Background

The implementation of Business Objects (BusObj) represents a significant improvement in query and reporting capability, compared to the query tool that it replaces, but it also means a change in approach when integration with Excel and Access is needed.

The single largest improvement with BusObj, in addition to more advanced querying capability (like unions, sub-queries, and so forth), is that highly complex reports can be produced directly. Historically, users had to rely on Excel, Access, and other tools in order to create all but the simplest reports. With BusObj however, users are encouraged to learn the query and report writing capabilities, and in most cases will find that BusObj will meet most reporting needs, even where complex calculation and formatting requirements exist.

Despite the increase in reporting functionality, there are cases where a BusObj query / report is not the end result of a process, but is rather a data source that starts yet another process. In these situations, Excel (via embedded queries using the former tool) and Access (via pass-through queries directly against the Oracle tables) are often used for these "other processes." In still other cases, custom applications (using Visual Basic, for example) are used. For business and architectural reasons beyond the scope of this document, the embedding / pass-through approaches do not apply with BusObj. This document discusses the recommended approach for using BusObj to serve as that similar data source. While the approach does require a modification to the current process, the conversion is not complex and in most cases will offer increased functionality as well.

High-Level Solution

While many different solutions of various complexity were considered / prototyped, the recommended solution ends up being rather straightforward in approach. Use the strengths and built-in functionalities of the various tools, and if desired, tie them together with automation (VBA procedures):

- Business Objects: Create a document (.rep file) with data providers (queries) as needed. Optionally, report tabs can also be built in the document where helpful. Export the data providers and / or report tabs (several formats are supported).
- Excel: Use the Get External Data functionality to bring the exported data into the spreadsheet.
- Access: Use the Get External Data functionality to bring the exported data into the database.
- Custom applications: Use ADO (ActiveX Data Objects) to read the text files. Please note that use of ADO is beyond the scope of this document, but is briefly mentioned here for completeness. While this technique (ADO) has been used successfully and details are available upon request, the remainder of this document will focus on Excel and Access only.

The following sections will first outline the manual steps to accomplish the integration. A final section will discuss automation options. *DO NOT* be tempted to rush ahead to the automation section, as the majority of the manual steps are still necessary as a foundation for the automation.

Business Objects

There are two options for exporting data from a BusObj document:

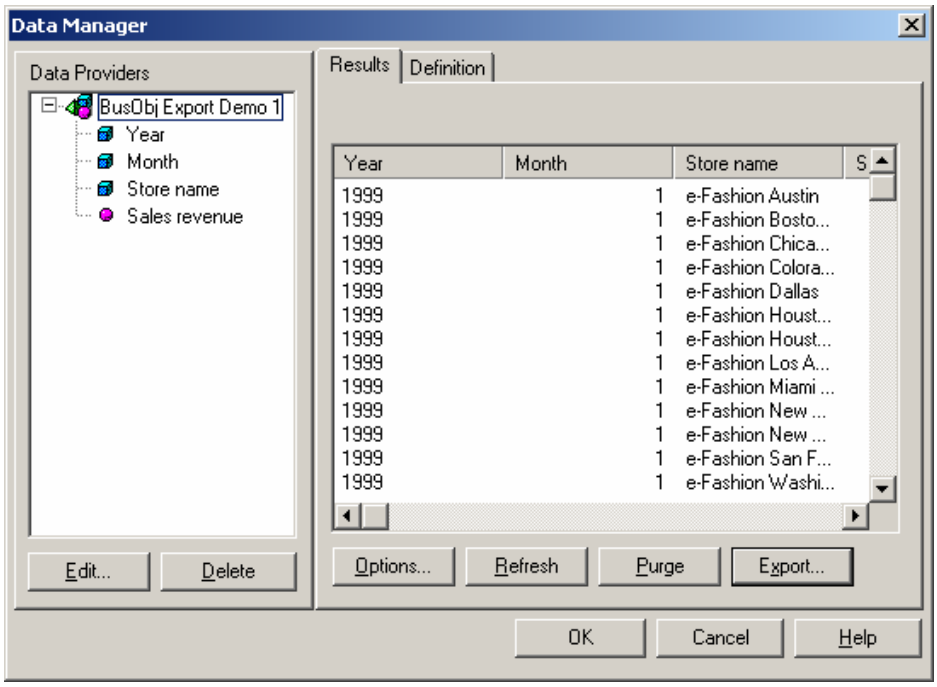
- Export the contents of each data provider.
- Export the contents of each report tab.

