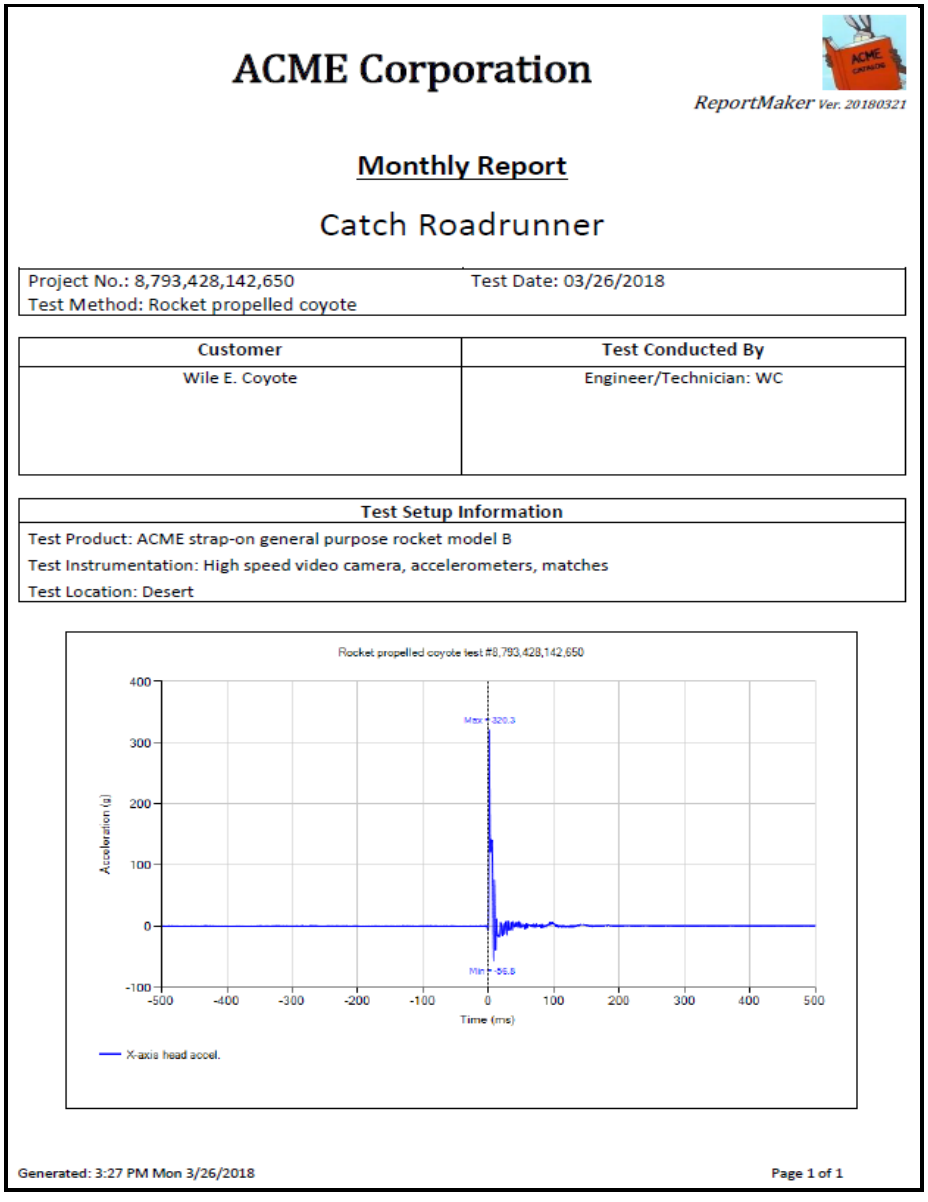
Using Template Bookmarks for Automating Microsoft Word Reports

Darryl Bryk
U.S. Army RDECOM-TARDEC
Warren, MI 48397



ACME Corporation

ReportMaker Ver. 20180321

Monthly Report

Catch Roadrunner

Project No.: 8,793,428,142,650 Test Date: 03/26/2018
Test Method: Rocket propelled coyote

Customer	Test Conducted By
Wile E. Coyote	Engineer/Technician: WC

Test Setup Information
Test Product: ACME strap-on general purpose rocket model B
Test Instrumentation: High speed video camera, accelerometers, matches
Test Location: Desert

Generated: 3:27 PM Mon 3/26/2018 Page 1 of 1

Figure 1 - Example of an automated report built from a template

Introduction

A C# method is discussed for using bookmarks in Microsoft Word templates to automate reports. Automating reports using templates can enable the user or client to include custom designs and information. For example, the company's logo design can be put into the template header. Each client can have their own template report style. However, some information may need to be placed in specific locations in the report, or there may be user input to insert, that is only known just prior to the report being generated, and this is where bookmarks can be used.

