Mike Lanham

VBA Developer/PM- Marshall Construction (2006-2012)

##### Contents:

##### [VBA Development](#VBADevelopment).

##### [Quote-Takeoff Application](#QuoteTakeoffApplication).

##### [Schedule Board Application](#ScheduleBoardApplication).

##### [Design](#Design).

##### [Application.Open](#ApplicationOpen).

##### [Using Procedures](#UsingProcedures).

##### [Parsing](#Parsing).

##### [Find.Text](#FindText).

##### [Retrieving Data](#RetrievingData).

##### [Data Servers](#DataServers).

##### [Outlook MAPIFolders](#OutlookMapiFolders).

##### [Data Transfer](#DataTransfer).

##### [Schedule Board](#ScheduleBoard).

##### [Schedule Board Webpage](#ScheduleBoardWebpage).

##### [ASP.NET Framework](#AspNet).

##### [JavaScript](#JavaScript).

##### [CSS Formatting](#CSSFormating).

##### [FTP Assembly](#FTPAssembly).

VBA Development**.** This document shows details of the tools and techniques used to develop office automation applications. These projects consist of four main parts: Quote (Word VBA), Takeoff (Excel VBA), Schedule Board (Outlook VBA), and Schedule Webpage (Visual Studio .NET). Over 20,000 lines of code were written for these projects. A description of the applications’ functionality with code snippets follows.

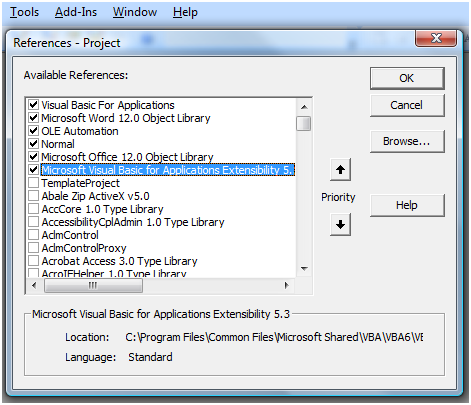
Quote-Takeoff Application. Designed and developed office automation systems for quoting jobs. The application uses MS Word as the user interface with MS Excel as the data source. The pairing of these office applications is done to provide the end users with a familiar interface.

Schedule Board Application**.** Outlook is used for the scheduling and resource assignment. Outlook is extended with a time -line type of interface allowing the user to add, edit and delete tasks directly on the graphical interface. In addition, a webpage is provided for access to the Schedule Board when out of the office. The webpage has full schedule update functionality, with authorization security (username-password) features.

Design**.** Office automation with VBA exposes in Word, the Excel object library. VBA procedures using Excel objects and saved in the Excel Project are callable in the Word Project, by setting the relevant References in the VBIDE. By setting References to a dynamic-link libraries (DLL) through a type library (TLB), a developer has access to functions in many DLLs, including custom assemblies written with Visual Studio in C+ (and most other languages). However, not all DLLs proved type libraries. The DLLs that form the Windows API do not provide type libraries. To call a function in the Windows API, a Declare statement is used to load the DLL. Here is an example of the Declare statement for the GetComputerName function:

Private Declare Function GetComputerName Lib "kernel32" \_

Alias "GetComputerNameA" (ByVal lpBuffer As String, nSize As Long) As Long

When a TLB is provided, the controls of an object library are exposed by setting the References in the VBIDE. In the VBIDE, selecting the Tools>References Menu bring s up a dialog box listing the TLB name for the available DLLs. Here is an image of the dialog box:

Application.Open**.** After setting the References, the next step in an automation project is opening one application from another. Before a documents can be opened, an the application must be loaded. This example first tries to access Excel. If an error is thrown, then the Excel application is loaded. After Excel is activated, the VBAUtility.xlsm workbook is opened. The workbook is opened, but not visible to the user (oExcel.Visible= False).

