public void insertMP3(ref MP3 paramMP3)
{
    try
    {
        workingMP3 = paramMP3;
        hasActiveThreads = true;

        //connect to database
        if (connectDB())
        {
            //INSERT new details
            updateArtist();
            updateAlbum();
            getGenreNum();
            getFolderNum(workingMP3.filePath);
            insertMP3Details();
            disconnectDB();
            return;
        }
    }
    catch(Exception ex)
    {
        if (hasActiveThreads)
        {
            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);
        }
    }
    finally
    {
        disconnectDB();
        hasActiveThreads = false;
    }
}

/// <summary>
/// Takes a Dns node name of the PC and retruns all folders oot folders registered
/// for the Dns node name
/// </summary>
/// <param name="node">The Dns node name of the PC that you wish to get the root
folders
/// for</param>
/// <returns>A <see cref="System.Data.DataSet">DataSet </see>which contains all
stored
/// root folders registered for the Dns node name</returns>
public DataSet GetRootFolders(string node)
{
    try
    {
        //connect to database
        if (connectDB())
        {
            //build query
            string sql = "SELECT folderName FROM tblFolders WHERE isRoot=1 AND

```

```

nodeName='" + node + "'";

        //build SQL objects
        OleDbDataAdapter daFolders=new OleDbDataAdapter(sql,connString);
        DataSet dsFolders = new DataSet();

        //fill DataSet and return it
        daFolders.FillSchema(dsFolders,SchemaType.Mapped,"Folders");
        daFolders.Fill(dsFolders,"Folders");
        disconnectDB();
        return dsFolders;
    }
    else
    {
        return null;
    }
}
catch(Exception ex)
{
    if (hasActiveThreads)
    {
        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);
    }
    return null;
}
finally
{
    disconnectDB();
}
}

/// <summary>
/// Deletes all files associated with the current folder ID
/// </summary>
public void RemoveFiles()
{
    try
    {
        //connect to database
        if (connectDB())
        {
            //create sql objects
            OleDbCommand com = new OleDbCommand();
            com.Connection = connection;
            com.CommandType = CommandType.Text;

            //create query
            char chQuote = '\'';
            string sql = @"DELETE FROM tblFiles " +
                "WHERE tblFiles.folderID = " + AutoFolderID;

            //execute query
            com.CommandText = sql;
            com.ExecuteNonQuery();
        }
    }
    catch(Exception ex)
    {
        if (hasActiveThreads)
        {
            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);
        }
    }
    finally

```

```

    {
        disconnectDB();
    }
}

/// <summary>
/// Takes a folder name and removes this folder and all associated files from
/// the underlying database
/// </summary>
/// <param name="fold">the folder name to remove from the database. Cascade changes
/// is turned on within the database, so all files associated with this folder will
/// also be released</param>
public void RemoveFolder(string fold)
{
    try
    {
        //connect to database
        if (connectDB())
        {
            //create sql objects
            OleDbCommand com = new OleDbCommand();
            com.Connection = connection;
            com.CommandType = CommandType.Text;

            //create query
            char chQuote = '\'';
            string sql = @"DELETE FROM tblFolders " +
                "WHERE ((tblFolders.folderName)=" + chQuote +
                fold.Trim() + chQuote + ")";

            //execute query
            com.CommandText = sql;
            com.ExecuteNonQuery();
        }
    }
    catch(Exception ex)
    {
        if (hasActiveThreads)
        {
            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);
        }
    }
    finally
    {
        disconnectDB();
    }
}

/// <summary>
/// If an Folder exists, fetches the folderID ID into the AutoFolderID field,
/// otherwise creates a new folder an then fetches its folderID ID into the
AutoFolderID field
/// </summary>
/// <param name="node">The Dns node name of the current PC</param>
/// <param name="fldr">The presently scanned folder</param>
/// <param name="isRoot">True if the folder is a root folder</param>
/// <param name="parentFolderID">The parent folder ID</param>
/// <returns></returns>
public bool updateFolder(string node,string fldr, int isRoot, int parentFolderID)
{
    //Get the Folder data
    try
    {
        //create query
        string sql = "SELECT * FROM tblFolders WHERE folderName = " +
            '\'' + fldr + '\'';