In the above example report, bookmarks were placed into the template for the test date, subtitle, project no., test method, customer, engineer/technician, and the information in the Test Setup table. A template can be designed in Microsoft Word and bookmarks inserted at the desired locations. The report maker application can then read the bookmarks from the template and insert the text at the bookmark location. Some known information, like the current date can be inserted automatically, or for specific user information, a textbox dialog can prompt the user for the text to be inserted for each bookmark field. When a bookmark is defined in Word it must be given a name, and these names can be used as textbox labels to identify each bookmark.

A bookmark may be inserted into a Microsoft Word 2013 template as follows:

- Position the cursor to the desired bookmark position in the document
- Select the ribbon "INSERT" tab
- From the "Links" dropdown select "Bookmark"
- Type the name of the bookmark (note: spaces in the name are not allowed)
- Click Add

(To make bookmarks visible: go to File > Options > Advanced, then under "Show document content" check "Show bookmarks").

Using the Code

The code below reads the bookmarks from a Word template document and returns them in top-down order in a List<string>.

```
using System.Linq; // For LINQ .OrderBy
using Microsoft.Office.Interop.Word; // For _Document, _Application, Bookmark

//---------------------------------------------------------------------------
// Returns ordered list of bookmarks found in doc.
//---------------------------------------------------------------------------
public static List<string> GetBookmarks(string fname) {
    _Application WDapp = new Microsoft.Office.Interop.Word.Application();
    _Document doc; // _ removes compiler ambiguity warning on Close()

    List<Bookmark> bmarks = new List<Bookmark>();

    try {
        doc = WDapp.Documents.Add(fname);

        // Get list as they come from Word (alphabetically)
        foreach (Bookmark b in doc.Bookmarks) {
            bmarks.Add(b);
        }
    }
    catch (Exception err) {
```

```
            MessageBox.Show("Error: " + err.Message, "GetBookmarks()", MessageBoxButtons.OK,
                MessageBoxIcon.Error);
            return null;
        }

        // Re-sort list in order of appearance
        bmarks = bmarks.OrderBy(b => b.Start).ToList(); // LINQ

        List<string> slBMarks = new List<string>();
        foreach (Bookmark b in bmarks) {
            slBMarks.Add(b.Name); // Accumulate bookmark names
        }

        try { // Crashes if already closed
            doc.Close(WdSaveOptions.wdDoNotSaveChanges);
            WDapp.Quit(WdSaveOptions.wdDoNotSaveChanges);
        }
        catch { }

        return slBMarks;
    }
```

The template file name is passed in the variable fname. The bookmarks are read from the Word template and added to a List<Bookmark> object. Since the bookmarks are returned from Word in alphabetical order, the LINQ statement re-sorts the bookmarks in reading order (i.e. top-down as found in the template). The method returns the ordered bookmark names in a List<string> object. Once the bookmark names are known, the class method InsertBookmarkText (shown below) can be called to insert the text at the bookmark location.

```
    //---------------------------------------------------------------------------
    // Inserts text at bookmark, returns true if didn't fail
    //---------------------------------------------------------------------------
    private bool InsertBookmarkText(string bookmarkName, string text) {
        try {
            if (_doc.Bookmarks.Exists(bookmarkName)) {
                _doc.Bookmarks.get_Item(bookmarkName).Range.Text = text;
                return true;
            }
        }
        catch (Exception ex) {
            #if DEBUG
                MessageBox.Show(ex.Message, "InsertBookmarkText()", MessageBoxButtons.OK,
                    MessageBoxIcon.Error);
            #endif
        }

        return false; // Error return
    }
```

The method takes the bookmark name and text to be inserted at that bookmark position as input. The name of the bookmark is used by the method to locate the bookmark in the document. Note that _doc used here is a _Document object as in GetBookmarks above, and has already been initialized before the call to this method by the class constructor. The exception “catch” here only displays a message in Debug mode, because not finding the bookmark does not normally warrant pausing the application to show the error. For example, the code could use this method to search for several different bookmarks, that are used in some templates but not others, and only insert text for the ones found.