**Sub OpenExcel**()

Dim oExcel as Excel.Application

Dim oWB as Workbook

Dim workbookPath As String

On Error Resume Next

Set oExcel = GetObject(, "Excel.Application")

If Err Then

Set oExcel = New Excel.Application

End If

workbookPath = ThisDocument.path & "\VBAUtility.xlsm"

Set oWB = oExcel.Workbooks.Open(workbookPath)

oExcel.**Visible = False**

End Sub

Using Procedures**.** Once an application is referenced and opened, then all the objects and procedures of that application are available. For example, in Word, once Excel is referenced and a workbook is opened, then all the procedures in that workbook can be called from Word. In this example, a list of items from an Excel listbox is parsed, then a procedure called, Call **ExportVBComponent**(VBComp, folderName, ProcName) to populate a Word listbox.

Public **Sub viewSelectedItemExcel**(ByVal FileToGet As String)

'view selected item

Initialize:

Dim oExcel As Excel.Application

Dim oWb as Workbook

Dim ProcName As String

Dim compString As String

Dim folderName As String

Dim fileName As String

Dim I As Long

Dim S As String

'open Excel VBIDE to view procedure

On Error Resume Next

Set oExcel = GetObject(, "Excel.Application")

If Err Then

Set oExcel = New Excel.Application

End If

On Error GoTo ErrorHandler

Set oWb = oExcel.Workbooks.Open(FileToGet)

oWb.Windows(1).Visible = False

Set VBProj = oWb.VBProject

Main:

'parse selected item for procedure name

On Error Resume Next

folderName = Documents("VBAUtility.docm").path & "\VBAModules\VBAModulesExcel"

Dim columnNumber As Long

columnNumber = 0

With ControlPanel.ProcedureListBox

For I = 0 To .ListCount - 1

If .Selected(I) Then

S = .List(I, columnNumber)

compString = Left(S, InStr(S, ":") - 1)

Set VBComp = VBProj.VBComponents(compString)

If VBComp Is Nothing Then fileName = compString & ".doc"

ProcName = Right(S, Len(S) - (InStr(S, ":") + 1))

ProcName = Left(ProcName, InStr(ProcName, ",") - 1)

'call sub to show selected Excel procedure in Word

Call **ExportVBComponent**(VBComp, folderName, ProcName)

Exit Sub

End If

Next I

End With

Cleanup:

Set VBProj = Nothing

Set VBComp = Nothing

compString = vbNullString

Exit Sub

ErrorHandler:

MsgBox "An error occurred while processing 'viewSelectedItemExcel'."

Resume Cleanup

End Sub

Parsing**.** In most automation projects, parsing strings is required. Parsing is used to retrieve the part of string needed for the task at hand. In the previous snippet, each item of a Excel listbox is parsed to find the procedure name (ProcName), then that ProcName is used as an argument in the ExportVBComponent() function to populate Word listbox. The string functions: Left, Right, Len, InStr and InStrRev are four of the most useful functions for parsing.

Find.Text**.** Finding the string to parse is usually the first step in a parsing procedure. The following is an example of finding and concatenating a string to type the text into a document.

Sub walkoutCompaction()

Dim compactionFound As Boolean

With Selection

.Find.Text = "compaction"

compactionFound = .Find.Execute

If compactionFound Then

.Extend

.EndKey

.TypeText "compaction of inside and” & \_

“ outside of walkout and garage areas"

Else

.Find.Text = "labour, forms"

compactionFound = .Find.Execute

.MoveUp Unit:=wdLine, Count:=1

.InsertBefore vbCr

.EndKey

.TypeText "compaction of inside and” & \_

“outside of walkout and garage areas"

End If

End With

End Sub

Retrieving Data**.** Populating a Word document from a data source is key to many application. Using automation, Word can pull data directly, using Excel object properties. For example, this Word Sub retrieves FileToGet = Range("A1").Value and populates an array using an Excel range, proceduresTable, named programmatically. In addition, an Excel user defined function, **proceduresCount** is Run to determine the number of procedures, to size an array:

Sub proceduresData(Optional ByVal proceduresCount As Long)

Initialize:

'copy procedures from Excel to populate this Word document

On Error GoTo ErrorHandler:

Const procedureName = "proceduresData"

StatusBar = procedureName

Dim I As Long

Dim dataArray() As Variant

Main:

'run function in Excel to count procedures

proceduresCount = Run("AutomationModule.**proceduresCount**")

FileToGet = Range("A1").Value

'dimension variables with count

ReDim dataArray(proceduresCount, 2)

Range("A3:B" & proceduresCount + 2).Name = "proceduresTable"

dataArray = Range("proceduresTable")

For I = 1 To UBound(dataArray)

If dataArray(I, 1) <> vbNullString Then

ControlPanel.ProcedureListBox.AddItem dataArray(I, 1) & \_

", Lines: " & dataArray(I, 2)

End If

Next I

Cleanup:

'Kill dataArray

Exit Sub

ErrorHandler:

Debug.Print "An error was thrown by " & procedureName & \_

vbCr & Err.Number & ": " & Err.Description

Resume Cleanup:

End Sub

Data Servers**.** The automation methods require that the server application, in this case, Excel be activated. Opening a workbook in this way is slow. It can be faster to use an ADO recordset to retrieve data. Similar methods of data retrieval are implemented for most database management systems (DBMS). This example is a function that returns a variant array populated by an ADODB.recordset, retrieved from Excel . In the SQL Select statement, the From clause uses a variable, sectionName, containing a name range, in place of a table name, as is used with other databases.

Function recordsetArray(ByVal sectionName As String) As Variant

Dim cn As New ADODB.Connection

Dim rs As ADODB.Recordset

Dim arrRecordArray() As Variant

Dim newPath As String

newPath = ThisDocument.Path

With cn

.Provider = "MSDASQL"

.ConnectionString = "Driver={Microsoft Excel Driver (\*.xls)};" & \_

"DBQ=" & newPath & "\TakeoffTables.xls; "

.CursorLocation = CursorLocationEnum.adUseClient

.Open

End With

sectionName = "selected" & sectionName

On Error Resume Next

Set rs = cn.Execute("SELECT \* FROM " & sectionName & \_

" WHERE Count <> null")

‘populate Array.

arrRecordArray = rs.GetRows

recordsetArray = arrRecordArray

End Function

Outlook MAPIFolders**.** Microsoft Outlook is significantly different from Word or Excel. The Outlook object model is generally used to access items in folders. The first step in many procedures is to get the appropriate folder. There are 16 standard Outlook MAPIFolders. The following example gets the olFolderCalendar folder. The items in this folder are the calendar items. This procedure filters items for a range of dates. The properties of each item are exposed. In the follow code the item properties are read into an array for used by other procedures.

'this procedure gets the Scheduled Calendar Items from Outlook

Function jobtasksArray(Optional ByRef Cancelled As Boolean) As Variant()

Dim olnameSpace As Outlook.NameSpace

Dim fld As Outlook.MAPIFolder

Dim itms As Outlook.Items

Dim ritms As Outlook.Items

Dim lngCount As Long

Dim strStartDate As String

Dim dteEnd As Date

Dim strDateRange As String

Dim strEndDate As String

Dim I As Integer

Set olnameSpace = Application.GetNamespace("MAPI")

Set fld = olnameSpace.GetDefaultFolder(9) 'olFolderCalendar

Set itms = fld.Items

startDateForm.Show

strStartDate = startDateForm.startDateTextBox.Value

If startDateForm.startDateTextBox.Value = "cancel" Then

Cancelled = True

controlPanel.Hide

Exit Function

End If

If strStartDate = "" Then

greenBoard.Hide

dteStart = 0

Exit Function

End If

dteStart = CDate(strStartDate)

GoTo endDate

endDate:

dteEnd = dteStart + 14

CreateFilter:

'Create date range string

strStartDate = dteStart & " 12:00 AM"

strEndDate = dteEnd & " 11:59 PM"

strDateRange = "[Start] >= """ & strStartDate & \_

""" and [Start] <= """ & strEndDate & """"

dateRangeCaption = DateValue(strStartDate) & " - " & DateValue(strEndDate)

itms.IncludeRecurrences = True

itms.Sort Property:="[Start]", Descending:=False

If strDateRange <> "" Then

Set ritms = itms.Restrict(strDateRange)

ritms.Sort Property:="[Start]", Descending:=False

'Get an accurate count

'this is where all the filtered items are accessed

lngCount = 0

For Each itm In ritms

lngCount = lngCount + 1

Next itm

Else

Set ritms = itms

End If

numberScheduledTasks = lngCount

'here we store all Calendar items that start between the dates input and their key parameters

ReDim itemArray(lngCount, 9)

Dim workdayStart As Date

I = 0

For Each itm In ritms

I = I + 1

itemArray(I, 1) = itm.Subject

itemArray(I, 2) = itm.Start

itemArray(I, 3) = itm.End

itemArray(I, 4) = itm.Duration

itemArray(I, 5) = itm.Categories

itemArray(I, 6) = itm.RequiredAttendees

itemArray(I, 7) = itm.Body

'workdayStart, 7 am, for the day the item is schedule for

workdayStart = DateAdd("h", 7, DateValue(itm.Start))

'minutes between workdayStart, 7 am, and JobTask start

itemArray(I, 8) = dateDiff("n", workdayStart, itm.Start)

'days between filter Date range and JobTaks start

itemArray(I, 9) = dateDiff("d", dteStart, itm.Start)

'work backwards from minutes offset for jobTask to find the Time

Dim startTime As Date

startTime = DateAdd("n", itemArray(I, 8), workdayStart)

Next itm

'add two more column to the itemArray for color code and jobNumber

ReDim Preserve itemArray(lngCount, 9)

'this is the array with all the job tasks and the 9 parameters needed for taskControls

On Error GoTo ErrorHandlerExit

jobtasksArray = extendedArray

ErrorHandlerExit:

Exit Function

ErrorHandler:

MsgBox prompt:="Error No: " & Err.Number & "; Description: " & \_

Err.Description, \_

Buttons:=vbOKOnly + vbCritical, \_

Title:="Error Getting Calendar"

Resume ErrorHandlerExit

End Function

Here the olFoldersContacts is used to populate a listbox for the user to make a resource selection.

Sub contactsList()

Dim myNamespace As Outlook.NameSpace

Dim myContacts As Outlook.Items

Dim myItems As Outlook.Items

Dim myItem As Object

Set myNamespace = Application.GetNamespace("MAPI")

Set myContacts = myNamespace.GetDefaultFolder(olFolderContacts).Items

Set myItems = myContacts

For Each myItem In myItems

If (myItem.Class = olContact) Then

taskAppointmentForm.contactsComboBox.**AddItem** myItem.fullName

End If

Next

End Sub

Data Transfer**.** In this example, the contacts are imported from Excel using a DAO.Database.Recordset. The SQL Select statement uses an Excel named range in place of a table name in the From clause. The MAPIFolder used is the olFolderContacts.

'this code get the Contacts from Excel

Sub transferContacts()

OpenConnectionToExcel:

Dim db As DAO.Database

Dim rs As DAO.Recordset

Dim contactsFile As String

Dim iRecord As Long

contactsFile = relativePathModule.myPath & "\My Projects\Project.Scheduler\contacts.xls"

Set db = OpenDatabase(Name:=contactsFile, Options:=False, ReadOnly:=False, Connect:="Excel 8.0")

Set rs = db.OpenRecordset("SELECT \* FROM `conactsTable` WHERE firstName <> null")

Dim totalCount As Long

rs.MoveLast

totalCount = rs.recordCount

rs.MoveFirst

CreateContactItems:

On Error Resume Next

Dim myNamespace As Outlook.NameSpace

Dim myFolder As Outlook.Folder

Dim myItem As Outlook.ContactItem

Dim fullName As String

Set myNamespace = Application.GetNamespace("MAPI")

Set myFolder = myNamespace.GetDefaultFolder(olFolderContacts)

Set myItem = myFolder.Items.Add

While Not rs.EOF

Set myItem = Outlook.CreateItem(olContactItem)

fullName = rs.Fields(3).Value & ", " & rs.Fields(1).Value

Call findName(fullName) 'check for duplicates

myItem.FileAs = rs.Fields(3).Value & ", " & rs.Fields(1).Value

myItem.LastName = rs.Fields(3).Value

myItem.FirstName = rs.Fields(1).Value

myItem.CompanyName = rs.Fields(5).Value

myItem.BusinessAddress = rs.Fields(8).Value

myItem.MobileTelephoneNumber = rs.Fields(40).Value

myItem.BusinessTelephoneNumber = rs.Fields(31).Value

myItem.HomeTelephoneNumber = rs.Fields(37).Value

myItem.Email1Address = rs.Fields(57).Value

myItem.Email1DisplayName = rs.Fields(59).Value

iRecord = iRecord + 1

If iRecord > totalCount Then GoTo ErrorHandler 'stop loop if count exceeds total records spreadsheet

myItem.Save

rs.MoveNext

Wend

MsgBox prompt:="Import Complete: " & iRecord, \_

Buttons:=vbOKOnly + vbInformation, \_

Title:="Import Contacts"

ErrorHandlerExit:

GoTo ExitSub

ErrorHandler:

MsgBox prompt:="Error No: " & Err.Number & "; Description: ", \_

Buttons:=vbOKOnly + vbCritical, \_

Title:="Import Appointments"

Resume ErrorHandlerExit

ExitSub:

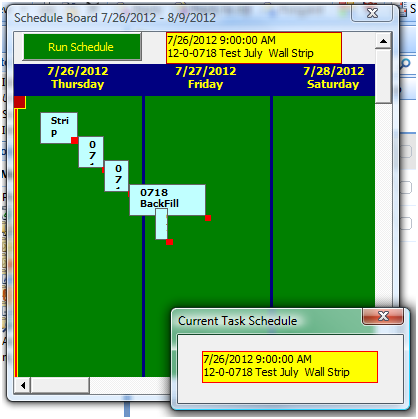
rs.Close

db.Close

Set rs = Nothing

Set db = Nothing

End Sub

Schedule Board**.** The Outlook standard calendar interface is a table, with cells representing days. However, Project Schedulers often prefer to view a calendar as a time line. The Schedule Board is such a graphical interface. Here is an image:

Each schedule item on the GreenBoard is represented with a textbox; there can be hundred of schedule items over a two week period. Controls are added to the GreenBoard dynamically. To add controls dynamically, using the minimum number lines of code, Class Modules with EventHandlers are used. The following code snippet is in the code behind the GreenBoard. The code calls the Class Modules, taskEventHander, which uses mouse events to implement drag and drop functionality to the dynamically added textboxes. A label is also added dynamically for each schedule task to provide a handle for resizing the control (the small red squares in the image). The resizeEventHandler is another Class Module, not shown.

‘this adds a textbox for each scheduled item

Sub addJobTasks(Optional ByRef Cancelled As Boolean)

On Error GoTo ExitSub

Dim taskControl As MSForms.TextBox

Dim resizeControl As MSForms.Label

Dim I As Integer

Dim priorEnd As Integer

Dim priorTop As Integer

priorEnd = 0

'the expanded array function adds the color code to the jobTasksArray

'the jobTasksArray function gets Calendar item data to put into the

'jobTasksArray

'the function gets JobTasks,stores the items in newArray, so that don't have to

'go back to the array 'function

newArray = jobTasksGetModule.jobtasksArray

'If Cancelled Then Exit Sub

'synch Day Labels with start date, dteStart

If dteStart = 0 Then Exit Sub

synchDayLabels

'add a textbox for each JobTask

For I = 1 To UBound(newArray(), 1)