Export data provider

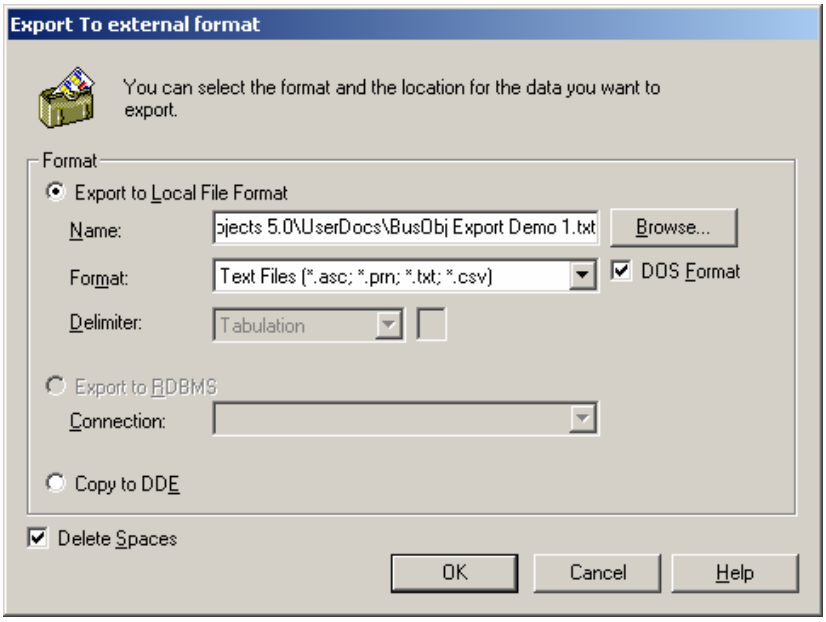
This option is preferred when the raw data meets the need directly. The following formats are supported by Excel and Access, and are used most often when exporting data providers:

- Tab-delimited text file (.txt extension): A tab character separates each field. In most cases this is the preferred format, since it has none of the limitations discussed in the following formats.
- Comma-delimited text file (.csv extension): A comma character separates each field. The risk of this format is that a comma character naturally occurring in the data will erroneously be interpreted as a field separator.
- Space-delimited text file (.prn extension): A space character separates each field. The risk of this format is that a space character naturally occurring in the data will erroneously be interpreted as a field separator.
- Excel 97 file (.xls extension, compatible with 97/2000/2002/2003): Slower to export than text files, and is limited to 256 columns and 65,535 rows of data.
- Excel file (.xls extension, older format for backwards compatibility): Slower to export than text files, and is limited to 256 columns and 16,834 rows of data.
- Dbase file (.dbf extension): Slower to export than text files, and since Dbase field names are only 10 characters long all column names must be unique in the first 10 characters.

Data providers are exported from the Data Manager (Data, View Data... or the View Data icon on the standard toolbar).



Select the desired data provider on the left of the Data Manager screen, and click Export... from the Results tab. The following dialogue box will appear:



Feel free to experiment with the other formats, but for purposes of this document only the tab-delimited (.txt file) will be described. The steps for other formats are similar. From the above dialogue box, use the following steps to export in tab-delimited format:

- In the format drop-down box, be sure Text Files (the default) is chosen.
- Use the Browse... button (or enter manually) to specify a location and name for the file. Be sure to include the .txt extension, since this will ensure the use of the tab character as the delimiter.
- An alternative is to choose All Files as the format, which will allow any file extension to be used, but be sure to specify Tabulation in the Delimiter drop-down box.
- Click OK, and the export will occur.

Repeat the steps for each data provider.