For information that must be prompted for, a dialog can be generated to prompt the user for the text to be inserted at each bookmark.

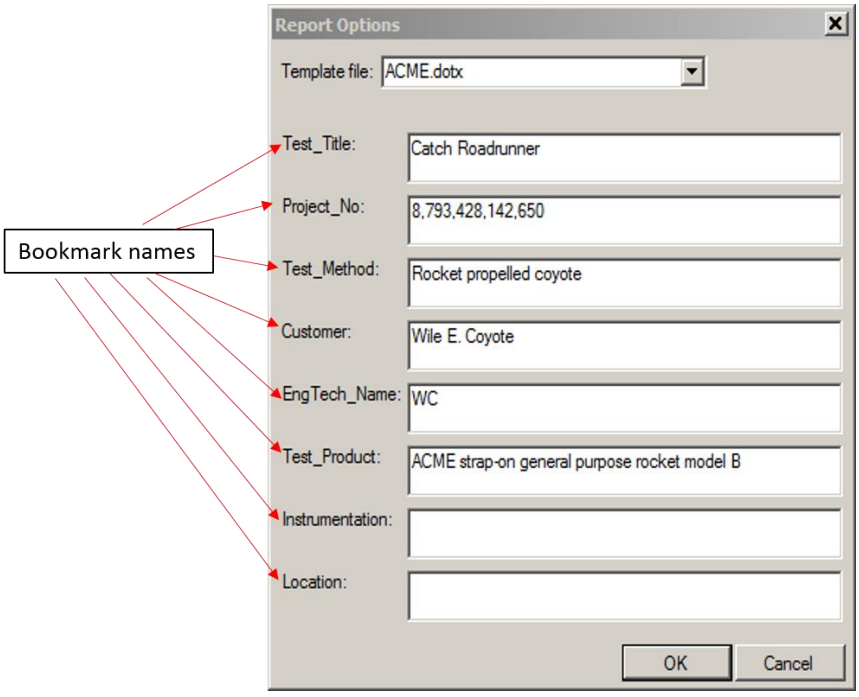


Figure 2 - A dynamic textbox dialog that updates itself based on the selected template file

The dialog above was designed with a combo box to select the template file to be used for the report. Since each template may contain different bookmarks, the dialog was made to automatically insert textboxes for only the bookmarks that are found. When the template file is chosen from the combo box, the textboxes are updated to reflect just the ones found in that template. It uses the bookmark names for the textbox labels so the user can identify which bookmark item it refers to. A "Panel" control is utilized as the container for all the textboxes so that textbox updates can be done easily by simply using the panel control's Clear method. The panel can also be anchored (properties: Top, Bottom, Left) to the form for automatic resizing.

The Dialog method below (see the CReportDlg class) sets up the dialog. The private member variables _path and _templatePath are saved for use by other class methods. The FilesInDir method (see CReportDlg class) returns a List<string> of files found matching the file mask in the passed folder, which in this case are template files, and these file names are then added to the combo box drop down.

```
using System.IO; // For File IO

//---------------------------------------------------------------------------
// Init. dialog
//---------------------------------------------------------------------------
public List<string> Dialog() {
    System.Windows.Forms.Cursor.Current = Cursors.WaitCursor;

    _path = System.Windows.Forms.Application.StartupPath + "\\";
    _templatePath = Path.Combine(_path, _templatePath); // Append def. filename
    string fname = Path.GetFileName(_templatePath);

    // Get template files for combo box
```

```
        List<string> tfiles = FilesInDir(_path, new string[] { "*.dotx" });
        if (tfiles != null) {
            foreach (string f in tfiles) {
                this.cbxTemplate.Items.Add(f);
            }

            if (fname != "")
                cbxTemplate.Text = fname; // Last one used
            else if (cbxTemplate.Items.Count > 0)
                cbxTemplate.Text = cbxTemplate.Items[0].ToString(); // Def. to 1st one
        }

        System.Windows.Forms.Cursor.Current = Cursors.Default;
        ShowDialog();

        return _lResults;
    }
```

