```
using System.IO;
using JRO;
using System;
using System.Windows.Forms;

// This is the code for your desktop app.
// Press Ctrl+F5 (or go to Debug > Start Without Debugging) to run your app.

namespace TestReadAccessDB
{
    public partial class FrmMain : Form
    {
        public FrmMain()
        {
            InitializeComponent();
        }

        public static class Gl // global variables
        {
            // Word
            public static Microsoft.Office.Interop.Word.Application WordApp =
                new Microsoft.Office.Interop.Word.Application();
            public static Microsoft.Office.Interop.Word.Document WordDoc;
            public static string sWordPath;
            public static string sWordFileName;
            public static string sTextPath; // list of Word document paths
            public static int iNumWordFiles; // lines in above text file, or 1
            public static int iCurrWordFileNum;
            // Access
            public static ADODB.Connection cn = new ADODB.Connection();
            public static ADODB.Recordset rs = new ADODB.Recordset();
            public static string sConn;
            public static string sPart1AccessMDBString =
              "Provider=Microsoft.Jet.OLEDB.4.0;Data Source=";
            public static string sPart2AccessString = ";Jet OLEDB:Database
              Password=''";
            public static object dummy = Type.Missing; // for Access INSERT/UPDATE/
              DELETE
            // general
            public static int x; // keeps track of what char in the Word doc we're on
            public static int iOrder; // order of items in worship,
              WorshipActivityInstances
            public static int iLineNo; // line being processed on current document
            public static string sWorshipDate;
            public static string sPageNo;
            public static string sKey;
            public static Boolean bIsKey;
            public static string sSongTitle;
            public static char cSongType; // from check song form
        } // Gl - global variables

        private int ActivityLeaderID(string psActivityLeaderName)
```

```
        {
            // This function receives an activity leader's name,
            // and looks for the person's ID. If the person is
            // not in the Persons table, the person is added
            // to it. Either way, the person's ID is returned.
            if(psActivityLeaderName == ""
            || psActivityLeaderName == "?")
                return 0; // unknown
            ADODB.Recordset rs = new ADODB.Recordset();
            string[] asNames; // will hold first and last name separately
            int iID;
            string sFirstName;
            string sLastName;
            asNames = psActivityLeaderName.Split(' ');
            switch(asNames.Length)
            {
                case 2: // assumed first and last name, 1 space
                    sFirstName = asNames[0]; // first name
                    sLastName = asNames[1]; // last name
                    break;
                case 3: // assumed first name, and 2-part last name, 2 spaces
                    sFirstName = asNames[0]; // first name
                    sLastName = asNames[1] + " " + asNames[2]; // last name
                    break;
                case 1: // assumed first name only, no spaces
                    sFirstName = asNames[0]; // first name
                    sLastName = ""; // last name
                    break;
                case 0:
                    MessageBox.Show("Activity leader on line " + Gl.iLineNo +
                        "is blank; please fix."); // should never happen here
                    return 0;
                default:
                    MessageBox.Show("The name \"" + psActivityLeaderName +
                        "\" contains " + asNames.Length + " parts. Unable to use.");
                    return 0;
            }
            rs.Open("SELECT ID FROM Persons" +
