


# OUTLOOK CALENDAR TO WORKFRONT TIME-OFF INTEGRATION SETUP



Zachary Pettit  
November 5, 2018

# Contents

1	OVERVIEW .....	3
1.	SYSTEM REQUIREMENTS .....	3
	CUSTOM POWER SETTINGS FOR PHYSICAL PC.....	3
2.	ASSUMPTIONS .....	4
2	POWERSHELL .....	5
1.	INTRODUCTION TO WINDOWS POWERSHELL.....	5
2.	SET EXECUTION POLICY.....	5
3.	GET-OUTLOOKCALENDAR .....	5
4.	CALL_GET-OUTLOOKCALENDAR .....	7
3	ACCESS .....	8
1.	INTRODUCTION TO MICROSOFT (MS) ACCESS .....	8
	THE CODE.....	9
2.	HOW-TO UPDATE WORKFRONT INFORMATION .....	11
	STEP 1 LOGIN .....	11
	STEP 2 REFRESH DB.....	13
	STEP 3 PUT ALL .....	15

4	TASK SCHEDULER .....	17
1.	INTRODUCTION TO TASK SCHEDULER .....	17
2.	POWERSHELL SCRIPT .....	17
3.	ACCESS DB.....	19
5	GENERAL ADJUSTMENTS .....	20
1.	CHANGING PATH TO CALDATA .....	20
6	ERROR HANDLING.....	24
1.	POWERSHELL .....	24
	EMPTY CSV .....	24
2.	ACCESS .....	25
	INVALID USE OF NULL.....	25
	LIBRARY NOT FOUND.....	25
	ERROR LOADING DLL .....	26
3.	WORKFRONT .....	26
	AUTHENTICATION EXCEPTION.....	26
	SUBSCRIPT OUT OF RANGE.....	27
4.	OUTLOOK .....	27
	YOU ARE NOT CURRENTLY SIGNED IN.....	27

## 1 OVERVIEW

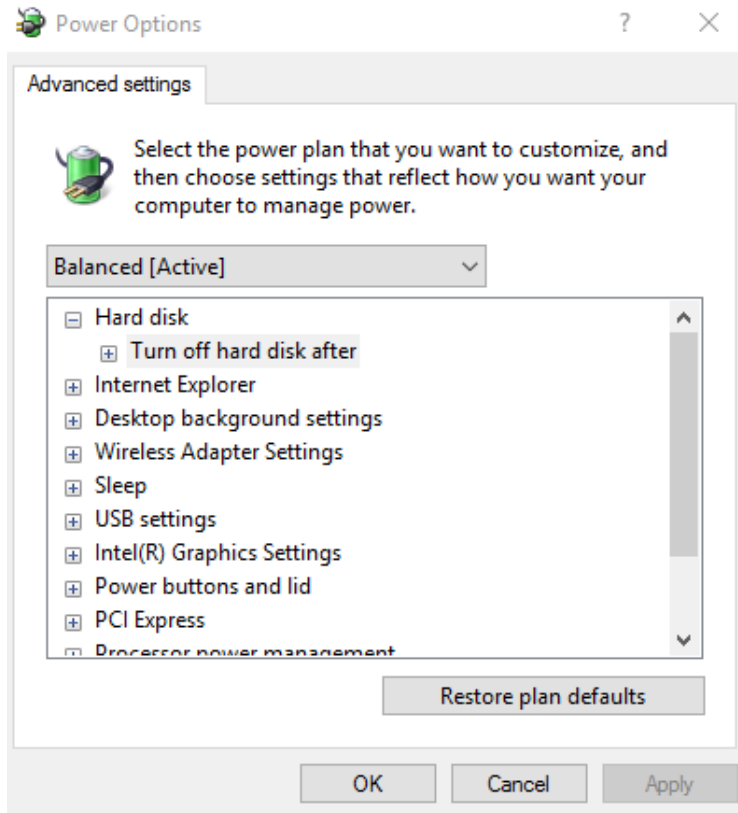
This Outlook—Workfront Integration is designed to create a link that automatically reflects each entry into the Agency's shared Leave Calendar with each user in the Organization's Workfront Time-Off Calendars. This feat is accomplished via a two-part solution. First, the Leave Calendar data is extracted from Microsoft (MS) Outlook and copied to a .csv file. Next, that file is drawn into MS Access using a linked table where it is compared to the contacts that Access GETs from Workfront and organized by user and uploaded to each user's Reserved Time field on Workfront as bulk PUT methods. Each run of this process must include all of the data that is expected to be reflected in each user's Workfront Time-Off Calendar. This is necessary due to Workfront's REST-ful architecture which restricts the API from retaining memory of previous uploads. This increases the data load on the solution compared to only uploading changes since the last upload, but it also means that this solution is a relatively direct sync between Outlook and Workfront. The Workfront Time-Off Calendars will always be point-by-point reflections of the Outlook Leave Calendar even when events are deleted or rescheduled.

### 1. SYSTEM REQUIREMENTS

- Dedicated Windows desktop or virtual machine (VM) running Windows 10
  - 1 GHz processor
  - 2GB RAM
  - Microsoft DirectX 9 graphics device with WDDM driver
  - Reliable Ethernet or Wi-Fi connection
- Powershell installed
- Task Scheduler installed
- Microsoft Access 2016 installed
- Outlook Application is installed and associated with the Department's shared Leave Calendar
- The user setting up this system has administrative privileges over their device
- The user setting up or otherwise referenced through this system has a Plan License associated with their Workfront profile

### CUSTOM POWER SETTINGS FOR PHYSICAL PC

To run MS Access through Task Scheduler, you will need to keep the computer logged in. The computer can be locked and in power-save mode, but the Task Scheduler will cease to run if the computer is fully logged out or shut down. To set up your computer to stay in power save mode without shutting down, navigate to Control Panel > All Control Panel Items > Power Options. This can be found by searching for "power and sleep settings" in the global navigation bar and select "Additional Power Settings." Then, by selectin "Create a Power Plan" and creating a power plan with any settings and selecting "Change Plan Settings" > "Change Plan Advanced Settings" causing the Power Options dialog box to open:



In the Power Options dialog box, type “0” in the text box denoted “Hard Disk” > “Turn off hard disk after” > “Plugged in (Minutes)” to set the time to shut down to “Never.”

Then, in the Power Options dialog box, type “0” in the text box denoted “Sleep” > “Hibernate after” > “Plugged in (Minutes)” to set the time to hibernation to “Never.”

## 2. ASSUMPTIONS

1. This setup procedure is performed on the dedicated machine described in **Section 1.1**.
2. All Department Leave Events are logged in the single, shared calendar.
3. All users submitting Leave Events have an associated Workfront account using the same email that they use to submit Leave Events. Any submissions from emails without associated Workfront accounts will be dumped in the suspenseData table and will not be uploaded.
4. The blank database provided assumes that Meeting Organizer is generated using following formula: “LASTNAME Firstname M.I.”  
This may need to be adjusted depending on your system, check **SupportModule.Organizer** to edit this logic (reference **Section 3.1**).
5. This program’s run time is directly related to the number of Workfront users. Our Workfront account include ~100 users an organization with thousands or tens of thousands of Workfront user accounts that need to be updated may see very different results when executing Access. Similarly, our implementation only uploads ~1000 events per run, if your implementation includes orders of magnitude of additional events, then you may see very different results when executing Powershell. The number of events can be controlled indirectly by dictating the time interval that Powershell draws from Outlook using the procedure laid out in **Section 2.3**.

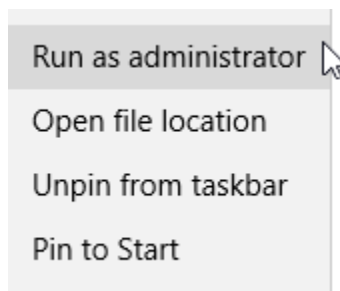
## 2 POWERSHELL

### 1. INTRODUCTION TO WINDOWS POWERSHELL

Powershell is a scripting language built into Windows OS. It is designed to form a shell around other programs and to create links where they do not exist natively. To open Powershell, use the global navigation bar to look up “powershell.” Open it in its folder and make a note of the path to its location for later (e.g. “C:\Windows\System32\WindowsPowerShell\v1.0\powershell.exe”). Most Windows systems will offer you a choice between Powershell and Powershell ISE. Both function the same, but ISE includes a more user friendly interface. You can choose either tool to set up this integration solution, but you should complete all steps using only one of the two tools because the two do not share data so shifting from one to the other over the course of this setup process may cause you to miss critical steps. I recommend using Powershell classic for this setup because whichever tool you choose will become unavailable while it is running and I tend to prefer to keep Powershell ISE available to test Powershell scripts manually.

### 2. SET EXECUTION POLICY

Before proceeding, you will need to create a new cmdlet in Powershell that can be used to access data from the default Outlook calendar. To begin, open powershell.exe as an administrator by right clicking on the Powershell application and selecting “Run as administrator:”



Then, enter the command `[get-execution policy]`. If Powershell returns “Restricted,” then enter the command `[set-execution policy unrestricted]` to authorize Powershell to run scripts.

### 3. GET-OUTLOOKCALENDAR

First, you will need to create a new cmdlet in Powershell that can be used to access data from the default Outlook calendar. To begin, open powershell.exe. Next, open your profile by using the command `[notepad $profile]`. If you do not have a profile (`[notepad $profile]` will return an error), then create a new profile using:

```
if(!(Test-Path -Path $profile)){New-Item -ItemType File -Path $profile -Force}
```

then, you should successfully be able to use `[notepad $profile]` to edit your profile. To do so, paste the following code into your profile:

```
Function Get-OutlookCalendar
```

```
{  
    Add-type -assembly "Microsoft.Office.Interop.Outlook" | out-null  
    $olFolders = "Microsoft.Office.Interop.Outlook.OlDefaultFolders" -as [type]  
    $outlook = new-object -comobject outlook.application  
    $namespace = $outlook.GetNameSpace("MAPI")  
    $folder = $namespace.getDefaultFolder($olFolders::olFolderCalendar)  
    $folder.items |  
        Select-Object -Property Organizer, Start, End  
} #end function Get-OutlookCalendar  
  
$startDate = Get-Date  
$startDate = $startDate.AddYears(-1)  
$endDate = Get-Date  
$endDate = $endDate.AddYears(1)
```

This will create/modify the automatic profile associated with your Powershell program. So, every time powershell.exe is opened in the future, this profile will be loaded including the Get-OutlookCalendar custom cmdlet.

The variables \$startDate and \$endDate can be edited to reflect the length of time you want to cover. The time interval reflected in this code is a range from one year before the current date to one year after the current date.

#### 4. CALL\_GET-OUTLOOKCALENDAR

Next, you will need to create a .ps1 file to call the Get-OutlookCalendar cmdlet you just created. Simply save the attached file titled “Call\_Get-OutlookCalendar” to your computer or create a new .ps1 file containing the Powershell code:

```
Get-OutlookCalendar | where-object { $_.start -gt $startDate -AND $_.start -lt ` $endDate } |  
sort-object Organizer | Export-Csv -Path "C:\Users\Public\calData.csv" -Force  
-notypeinformation
```

You can replace “C:\Users\Public\calData.csv” with the path and name of the .csv file you want the program to output. But, if you do, review **Section 5.1** to adjust for that change in your version of the Access database.

Next, test-run the script by inputting the file location of your .ps1 file into Powershell. You may need to use the Unblock-File cmdlet to authorize this file to run automatically. To unblock the file, enter the following code:

```
Unblock-File -Path C:\Users\Public\Call_Get-OutlookCalendar.ps1
```

Replace “C:\Users\Public\Call\_Get-OutlookCalendar” with your path to Call\_Get-OutlookCalendar.ps1 or whatever name you assigned the .ps1 call file for the Get-OutlookCalendar cmdlet. Next, run the ps1 file in Powershell to confirm that it has been unblocked.