        //create SQL objects
        daFolder=new OleDbDataAdapter(sql,connString);
    }
}

```

```

        OleDbCommandBuilder cbFolder = new OleDbCommandBuilder(daFolder);
        DataSet dsFolder = new DataSet();

        //fill DataSet
        daFolder.FillSchema(dsFolder, SchemaType.Mapped, "Folders");
        daFolder.Fill(dsFolder, "Folders");
        daFolder.RowUpdated+=new OleDbRowUpdatedEventHandler(daFolder_RowUpdated);

        //if there are no rows, add a new one
        if (dsFolder.Tables["Folders"].Rows.Count == 0)
        {
            DataRow newRow=dsFolder.Tables["Folders"].NewRow();
            newRow["nodeName"]= node;
            newRow["folderName"]= fldr;
            newRow["isRoot"]= isRoot;
            newRow["parentFolderID"]= parentFolderID;
            dsFolder.Tables["Folders"].Rows.Add(newRow);
            daFolder.Update(dsFolder.Tables["Folders"]);
            return false;
        }
        else
        {
            //fetch the folderID ID into the AutoFolderID field
            AutoFolderID = Int16.Parse(dsFolder.Tables["Folders"].Rows[0]["folderID"]
].ToString());
            return true;
        }
    }
    catch(Exception ex)
    {
        if (hasActiveThreads)
        {
            MessageBox.Show("Problem with DB access-\n\n updateFolder: "
                + "\r\n\r\n query: "
                + "\r\n\r\n\r\n" + ex.ToString(),
                "Database Access ERROR", MessageBoxButtons.OK,
                MessageBoxIcon.Error);
        }
        return false;
    }
}

/// <summary>
/// Fetches all sub folders for a given parent folder using the AutoFolderID of the
/// parent folder
/// </summary>
/// <param name="parentFolderID">The AutoFolderID of the parent folder</param>
/// <returns>A <see cref="System.Data.DataSet">DataSet </see>which contains all the
sub folders
/// of the folder that has the AutoFolderID specified by the parentFolderID
parameter
/// </returns>
public DataSet getAllLoggedSubFolders(int parentFolderID)
{
    try
    {
        //build query
        string sql= "SELECT tblFolders.folderName " +
            "FROM tblFolders " +
            "WHERE tblFolders.parentFolderID =" + parentFolderID;

        //build SQL objects
        OleDbDataAdapter da=new OleDbDataAdapter(sql,connString);
        DataSet ds = new DataSet();

        //Fill DataSet and return it
        da.FillSchema(ds, SchemaType.Mapped, "subFolders");
        da.Fill(ds, "subFolders");
        return ds;
    }
    catch(Exception ex)
    {

```

```
        MessageBox.Show("Problem with DB access-\n\n  getAllLoggedSubFolders: "
            + "\r\n\r\n          query: "
            + "\r\n\r\n\r\n" + ex.ToString(),
            "Database Access ERROR", MessageBoxButtons.OK,
            MessageBoxIcon.Error);
        return null;
    }
}

/// <summary>
/// Takes a folderName and returns the AutoFolderID for it
/// </summary>
/// <param name="folderName">the folder to get the AutoFolderID for</param>
/// <returns>int which represents the AutoFolderID associated with the
/// folderName parameter</returns>
public int getFolderNum(string folderName)
{
    try
    {
        //create query
        char chQuote = '\'';
        string sql = "SELECT folderID " +
            "FROM tblFolders " +
            "WHERE ((tblFolders.folderName)=" + chQuote +
            folderName + chQuote + ")";

        //create sql objects
        OleDbCommand com = new OleDbCommand();
        com.Connection = connection;
        com.CommandType = CommandType.Text;

        //execute query
        com.CommandText = sql;
        AutoFolderID = (int)com.ExecuteScalar();
        return AutoFolderID;
    }
    catch(Exception ex)
    {
        if (hasActiveThreads)
        {
            MessageBox.Show("Problem with DB access-\n\n  getFolderNum: "
                + "\r\n\r\n          query: "
                + "\r\n\r\n\r\n" + ex.ToString(),
                "Database Access ERROR", MessageBoxButtons.OK,
                MessageBoxIcon.Error);
            disconnectDB();
            return -1;
        }
    }
    return -1;
}