                " WHERE FirstName = " + Literal(sFirstName) +
                " AND LastName = " + Literal(sLastName), Gl.cn,
                ADODB.CursorTypeEnum.adOpenDynamic,
                ADODB.LockTypeEnum.adLockOptimistic);
            if(rs.BOF && rs.EOF) // person not found, add
            {
                iID = NextAvailableIDNumber("Persons");
                Gl.cn.Execute("INSERT INTO Persons VALUES(" +
                    iID +
                    ", " + Literal(sFirstName) +
                    ", " + Literal(sLastName) + ", '')",
                    out Gl.dummy, 0);
            }
            else // person found
```

```
            {
                rs.MoveFirst();
                iID = rs.Fields["ID"].Value;
            }
            return iID;
        } // ActivityLeaderID

        private void AddSong()
        {
            string sBookID;
            int iID;
            // see if song is already in the Songs table
            iID = GetSongID();
            if (iID != 0) // song in input file found
                Gl.cSongType = 'Y';
            else
            {
                // not in Songs table, see if misspelled, truncated, actually need to
                  add,
                // pull up Check Song form, which will set cSongType to 'S', 'N' or
                  'E'
                // and put song title to be used in sSongTitle, if not already there
                frmCheckSong formCheckSong = new frmCheckSong();
                formCheckSong.ShowDialog();
            }
            switch(Gl.cSongType)
            {
                case 'Y': // do nothing, we already have the ID
                    break;
                case 'N':
                case 'E': // add new song, from input file ('N'), or user typed on
                  form ('E')
                {
                    // add song
                    iID = NextAvailableIDNumber("Songs");
                    if (Gl.sPageNo == "")
                        sBookID = "NULL";
                    else
                        sBookID = DefaultBookID();
                    Gl.cn.Execute("INSERT INTO Songs VALUES (" + iID +
                        ", " + Literal(Gl.sSongTitle) + ", '', '', '', " +
                        Literal(Gl.sKey) + ", " + Literal(Gl.sPageNo) + ", " + sBookID
                       +
                        ", NULL, NULL, NULL, NULL, NULL, NULL, NULL, NULL)",
                        out Gl.dummy, 0);
                        break;
                }
                case 'S': // existing song selected from drop-down in Check Song form
                    iID = GetSongID();
                    break;
            } // switch
            // either way, add record to WorshipActivityInstances;
```

```
            // worship leader will be filled in later, 0 for now
            Gl.cn.Execute("INSERT INTO WorshipActivityInstances VALUES (" +
                NextAvailableIDNumber("WorshipActivityInstances") + ", 1, " + iID +
                ", 0, " + Gl.iOrder + ", " + Literal(Gl.sWorshipDate) + "," +
                " '" + Gl.sWordFileName + "', NULL, '" +
                DateTime.Today.ToShortDateString() + "')", out Gl.dummy, 0);
        } // AddSong

        private void CmdAccessBrowse_Click(object sender, EventArgs e)
        {
            OpenFileDialog dlgAccess = new OpenFileDialog
            {
                InitialDirectory = "c:\\",
                Filter = "Database files (*.mdb)|*.mdb",
                FilterIndex = 0,
                RestoreDirectory = true
            };
            if (dlgAccess.ShowDialog() == DialogResult.OK)
                txtAccessFilePath.Text = dlgAccess.FileName;
        } // CmdAccessBrowse_Click

        private void CmdCompactRepair_Click(object sender, EventArgs e)
        {
            int x;
            int iDotBeforeSuffix;
            string sTempAccessFilePath = "";
            x = txtAccessFilePath.Text.Length - 1;
            while (txtAccessFilePath.Text.Substring(x, 1) != ".")