### 3 ACCESS

#### 1. INTRODUCTION TO MICROSOFT (MS) ACCESS

MS Access can be found by entering “access” into the global navigation bar. Before opening the program, select “Show in File” and make a note of the path to MSACCESS.EXE. Inside Access, open your version of the Integration Database: inside the database, begin by opening the form: ToWorkfront.

This is the focal point of the database. Every other function of the database is either accessed through or accesses functions in this form:

**Figure 1**

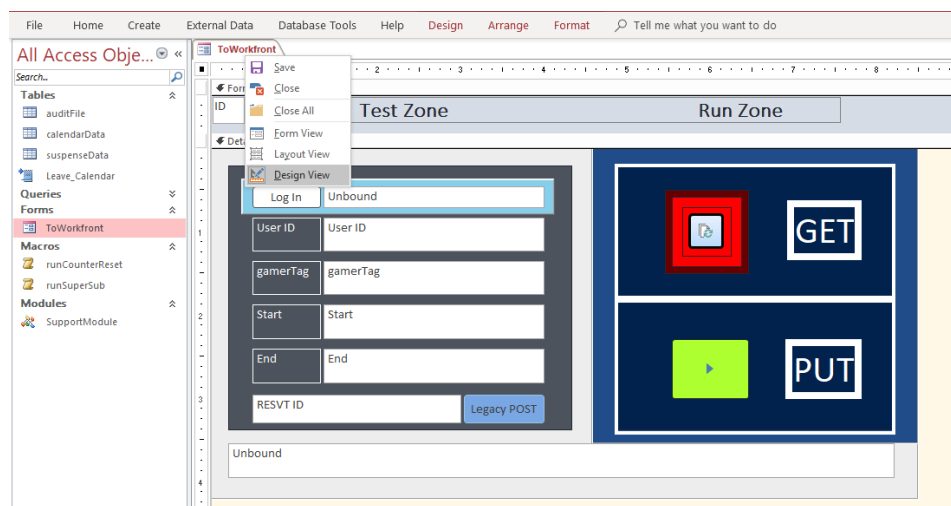
The ToWorkfront form depicted in **Figure 1** is accessed by double clicking “ToWorkfront” under “Forms” in the menu to the left of the screen from anywhere in the database as seen in **Figure 1.1**. The ToWorkfront form displays records from the calendarData table which, in turn, is populated by re-organizing the data from the Leave\_Calendar linked table. The first action after accessing ToWorkfront should always be clicking the button marked “Log in” in **Figure 1.2** which will submit your login information, once you’ve set it, **Section 3.2.1**, to provide you with a Session ID which is used to execute the other form functions. This ID is a temporary authentication code that is refreshed with each login.

After logging in, the next step should be to select the button labelled “GET,” **Figure 1.3**, which will refresh the entire database using the Leave\_Calendar table and a list of active users that it GETs from Workfront. This refreshes the data behind the form too so it will need to close the form. Simply double click “ToWorkfront” in **Figure 1.1** to re-open it afterwards. Next, log in again, **Figure 1.2** and select the button labelled “PUT,” **Figure 1.4**. This will take all of the data from the calendarData table, organize it into the contact list stored in the auditFile table and then submit the Leave Calendar data to Workfront by user. The macro titled “runSuperSub” performs this entire process automatically, simply double click it to test. The macro is what is called by the Task Scheduler.

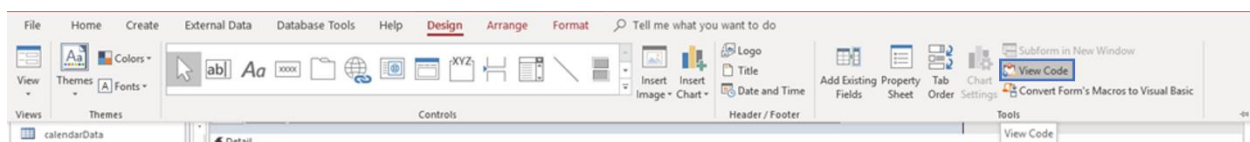
You can also use the arrows to manually navigate calendarData records, **Figure 1.5**. The Legacy POST button, **Figure 1.6**, is not part of the normal operation of this database, but it can be used to manually test individual records; however, if it is pressed, it will completely replace all of the current user’s RESVT data with the single record POST-ed. This can be undone by selecting PUT All, **Figure 1.3**, to replace all user data with the current Leave Calendar data, but it is still highly recommended that you follow the procedure illustrated in **Section 2.2** to switch to the Workfront sandbox before using this function.

## THE CODE

In the ToWorkfront form, right click on the tab above the form denoted “ToWorkfront and select “Design View” from the dropdown menu:



The form is now editable and the buttons cannot be pressed until you return to Form View. Next, select “View Code” from the Design tab in the menu at the top of the screen to view the code behind the form:



Which will open a new window containing the VBA code that guides the ToWorkfront form:

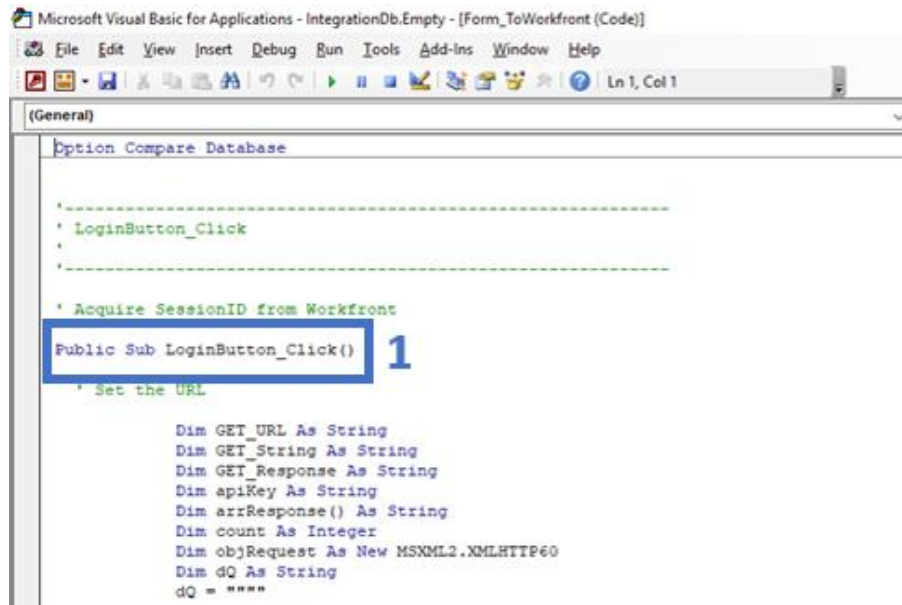


Figure 2

This is the code behind ToWorkfront. Each block of code is parsed into public subs with unique names as depicted in **Figure 2.1**. This document refers to subs using the format **[Form/Module].[Sub/Function]** e.g. **ToWorkfront.LoginButton\_Click**.

All VBA code that is not stored behind ToWorkfront is in the Support Module, accessible by double clicking “Support Module” from the menu to the left of the screen when viewing the Access Database:

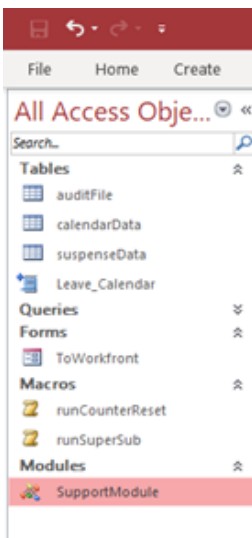


Figure 3

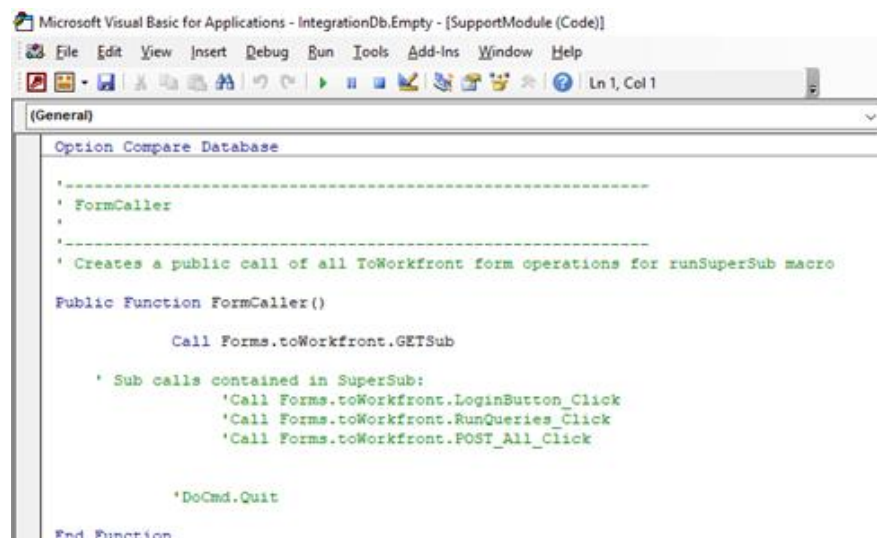


Figure 4

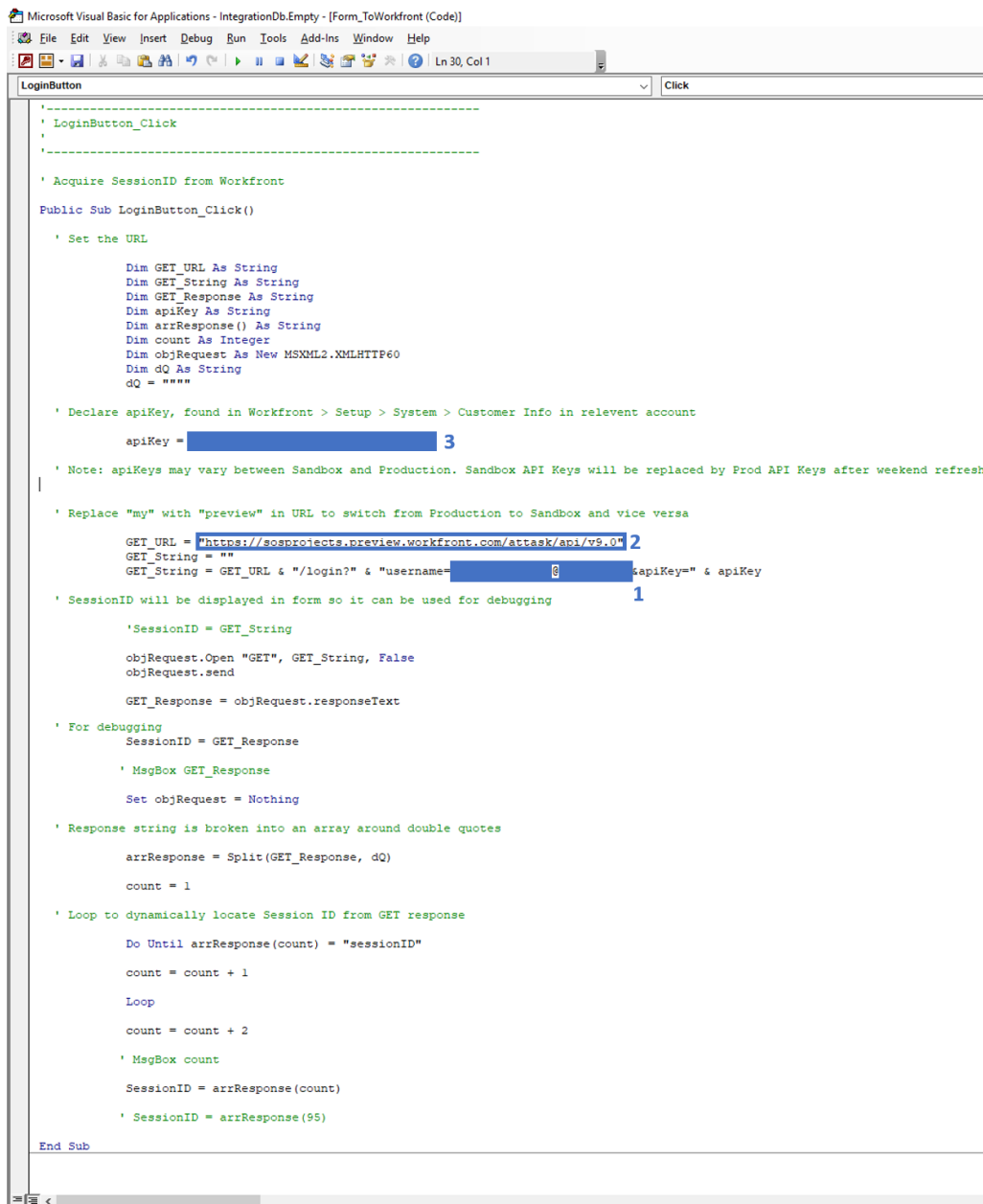
The Support Module is accessed by double clicking “Support Module” in the menu bar, **Figure 3**, which will access the Support Module code, **Figure 4**, which supplements to the code behind ToWorkfront.