/// <summary>
/// check to see if an entry exists in the database that matches the
/// current AutoFolderID
/// </summary>
/// <returns>true if the database contains this folder</returns>
public bool folderFileExists()
{
    try
    {
        //build query
        string sql = "SELECT tblFiles.fileName " +
            "FROM tblFiles " +
            "WHERE tblFiles.folderID=" + AutoFolderID ;

        //build SQL objects
        OleDbDataAdapter daFolder=new OleDbDataAdapter(sql,connString);
        DataSet dsFolder = new DataSet();

        //fill DataSet, and return status
        daFolder.FillSchema(dsFolder,SchemaType.Mapped,"Folder");
    }
}
```

```

        daFolder.Fill(dsFolder, "Folder");
        return (dsFolder.Tables["Folder"].Rows.Count > 0);
    }
    catch(Exception ex)
    {
        if (hasActiveThreads)
        {
            MessageBox.Show("Problem with DB access-\n\n folderFileExists: "
                + "\r\n\r\n query: "
                + "\r\n\r\n\r\n" + ex.ToString(),
                "Database Access ERROR", MessageBoxButtons.OK,
                MessageBoxIcon.Error);
        }
        return false;
    }
}

/// <summary>
/// Add all the genres contained within the arGenres parameter
/// to the underlying database
/// </summary>
/// <param name="arGenres">An ArrayList from the <see cref="SB54_CSAI.
Singleton_Genres.MThreadSingleton_genres">
/// genres singleton </see> that is enumerated through to add genre entries to the
database</param>
public void createGenres(ArrayList arGenres)
{
    this.arGenres = arGenres;
    //only add the genres if they have not been done already
    if (!genresAdded)
    {
        try
        {
            //connect to database
            if (connectDB())
            {
                //create sql objects
                OleDbCommand com = new OleDbCommand();
                com.Connection = connection;
                com.CommandType = CommandType.Text;

                //if the genres already exist, job is already done, so return
                if (genresExist())
                {
                    return;
                }
                else
                {
                    IEnumerator enGenres = arGenres.GetEnumerator();
                    //reset incoming genre IEnumerator
                    enGenres.Reset();
                    //add all genres to database
                    while(enGenres.MoveNext())
                    {
                        //build query and execute, once per Genre read
                        char chQuote = '\'';
                        string sql = "INSERT INTO tblGenre (genreName) VALUES (" +
                            chQuote + (string)enGenres.Current + chQuote + ")";
                        com.CommandText = sql;
                        com.ExecuteNonQuery();
                    }
                    disconnectDB();
                    genresAdded = true;
                }
            }
        }
        catch(Exception ex)
        {
            if (hasActiveThreads)
            {

```

```
        MessageBox.Show("Problem with DB access-\n\n    createGenres: "
            + "\r\n\r\n        query: "
            + "\r\n\r\n\r\n" + ex.ToString(),
            "Database Access ERROR", MessageBoxButtons.OK,
            MessageBoxIcon.Error);
    }
}

/// <summary>
/// Disconnected from the database if connection is ConnectionState.Open
/// </summary>
public void disconnectDB()
{
    try
    {
        //see if the connection is currently open
        if (connection.State == ConnectionState.Open)
        {
            connection.Close();
        }
    }
    catch(Exception ex)
    {
        if (hasActiveThreads)
        {
            MessageBox.Show("Problem with DBAccess-\n\n    disconnectDB: "
                + "\r\n\r\n        " +
                "\r\n\r\n\r\n" + ex.ToString(),
                "Database Access ERROR", MessageBoxButtons.OK,
                MessageBoxIcon.Error);
        }
    }
}

/// <summary>
/// Connects to the database if it is not already connected
/// </summary>
/// <returns>true if managed to open connection to database</returns>
public bool connectDB()
{
    // Open the connection
    try
    {
        //is the connection already open, maybe in the middle of a multiple
        //sql queries that relies on open connection
        if (connection != null && connection.State == ConnectionState.Open)
        {
            return true;
        }
        else
        {
            // Connection object
            connection = new OleDbConnection(connString);
            connection.Open();
            return true;
        }
    }
    catch(Exception ex)
    {
        if (hasActiveThreads)
        {
            MessageBox.Show("Problem with DBAccess-\n\n    connectDB : "
                + connString + "\r\n\r\n        " +
                "\r\n\r\n\r\n" + ex.ToString(),
                "Database Access ERROR", MessageBoxButtons.OK,
                MessageBoxIcon.Error);
        }
        return false;
    }
}
}
```