The code below is the combo box event handler for when the user changes to a different template file. As mentioned above, a panel is used as the container for the textboxes (called pnlBookmarks) so that all the previous textboxes can be easily disposed of (by calling Clear), and also the panel can then be docked to the form for automatic resizing when the textboxes are changed. The bookmarks are read from the new template and a textbox with label is added for each bookmark in the foreach loop. The call to CMsgDlg.AddTextBox adds the textboxes to the panel. The panel control is passed to the CMsgDlg() constructor so that the AddTextBox method can access the control to be used for putting the textboxes onto. For more information, and to download the CMsgDlg class, see the Codeproject article: www.codeproject.com/Articles/1239528/A-Multipurpose-Message-Dialog.

The foreach loop labeled with the comment: “Check for prev. bookmark defaults” checks for previously entered textbox entries to provide defaults to the user for ease of use. This check is made by looking at the textbox tags which are used to hold the bookmark names. The AddTextBox method stores the passed textbox label, which in this case, is the bookmark name, in the textbox tag for this purpose of identification. If the new bookmark name matches the textbox tag, then the previously saved textbox string is used for the default input. In the CReportDlg.btnOK_Click function, the bookmark name and the text entered by the user is appended together in one string (to be returned to the caller), separated by a ‘|’ character for parsing later. This is why the Split(‘|’) is done to parse out the bookmark name. The textbox positions and maximum width is checked so they can be aligned, and then finally the dialog’s form is sized for proper fitting of the textboxes.

```
    //---------------------------------------------------------------------------
    // cbxTemplate change handler sets up textboxes for bookmarks.
    // A panel is used for bookmark controls for easier updating.
    //---------------------------------------------------------------------------
    private void cbxTemplate_SelectedIndexChanged(object sender, EventArgs e) {
        pnlBookmarks.Controls.Clear(); // Remove all prev.
        this.Size = new Size(_startformWidth, _startformHeight);

        // Add new bookmarks textboxes
        List<string> bms = CReport.GetBookmarks(Path.Combine(_path, this.cbxTemplate.Text));
        if (bms != null && bms.Count > 0) {
            string strPad = string.Format("{0:-30}\n ", " "); // Default empty string
            int maxlblRight = 0;
            foreach (string bookmarkName in bms) {
                string strDef = strPad;
```

```
                // Check for prev. bookmark defaults
                foreach (string bm_text in _lbmks) {
                    string[] str = bm_text.Split('|');
                    if (str[0] == bookmarkName) {
                        if (str[1] != "") strDef = str[1];
                        break;
                    }
                }

                Control tbx = new CMsgDlg(pnlBookmarks).AddTextBox(bookmarkName + ":",
                     strDef);
                tbx.Text = tbx.Text.Trim(new char[] { ' ', '\n', '\r' }); // Trim \n's
                tbx.Name = bookmarkName; // Name for return ID
                Label lbl = (Label)tbx.Tag; // CMsgDlg stores tbx label object in Tag
                lbl.Left = 0; // Align labels
                if (maxlblRight < lbl.Right) maxlblRight = lbl.Right; // Save max.
                tbx.Left = lbl.Right;
                tbx.Width = this.Width - _margin * 2; // Make to form width
            }
            this.Width = pnlBookmarks.Width + pnlBookmarks.Margin.Right + _margin * 2;

            // Re-align all to max lbl.Right
            foreach (Control ctl in pnlBookmarks.Controls) {
                if (ctl is TextBox) ctl.Left = maxlblRight;
            }
        }
        this.Height = _startformHeight + pnlBookmarks.Height;
    }
```

The CMsgDlg.AddTextBox method is called in a loop to add the appropriate number of textboxes and their labels as prompts for the bookmark information to be inserted into the report. Its method is declared as:

```
public TextBox AddTextBox(string lblText, string tbxText) {…
```

The first passed parameter, lblText, will become the label for the textbox. Here the bookmark name followed by a colon (":") is used. The second parameter, tbxText, is passed to the method to use as a default for the textbox. Its also used to calculate the width for the textbox, and number of lines it should have (by number of embedded '\n'). In the usage above, the default textbox string strPad, is padded to 30 characters to provide a wider textbox for input, and the '\n' is added to make the textbox have two lines. For more information, and to download the CMsgDlg class, see the Codeproject article: www.codeproject.com/Articles/1239528/A-Multipurpose-Message-Dialog.