                x--;
            iDotBeforeSuffix = x;
            for(x = 0; x != iDotBeforeSuffix; x++)
                sTempAccessFilePath += txtAccessFilePath.Text.Substring(x, 1);
            sTempAccessFilePath += "_TEMP";
            while(x <= txtAccessFilePath.Text.Length - 1)
            {
                sTempAccessFilePath += txtAccessFilePath.Text.Substring(x, 1);
                x++;
            }
            string fOldAccessFile;
            string fNewAccessFile;
            fOldAccessFile = Gl.sPart1AccessMDBString + txtAccessFilePath.Text +
              Gl.sPart2AccessString;
            fNewAccessFile = Gl.sPart1AccessMDBString + sTempAccessFilePath +
              Gl.sPart2AccessString;
            JetEngine engine = new JetEngine();
            engine.CompactDatabase(fOldAccessFile, fNewAccessFile);
            File.Delete(txtAccessFilePath.Text);
            File.Move(sTempAccessFilePath, txtAccessFilePath.Text);
            MessageBox.Show("Database compact and repaired successfully!");
        } // CmdCompactRepair_Click

        private void CmdExit_Click(object sender, EventArgs e)
```

```
        {
            Gl.WordApp.Quit();
            Close();
        } // CmdExit_Click

        private void CmdImport_Click(object sender, EventArgs e)
        {  // THE MAIN PROCEDURE OF THE WHOLE IMPORT
            Gl.cn.Open(Gl.sConn); // open Access DB
            if (radWord.Checked)
            {
                Gl.sWordPath = txtWordFilePath.Text;
                Gl.WordDoc = Gl.WordApp.Documents.Add(txtWordFilePath.Text);
                Gl.iCurrWordFileNum = 1;
                Gl.iNumWordFiles = 1;
                ProcessOneInputDocument();
            }
            else
                ProcessAllInputDocumentsInTextDocument();
            Gl.cn.Close(); // close Access DB
            lblProcessing.Text = "";
            MessageBox.Show("Done.");
        } // CmdImport_Click

        private void CmdWordBrowse_Click(object sender, EventArgs e)
        {
            OpenFileDialog dlgAccess = new OpenFileDialog();
            dlgAccess.InitialDirectory = "c:\\";
            if (radWord.Checked)
                dlgAccess.Filter = "Word files (*.doc, *.docx)|*.doc;*.docx";
            else
                dlgAccess.Filter = "Text files (*.txt)|*.txt";
            dlgAccess.FilterIndex = 0;
            dlgAccess.RestoreDirectory = true;
            if (dlgAccess.ShowDialog() == DialogResult.OK)
            {
                txtWordFilePath.Text = dlgAccess.FileName;
            }
        } // CmdWordBrowse_Click

        private string DefaultBookID()
        {
            int iID;
            ADODB.Recordset rs = new ADODB.Recordset();
            rs.Open("SELECT TOP 1 DefaultBookID FROM Misc", Gl.cn,
                ADODB.CursorTypeEnum.adOpenDynamic,
                ADODB.LockTypeEnum.adLockOptimistic);
            if (rs.BOF && rs.EOF)
                iID = 0;
            else
            {
                rs.MoveFirst();
                iID = rs.Fields["DefaultBookID"].Value;
```

```
            }
            rs.Close();
            return iID.ToString();
        } // DefaultBookID

        private void ExtractKey()
        {
            Gl.sKey = "";
            while (Gl.WordDoc.Characters[Gl.x].Text != ")")
            {
                Gl.sKey += Gl.WordDoc.Characters[Gl.x].Text;
                Gl.x++;
            }
            // if the key has a dot on the end of the name (ex: "E min."), remove
            if (Gl.sKey.Substring(Gl.sKey.Length - 1, 1) == ".")