## 2. HOW-TO UPDATE WORKFRONT INFORMATION

Note: To shift from the sandbox to the active account, use CTRL + F (in both the ToWorkfront code and the SupportModule code), search for “preview” and replace all with “my” and vice-versa to shift from active to sandbox.

### STEP 1 LOGIN

The login function is a pre-requisite to perform each of the other two central ToWorkfront functions. The login button, pictured in **Figure 1.2**, is editable via the sub: **ToWorkfront.LoginButton\_Click**, navigated to in **Section 3.1.1**:



```
Microsoft Visual Basic for Applications - IntegrationDb.Empty - [Form_ToWorkfront (Code)]
File Edit View Insert Debug Run Tools Add-Ins Window Help
Ln 30, Col 1

LoginButton Click

'-----
' LoginButton_Click
'-----

' Acquire SessionID from Workfront
Public Sub LoginButton_Click()

    ' Set the URL

    Dim GET_URL As String
    Dim GET_String As String
    Dim GET_Response As String
    Dim apiKey As String
    Dim arrResponse() As String
    Dim count As Integer
    Dim objRequest As New MSXML2.XMLHTTP60
    Dim dQ As String
    dQ = ""

    ' Declare apiKey, found in Workfront > Setup > System > Customer Info in relevent account
    apiKey = [REDACTED] 3

    ' Note: apiKeys may vary between Sandbox and Production. Sandbox API Keys will be replaced by Prod API Keys after weekend refresh

    ' Replace "my" with "preview" in URL to switch from Production to Sandbox and vice versa
    GET_URL = "https://spsprojects.preview.workfront.com/ataask/api/v9.0" 2
    GET_String = ""
    GET_String = GET_URL & "/login?" & "username=[REDACTED]8" & "apiKey=" & apiKey

    ' SessionID will be displayed in form so it can be used for debugging 1
    GET_Response = GET_String

    objRequest.Open "GET", GET_String, False
    objRequest.send

    GET_Response = objRequest.responseText

    ' For debugging
    SessionID = GET_Response
    MsgBox GET_Response

    Set objRequest = Nothing

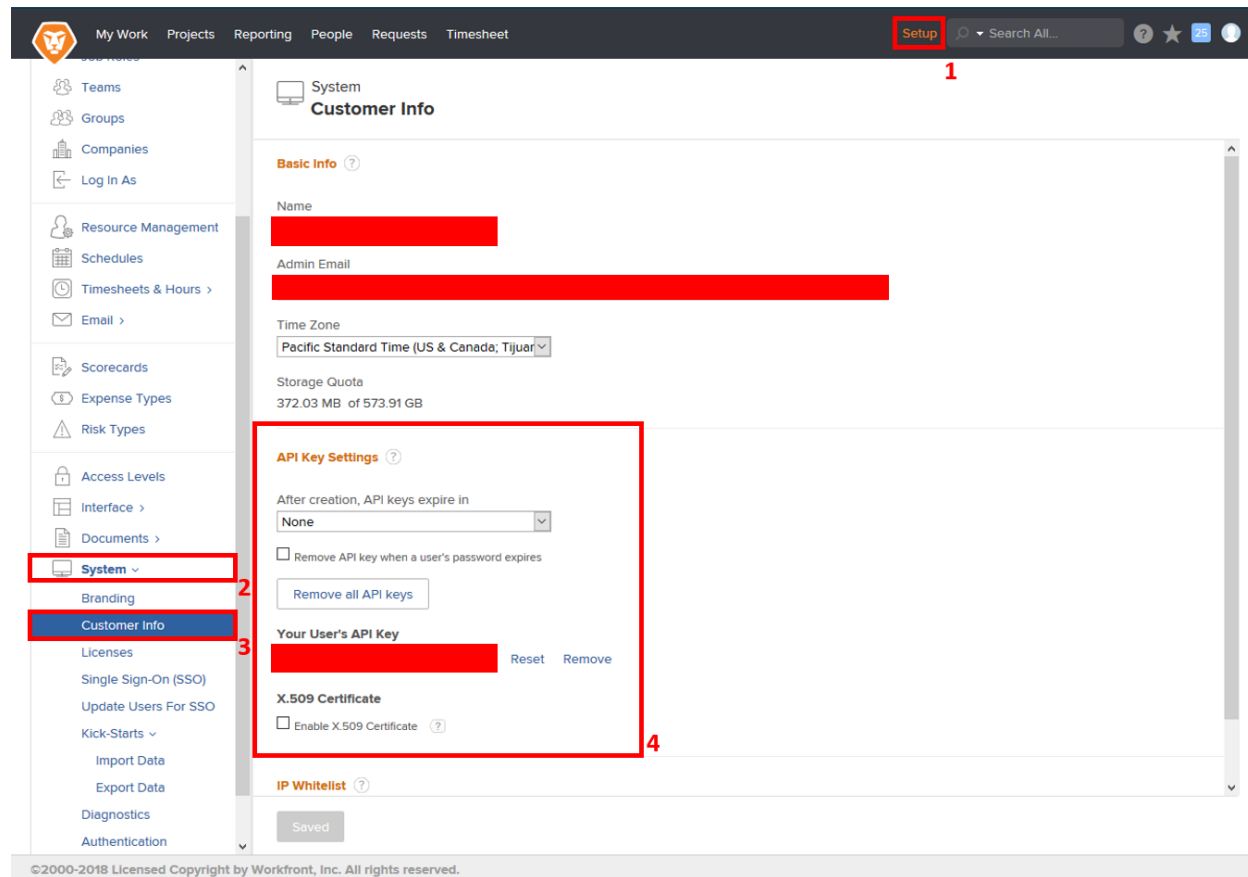
    ' Response string is broken into an array around double quotes
    arrResponse = Split(GET_Response, dQ)
    count = 1

    ' Loop to dynamically locate Session ID from GET response
    Do Until arrResponse(count) = "sessionID"
        count = count + 1
    Loop
    count = count + 2
    MsgBox count
    SessionID = arrResponse(count)
    ' SessionID = arrResponse(95)

End Sub
```

Figure 5

To update the login parameters, you will first need to update the username to the email for the dedicated Workfront account with a Plan license, **Figure 5.1**. Next, update the Workfront URL in **Figure 5.2** to reflect the relevant Workfront website. Finally, you will need an API Key associated with the dedicated Workfront account, **Figure 5.3**. This can be found/created in Workfront Setup:

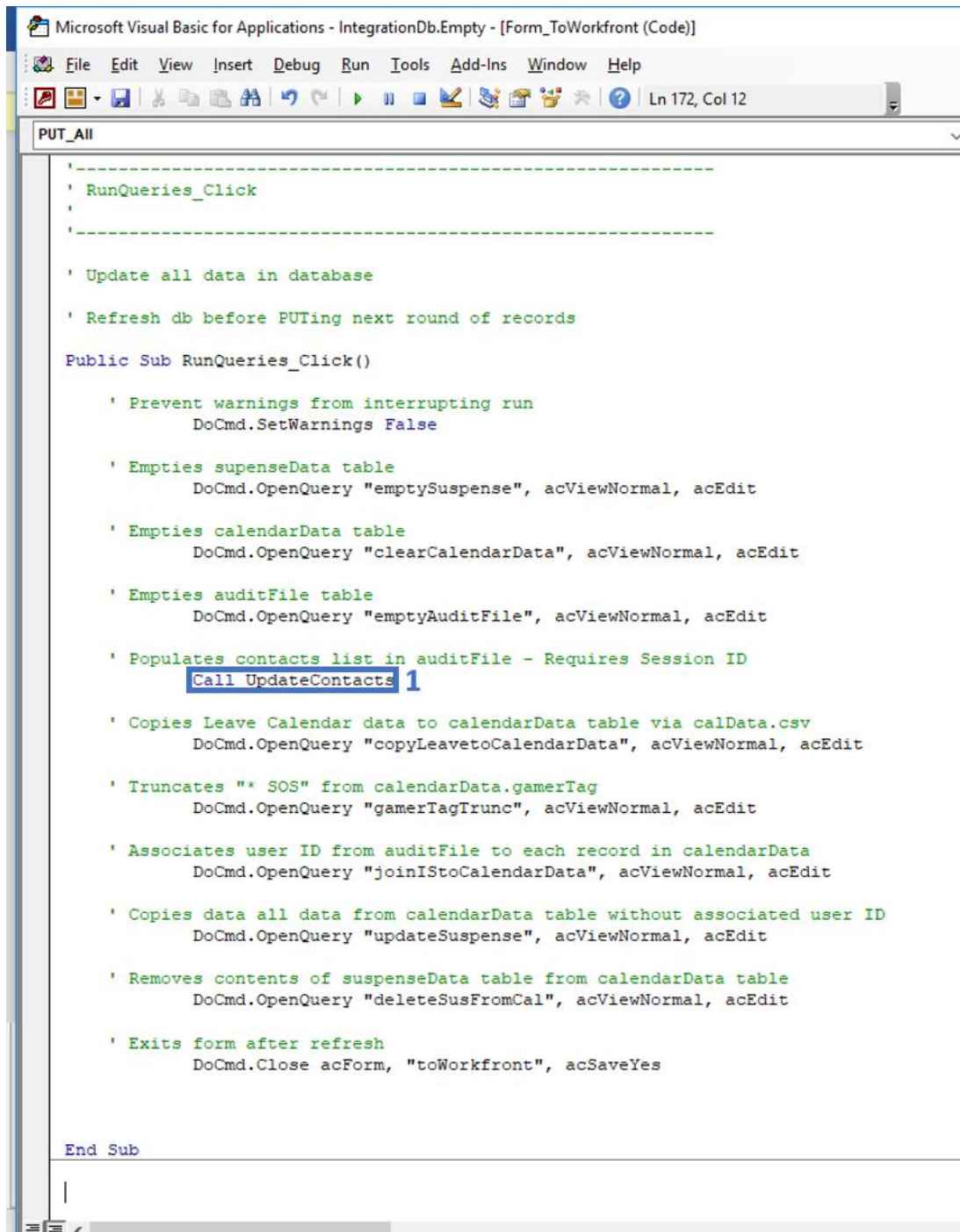


**Figure 6**

To access the Setup menu, select “Setup” from the navigation bar at the top of the page from anywhere in Workfront, **Figure 6.1**. Then, select “System” **Figure 6.2** > “Customer Info” **Figure 6.3**, to access the System Customer Info page. In the API Key Settings, **Figure 6.4**, you can view all active API Keys, create API Keys, and adjust other API Key settings. This key will serve as the password associated with your account to allow the **ToWorkfront.LoginButton\_Click** sub to write to your users’ Time-Off Calendars.