```
    /// <summary>
    /// Takes the values contained in the classifiedType parameter and returns a DataSet
    with the
    /// corresponding meta data associated with the classifiedType
    /// </summary>
    /// <param name="classifiedType">Comes from <see cref="SB54_CSAI.MediaLibrary.
ClassifiedTreeNode">GenreTreeNode </see>
    /// and can be one of three values ByArtist, ByAlbum, ByGenre</param>
    /// <returns>A <see cref="System.Data.DataSet">DataSet </see>which contains all the
meta data associated
    /// with the classifiedType. For example if the classifiedType was ByArtist the
DataSet would be filled
    /// as follows
    /// <br></br>
    /// <br></br>
    /// SELECT tblArtist.artistName FROM tblArtist INNER JOIN tblFiles ON tblArtist.
artistID = tblFiles.artistID
    /// </returns>
    public DataSet getClassifiedData(string classifiedType)
    {
        string sql="";

        if (classifiedType.Equals("ByArtist"))
        {
            sql = "SELECT tblArtist.artistName " +
                "FROM tblArtist INNER JOIN tblFiles ON " +
                "tblArtist.artistID = tblFiles.artistID";
        }
        else if (classifiedType.Equals("ByAlbum"))
        {
            sql = "SELECT DISTINCT tblAlbum.albumName " +
                "FROM tblAlbum INNER JOIN tblFiles ON tblAlbum.albumID = tblFiles.
albumID";
        }
        else if (classifiedType.Equals("ByGenre"))
        {
            sql = "SELECT DISTINCT tblGenre.genreName " +
                "FROM tblGenre INNER JOIN tblFiles ON tblGenre.genreID = tblFiles.
genreID";
        }

        //Get the Classified data based on logic above data
        try
        {
            //build SQL objects
            OleDbDataAdapter da=new OleDbDataAdapter(sql,connString);
            DataSet ds = new DataSet();

            //Fill DataSet and return it
            da.FillSchema(ds,SchemaType.Mapped,"ClassifiedData");
            da.Fill(ds,"ClassifiedData");

            return ds;
        }
        catch(Exception ex)
        {
            MessageBox.Show("Problem with DB access-\n\n getClassifiedData: "
                + "\r\n\r\n query: "
                + "\r\n\r\n\r\n" + ex.ToString(),
                "Database Access ERROR", MessageBoxButtons.OK,
                MessageBoxIcon.Error);
            return null;
        }
    }

    /// <summary>
    /// Takes the values contained in the classifiedType/classifiedID parameters and
returns a DataSet
    /// with the corresponding meta data associated with the classifiedType/
```

```

classifiedID
    /// </summary>
    /// <param name="classifiedType">Comes from <see cref="SB54_CSAI.MediaLibrary.
ClassifiedTreeNode">GenreTreeNode </see>
    /// and can be one of three values ByArtist, ByAlbum, ByGenre</param>
    /// <param name="classifiedID">Comes from <see cref="SB54_CSAI.MediaLibrary.
ClassifiedTreeNode">GenreTreeNode </see>
    /// and can any unique value that can be used to fetch meta data from the underlying
database</param>
    /// <returns>A <see cref="System.Data.DataSet">DataSet </see>which contains all the
meta data associated
    /// with the classifiedType/classifiedID. For example if the classifiedType was
ByArtist and the
    /// classifiedID was "RammStein" the DataSet would be filled as follows
    /// <br></br>
    /// <br></br>
    /// SELECT tblFolders.folderName, tblFiles.fileName FROM tblArtist INNER JOIN
(tblFolders INNER JOIN tblFiles ON
    /// tblFolders.folderID = tblFiles.folderID) ON tblArtist.artistID = tblFiles.
artistID
    /// WHERE (((tblArtist.artistName)='RammStein'))
    /// </returns>
    public DataSet getClassifiedNodeData(string classifiedType,string classifiedID)
    {
        string sql="";