                Gl.sKey = Gl.sKey.Substring(0, Gl.sKey.Length-1);
            // see if it's really a key, and not part of the song title
            switch (Gl.sKey)
            {
                case "C":
                case "C min":
                case "C#":
                case "C# min/E":
                case "C# min":
                case "Db":
                case "Db min":
                case "D":
                case "D min":
                case "D#":
                case "D# min":
                case "Eb":
                case "Eb min":
                case "E":
                case "E min":
                case "E/F#":
                case "F":
                case "F/G":
                case "F min":
                case "F#":
                case "F# min":
                case "F# min./A":
                case "Gb":
                case "Gb min":
                case "G":
                case "G min":
                case "G#":
                case "G# min":
                case "Ab":
                case "Ab min":
                case "A":
                case "A min":
                case "A/B":
```

```
                case "A#":
                case "A# min":
                case "Bb":
                case "Bb min":
                case "B":
                case "B min":
                    Gl.bIsKey = true;
                    break;
                default:
                    Gl.bIsKey = false;
                    // add "non-key" to end of song title
                    Gl.sSongTitle += " (" + Gl.sKey + ")";
                    break;
            }
        } // ExtractKey

        private void ExtractPageNumber()
        {
            Gl.sPageNo = "";
            while (Gl.WordDoc.Characters[Gl.x].Text != ")")
            {
                Gl.sPageNo += Gl.WordDoc.Characters[Gl.x].Text;
                Gl.x++;
            }
        } // ExtractPageNumber

        private int GetSongID()
        {
            int iID;
            Gl.rs.Open("SELECT ID FROM Songs" +
                " WHERE SongTitle = " + Literal(Gl.sSongTitle) +
                " OR AltSongTitle1 = " + Literal(Gl.sSongTitle) +
                " OR AltSongTitle2 = " + Literal(Gl.sSongTitle), Gl.cn,
                ADODB.CursorTypeEnum.adOpenDynamic,
                ADODB.LockTypeEnum.adLockOptimistic);
            if (!Gl.rs.BOF || !Gl.rs.EOF) // found the song, get its ID
            {
                Gl.rs.MoveFirst();
                iID = Gl.rs.Fields["ID"].Value;
            }
            else // song not there, ID set to 0
                iID = 0;
            Gl.rs.Close();
            return iID;
        } // GetSongID

        private Boolean IsNumeric(string psChar)
        {
            char cChar;
            cChar = psChar[0];
            if (cChar >= '0' && cChar <= '9')
                return true;
```

```
            else
                return false;
        } // IsNumeric

        private string Literal(string psInput)
        {
            // This function receives a string, and outputs
            // the same string, except it has single quotes
            // added to both ends, and also doubles any
            // single quotes in the text, to make it clear
            // they are single quote characters, and not
            // delimiters. Example: "can't" becomes "'can''t'".
            // It also does the same thing to "irregular" apostrophes
            // from Word, converting them to "normal" apostrophes.
            int x;
            string sOutput = "'";
            for (x = 0; x <= psInput.Length - 1; x++)
                switch(psInput.Substring(x, 1))
                {
                    case "'": // "normal" apostrophe
                    case "’": // "irregular" apostrophe
                        sOutput += "''";
                        break;
                    default:
                        sOutput += psInput.Substring(x, 1);
                        break;
                }
            sOutput += "'";
            return sOutput;
        } // Literal

        private int NextAvailableIDNumber(string psTableName)
        {
            // DO NOT USE THIS WITH TABLE "Users"!!
            // This function receives a table name, and returns the next
            // available ID number which should be used. I don't like the way the
            // default Access AutoNumber process leaves holes in the numbering
            // if something is removed; this will fill them in.
            ADODB.Recordset rs = new ADODB.Recordset();
            int iNextIDNumber;
            Boolean bAvailableNumberFound;
            rs.Open("SELECT ID FROM " + psTableName, Gl.cn,
                ADODB.CursorTypeEnum.adOpenDynamic,
                ADODB.LockTypeEnum.adLockOptimistic);
            iNextIDNumber = 1;
            // if no records, it will stay at 1, else code below
            if(!rs.BOF || !rs.EOF) // find first hole, if any, or tack to end
            {
                bAvailableNumberFound = false;
                rs.MoveFirst();
                while(!rs.EOF && !bAvailableNumberFound)
                {
```

```
                if (rs.Fields["ID"].Value > iNextIDNumber) // found a hole
                    bAvailableNumberFound = true;
                else
                {
                    iNextIDNumber++;
                    rs.MoveNext();
                }
            }
        }
        rs.Close();
        return iNextIDNumber;
    } // NextAvailableIDNumber

    private void NukeRecordsForWorshipDate()
    {
        // Apparently the user has chosen to replace existing records for
        // the worship date with ones generated by the new input document.