## STEP 2 REFRESH DB

After logging in, the next database function is the **ToWorkfront.RunQueries\_Click** sub which refreshes all table data:



```
Microsoft Visual Basic for Applications - IntegrationDb.Empty - [Form_ToWorkfront (Code)]
File Edit View Insert Debug Run Tools Add-Ins Window Help
Ln 172, Col 12
PUT_All
'-----
' RunQueries_Click
'-----
' Update all data in database
' Refresh db before PUTing next round of records
Public Sub RunQueries_Click()
    ' Prevent warnings from interrupting run
    DoCmd.SetWarnings False

    ' Empties suspenseData table
    DoCmd.OpenQuery "emptySuspense", acViewNormal, acEdit

    ' Empties calendarData table
    DoCmd.OpenQuery "clearCalendarData", acViewNormal, acEdit

    ' Empties auditFile table
    DoCmd.OpenQuery "emptyAuditFile", acViewNormal, acEdit

    ' Populates contacts list in auditFile - Requires Session ID
    Call UpdateContacts 1

    ' Copies Leave Calendar data to calendarData table via calData.csv
    DoCmd.OpenQuery "copyLeavetoCalendarData", acViewNormal, acEdit

    ' Truncates "*" SOS" from calendarData.gamerTag
    DoCmd.OpenQuery "gamerTagTrunc", acViewNormal, acEdit

    ' Associates user ID from auditFile to each record in calendarData
    DoCmd.OpenQuery "joinISStoCalendarData", acViewNormal, acEdit

    ' Copies data all data from calendarData table without associated user ID
    DoCmd.OpenQuery "updateSuspense", acViewNormal, acEdit

    ' Removes contents of suspenseData table from calendarData table
    DoCmd.OpenQuery "deleteSusFromCal", acViewNormal, acEdit

    ' Exits form after refresh
    DoCmd.Close acForm, "toWorkfront", acSaveYes

End Sub
```

Figure 7

The DoCmd.OpenQuery methods each call one of the database queries that update table data based on the Leave\_Calendar linked table. These should not require any modification after calData.csv has been successfully linked to the Leave\_Calendar table; however, in **Figure 7.1**, the **ToWorkfront.Run\_Queries\_Click** sub calls the **SupportModule.UpdateContacts** function which updates the contacts list in the auditFile table based upon contacts data from Workfront and will require adjustment whenever changes to the active Workfront account are made. To make these changes, navigate to **SupportModule.UpdateContacts**:

```

' UpdateContacts
'
' -----
' Automatically populates auditFile contacts from Workfront user info, only includes active users

Public Function UpdateContacts()

    Dim GET_String As String
    Dim GET_URL As String
    Dim SessionID As String
    Dim objRequest As New MSXML2.XMLHTTP60

    GET_URL = "https://spsprojects.preview.workfront.com/attask/api/v9.0" 1

    SessionID = Forms.toWorkfront.SessionID

    GET_String = GET_URL & "/user/search?" & "sessionID=" & SessionID

    objRequest.Open "GET", GET_String, False
    objRequest.send
    Forms.toWorkfront.displayBox = objRequest.responseText

    '$$LIMIT adjusts maximum number of users, default is 100

    GET_String = GET_URL & "/user/search?$$LIMIT=1000" & "sessionID=" & SessionID 2

    objRequest.Open "GET", GET_String, False
    objRequest.send
    Forms.toWorkfront.displayBox = objRequest.responseText

    ' Error check

    Dim checker As Integer
    checker = 0
    checker = InStr(Forms.toWorkfront.displayBox, "error")
    If (checker <> 0) Then
        MsgBox Forms.toWorkfront.displayBox
        Exit Function
    End If

    ' Response string is organized into auditFile

    Call Organizer 3

    ' Rinse and repeat for deactivated users

    GET_String = GET_URL & "/user/search?filters={" & dQ & "isActive" & dQ & " & " & dQ & "false" & dQ & "}&sessionID=" & SessionID
    objRequest.Open "GET", GET_String, False
    objRequest.send
    Forms.toWorkfront.displayBox = objRequest.responseText

    ' Error check

    checker = 0
    checker = InStr(Forms.toWorkfront.displayBox, "error")
    If (checker <> 0) Then
        MsgBox Forms.toWorkfront.displayBox
        Exit Function
    End If

    ' Response string is organized into auditFile

    Call Organizer

End Function

```

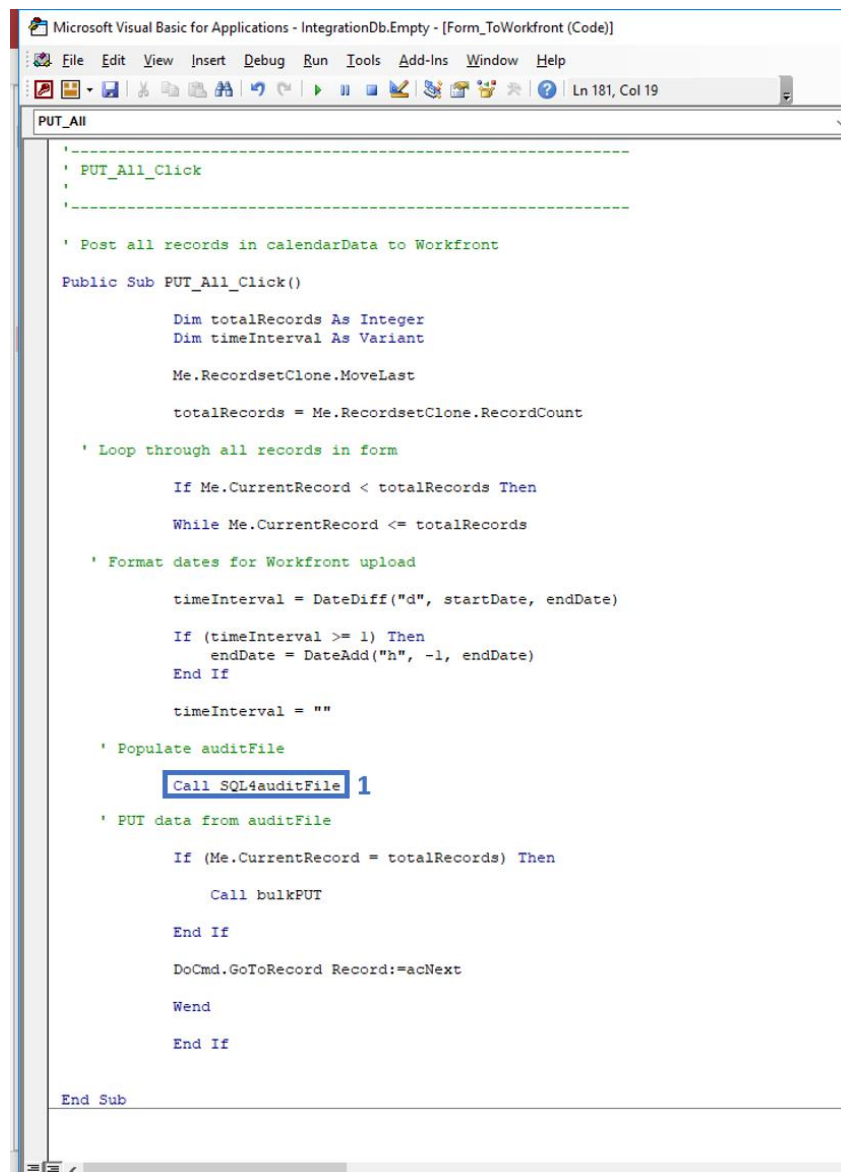
**Figure 8**

**ToWorkfront.UpdateContacts** uses the Session ID acquired from **ToWorkfront.LoginButton\_Click** to GET all active Workfront users from Workfront. The GET\_URL, **Figure 8.1**, will need to be updated if the target Workfront site is changed. The “\$\$LIMIT” function, **Figure 8.2**, can be used to set the maximum number of users that can be generated from Workfront, the current limit is 1000. After the GET, the sub organizes the associated PUT\_String and response RESVT ID into the auditFile table by calling the **ToWorkfront.Organizer** sub, **Figure 8.3**.



### STEP 3 PUT ALL

The final step of the Access process is to PUT all Leave Data to Workfront. To understand this process, begin by navigating to **ToWorkfront.PUT\_All\_Click**:



```
Microsoft Visual Basic for Applications - IntegrationDb.Empty - [Form_ToWorkfront (Code)]
File Edit View Insert Debug Run Tools Add-Ins Window Help
Ln 181, Col 19

PUT_All

' PUT_All_Click
'
'-----
' Post all records in calendarData to Workfront

Public Sub PUT_All_Click()

    Dim totalRecords As Integer
    Dim timeInterval As Variant

    Me.RecordsetClone.MoveLast

    totalRecords = Me.RecordsetClone.RecordCount

    ' Loop through all records in form

    If Me.CurrentRecord < totalRecords Then
        While Me.CurrentRecord <= totalRecords

            ' Format dates for Workfront upload

            timeInterval = DateDiff("d", startDate, endDate)

            If (timeInterval >= 1) Then
                endDate = DateAdd("h", -1, endDate)
            End If

            timeInterval = ""

            ' Populate auditFile

            Call SQL4auditFile 1

            ' PUT data from auditFile

            If (Me.CurrentRecord = totalRecords) Then

                Call bulkPUT

            End If

            DoCmd.GoToRecord Record:=acNext

        Wend

    End If

End Sub
```

**Figure 9**

The **ToWorkfront.PUT\_All\_Click** sub navigates through each record in calendarData and, as it runs through each record, calls the **ToWorkfront.SQL4auditFile** sub, **Figure 9.1**, to add it to the auditFile for upload. Finally, once the last record has been reached, **ToWorkfront.bulkPUT**, **Figure 9.2**, is called to PUT all of the data compiled in the auditFile table to Workfront. The **ToWorkfront.SQL4auditFile** is a data organizing sub, using built in SQL, similar to the **ToWorkfront.RunQueries\_Click** sub, and should not need to be updated. However, the **ToWorkfront.bulkPUT** will interface with Workfront and therefore, will require adjustment to reflect the active Workfront account:



```

Microsoft Visual Basic for Applications - IntegrationDb.Empty - [Form_ToWorkfront (Code)]
File Edit View Insert Debug Run Tools Add-Ins Window Help
Ln 331, Col 1
(General) bulkPUT

'-----
' bulkPUT()
'-----

' PUT all Leave data compiled in auditFile to Workfront

Public Sub bulkPUT()

    Dim auditFile As DAO.Recordset
    Dim dbsLeave As DAO.Database
    Dim Bulk_PUT As String
    Dim PUT_URL As String
    Dim objRequest As New MSXML2.XMLHTTP60
    Dim PUT_Response As String
    Dim arrResponse() As String
    Dim count As Integer
    Dim checker As Integer
    Dim dQ As String
    dQ = ""

    Set dbsLeave = CurrentDb
    Set auditFile = dbsLeave.OpenRecordset("SELECT * FROM auditFile")

    PUT_URL = "https://sosprojects.preview.workfront.com/attask/api/v9.0" 1

    'Check to see if the recordset actually contains rows
    If Not (auditFile.EOF And auditFile.BOF) Then
        auditFile.MoveFirst
    End If

    Do Until auditFile.EOF = True
        ' Bulk_PUT = ""

        'Save fully executed POST_String into Bulk_PUT variable
        ' Bulk_PUT = "" & auditFile!POST_String

        If (auditFile!POST_String <> "") Then

            auditFile.Edit
            auditFile("PUT_String") = PUT_URL & "/user/" & auditFile("User ID") & "?updates={reservedTimes:[" &
            auditFile.Update

            objRequest.Open "PUT", auditFile("PUT_String"), False
            objRequest.send
            PUT_Response = objRequest.responseText

        ' Error check

        'MsgBox PUT_Response

        checker = InStr(PUT_Response, "error")
        If (checker <> 0) Then
            MsgBox PUT_Response
        End If
        End If

        ' Loop to dynamically locate RESVT ID from PUT response

        arrResponse = Split(PUT_Response, dQ)
        count = 0

        Do Until arrResponse(count) = "ID"
            count = count + 1
        Loop

        count = count + 2

        auditFile.Edit
        auditFile("RESVT ID") = arrResponse(count)
        auditFile.Update

        End If

        auditFile.MoveNext

    Loop

    auditFile.Close
    dbsLeave.Close

End Sub

```

**Figure 10**

In the **ToWorkfront.bulkPUT** sub, the PUT\_URL, **Figure 10.1**, will need to be updated to reflect any change to the target Workfront site.