        if (classifiedType.Equals("ByArtist"))
        {
            sql = "SELECT tblFolders.folderName, tblFiles.fileName " +
                "FROM tblArtist INNER JOIN (tblFolders INNER JOIN tblFiles ON " +
                "tblFolders.folderID = tblFiles.folderID) ON tblArtist.artistID =
tblFiles.artistID " +
                "WHERE (((tblArtist.artistName)='" + classifiedID + "'));"
        }
        else if (classifiedType.Equals("ByAlbum"))
        {
            sql = "SELECT tblFolders.folderName, tblFiles.fileName " +
                "FROM tblAlbum INNER JOIN (tblFolders INNER JOIN tblFiles ON " +
                "tblFolders.folderID = tblFiles.folderID) ON tblAlbum.albumID = tblFiles
.albumID " +
                "WHERE (((tblAlbum.albumName)='" + classifiedID + "'));"
        }
        else if (classifiedType.Equals("ByGenre"))
        {
            sql = "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)='" + classifiedID + "'));"
        }

        //Get the Classified data based on logic above data
        try
        {
            //build SQL objects
            OleDbDataAdapter da=new OleDbDataAdapter(sql,connString);
            DataSet ds = new DataSet();

            //fill dataSet and return it
            da.FillSchema(ds,SchemaType.Mapped,"ClassifiedNodeData");
            da.Fill(ds,"ClassifiedNodeData");

            return ds;
        }
        catch(Exception ex)
        {
            MessageBox.Show("Problem with DB access-\n\n    getClassifiedNodeData: "
                + "\r\n\r\n        query: "
                + "\r\n\r\n\r\n\r\n" + ex.ToString(),
                "Database Access ERROR", MessageBoxButtons.OK,
                MessageBoxIcon.Error);
            return null;
        }
    }

```

```

    }
}
#endregion
#region Private Methods / Properties
/// <summary>
/// updates all MP3 details for a given MP3 file
/// </summary>
private void updateMP3Details()
{
    try
    {
        //create sql objects
        OleDbCommand com = new OleDbCommand();
        com.Connection = connection;
        com.CommandType = CommandType.Text;

        //update the MP3 Details
        char chQuote = '"';
        string sql = "UPDATE tblFiles SET artistID = " + AutoArtistID +
            ", albumID = " + AutoAlbumID +
            ", [year] = '" + workingMP3.id3Year +
            "', comment = '" + workingMP3.id3Comment +
            "', genreID = " + AutoGenreID +
            " WHERE folderID=" + AutoFolderID +
            " AND fileName=" + chQuote + workingMP3.fileFileName +
            chQuote;

        com.CommandText = sql;
        com.ExecuteNonQuery();
    }
    catch(OleDbException ex)
    {
        if (hasActiveThreads)
        {
            MessageBox.Show("Problem with DB access-\n\n updateMP3Details: "
                + "\r\n\r\n query: "
                + "\r\n\r\n\r\n" + ex.ToString(),
                "Database Access ERROR", MessageBoxButtons.OK,
                MessageBoxIcon.Error);
        }
    }
}

/// <summary>
/// inserts a new MP3 file
/// </summary>
private void insertMP3Details()
{
    try
    {
        //create sql objects
        OleDbCommand com = new OleDbCommand();
        com.Connection = connection;
        com.CommandType = CommandType.Text;

        string year = (!workingMP3.id3Year.Equals(String.Empty) ? workingMP3.id3Year
            : DateTime.Now.Year.ToString());
        string comment = (!workingMP3.id3Comment.Equals(String.Empty) ? workingMP3.
            id3Comment : "No Comment exists");

        //CMD FORMAT = INSERT INTO tblFiles (folderID,fileName,artistID,albumID,
        year,comment,genreID)
        // VALUES (556,"file1",2,2,2004,"test",2)
        string sql = "INSERT INTO tblFiles (folderID,fileName,artistID,albumID,
        [year],comment,genreID) "
        + "VALUES (" + AutoFolderID + "," + "'" + workingMP3.
        fileFileName + "'" + "," +
        + AutoArtistID + "," + AutoAlbumID + "," + "'" + year + "',"
        + "'" + comment + "'," + AutoGenreID + ")";

        com.CommandText = sql;
        com.ExecuteNonQuery();
    }
}

```

```

        catch(OleDbException ex)
        {
            if (hasActiveThreads)
            {
                MessageBox.Show("Problem with DB access-\n\n insertMP3Details: "
                    + "\r\n\r\n          query: "
                    + "\r\n\r\n\r\n" + ex.ToString(),
                    "Database Access ERROR", MessageBoxButtons.OK,
                    MessageBoxIcon.Error);
            }
        }
    }