Export Report Tab

This option is preferred when additional calculations, formatting, filtering, cross-tabs, sorting, ranking, summarizing, and so forth are beneficial. Data (in tab-delimited text format, minus formatting like fonts, colors, borders, and so forth) will be exported exactly as displayed on the report. Steps are as follows:

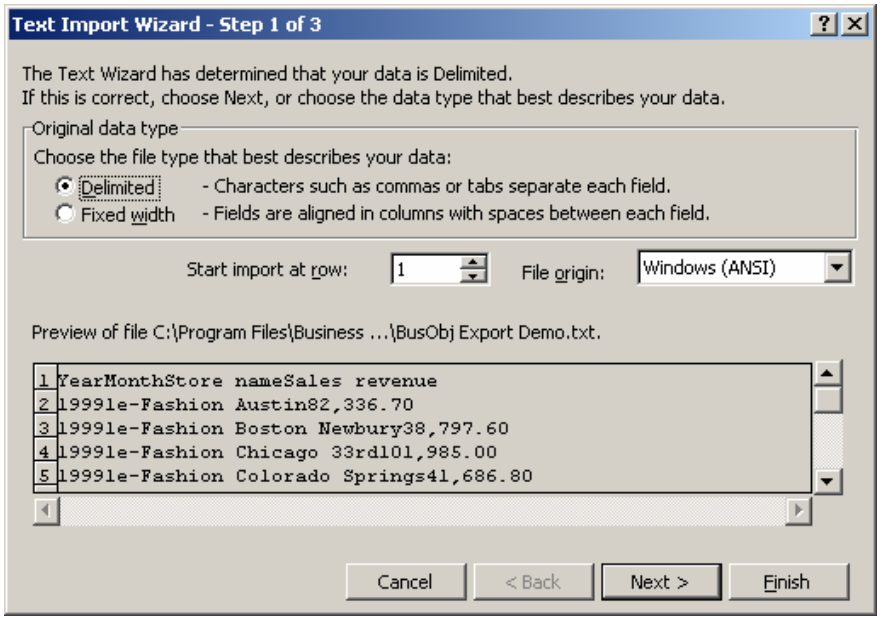
- Activate the desired report tab.
- Click File, Save As....
- Browse to the desired location, and fill in the desired file name.
- Change the Save as type to Text file.
- Click Save, and the tab-delimited file will be exported.

Repeat the steps for each report tab.

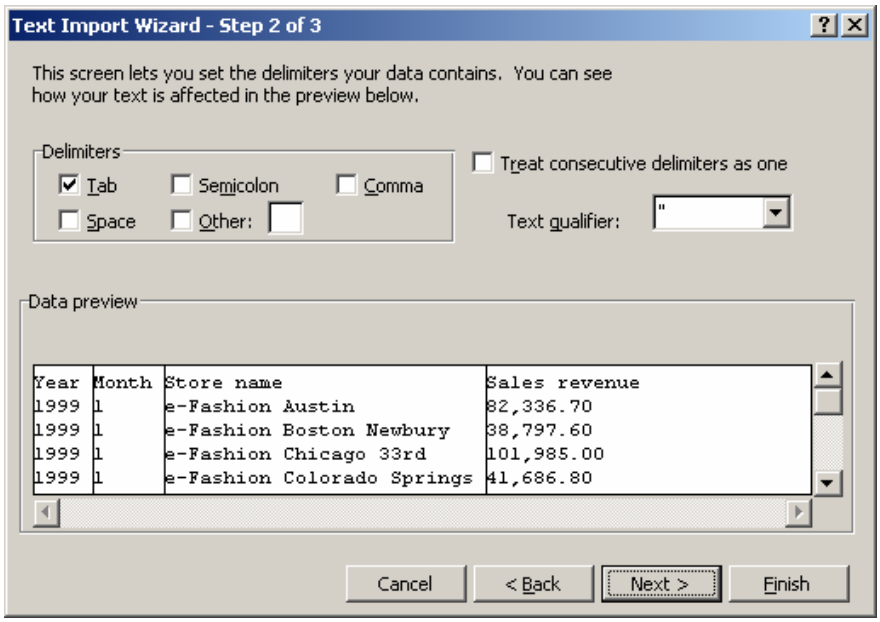
Microsoft Excel

Now that we have created the tab-delimited text file from BusObj, we will use the built-in "Get External Data" functionality of Excel to create a link to that exported data. The steps for building an external link are as follows:

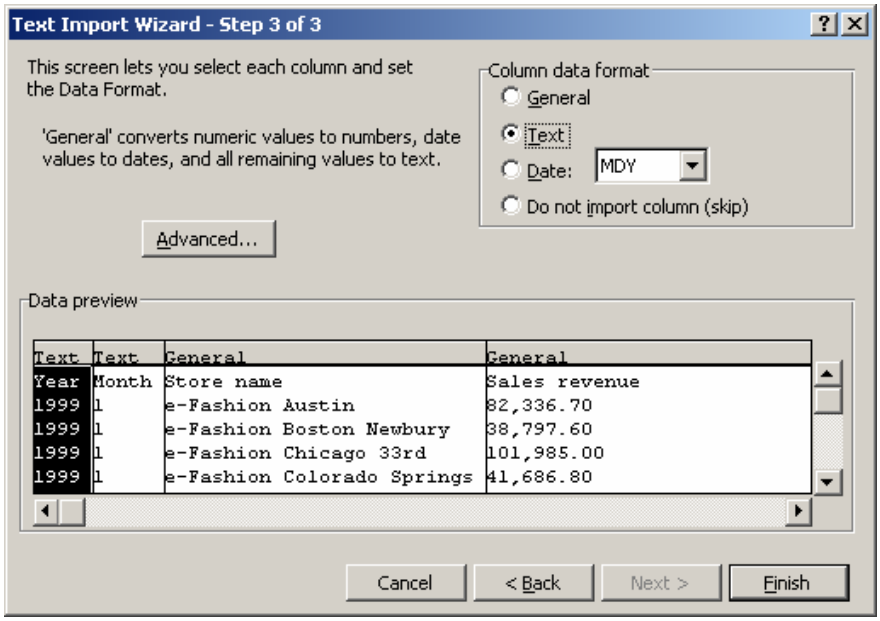
- Office 2000: Data, Get External Data..., Import Text File...
- Office 2002/2003: Data, Import External Data..., Import Data...
- Browse to the appropriate directory, choose the text file, and click Import...
- This will start the Text Import Wizard (Step 1 of 3):



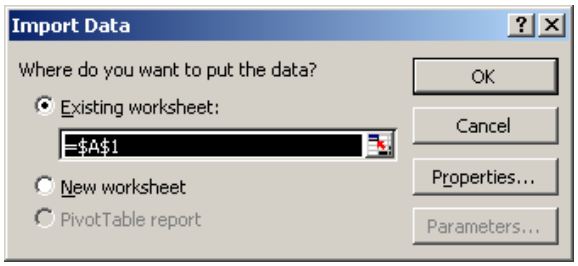
Defaults are probably appropriate. Delimited is the file type. Set the "Start import at row" setting to 2 if the text file contains field headings and you prefer not to have the field headings included in the link. Click Next to advance to Step 2 of 3.



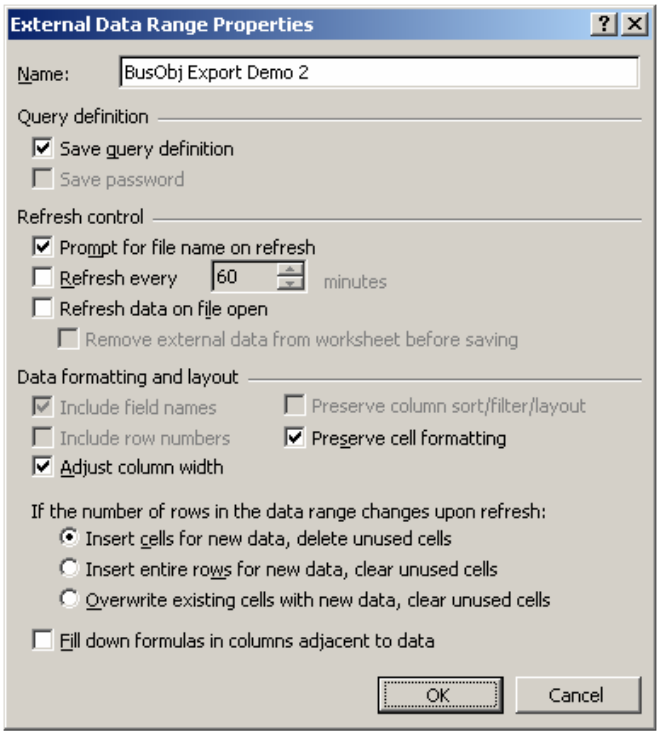
Again, the defaults are appropriate. The BusObj text extract is a tab-delimited file. Click Next to advance to Step 3 of 3.



This screen offers one of the benefits to using this "Get External Data" method. By default, columns are imported with a General format (numeric values convert to numbers, date values convert to dates, all others to text). This is often a problem where columns that contain only digits (like Year in the example above) are erroneously interpreted as numbers and are formatted by Excel with currency signs, commas, decimals, and so forth. These columns can be set explicitly to text in advance by selecting each column, and setting the Column Data Format to text. Click Finish to complete the Text Import Wizard, and continue setting up the Import Data parameters.