        Gl.cn.Execute("DELETE FROM WorshipActivityInstances" +
            " WHERE ActivityDate = #" + Gl.sWorshipDate + "#",
            out Gl.dummy, 0);
    } // NukeRecordsForWorshipDate

    private void ProcessAllInputDocumentsInTextDocument()
    {
        // If this is being run, the user has chosen to read
        // a text document listing Word documents (each line has
        // the path to one Word document). This loops through the
        // text document, and processes each Word deocument whose
        // path is in it.
        int x;
        string[] asTextLines = System.IO.File.ReadAllLines(Gl.sTextPath);
        Gl.iNumWordFiles = asTextLines.Length;
        for(x=0;x<=asTextLines.Length-1;x++)
        {
            Gl.iCurrWordFileNum = x + 1;
            Gl.WordDoc = Gl.WordApp.Documents.Add(asTextLines[x]);
            Gl.sWordPath = asTextLines[x];
            ProcessOneInputDocument();
        }
    } // ProcessAllInputDocumentsInTextDocument

    private void ProcessOneInputDocument()
    {
        Gl.iLineNo = 0;
        UpdateLabel();
        if (!ValidFirstLineDate())
            return;
        while (Gl.x <= Gl.WordDoc.Characters.Count) // not end of document
        {
            ProcessOneLine();
            SkipToBeginningOfNextLine();
        }
```

```
        } // ProcessOneInputDocument

        private void ProcessOneLine()
        {
            // This will process one line of the input file. It
            // looks to see if the first character of the line is
            // bold or not. If not, it's a song line; if so, it's
            // something else.
            Gl.iOrder++;
            if(Gl.WordDoc.Characters[Gl.x].Bold == -1) // bold
                ProcessOneNonSongActivity();
            else // 0, not bold
                ProcessOneSong();
        } // ProcessOneLine

        private void ProcessOneNonSongActivity()
        {
            string sActivityType;
            string sActivityLeaderName;
            int iActivityID;
            // line begins with *, last line, we don't do anything
            // with it, skip
            if (Gl.WordDoc.Characters[Gl.x].Text == "*")
                return;
            sActivityType = "";
            while(Gl.WordDoc.Characters[Gl.x].Text != "-"
            && Gl.WordDoc.Characters[Gl.x].Text != "\t")
            {
                sActivityType += Gl.WordDoc.Characters[Gl.x].Text;
                Gl.x++;
            }
            sActivityLeaderName = "";
            // we're on tab or dash; if dash, get activity leader
            // otherwise, there isn't one listed
            if (Gl.WordDoc.Characters[Gl.x].Text == "-")
            {
                Gl.x++;
                while (Gl.WordDoc.Characters[Gl.x].Text != "\t")
                {
                    sActivityLeaderName += Gl.WordDoc.Characters[Gl.x].Text;
                    Gl.x++;
                }
            } // if dash, get activity leader
            // adjust for "backwards compatibility" for some older terms we don't use ↵
              in the list now
            switch(sActivityType)
            {
                case ("Shepherd’s Prayer"): //apostrophe not standard, but from Word
                case ("shepherd’s prayer"): //apostrophe not standard, but from Word
                case ("Shepherd's Prayer"): //normal apostrophe
                case ("shepherd's prayer"): //normal apostrophe
                    sActivityType = "opening prayer";
```

```
                break;
            case "Song leader":
            case "song leader":
                sActivityType = "worship leader";
                break;
            case "Sermon":
                sActivityType = "sermon";
                break;
            case "kids leave on next song": // don't do anything with this, move on
                return;
        }
        if (sActivityType.Length >= 6
        && (sActivityType.Substring(0, 6) == "sermon"
        || sActivityType.Substring(0, 6) == "Sermon")) // "sermon" plus other stuff
            sActivityType = "sermon"; // just "sermon"
        if (sActivityType != "worship leader")
        {
            Gl.rs.Open("SELECT ID FROM WorshipActivities" +
                " WHERE Activity = " + Literal(sActivityType),
                Gl.cn, ADODB.CursorTypeEnum.adOpenDynamic,
                ADODB.LockTypeEnum.adLockOptimistic);
            if(!Gl.rs.BOF || !Gl.rs.EOF) // found a record
            {
                iActivityID = Gl.rs.Fields["ID"].Value;
                Gl.cn.Execute("INSERT INTO WorshipActivityInstances VALUES (" +