'add the textbox controls

Set taskControl = Me.Controls.Add("forms.textbox.1", "Task" & I, True)

'add Task# control to the taskControls array

If TypeName(taskControl) = "TextBox" Then

taskControls.Add Me.Controls("Task" & I)

End If

'set the attributes of the new Task control, using newArray as the data source

With taskControl

greenBoard.Caption = "Schedule Board " & dateRangeCaption

.Enabled = True

.Height = 24

If newArray(I, 9) = "" Then

MsgBox newArray(I, 1) & " Color is null."

End If

.BackColor = newArray(I, 11)

If .BackColor = 0 Then .BackColor = newArray(I - 1, 11)

.BorderStyle = 1

.TextAlign = fmTextAlignLeft

.Font.Size = 7

.Font.Bold = True

.WordWrap = True

.MultiLine = True

.ControlTipText = newArray(I, 1) '.Subject=1

.Top = 42 + 18 \* I 'priorTop

.text = newArray(I, 10) & vbCr & newArray(I, 5)

'jobNumer=8 '.Categories=5

.Width = newArray(I, 4) / 6.25 '.Duration

If .Width < 4 Then .Width = 4

.Left = newArray(I, 9) \* 96 + newArray(I, 8) / 6.25

'offset from 7 am for the day the item is schedule for

If newArray(I, 8) = 0 Then .Left = .Left + 2

If newArray(I, 8) = 96 Then .Left = .Left - 1

priorEnd = newArray(I, 8) + newArray(I, 4)

priorTop = .Top

End With

'add a resizeControl for each taskControl

Set resizeControl = Me.Controls.Add("forms.Label.1", "Resize" & I, True)

'add Task# control to the taskControls array

If TypeName(resizeControl) = "Label" Then

resizeControls.Add Me.Controls("Resize" & I)

End If

With resizeControl

.Left = Me.Controls("Task" & I).Left + Me.Controls("Task" & I).Width- 1

.Top = Me.Controls("Task" & I).Top + Me.Controls("Task" & I).Height - 1

.Width = 5

.Height = 5

.MousePointer = 8

.BackStyle = 1

.BackColor = vbRed

resizeControlsArray(I) = .Name

End With

Next I

ExitSub:

End Sub

'this uses the Class Module taskEventHandler to minimize repeating code for each added ‘TaskControl

Sub addTasksHandler()

Set taskCollectionOfEventHandlers = New Collection

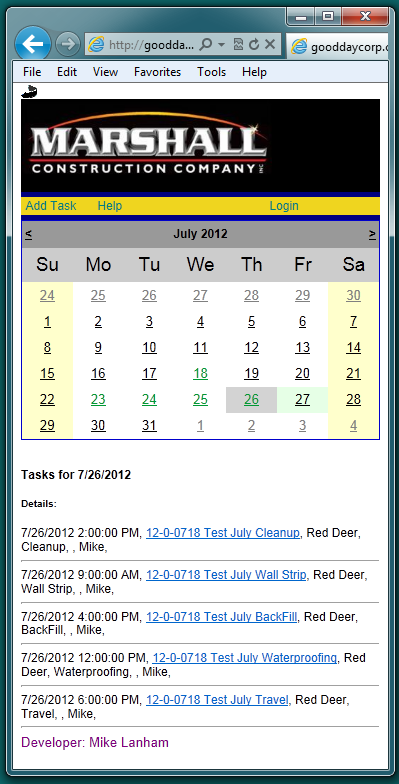
Dim I As Integer

Dim oControl As Control

Dim oEventHandler As taskEventHandler

For Each oControl In Me.Controls 'taskControls, textbox

If TypeName(oControl) = "TextBox" Then

 Set oEventHandler = New taskEventHandler

Set oEventHandler.taskControl = oControl

taskCollectionOfEventHandlers.Add oEventHandler

End If

Next oControl

End Sub

Schedule Board Webpage**.** Access to the Scheduler is provide by the [www.gooddaycorp.com](http://www.gooddaycorp.com) website. There are 4 parts to the web. The web form controls, data access code behind, JavaScript functionality and CSS formatting. The connection between the website and the Schedule Board in Outlook is through a COM assembly built with Visual Studio that provides a TLB for easy References.