After the user has entered the desired text for each bookmark and closed the dialog, the InsertBookmarkText method (discussed above) is used to insert the text strings. This method is called repeatedly in a loop to insert each text string at its proper bookmark position (see the CReport method for more details).

Conclusion

It can be seen that using bookmarks with templates for automating reports can be a useful option for the developer. The methods discussed here will hopefully offer a good starting point for utilizing these features.

Appendix A

The CReportDlg class

```
using System;
using System.Collections.Generic;
using System.ComponentModel;
using System.Data;
using System.Drawing;
using System.Linq;
using System.Text;
using System.Windows.Forms;
using System.Diagnostics;   // For debug
using System.IO;            // For File IO

namespace MyApp {
//-----------------------------------------------------------------------
// Dialog class handles Report input parameters, bookmark inputs from
// template files, and maintains defaults.
//-----------------------------------------------------------------------
public partial class CReportDlg : Form {
    private List<string> _lResults = new List<string>(); // List return
    private string _path = "";
    private int _startformHeight; // Starting form size
    private int _startformWidth;

    // Static to save as defaults
    private static string _templatePath = "";
    private static List<string> _lbmks = new List<string>(); // Bookmarks

    private const int _margin = 5;

    //-----------------------------------------------------------------------
    // Constructor
    //-----------------------------------------------------------------------
    public CReportDlg() {
        InitializeComponent();
        _startformHeight = this.Height; // Save orig. size for
               cbxTemplate_SelectedIndexChanged()
        _startformWidth = this.Width;
    }

    //-----------------------------------------------------------------------
    // Init. dialog
    //-----------------------------------------------------------------------
    public List<string> Dialog() {
        System.Windows.Forms.Cursor.Current = Cursors.WaitCursor;

        _path = System.Windows.Forms.Application.StartupPath + "\\";
        _templatePath = Path.Combine(_path, _templatePath); // Append def. filename
        string fname = Path.GetFileName(_templatePath);

        // Get template files for cbx
        List<string> tfiles = FilesInDir(_path, new string[] { "*.dotx" });
        if (tfiles != null) {
            foreach (string f in tfiles) {
```

```
                this.cbxTemplate.Items.Add(f);
            }

            // Changing cbxTemplate.Text causes call to cbxTemplate_SelectedIndexChanged()
            if (fname != "")
                cbxTemplate.Text = fname; // Last one
            else if (cbxTemplate.Items.Count > 0)
                cbxTemplate.Text = cbxTemplate.Items[0].ToString(); // Def. to 1st one
        }

        System.Windows.Forms.Cursor.Current = Cursors.Default;
        ShowDialog();

        return _lResults;
    }

    //---------------------------------------------------------------------------
    // cbxTemplate change handler sets up textboxes for bookmarks.
    // A panel is used for bookmark controls for easier updating.
    //---------------------------------------------------------------------------
    private void cbxTemplate_SelectedIndexChanged(object sender, EventArgs e) {
        pnlBookmarks.Controls.Clear(); // Remove all prev.
        this.Size = new Size(_startformWidth, _startformHeight);

        // Add new bookmarks textboxes
        List<string> bms = CReport.GetBookmarks(Path.Combine(_path, this.cbxTemplate.Text));
        if (bms != null && bms.Count > 0) {
            string strPad = string.Format("{0:-30}\n ", " ");
            int maxlblRight = 0;