    /// <summary>
    /// If an Artist exists for the current MP3, fetches the Artist ID into the
    AutoArtistID field,
    /// otherwise creates a new Artist an then fetches its Artist ID into the
    AutoArtistID field
    /// </summary>
    private void updateArtist()
    {
        //Get the Artist data
        try
        {
            //build query
            string sql = "SELECT * FROM tblArtist WHERE artistName= " +
                "'" + @workingMP3.id3Artist + "'";

            //build sql objects
            daArtist=new OleDbDataAdapter(sql,connString);
            OleDbCommandBuilder cbArtist = new OleDbCommandBuilder(daArtist);
            DataSet dsArtist = new DataSet();

            //fill DataSet
            daArtist.FillSchema(dsArtist,SchemaType.Mapped,"Artist");
            daArtist.Fill(dsArtist,"Artist");
            daArtist.RowUpdated+=new OleDbRowUpdatedEventHandler(daArtist_RowUpdated);

            //if there are no records add one
            if (dsArtist.Tables["Artist"].Rows.Count == 0)
            {
                DataRow newRow=dsArtist.Tables["Artist"].NewRow();
                newRow["artistName"]= workingMP3.id3Artist;
                dsArtist.Tables["Artist"].Rows.Add(newRow);
                daArtist.Update(dsArtist.Tables["Artist"]);
            }
            else
            {
                //get the Artist ID
                AutoArtistID = Int16.Parse(dsArtist.Tables["Artist"].Rows[0]["artistID"]);
            }
        }
        .ToString());
    }
    catch(Exception ex)
    {
        if (hasActiveThreads)
        {
            MessageBox.Show("Problem with DB access-\n\n updateArtist: "
                + "\r\n\r\n          query: "
                + "\r\n\r\n\r\n" + ex.ToString(),
                "Database Access ERROR", MessageBoxButtons.OK,
                MessageBoxIcon.Error);
        }
    }
}

/// <summary>
/// Gets the genre ID and places it into the AutoGenreID field for a specific
genreName
/// </summary>
private void getGenreNum()
{
    try
    {

```

```

        //create sql objects
        OleDbCommand com = new OleDbCommand();
        com.Connection = connection;
        com.CommandType = CommandType.Text;

        //build query
        string genre = (string)arGenres[(int)workingMP3.id3Genre];
        string sql = "SELECT genreID FROM tblGenre WHERE genreName= " +
            "'" + genre + "'";

        //run query
        com.CommandText = sql;
        AutoGenreID = (int)com.ExecuteScalar();
    }
    catch(Exception ex)
    {
        if (hasActiveThreads)
        {
            MessageBox.Show("Problem with DB access-\n\n getGenreNum: "
                + "\r\n\r\n query: "
                + "\r\n\r\n\r\n" + ex.ToString(),
                "Database Access ERROR", MessageBoxButtons.OK,
                MessageBoxIcon.Error);
        }
    }
}

/// <summary>
/// If an Album exists for the current MP3, fetches the Album ID into the
AutoAlbumID field,
/// otherwise creates a new Album an then fetches its Album ID into the AutoAlbumID
field
/// </summary>
private void updateAlbum()
{
    //Get the Album data
    try
    {
        //build query
        string sql = @"SELECT * FROM tblAlbum WHERE albumName= " +
            "'" + workingMP3.id3Album + "'";

        //build sql objects
        daAlbum=new OleDbDataAdapter(sql,connString);
        OleDbCommandBuilder cbArtist = new OleDbCommandBuilder(daAlbum);
        DataSet dsAlbum = new DataSet();

        //fill DataSet
        daAlbum.FillSchema(dsAlbum, SchemaType.Mapped, "Album");
        daAlbum.Fill(dsAlbum, "Album");
        daAlbum.RowUpdated+=new OleDbRowUpdatedEventHandler(daAlbum_RowUpdated);