ASP.NET**.** Visual Studio was used to develop this website. This ASP.NET website is built using ASP.NET 3.5 framework, with C#. The main functionality of the web is provided by the DataCalendar assembly by Mike Ellison (2004), downloaded from [The Code Project](http://www.codeproject.com/Articles/5347/DataCalendar). The site is designed with a MasterPage that contains the links to the style pages (**CSS Formating**), the Membership controls (login) and the **JavaScript**. The content pages contain the web form controls: calendar control, a repeater, databound to the calendar and the data source connection script. The data source is an Excel file that is transferred to the website using an **FTP Assembly**. The Excel file is populated from the Outlook Schedule Board, VBA.

JavaScript**.** Client side Javascript livens up the webpage by animating the Membership controls, to popup, with fade and resizing. The popups are set to grow on the screen for 1000 milliseconds, while the background is grays-out. Here is a snippet:

function greyout(d, z) {

var obj = document.getElementById('greyout');

if (!obj) {

appendElement(document.body, 'div', 'greyout');

obj = document.getElementById('greyout');

obj.style.position = 'absolute';

obj.style.top = '0px';

obj.style.left = '0px';

obj.style.background = '#111';

obj.style.opacity = '.3';

obj.style.filter = 'alpha(opacity=30)';

}

if (d) {

var ch = document.documentElement.clientHeight ? document.documentElement.clientHeight : document.body.clientHeight;

var cw = document.documentElement.clientWidth ? document.documentElement.clientWidth : document.body.clientWidth;

var sh = document.documentElement.scrollHeight ? document.documentElement.scrollHeight : document.body.scrollHeight;

if (document.body.scrollHeight) sh = Math.max(sh, document.body.scrollHeight)

var sw = document.documentElement.scrollWidth ? document.documentElement.scrollWidth : document.body.scrollWidth;

if (document.body.scrollWidth) sh = Math.max(sh, document.body.scrollWidth)

var wh = window.innerHeight ? window.innerHeight : document.body.offsetHeight;

if (!z) { z = 50 }

obj.style.zIndex = z;

obj.style.height = Math.max(wh, Math.max(sh, ch)) + 'px';

obj.style.width = Math.max(sw, cw) + 'px';

obj.style.display = 'block';

addEvent(window, 'resize', greyoutResize);

}

else {

obj.style.display = 'none';

removeEvent(window, 'resize', greyoutResize);

}

}

function greyoutResize() {

var ch = document.documentElement.clientHeight ? document.documentElement.clientHeight : document.body.clientHeight;

var cw = document.documentElement.clientWidth ? document.documentElement.clientWidth : document.body.clientWidth;

var sh = document.documentElement.scrollHeight ? document.documentElement.scrollHeight : document.body.scrollHeight;

if (document.body.scrollHeight) sh = Math.max(sh, document.body.scrollHeight)

var sw = document.documentElement.scrollWidth ? document.documentElement.scrollWidth : document.body.scrollWidth;

if(document.body.scrollWidth) sh = Math.max(sh,document.body.scrollWidth)

var wh = window.innerHeight ? window.innerHeight : document.body.offsetHeight;

var obj = document.getElementById('greyout');

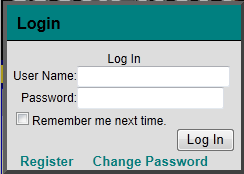
obj.style.height = ch+'px';

obj.style.width = cw+'px';

obj.style.height = Math.max(wh,Math.max(sh,ch))+'px';

obj.style.width = Math.max(sw,cw)+'px';