## 4 TASK SCHEDULER

### 1. INTRODUCTION TO TASK SCHEDULER

The Task Scheduler is a component of Windows that automatically performs routine tasks. The Task Scheduler can open applications and execute simple actions automatically as scheduled by following a series of rigorously defined instructions. First, the Task Scheduler is triggered when a specific condition is met. It will then follow the recipe laid out below “Actions” to execute the required task. It can perform also perform the task repeatedly after it is triggered.

For this project, the Task Manager needs to perform two separate actions, the highest imperative when setting up these actions is to consider how the Task Manager’s literal interpretation of your instructions will affect its interactions with the two programs it needs to call. For example, if the Task Manager attempts to open at a scheduled time, but MS Access or Powershell is opened by the user at the same time, then the task may fail. This happens because the user has priority access and the Task Scheduler does not know to wait until the application becomes available. Similarly, if the Access action is scheduled to run too closely to the Powershell action, then the Access task will fail because Powershell is given priority over the .csv file that both programs require. If Powershell is interrupted then it also may dump an empty .csv file or simply delete the existing .csv rather than creating a fully realized copy of the Leave Calendar data.

### 2. POWERSHELL SCRIPT

To access the Task Scheduler, type “task scheduler” into the global navigation bar and open Windows Task Scheduler:

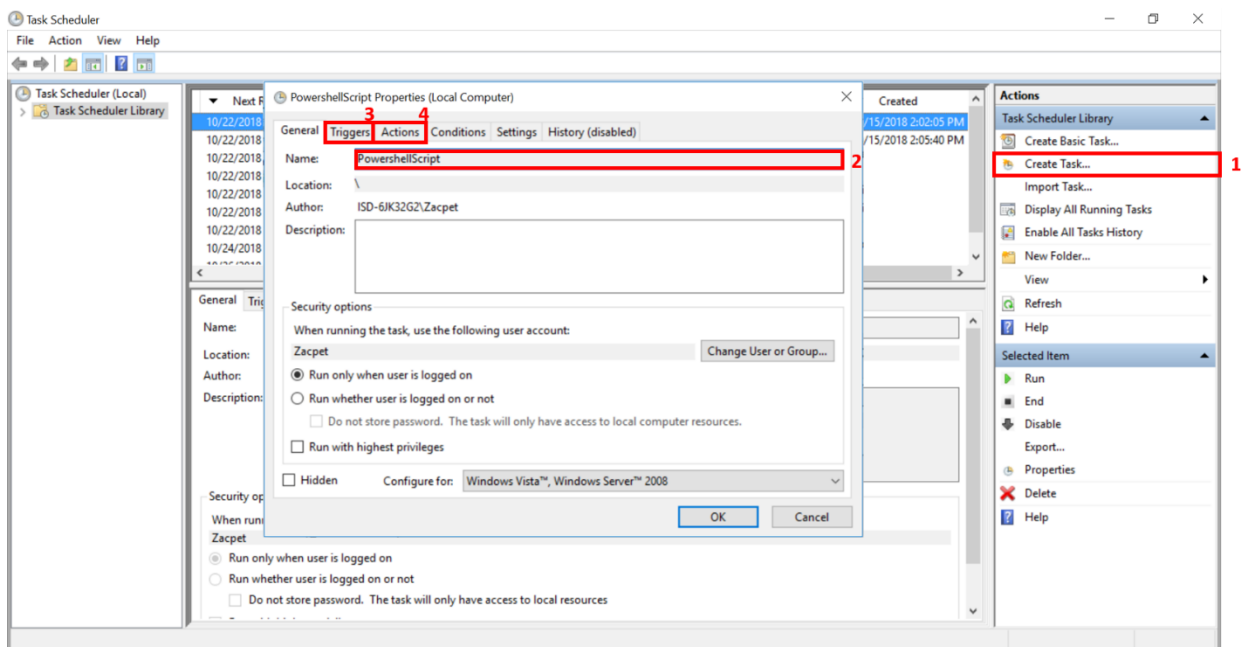
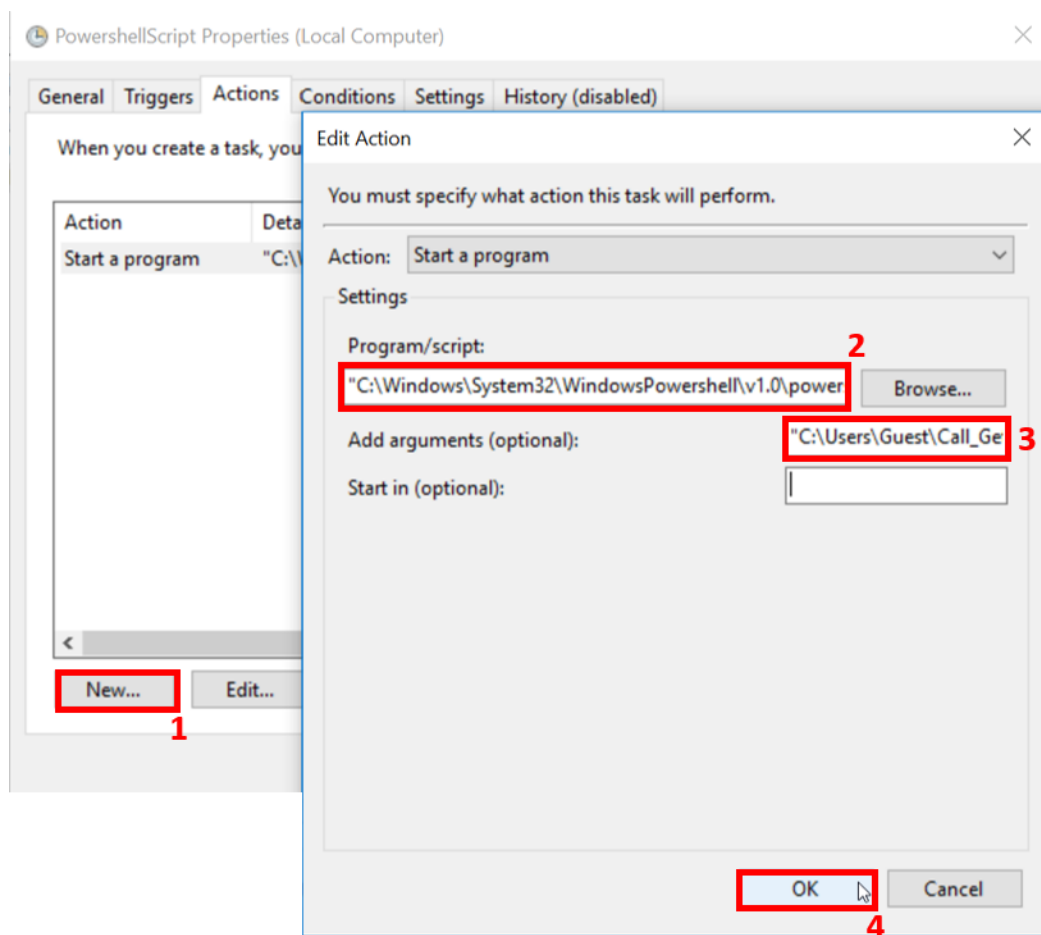


Figure 11

In Task Scheduler, create a new task by selecting the menu option denoted “Create Task,” **Figure 11.1**. Then, a new dialog box will open. In the dialog box, name the first task “PowershellScript” or something similar in the “Name” text box, **Figure 11.2**. Do not change any of the other options from the defaults, especially not “Run whether user is logged on or not.” Selecting this option will cause the task to fail because it will not load the profile you made in **Section 2**. Next, select “Triggers,” **Figure 11.3**, and set any time to trigger the first run. You can also choose how often you want this to manually repeat. The regularity in which the Task Scheduler can run the two tasks in coordination will vary by PC. In my experience, setting this task to run on the hour and every thirty minutes afterwards has run consistently and without error. Finally, select “Actions,” **Figure 11.4** to setup the actions that the task will run after it is triggered:



**Figure 12**

In the Actions tab, select “New...” **Figure 12.1** and an additional dialog box will open. In the new dialog box in the text box titled “Program/script:” **Figure 12.2**, type in:

`"C:\Windows\System32\WindowsPowerShell\v1.0\powershell.exe"`

Including the quotation marks, or whatever the path is to your powershell.exe file found in **Section 2.1**.

Next, in the text box denoted “Add arguments (optional)” type in:

"C:\Users\Public\Call\_Get-OutlookCalendar.ps1"

Including the quotation marks, or whatever the path and name is of your .ps1 call file created in **Section 2.4**. Finally click “OK,” **Figure 12.4**, and then click “OK again, once you are certain all of the information is correct and that you have included a trigger in the near future.