                    NextAvailableIDNumber("WorshipActivityInstances") + ", "+
                  iActivityID + ", NULL, " +
                    ActivityLeaderID(sActivityLeaderName) + ", " + Gl.iOrder + ",
                  #" + Gl.sWorshipDate + "#," +
                    " '" + Gl.sWordFileName + "', NULL, #" +
                    DateTime.Today.ToShortDateString() + "#)", out Gl.dummy, 0);
            }
            else // no record for that worship activity
            {
                MessageBox.Show("Worship activity \"" + sActivityType + "\" not
                  found;" +
                    " will not be added. Maybe misspelled or program needs to be
                  updated.");
            }
            Gl.rs.Close();
        } // if
        else // worship leader
        {
            // find worship leader's ID and update ALL song records
            // (already added) with it for worship date
            Gl.cn.Execute("UPDATE WorshipActivityInstances SET PersonID = " +
                ActivityLeaderID(sActivityLeaderName) +
                " WHERE ActivityDate = #" + Gl.sWorshipDate +
                "# AND ActivityID = 1", // worship date, songs
                out Gl.dummy, 0);
```

```
            }
        } // ProcessOneNonSongActivity

        private void ProcessOneSong()
        {
            Gl.sSongTitle = "";
            Gl.sKey = "";
            Gl.sPageNo = "";
            // if 1st char "(", not really a song,
            // or EOF, skip
            if (Gl.WordDoc.Characters[Gl.x].Text == "("
            || Gl.x >= Gl.WordDoc.Characters.Count)
                return;
            // get song title
            while(Gl.WordDoc.Characters[Gl.x].Text != "("
            && Gl.WordDoc.Characters[Gl.x].Text != "\t")
            {
                Gl.sSongTitle += Gl.WordDoc.Characters[Gl.x].Text;
                Gl.x++;
            }
            Gl.sSongTitle = Gl.sSongTitle.Trim();
            // Now we're pointing to either a left paren or tab. If
            // tab, no key or page number. If left paren, it could be
            // a page number or key, and if page number, there could be a key
            // after that, both inside parentheses. Let's check for those...
            if (Gl.WordDoc.Characters[Gl.x].Text == "(")
            {
                Gl.x++; // at 1st char of key or page #
                // loop through any keys or page #s before hitting tab, or EOF
                while (Gl.WordDoc.Characters[Gl.x].Text != "\t")
                {
                    if (IsNumeric(Gl.WordDoc.Characters[Gl.x].Text)) // page #
                        ExtractPageNumber();
                    else // probably key, could be part of song title after left paren
                        ExtractKey();
                    // now pointing to right paren; move until we find char
                    // after left paren (beginning of another page # or key)
                    // or tab (done with page # and key)
                    //while (Gl.WordDoc.Characters[Gl.x].Text != "("
                    //&& Gl.WordDoc.Characters[Gl.x].Text != "\t")
                    while (Gl.WordDoc.Characters[Gl.x].Text != "("
                    && Gl.WordDoc.Characters[Gl.x].Text != "\t")
                            Gl.x++;
                    if (Gl.WordDoc.Characters[Gl.x].Text == "(")
                        Gl.x++;
                } // done extracting any page # &/or key
            }  // done extracting any page # &/or key IF any
            // Add song in Songs table if needed,
            // add record to WorshipActivityInstances
            AddSong();
        } // ProcessOneSong
```

```
        private void radWord_CheckedChanged(object sender, EventArgs e)
        {
            if (radWord.Checked) // Word document
            {
                lblWordOrText.Text = "Select the Word file (not needed for Compact && ↵
                  Repair):";
            }
            else // text document with list of Word documents
            {
                lblWordOrText.Text = "Select the text file (not needed for Compact && ↵
                  Repair):";
            }
            txtWordFilePath.Text = "";
        } // radWord_CheckedChanged

        private Boolean RecordsExistForWorshipDate()
        {
            // See if records exist for this worship date. If so,
            // user will be prompted to see if they want to nuke these
            // records and replace them. If not, the input document
            // will not be imported.