        //if there are no records add one
        if (dsAlbum.Tables["Album"].Rows.Count == 0)
        {
            DataRow newRow=dsAlbum.Tables["Album"].NewRow();
            newRow["albumName"] = workingMP3.id3Album;
            dsAlbum.Tables["Album"].Rows.Add(newRow);
            daAlbum.Update(dsAlbum.Tables["Album"]);
        }
        else
        {
            //get the Album ID
            AutoAlbumID = Int16.Parse(dsAlbum.Tables["Album"].Rows[0]["albumID"]);
        }
    }
    catch(Exception ex)
    {
        if (hasActiveThreads)
        {
            MessageBox.Show("Problem with DB access-\n\n updateAlbum: "
                + "\r\n\r\n query: "

```

```

        + "\r\n\r\n\r\n" + ex.ToString(),
        "Database Access ERROR", MessageBoxButtons.OK,
        MessageBoxIcon.Error);
    }
}

/// <summary>
/// Check to see if the workingMP3 file exists within the database,
/// this is used in conjunction with the <see cref="SB54_CSAI.MP3Editor.uctMP3Editor
">
/// MP3 Editor control </see> control, such that if a file exists in the database,
/// its details will be updated using the new ID3 tag information from the MP3
Editor
/// control GUI fields
/// </summary>
/// <returns>true if the database contains this file</returns>
private bool fileExists()
{
    //build query
    char chQuote = '"';
    string sql = "SELECT tblFolders.folderName, tblFiles.fileName " +
        "FROM tblFolders INNER JOIN tblFiles ON tblFolders.folderID = tblFiles.
folderID " +
        "WHERE (((tblFolders.folderName)=" + chQuote + workingMP3.filePath +
chQuote +
        ") AND ((tblFiles.fileName)=" + chQuote + workingMP3.fileFileName + chQuote
+ ")))";

    try
    {
        //build objects
        OleDbDataAdapter daFile=new OleDbDataAdapter(sql,connString);
        DataSet dsFile = new DataSet();

        //Fill the DataSet, and return rowcount, if > 0 file exists
        daFile.FillSchema(dsFile,SchemaType.Mapped,"File");
        daFile.Fill(dsFile,"File");
        return (dsFile.Tables["File"].Rows.Count > 0);
    }
    catch(Exception ex)
    {
        if (hasActiveThreads)
        {
            MessageBox.Show("Problem with DB access-\n\n fileExists: "
                + "\r\n\r\n query: "
                + "\r\n\r\n\r\n" + ex.ToString(),
                "Database Access ERROR", MessageBoxButtons.OK,
                MessageBoxIcon.Error);
        }
        return false;
    }
}

/// <summary>
/// Check to see if the genres exist within the database
/// </summary>
/// <returns>true if the genres table has already been populated,
/// false otherwise</returns>
private bool genresExist()
{
    //build query
    string sql = "SELECT tblGenre.genreName FROM tblGenre";

    //Get get Genres data
    try
    {
        //build SQL objects
        OleDbDataAdapter daGenres=new OleDbDataAdapter(sql,connString);
        DataSet dsGenres = new DataSet();

        //fill DataSet
        daGenres.FillSchema(dsGenres,SchemaType.Mapped,"Genres");
    }
}

```

```

        daGenres.Fill(dsGenres, "Genres");
        return (dsGenres.Tables["Genres"].Rows.Count > 0);
    }
    catch(Exception ex)
    {
        if (hasActiveThreads)
        {
            MessageBox.Show("Problem with DB access-\n\n    genresExist: "
                + "\r\n\r\n        query: "
                + "\r\n\r\n\r\n\r\n" + ex.ToString(),
                "Database Access ERROR", MessageBoxButtons.OK,
                MessageBoxIcon.Error);
        }
        return false;
    }
}