            foreach (string bookmarkName in bms) {
                string strDef = strPad;

                // Check for prev. bookmark defaults
                foreach (string bm_text in _lbmks) {
                    string[] str = bm_text.Split('|');
                    if (str[0] == bookmarkName) {
                        if (str[1] != "") strDef = str[1];
                        break;
                    }
                }

                Control tbx = new CMsgDlg(pnlBookmarks).AddTextBox(bookmarkName + ":",
                    strDef);
                tbx.Text = tbx.Text.Trim(new char[] { ' ', '\n', '\r' }); // Trim end \n's
                tbx.Name = bookmarkName; // Name for return ID
                Label lbl = (Label)tbx.Tag; // CMsgDlg stores tbx label object in Tag
                lbl.Left = 0; // Align labels
                if (maxlblRight < lbl.Right) maxlblRight = lbl.Right; // Save max.
                tbx.Left = lbl.Right;
                tbx.Width = this.Width - _margin * 2; // Make to form width
            }
            this.Width = pnlBookmarks.Width + pnlBookmarks.Margin.Right + _margin * 2;

            // Re-align all to max lbl.Right
            foreach (Control ctl in pnlBookmarks.Controls) {
                if (ctl is TextBox) ctl.Left = maxlblRight;
            }
```

```
        }
        this.Height = _startformHeight + pnlBookmarks.Height;
    }

    //---------------------------------------------------------------------------
    // btnOK handler; sets _lresult responses for return
    //---------------------------------------------------------------------------
    private void btnOK_Click(object sender, EventArgs e) {
        _lResults = new List<string>();

        // Save defaults
        _templatePath = cbxTemplate.Text;

        if (cbxTemplate.Text != "")
            _lResults.Add(Path.Combine(_path, cbxTemplate.Text));
        else _lResults.Add("");

        // Bookmarks
        if (pnlBookmarks.Controls.Count > 0) { // Don't clear prev. unless new (defaults)
            _lbmks = new List<string>();
            foreach (Control ctl in pnlBookmarks.Controls) {
                if (ctl is TextBox && ctl.Text != "") {
                    string str = ctl.Name + "|" + ctl.Text; // Format for parsing later
                    _lResults.Add(str);
                    _lbmks.Add(str);
                }
            }
        }

        Close();
    }

    //---------------------------------------------------------------------------
    // Cancel button handler
    //---------------------------------------------------------------------------
    private void btnCancel_Click(object sender, EventArgs e) {
        this.Close();
    }

    //---------------------------------------------------------------------------
    // Returns sorted list of files in passed directory matching file masks.
    // Set bSubs = true to incl. sub-dirs.
    //---------------------------------------------------------------------------
    private List<string> FilesInDir(string dir, string[] fmask, bool bSubs = false) {
        List<string> files = new List<string>();

        try {
            DirectoryInfo di = new DirectoryInfo(dir);
            SearchOption searchSubs = (bSubs) ? SearchOption.AllDirectories :
                SearchOption.TopDirectoryOnly;

            foreach (string m in fmask) {
                FileInfo[] fi = di.GetFiles(m, searchSubs);
                foreach (FileInfo f in fi) {
                    if (!f.Attributes.HasFlag(FileAttributes.Hidden)) files.Add(f.Name);
                }
            }
```

```
        }
        catch (System.Threading.ThreadAbortException) { }
        catch (Exception ex) {
            MessageBox.Show(ex.Message, "FilesInDir()", MessageBoxButtons.OK,
                MessageBoxIcon.Error);
        }
        files.Sort(); // Default sort

        return files;
    }
}
}
```