            ADODB.Recordset rs = new ADODB.Recordset();
            Boolean bRecordsExist;
            rs.Open("SELECT COUNT(*) AS RecCount FROM WorshipActivityInstances" +
                " WHERE ActivityDate = #" + Gl.sWorshipDate + "#", Gl.cn,
                ADODB.CursorTypeEnum.adOpenDynamic,
                ADODB.LockTypeEnum.adLockOptimistic);
            rs.MoveFirst();
            bRecordsExist = rs.Fields["RecCount"].Value > 0;
            rs.Close();
            return bRecordsExist;
        } // RecordsExistForWorshipDate

        private void SkipToBeginningOfNextLine()
        {
            while (Gl.x <= Gl.WordDoc.Characters.Count
            && Gl.WordDoc.Characters[Gl.x].Text != "\r")
                Gl.x++;
            Gl.x++;
            UpdateLabel();
        } // SkipToBeginningOfNextLine

        private void TxtAccessFilePath_TextChanged(object sender, EventArgs e)
        {
            // Only enable the Import button if there's something in both
            // txtAccessFilePath and txtWordFilePath. Check when either
            // changes and enable/disable the Run button accordingly.
            cmdImport.Enabled = txtAccessFilePath.Text != ""
                && txtWordFilePath.Text != "";
            // set connection string
            Gl.sConn = Gl.sPart1AccessMDBString + txtAccessFilePath.Text +          ↵
              Gl.sPart2AccessString;
```

```
            // Only enable the Compact & Repair button if there's something in
            // txtAccessFilePath
            cmdCompactRepair.Enabled = txtAccessFilePath.Text != "";
        } // TxtAccessFilePath_TextChanged

        private void TxtWordFilePath_TextChanged(object sender, EventArgs e)
        {
            // Only enable the Import button if there's something in both
            // txtAccessFilePath and txtWordFilePath. Check when either
            // changes and enable/disable the Run button accordingly.
            cmdImport.Enabled = txtAccessFilePath.Text != ""
                && txtWordFilePath.Text != "";
            if(radText.Checked)
                Gl.sTextPath = txtWordFilePath.Text;
        } // TxtWordFilePath_TextChanged

        private void UpdateLabel()
        {
            Gl.iLineNo++;
            lblProcessing.Text = "Processing \"" + Gl.sWordFileName +
                "\", file " + Gl.iCurrWordFileNum + " of " + Gl.iNumWordFiles +
                ", line " + Gl.iLineNo + "...";
        } // UpdateLabel

        private Boolean ValidFirstLineDate()
        { // All we want out of this line is the date, and to compare it
          // with the date in the file name. We'll make sure that the
          // date in the first line matches the date in the file name.
          // If not, they have to fix it first. Also, if there are already
          // records for the worship date, ask if the user wants to replace
          // them with information from this document. if not, do not move
          // on.