}

CSS Formatting**.** Formatting is specified in separate CSS files for cleaner code and easier maintenance. Here is a snippet:

On the Masterpage is this line of code:

<link href="Style/popup.css" rel="stylesheet" type="text/css" />

Which links to this CSS:

body

{

text-decoration: none;

font-family: Arial, Helvetica, sans-serif;

font-size: small;

font-weight: normal;

font-style: normal;

background-color: #FFFFFF;

}

.popup {

border: outset 4px Gray;

font-family: Arial, Helvetica, sans-serif;

font-size: 12px;

display: none;

position: fixed;

left: 10px;

top: 50px;

z-index: 100;

}

.popuptitle {

background-position: center;

background: teal;

color: black;

font-weight: bold;

font-size: 12pt;

height: 15px;

padding: 10px;

border-bottom: solid 1px black;

}

.popupbody {

background-position: inherit;

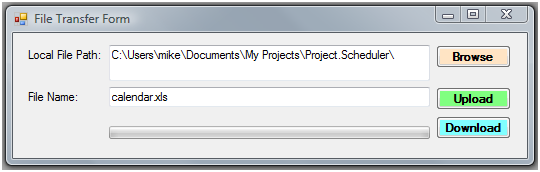
background: #ddd;

padding: 10px 5px 0px 5px;

text-align: left;

font-family: Arial, Helvetica, sans-serif;

}

FTP Assembly**.** The connection between the website and the Schedule Board in Outlook is through a COM assembly build with Visual Studio that provides a TLB for easy References. The Visual Studio project uses the Class Library template in VB. This template is COM Class that provides a TLB file when compiled with: Compile Option- “Register for COM interop” and Assembly Information- “Make assembly COM-Visible”, both checked . The FTP functionality is provided by System.Net.FtpWebRequest. This code snippet show the procedure: 

Public Sub ftpDownloadWebRequest(ByVal path As String, ByVal name As String)

Dim localFile As String

localFile = path & name

Const remoteFile As String = "/gooddaycorp.com/wwwroot/App\_Data/calendar.xls"

Const host As String = "ftp://xxxxxxxxxx@xxx.xxx.xxx.xxx"

Const username As String = "xxxxxxxxxxx"

Const password As String = "xxxxxxxxxxx"

'1. Create a request: must be in ftp://hostname format,

' not just ftp.myhost.com

Dim URI As String = host & remoteFile

Dim ftp As System.Net.FtpWebRequest = \_

CType(System.Net.FtpWebRequest.Create(URI), \_ System.Net.FtpWebRequest)

'2. Set credentials

ftp.Credentials = New \_

System.Net.NetworkCredential(username, password)

'3. Settings and action

ftp.KeepAlive = False

'we want a binary transfer, not textual data

ftp.UseBinary = True

'Define the action required (in this case, download a file)

ftp.Method = System.Net.WebRequestMethods.Ftp.DownloadFile

'4. If we were using a method that uploads data e.g. UploadFile

' we would open the ftp.GetRequestStream here an send the data

'5. Get the response to the Ftp request and the associated stream

Using response As System.Net.FtpWebResponse = \_

CType(ftp.GetResponse, System.Net.FtpWebResponse)

Using responseStream As IO.Stream = response.GetResponseStream

'loop to read & write to file

Using fs As New IO.FileStream(localFile, IO.FileMode.Create)

Dim buffer(2047) As Byte

Dim read As Integer = 0

Do

read = responseStream.Read(buffer, 0, buffer.Length)

fs.Write(buffer, 0, read)

'downloadForm.ProgressBar1.PerformStep()

Loop Until read = 0 'see Note(1)

responseStream.Close()

fs.Flush()

fs.Close()

End Using

responseStream.Close()

End Using

response.Close()

End Using

MsgBox("Download Complete")

'downloadForm.ProgressBar1.Value = 0

'6. Done! the Close happens because ftp goes out of scope

' There is no .Close or .Dispose for FtpWebRequest

End Sub