Boiler Plate
PowershellScript "C:\Windows\System32\WindowsPowerShell\v1.0\powershell.exe" "C:\Users\Public\Call_Get-OutlookCalendar.ps1"

### 3. ACCESS DB

To setup Access to run in the Task Scheduler, repeat the steps from **Section 4.2** to create a new task called “AccessDb” or something similar. Just like in the last step, do not change any of the default settings, especially not “Run whether user is logged on or not” because MS Access has a hard limit when interacting with Task Manager causing to be unable to run in Task Manager if the user is not logged in. Next, be sure to include a trigger in the near future, I recommend setting this task to run at the fifteen minute mark and every thirty minutes afterwards which allows it to run in conjunction with the Powershell script without overlapping. Finally, in the last dialog box, **Figure 12.2**, input:

"C:\Program Files (x86)\Microsoft Office\Office15\MSACCESS.EXE"

Including the quotation marks and with respect to case, or whatever path leads to your MSACCESS.EXE file located in **Section 3.1**. Next, below the first input, **Figure 12.3**, input:

"C:\Users\Public\IntegrationDb.Empty.accdb" /x runSuperSub

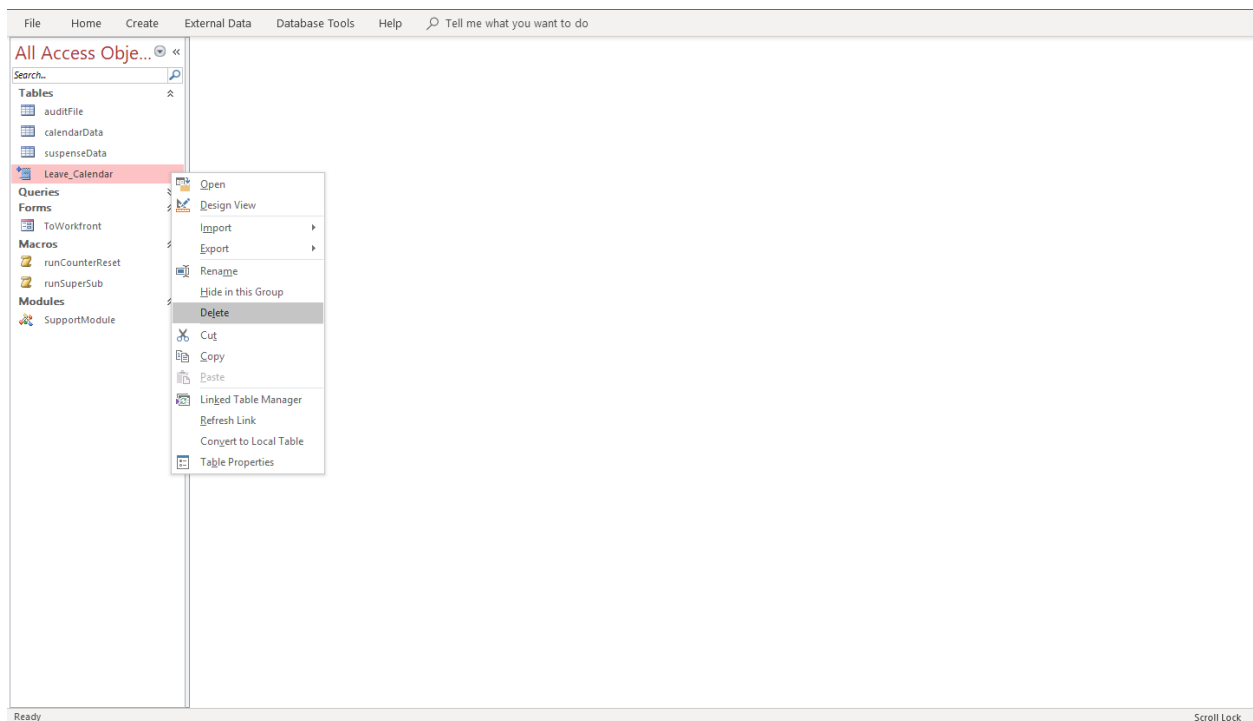
Including quotations, or whatever the path/name of your Access database is. The [/x runSuperSub] argument calls the runSuperSub macro in Access that instigates each of its core functions in sequence.

Boiler Plate
AccessDb "C:\Program Files (x86)\Microsoft Office\Office15\MSACCESS.EXE" "C:\Users\Public\IntegrationDb.Empty.accdb" /x runSuperSub

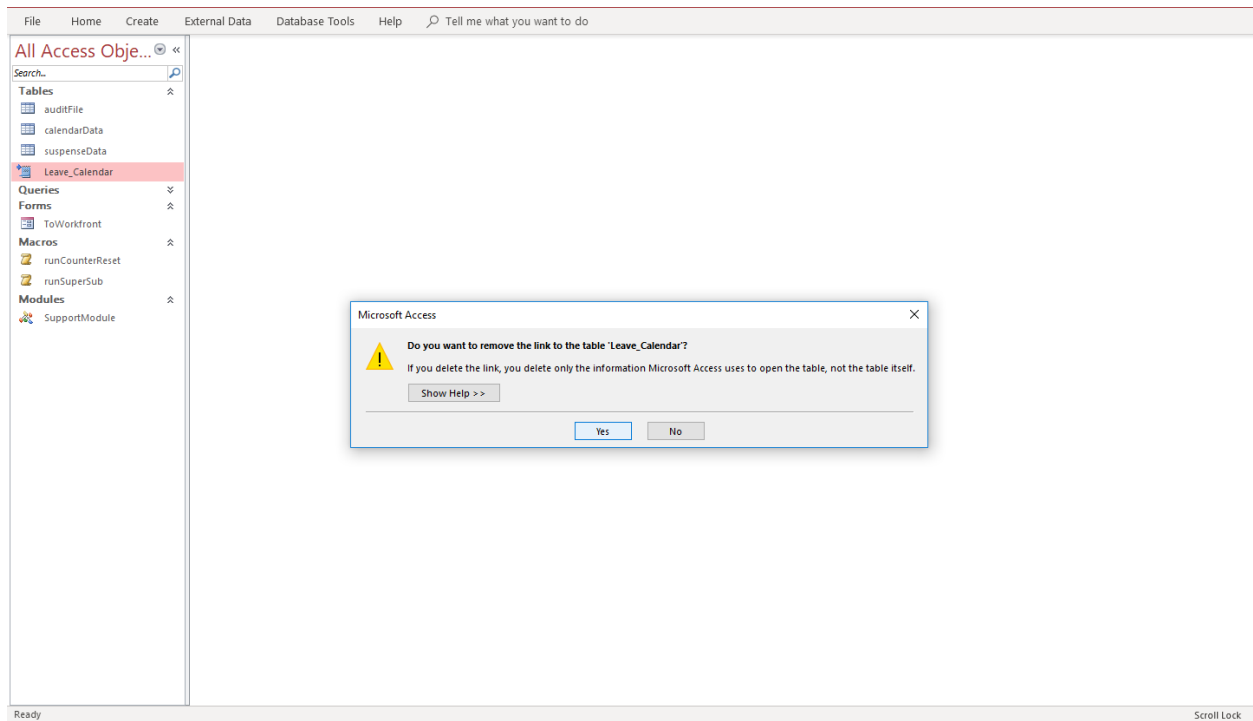
## 5 GENERAL ADJUSTMENTS

### 1. CHANGING PATH TO CALDATA

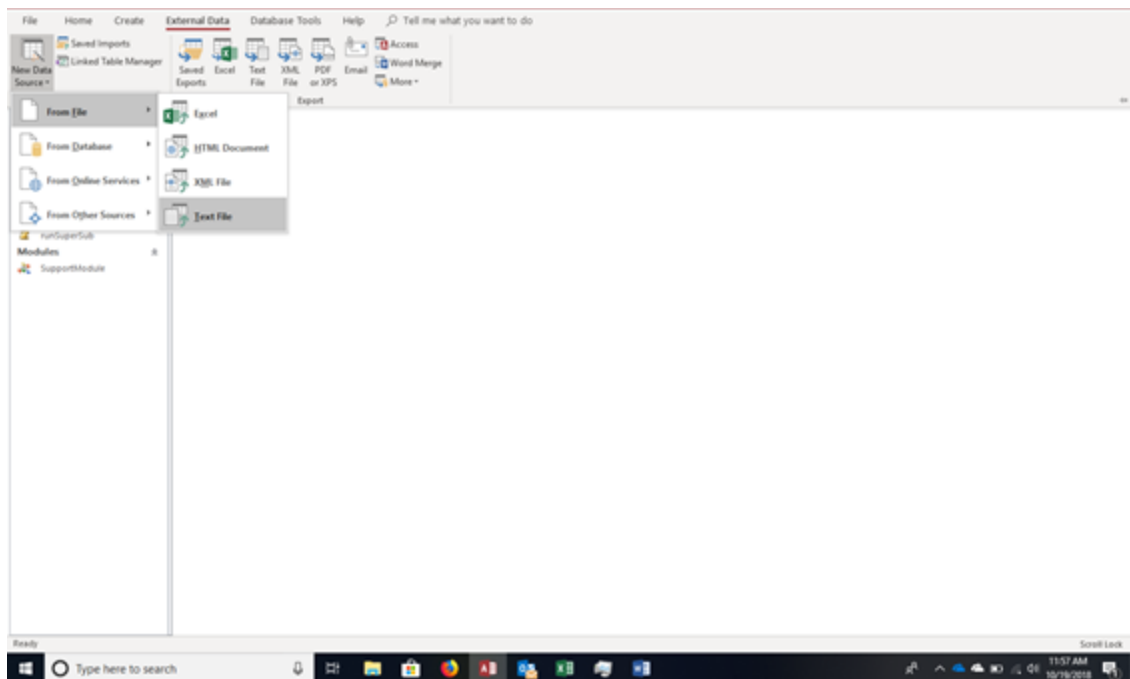
Most of the names and paths in this procedure can be changed at will, as long as they are made consistently across their respective references, but the calData file is special because of its significance as the point of contact between the two main programs in this integration: Powershell and Access. If you are going to change the path or name of the calData.csv file; then, after adding the new path/name to your Powershell Get-OutlookCalendar script, **Section 2.4**, you will need to delete the existing linked Leave\_Calendar table in your Access database and re-create it:



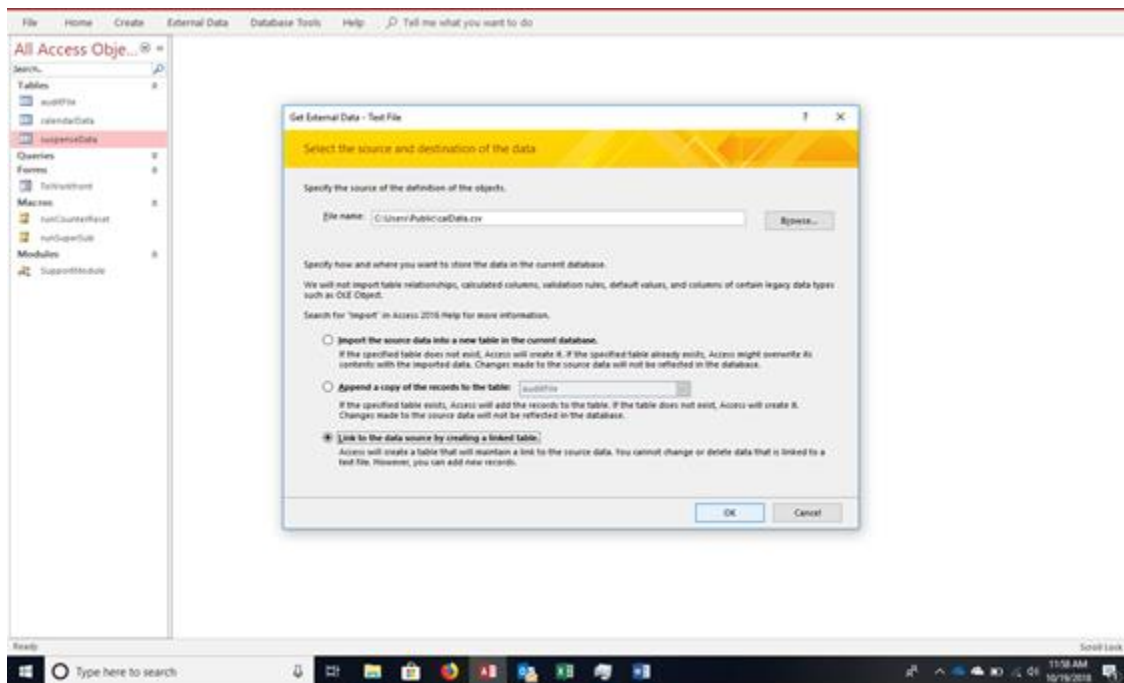
Access does include an option to replace the source file of an existing linked table with the new file location, but I could not get this function to work with the .csv file without corrupting the data so I recommend deleting and re-creating the table all together when changing this critical path.