            string sWorship3LetterMonth = "";
            string sDayOfMonth = "";
            string sDateFromPath = "";
            int y;
            Gl.sWorshipDate = "";
            Gl.x = 1;
            Gl.iOrder = 0;
            // skip to beginning of day of month
            while(!IsNumeric(Gl.WordDoc.Characters[Gl.x].Text))
                Gl.x++;
            // write each digit of day of month in variable
            while(IsNumeric(Gl.WordDoc.Characters[Gl.x].Text))
            {
                sDayOfMonth += Gl.WordDoc.Characters[Gl.x].Text;
                Gl.x++;
            }
            // skip to beginning of month name
            Gl.x++;
            // write each letter of month name in variable
            while(Gl.WordDoc.Characters[Gl.x].Text != " ")
```

```
            {
                sWorship3LetterMonth += Gl.WordDoc.Characters[Gl.x].Text;
                Gl.x++;
            }
            // skip to beginning of year
            Gl.x++;
            //write month in date variable
            switch(sWorship3LetterMonth)
            {
                case "Jan":
                    Gl.sWorshipDate = "1";
                    break;
                case "Feb":
                    Gl.sWorshipDate = "2";
                    break;
                case "Mar":
                    Gl.sWorshipDate = "3";
                    break;
                case "Apr":
                    Gl.sWorshipDate = "4";
                    break;
                case "May":
                    Gl.sWorshipDate = "5";
                    break;
                case "Jun":
                    Gl.sWorshipDate = "6";
                    break;
                case "Jul":
                    Gl.sWorshipDate = "7";
                    break;
                case "Aug":
                    Gl.sWorshipDate = "8";
                    break;
                case "Sep":
                    Gl.sWorshipDate = "9";
                    break;
                case "Oct":
                    Gl.sWorshipDate = "10";
                    break;
                case "Nov":
                    Gl.sWorshipDate = "11";
                    break;
                case "Dec":
                    Gl.sWorshipDate = "12";
                    break;
            } // switch
            Gl.sWorshipDate += "/" + sDayOfMonth + "/";
            // write year in date variable
            while (IsNumeric(Gl.WordDoc.Characters[Gl.x].Text))
            {
                Gl.sWorshipDate += Gl.WordDoc.Characters[Gl.x].Text;
                Gl.x++;
```

```
            }
            // compare date on 1st line of file to date in file name
            //sDateFromPath
            y = Gl.sWordPath.Length - 1;
            while (Gl.sWordPath[y] != '-')
                y--;
            y -= 2; // pointing to 1st month digit
            if(Gl.sWordPath[y] == '0') // skip leading 0 if there
                y++;
            // write month
            while (Gl.sWordPath[y] != '-')
            {
                sDateFromPath += Gl.sWordPath[y].ToString();
                y++;
            }
            y++;
            sDateFromPath += "/";
            if (Gl.sWordPath[y] == '0') // skip leading 0 if there
                y++;
            // write day
            while (Gl.sWordPath[y] != '.')
            {
                sDateFromPath += Gl.sWordPath[y].ToString();
                y++;
            }
            sDateFromPath += "/20";
            while(Gl.sWordPath[y] != '\\')
                y--;
            y++;
            sDateFromPath += Gl.sWordPath[y].ToString(); // year 10's digit
            y++;
            sDateFromPath += Gl.sWordPath[y].ToString(); // year 1's digit
            // extract file name from file path
            Gl.sWordFileName = "";
            y = Gl.sWordPath.Length - 1;
            while(Gl.sWordPath[y] != '\\')
                y--;
            y++;
            while(y <= Gl.sWordPath.Length - 1)
            {
                Gl.sWordFileName += Gl.sWordPath[y];
                y++;
            }
            SkipToBeginningOfNextLine();
            if(Gl.sWorshipDate != sDateFromPath)
            {
                MessageBox.Show("Worship dates on document (" + Gl.sWorshipDate +
                    ") and file name (" + sDateFromPath + ") are " +
                    "different; please fix.","Different Dates");
                this.Close();
            }
            if(RecordsExistForWorshipDate())
```

```
                if (MessageBox.Show("Records already exist for worship date " +
                Gl.sWorshipDate + ". Replace with information from this document?",
                "Existing Records For Date", MessageBoxButtons.YesNo) ==
                  DialogResult.Yes)
                    NukeRecordsForWorshipDate();
                else
                    return false;
            return true;
        } // ValidFirstLineDate
    } // class
} // namespace
```