Appendix B

The CReport class

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using System.Windows.Forms;
using Microsoft.Office.Interop.Word;
using System.IO;
using System.Drawing; // For Bitmap
using System.Diagnostics; // For debug

namespace MyApp {
public class CReport {
    private Microsoft.Office.Interop.Word._Application _WDapp; // _ for Quit()
    private _Document _doc; // _ removes compiler ambiguity warning on Close()
    private object _oEndOfDoc = "\\endofdoc";       // Predefined bookmark
    private object _oStartOfDoc = "\\startofdoc";   // Predefined bookmark

    //---------------------------------------------------------------------------
    // Constructor
    //---------------------------------------------------------------------------
    public CReport() {
        List<string> arstr = new CReportDlg().Dialog();
        if (arstr.Count() == 0) return;

        string _templatePathName = arstr[0];

        // Bookmarks text
        List<string> lbmks = new List<string>();
        for (int i = 3; i < arstr.Count; i++) lbmks.Add(arstr[i]);

        System.Windows.Forms.Cursor.Current = Cursors.WaitCursor;

        _WDapp = new Microsoft.Office.Interop.Word.Application();
        try {
            _doc = _WDapp.Documents.Add(_templatePathName);
        }
        catch (Exception err) {
            MessageBox.Show(err.Message, "CReport()", MessageBoxButtons.OK,
                MessageBoxIcon.Error);
            return;
        }

        _doc.ShowGrammaticalErrors = false;
        _doc.ShowSpellingErrors = false;
        _WDapp.Visible = true;

        // Header
        var header =
            _WDapp.Selection.Sections[1].Headers[WdHeaderFooterIndex.wdHeaderFooterPrimary].
            Range;
        if (SelectionFind("<app>", header)) {
```

```
    header.Text = System.Windows.Forms.Application.ProductName;

    if (SelectionFind("<ver>", header)) {
        header.Text = string.Format("Ver. {0}",
           System.Windows.Forms.Application.ProductVersion);
    }
}

// Footer
var footer =
   _WDapp.Selection.Sections[1].Footers[WdHeaderFooterIndex.wdHeaderFooterPrimary];
Range footer_rng = footer.Range;
footer_rng.ParagraphFormat.SpaceAfter = 0; // Points
footer_rng.ParagraphFormat.SpaceBefore = 0;
footer_rng.ParagraphFormat.LineSpacingRule = WdLineSpacing.wdLineSpaceSingle;
footer_rng.Collapse(WdCollapseDirection.wdCollapseEnd);

footer_rng.InsertAfter("Generated: " + DateTime.Now.ToString("h:mm tt ddd
    M/d/yyyy"));
footer_rng.Font.Size = 9;

// Page numbers - Don't know why this way, but works
footer_rng.InsertAfter("\t\tPage ");
footer_rng.Collapse(WdCollapseDirection.wdCollapseEnd);
footer_rng.Fields.Add(footer_rng, WdFieldType.wdFieldPage);
footer_rng = footer.Range;
footer_rng.InsertAfter(" of ");
footer_rng.Collapse(WdCollapseDirection.wdCollapseEnd);
footer_rng.Fields.Add(footer_rng, WdFieldType.wdFieldNumPages);
footer_rng.Fields.Update();
footer_rng.Collapse(WdCollapseDirection.wdCollapseEnd);

// Insert bookmark strings
foreach (string s in lbmks) {
    string[] astr = s.Split('|');
    InsertBookmarkText(astr[0], astr[1]);
}

_WDapp.Visible = true;
_WDapp.ActiveWindow.ScrollIntoView(_doc.Sentences[1]); // Scroll to top

string filename = "ACME";
SaveFileDialog saveFileDialog = new SaveFileDialog(); // Save file dialog
saveFileDialog.Filter = "PDF (*.pdf)|*.pdf|" +
                        "XPS Document (*.xps)|*.xps|" +
                        "Word Document (*.docx)|*.docx";
saveFileDialog.Title = "Save Report";
saveFileDialog.FileName = "Report - " + filename;

if (saveFileDialog.ShowDialog() == DialogResult.OK) {
    string fname = saveFileDialog.FileName;
    if (Path.GetExtension(fname.ToUpper()) == ".PDF") {
        _doc.ExportAsFixedFormat(fname, WdExportFormat.wdExportFormatPDF,
                        false, WdExportOptimizeFor.wdExportOptimizeForPrint);
    }
    else if (Path.GetExtension(fname.ToUpper()) == ".XPS") {
        _doc.ExportAsFixedFormat(fname, WdExportFormat.wdExportFormatXPS,
```

```
                                false, WdExportOptimizeFor.wdExportOptimizeForPrint);
            }
            else _doc.SaveAs(fname); // Word doc.
        }

        try { // Avoids crash if already closed
            _doc.Close(WdSaveOptions.wdDoNotSaveChanges);
            _WDapp.Quit(WdSaveOptions.wdDoNotSaveChanges);
        }
        catch { }
    }