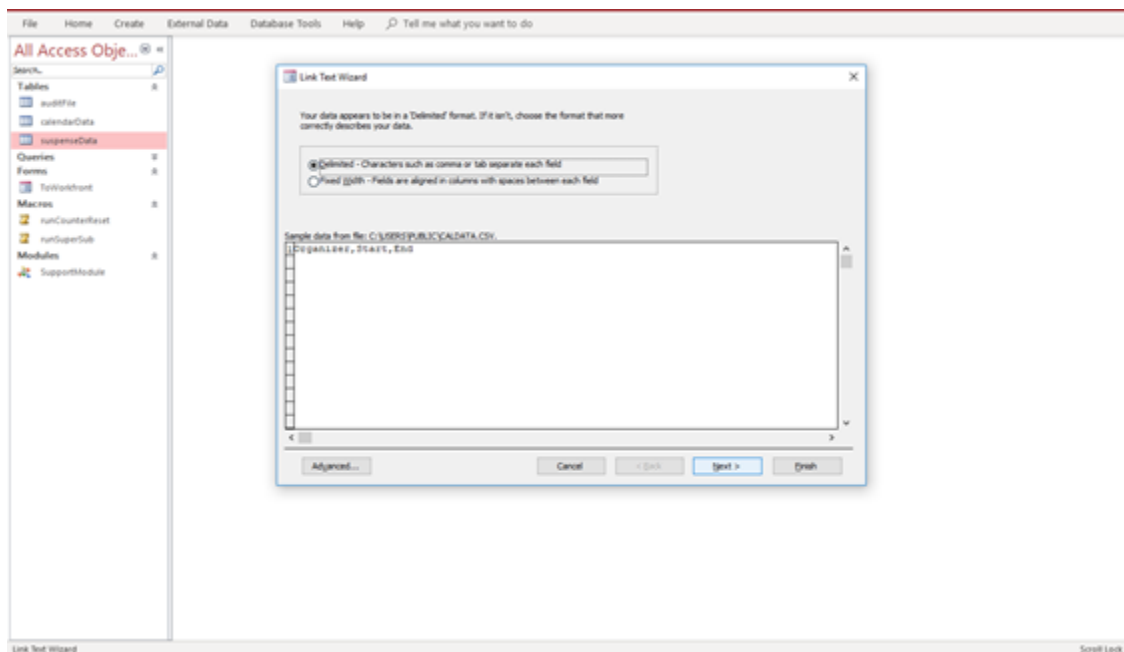
Click through warnings.



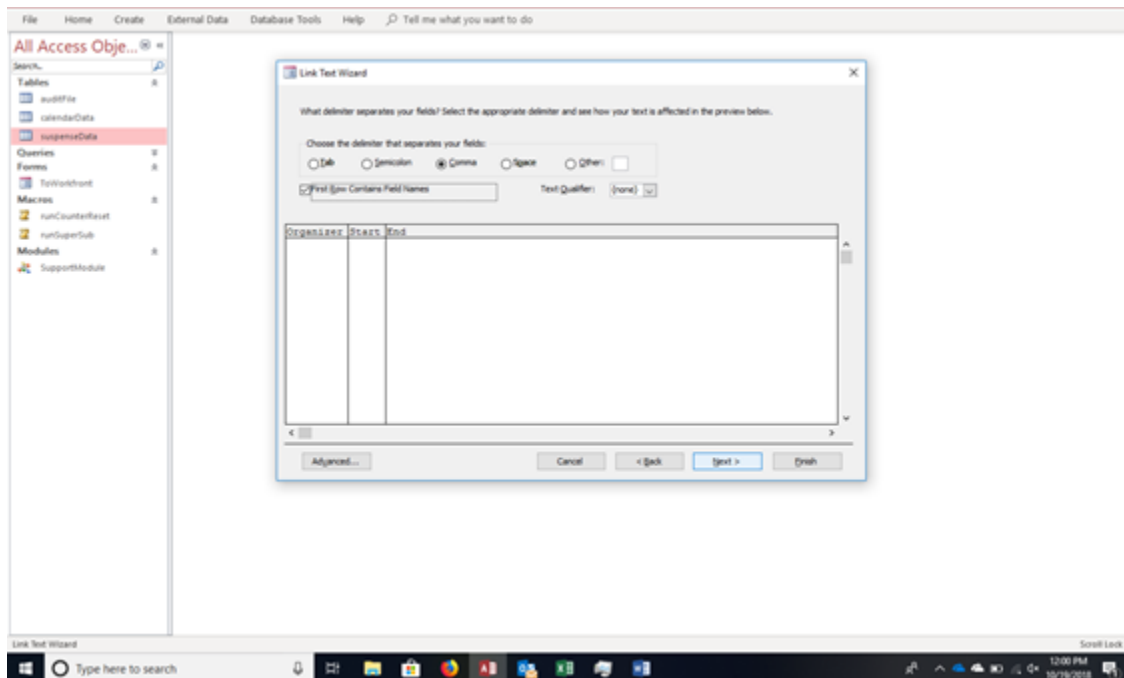
The .csv is a text file, so use New Data Source > From File > Text File to link to it.



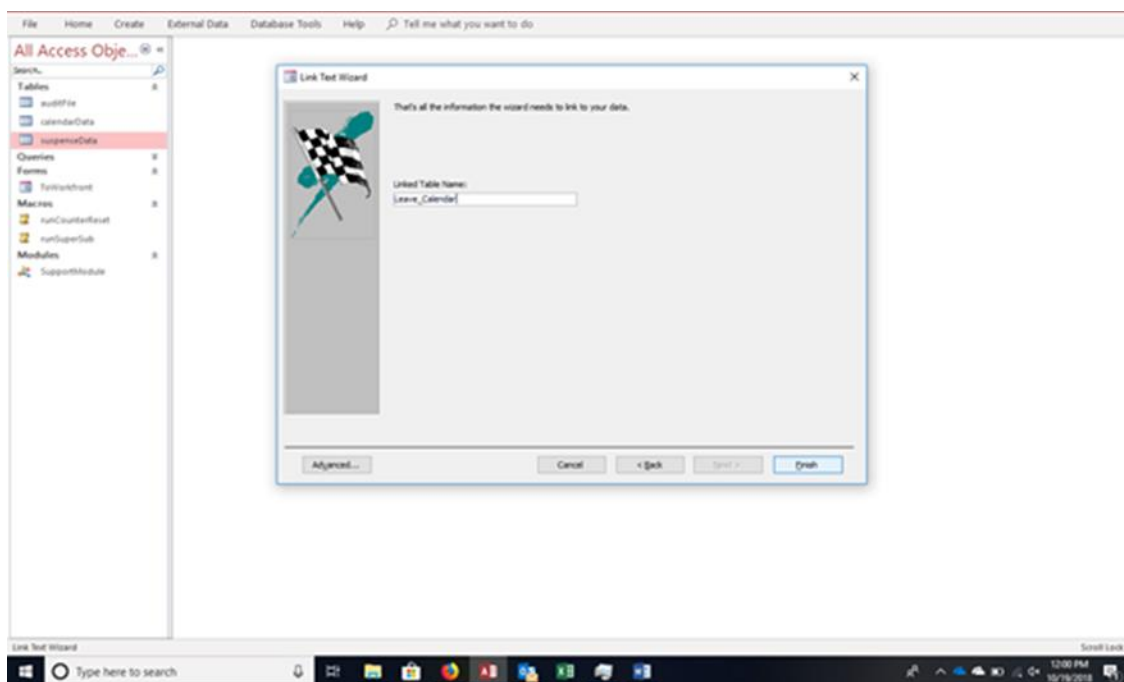
Choose “Link to the data source by creating a linked table” or else it will create a static copy that will not update during subsequent Powershell runs.



A .csv file is delimited by “,”s.



Check “First Row Contains Field Names.”



If you name this table something other than “Leave\_Calendar” then you will need to ensure that Access query references reflect that change.

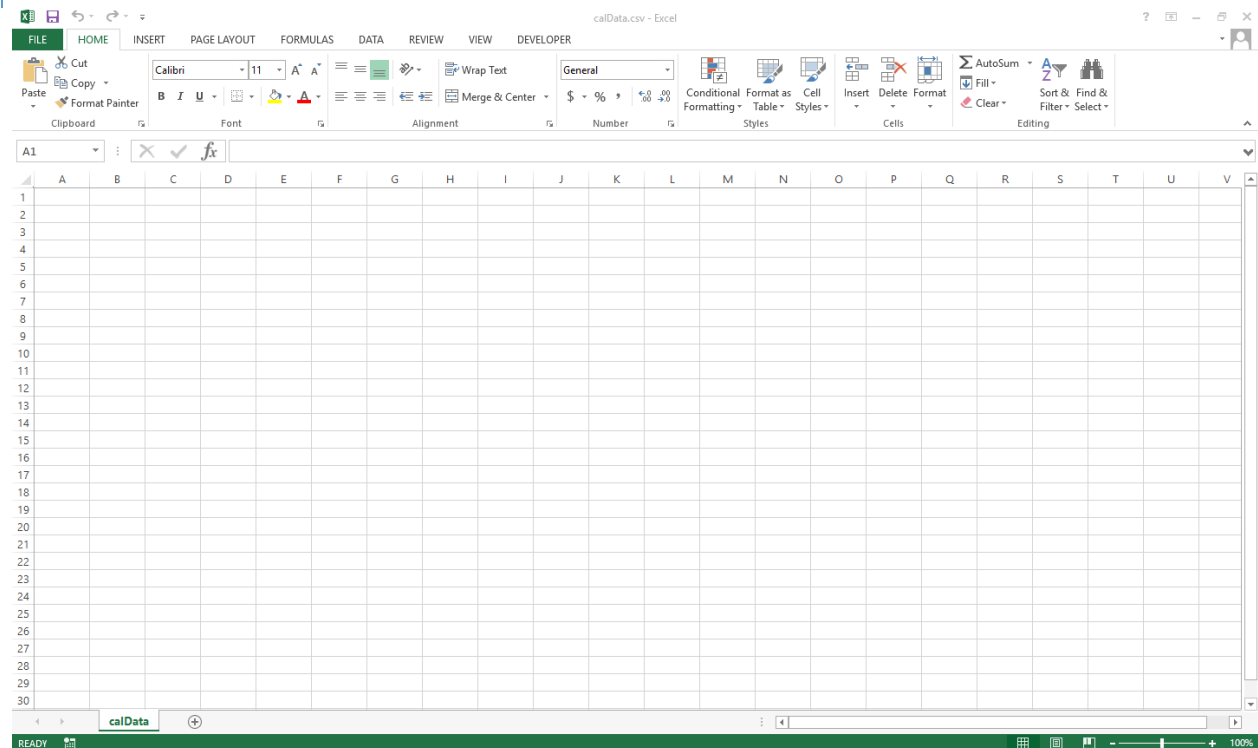


## 6 ERROR HANDLING

Windows Powershell and MS Access each include a number of built-in error display functions. The Access database is also designed to display any errors from Workfront.