/// <summary>
/// Event handler that occurs when the internal <see cref="System.Data.OleDb.
OleDbDataAdapter">
/// daArtist </see>row updated event happens. Basically when the DataSet bound to
the DataAdaptor
/// is updated. This event is used to retrieve the AutoNumber value of the
underlying table for
/// use with the internal field AutoArtistID. This value can then be used to update
all related
/// tables with the correct foreign key information
/// </summary>
/// <param name="sender">The source object that raised the event</param>
/// <param name="e">The event arguments</param>
private void daArtist_RowUpdated(object sender, OleDbRowUpdatedEventArgs e)
{
    try
    {
        //get the IDENTITY (AutoNum of the PRIMARY_KEY in the database, on DataSet
inserts)
        if(e.Status == UpdateStatus.Continue && e.StatementType == StatementType.
Insert)
        {
            OleDbCommand idCMD = new OleDbCommand("SELECT @@IDENTITY",
(OleDbConnection) e.Command.Connection);
            AutoArtistID = Int16.Parse(idCMD.ExecuteScalar().ToString());
        }
    }
    catch(Exception ex)
    {
        if (hasActiveThreads)
        {
            MessageBox.Show("Problem with DBAccess-\n\n    daArtist_RowUpdated : "
                + connString + "\r\n\r\n        " +
                "\r\n\r\n\r\n\r\n" + ex.ToString(),
                "Database Access ERROR", MessageBoxButtons.OK,
                MessageBoxIcon.Error);
        }
    }
}

/// <summary>
/// Event handler that occurs when the internal <see cref="System.Data.OleDb.
OleDbDataAdapter">
/// daAlbum </see>row updated event happens. Basically when the DataSet bound to the
DataAdaptor
/// is updated. This event is used to retrieve the AutoNumber value of the
underlying table for
/// use with the internal field AutoAlbumID. This value can then be used to update
all related
/// tables with the correct foreign key information
/// </summary>
/// <param name="sender">The source object that raised the event</param>
/// <param name="e">The event arguments</param>
private void daAlbum_RowUpdated(object sender, OleDbRowUpdatedEventArgs e)
{
    try

```

```
        {
            //get the IDENTITY (AutoNum of the PRIMARY_KEY in the database, on DataSet
inserts)
            if(e.Status == UpdateStatus.Continue && e.StatementType == StatementType.
Insert)
            {
                OleDbCommand idCMD = new OleDbCommand("SELECT @@IDENTITY",
(OleDbConnection) e.Command.Connection);
                AutoAlbumID = Int16.Parse(idCMD.ExecuteScalar().ToString());
            }
        }
    catch(Exception ex)
    {
        if (hasActiveThreads)
        {
            MessageBox.Show("Problem with DBAccess-\n\n daArtist_RowUpdated : "
+ connString + "\r\n\r\n" +
"\r\n\r\n\r\n" + ex.ToString(),
"Database Access ERROR", MessageBoxButtons.OK,
MessageBoxIcon.Error);
        }
    }
}

/// <summary>
/// Event handler that occurs when the internal <see cref="System.Data.OleDb.
OleDbDataAdapter">
/// daFolder </see>row updated event happens. Basically when the DataSet bound to
the DataAdaptor
/// is updated. This event is used to retrieve the AutoNumber value of the
underlying table for
/// use with the internal field AutoFolderID. This value can then be used to update
all related
/// tables with the correct foreign key information
/// </summary>
/// <param name="sender">The source object that raised the event</param>
/// <param name="e">The event arguments</param>
private void daFolder_RowUpdated(object sender, OleDbRowUpdatedEventArgs e)
{
    try
    {
        //get the IDENTITY (AutoNum of the PRIMARY_KEY in the database, on DataSet
inserts)
        if(e.Status == UpdateStatus.Continue && e.StatementType == StatementType.
Insert)
        {
            OleDbCommand idCMD = new OleDbCommand("SELECT @@IDENTITY",
(OleDbConnection) e.Command.Connection);
            AutoFolderID = Int16.Parse(idCMD.ExecuteScalar().ToString());
        }
    }
    catch(Exception ex)
    {
        if (hasActiveThreads)
        {
            MessageBox.Show("Problem with DBAccess-\n\n daArtist_RowUpdated : "
+ connString + "\r\n\r\n" +
"\r\n\r\n\r\n" + ex.ToString(),
"Database Access ERROR", MessageBoxButtons.OK,
MessageBoxIcon.Error);
        }
    }
}
#endregion
}
#endregion
}
```