    //---------------------------------------------------------------------
    // Inserts picture at range
    //---------------------------------------------------------------------
    private bool InsertPic(string fname, Range rng) {
        try {
            //Stream s = GetType().Assembly.GetManifestResourceStream(fname);
            Bitmap bmp = new Bitmap(fname);
            Clipboard.SetImage(bmp);
            rng.PasteAndFormat(WdRecoveryType.wdFormatOriginalFormatting);
            bmp.Dispose(); // Release mem.
        }
        catch { return false; }

        return true;
    }

    //---------------------------------------------------------------------
    // Inserts text at bookmark, returns true if didn't fail
    //---------------------------------------------------------------------
    private bool InsertBookmarkText(string bookmarkName, string text) {
        try {
            if (_doc.Bookmarks.Exists(bookmarkName)) {
                _doc.Bookmarks.get_Item(bookmarkName).Range.Text = text;
                return true;
            }
        }
        catch (Exception ex) {
            #if DEBUG
                MessageBox.Show(ex.Message, "InsertBookmarkText()", MessageBoxButtons.OK,
                    MessageBoxIcon.Error);
            #endif
        }

        return false; // Error return
    }

    //---------------------------------------------------------------------
    // Returns true with text selected if found
    //---------------------------------------------------------------------
    private bool SelectionFind(object text, Range rng = null) {
        try {
            _WDapp.Selection.Find.ClearFormatting(); // Clear prev. searches

            if (rng != null) {
```

```
                return rng.Find.Execute(text, Forward: true, Wrap:
                    WdFindWrap.wdFindContinue);
            }

            return _WDapp.Selection.Find.Execute(text, Forward: true, Wrap:
                WdFindWrap.wdFindContinue);
        }
        catch (Exception ex) {
            #if DEBUG
            MessageBox.Show(ex.Message, "SelectionFind()", MessageBoxButtons.OK,
                MessageBoxIcon.Error);
            #endif
        }

        return false;
    }

    //---------------------------------------------------------------------------
    // Returns ordered list of bookmarks found in doc.
    //---------------------------------------------------------------------------
    public static List<string> GetBookmarks(string fname) {
        _Application WDapp = new Microsoft.Office.Interop.Word.Application();
        _Document doc;
        List<Bookmark> bmarks = new List<Bookmark>();

        try {
            doc = WDapp.Documents.Add(fname);

            // Get list as they come from Word (alphabetically)
            foreach (Bookmark b in doc.Bookmarks) {
                bmarks.Add(b);
            }
        }
        catch (Exception err) {
            MessageBox.Show("Error: " + err.Message, "GetBookmarks()", MessageBoxButtons.OK,
                MessageBoxIcon.Error);
            return null;
        }

        // Re-sort list in order of appearance
        bmarks = bmarks.OrderBy(b => b.Start).ToList(); // LINQ

        List<string> slBMarks = new List<string>();
        foreach (Bookmark b in bmarks) {
            slBMarks.Add(b.Name);
        }

        try { // Crashes if already closed
            doc.Close(WdSaveOptions.wdDoNotSaveChanges);
            WDapp.Quit(WdSaveOptions.wdDoNotSaveChanges);
        }
        catch { }

        return slBMarks;
    }
```

```
    //--------------------------------------------------------------------------
    // Sets range format (null for no change)
    //--------------------------------------------------------------------------
    private void setFormat(Range rng, object fontSize = null, object fontBold = null,
                           object fontItalic = null, object fontUnderln = null,
                           object parSpaceBefore = null, object parSpaceAfter = null,
                           object parAlignment = null, object parLineSpacing = null) {
        if (fontSize != null) rng.Font.Size = (int)fontSize;
        if (fontBold != null) rng.Font.Bold = (int)fontBold;
        if (fontItalic != null) rng.Font.Italic = (int)fontItalic;
        if (fontUnderln != null)
            rng.Font.Underline = ((int)fontUnderln == 0) ? 0 : WdUnderline.wdUnderlineSingle;
        if (parSpaceBefore != null) rng.ParagraphFormat.SpaceBefore = (int)parSpaceBefore;
        if (parSpaceAfter != null) rng.ParagraphFormat.SpaceAfter = (int)parSpaceAfter;
        if (parAlignment != null)
            rng.ParagraphFormat.Alignment = (WdParagraphAlignment)parAlignment;
        if (parLineSpacing != null)
            rng.ParagraphFormat.LineSpacingRule = (WdLineSpacing)parLineSpacing;
    }
}
}
```