### 1. POWERSHELL

#### EMPTY CSV



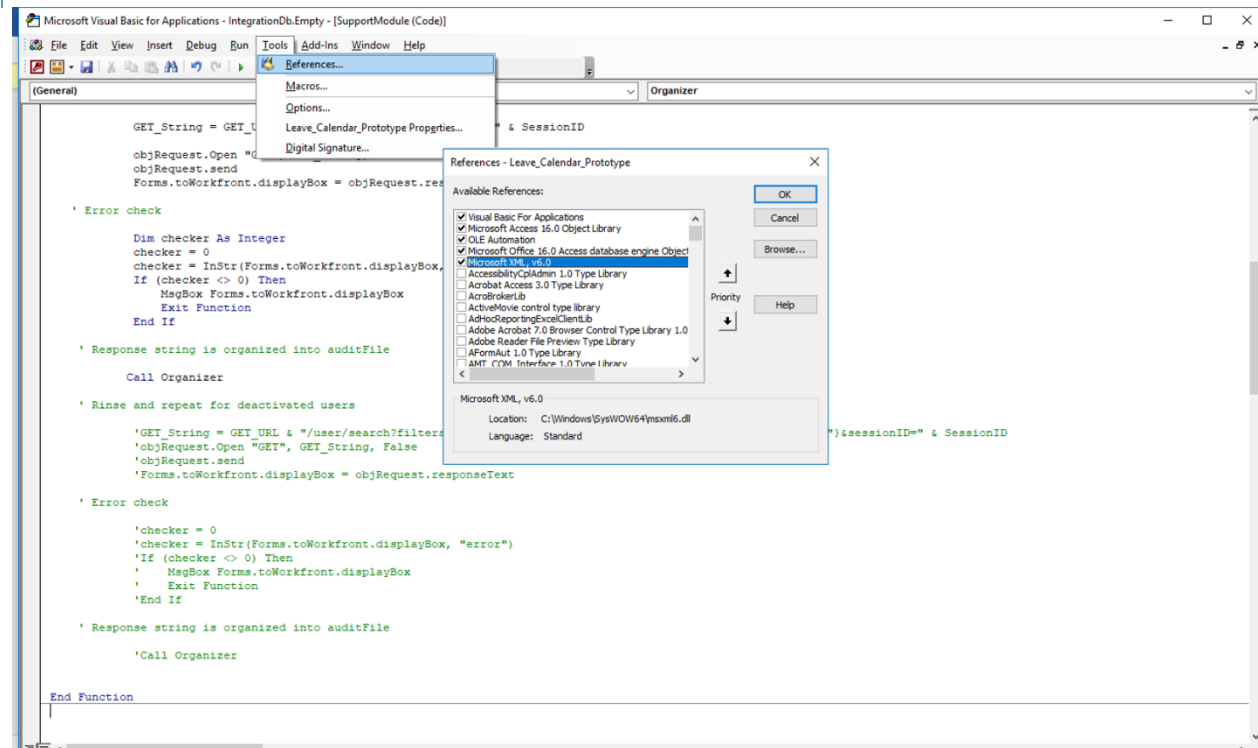
If your calData.csv file turns up empty after previously running correctly, that means the Powershell script was interrupted at some point during its previous run. This may or may not be coupled with an error message from Powershell. Common causes: manual use of Powershell at the same time as a script is trying to run, prematurely closing a running Powershell script, and scheduling MS Access to run too soon after Powershell and it tried to pull data from the .csv file before it was fully populated. The .csv can be repopulated through a successful Powershell run, but the core cause should be identified and addressed or it may happen again. MS Access will not update if the .csv is empty; so, whether or not Access runs again before Powershell is successfully executed, no inaccurate data will be uploaded to Workfront.

## 2. ACCESS

### INVALID USE OF NULL

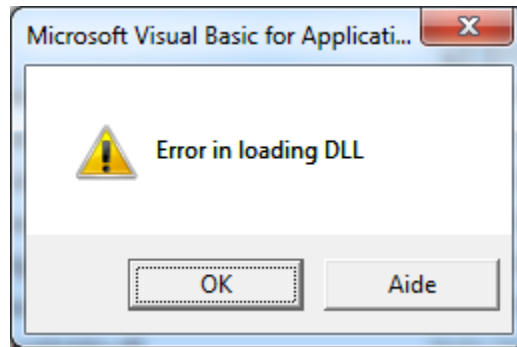
If you receive the error message “Invalid use of null” then the database most likely attempted to draw data from the calData.csv file, but the file does not exist. A non-existent .csv file implies that the Powershell script failed to execute correctly during its last run either due to a mistake during setup or an interruption during a previous run. See **Section 2** and **Section 6.1.1** for additional detail.

### LIBRARY NOT FOUND



If you encounter the error “reference library not found” or else a core procedure within the database’s execution is stopped because one of the functions is “not found”, then there is most likely a reference error. This often happens when switching from one version of Access to another and can be resolved by simply navigating to code behind the ToWorkfront form using the procedure outlined in **Section 3.1** and selecting Tools > References to open the reference library for the database. Within the reference library, uncheck any references denoted “Missing” and press “OK” to resolve this error.

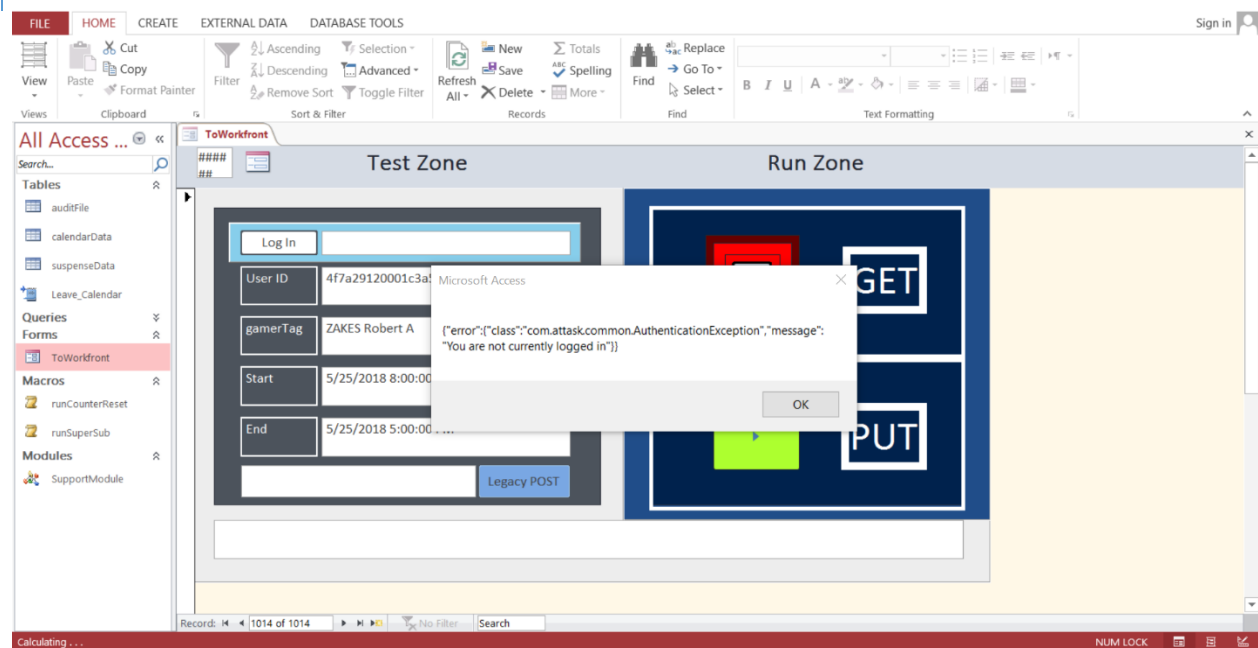
## ERROR LOADING DLL



If you encounter the error message “Error in loading DLL” then your Access .dll file may be corrupted. This is a problem that sometimes develops with older computers over subsequent upgrades to new operating systems that they were never designed to run. This issue can usually be mitigated in the short term by uninstalling and re-installing your database on the machine, but it will occur again and again until you replace the host PC with a newer model or otherwise upgrade the hardware for the existing model.

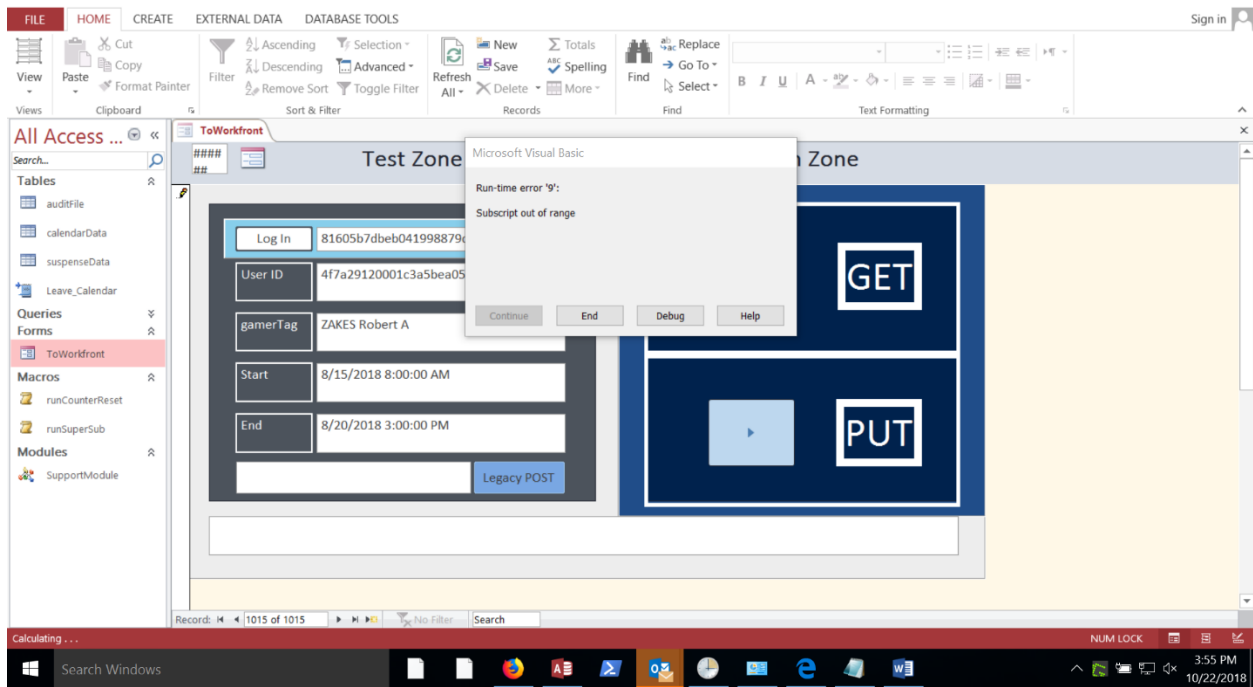
## 3. WORKFRONT

### AUTHENTICATION EXCEPTION



If you receive an authentication exception then you are either attempting to GET or PUT data without being logged in to Workfront. When you are logged in, a Session ID will be displayed next to “Log In” in the ToWorkfront form. If the Session ID is not generated by pressing the Log In button, then follow the **Section 3.2** to update your login credentials.

## SUBSCRIPT OUT OF RANGE



If you receive the error message “Subscript Out of Range,” then, you most likely attempted to PUT all of the data in the database twice in a row without GET-ing new data first to refresh the database. This can be resolved by simply running the GET procedure before running the PUT procedure. Remember to log in before running either the GET procedure or the PUT procedure, **Section 6.3**.

## 4. OUTLOOK

### YOU ARE NOT CURRENTLY SIGNED IN

If the Powershell script is interrupted by Outlook with the error message “You are not currently logged in,” or, while running automatically, the script appears to execute, but does not output anything; then your Outlook account may not be logged in or may have been changed through another machine requiring you to close and reopen the shared Outlook account. You will need to be signed in to the Outlook account associated with your organization’s Leave Calendar to execute the Powershell script. Furthermore, it must be the default account for the Outlook application on your computer, not an account solely accessed via the web.