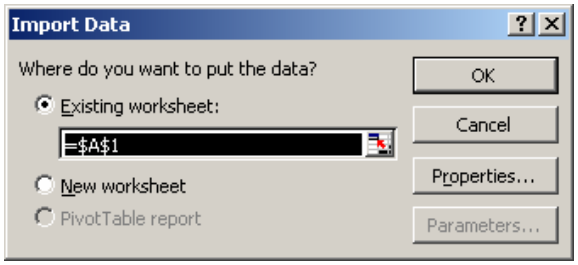
If needed, navigate to the upper left cell where you want the data to be placed in the spreadsheet. Before completing the import, we will want to change / review the Properties, so click the Properties... button.



This screen offers another set of advantages to this "Get External Data" approach. These generic Excel settings make this approach quite similar to (and even more flexible than) the embedded Q+E approach. Let's review each section of the External Data Range Properties:

- Name: By default, the link name is the same as the text file name (without the extension). This name will also be used to automatically assign a dynamically re-sized range name to the resulting data area.
- Query definition: If this is a one-time only import, you can clear the Save query definition checkbox. In most cases you will want to leave this setting checked, so that the settings are retained and future updates will simply require a refresh.
- Refresh control: If future refreshes of this link will use the same file location and name, you will want to deselect the Prompt for file name on refresh checkbox. You may also set the link to automatically be refreshed from the text file each time the Excel file is opened.
- Data formatting and layout:
    - Adjust column width: automatically performs a "best fit" on the columns
    - Preserve cell formatting: any custom formatting that you apply is retained when the link is refreshed
    - Insert cells: if the number of rows changes on refresh, cells below the data range are inserted / deleted as needed; columns to the left or right are unaffected
    - Insert rows: if the number of rows changes on refresh, entire rows are inserted / deleted as necessary, which may impact columns to the left or right of the data area
    - Overwrite: if the number of rows changes on refresh, new rows are overwritten and unused rows are cleared

Click the OK button to save the Properties and return to the Import Data screen.



Click OK to finish the Import process. Repeat for each text file.

Establishing these links is a one-time process. In the future, right-click in the data area, and to ...

- ... refresh the link, choose Refresh Data
- ... change the import text settings, choose Edit Text Import...
- ... change the “embedded” properties, choose Data Range Properties...

Microsoft Access

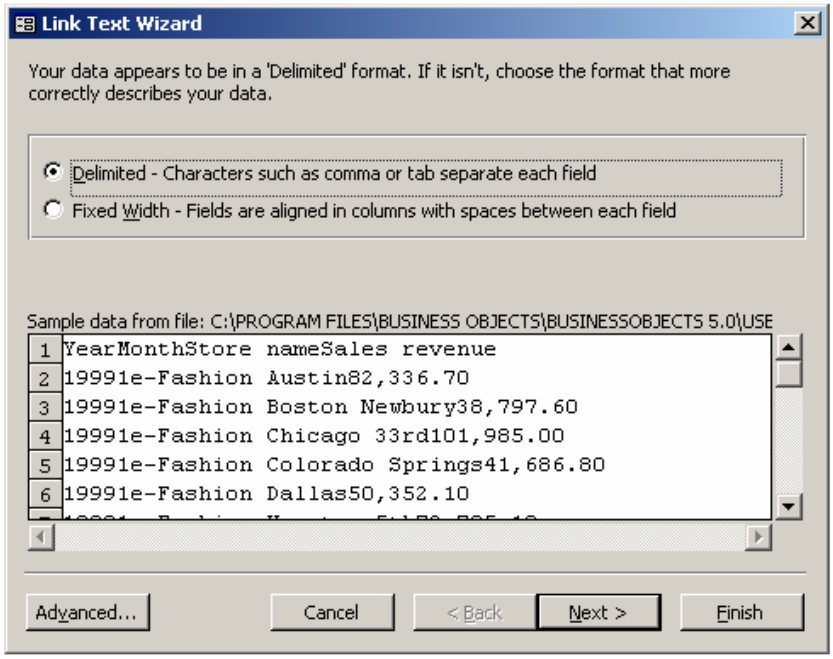
Using the tab-delimited text file created from BusObj, we will take advantage of the built-in "Get External Data" functionality of Access to utilize that exported data. There are two options:

- Link to the external data
- Import the external data

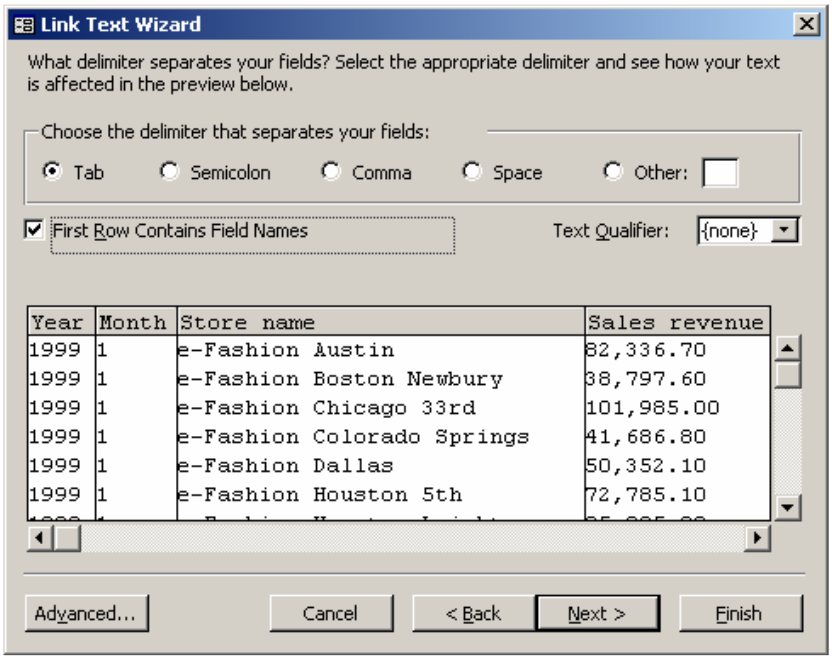
Link to external data

This option is the most similar to the pass-through query approach, in that the data itself remains external to Access. Any data validation, indexing for performance, storage within the database, and so forth are left up to Access to complete. Steps for linking to external data are as follows:

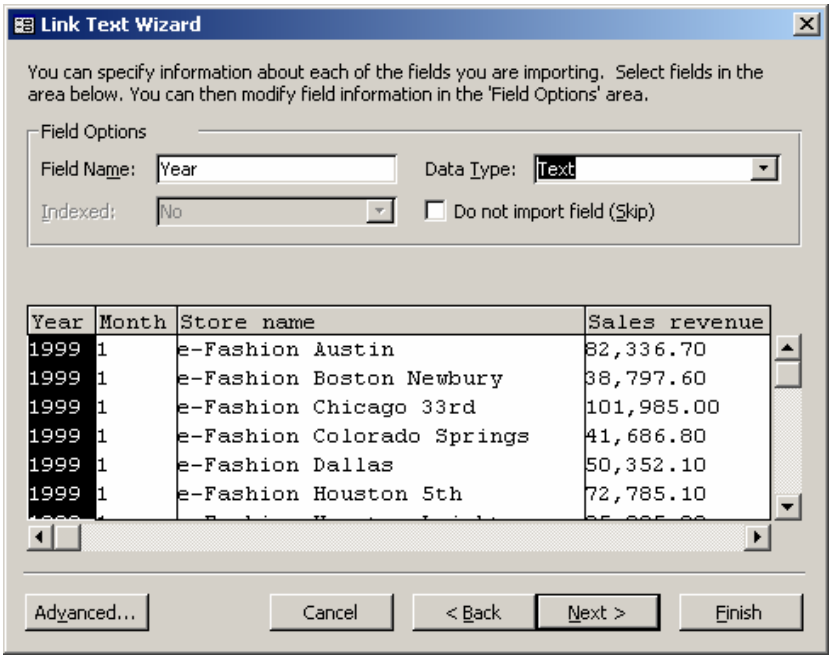
- From your database, choose File, Get External Data, Link Tables...
- In the Files of Type drop-down, choose Text Files
- Browse to the appropriate directory, choose the text file, and click Link...
- This will start the Link Text Wizard:



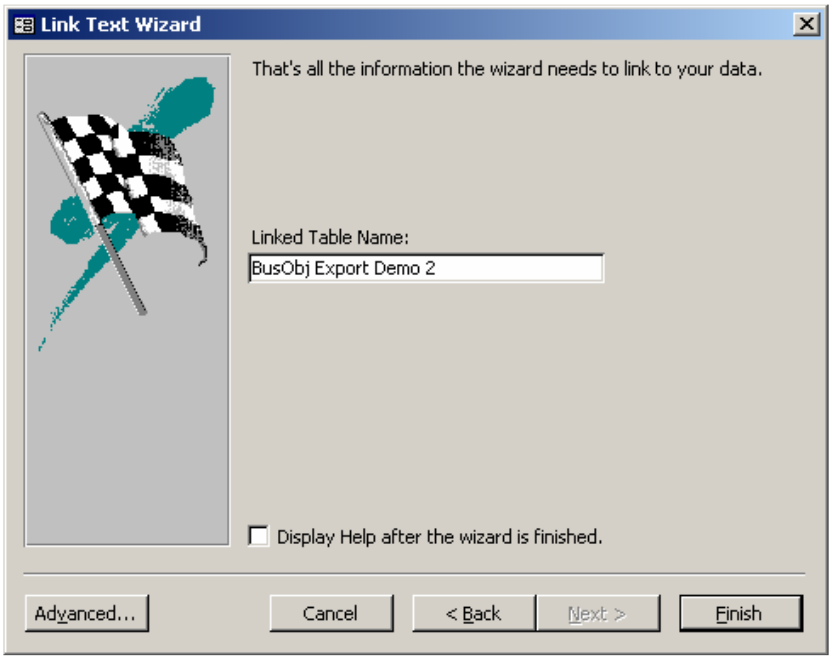
Defaults are probably appropriate. Delimited is the file type. Click Next to continue with the next step in the wizard.



Tab is the appropriate delimiter. If the first row of the text file contains field names, indicate so by selecting the checkbox. Click Next to continue with the next step in the wizard.



By default, Access does its best to interpret the appropriate data types. This is often a problem where columns that contain only digits (like account number in the example above) are erroneously interpreted as numeric fields. These columns can be set explicitly to text by selecting each column, and setting the Data Type to text. You can even set numeric columns to other data types if you like (Currency instead of Double, for example. Click Next to continue with the next step in the wizard.

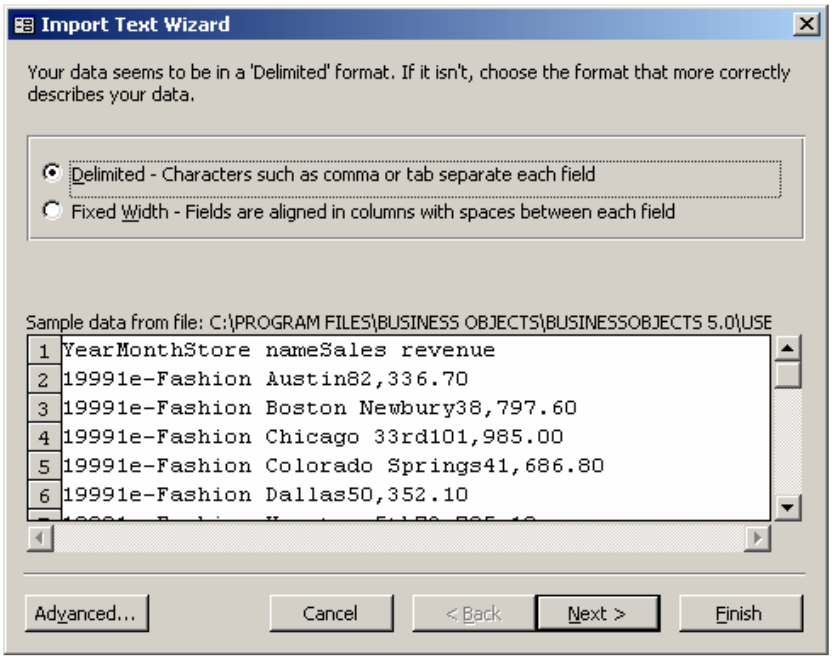


By default, the link name is the same as the text file name (without the extension). Change the name if desired, and click Finish to complete the Link Text Import Wizard. The linked table can now be referenced as needed (same as a pass-through query) by other steps in the Access process.

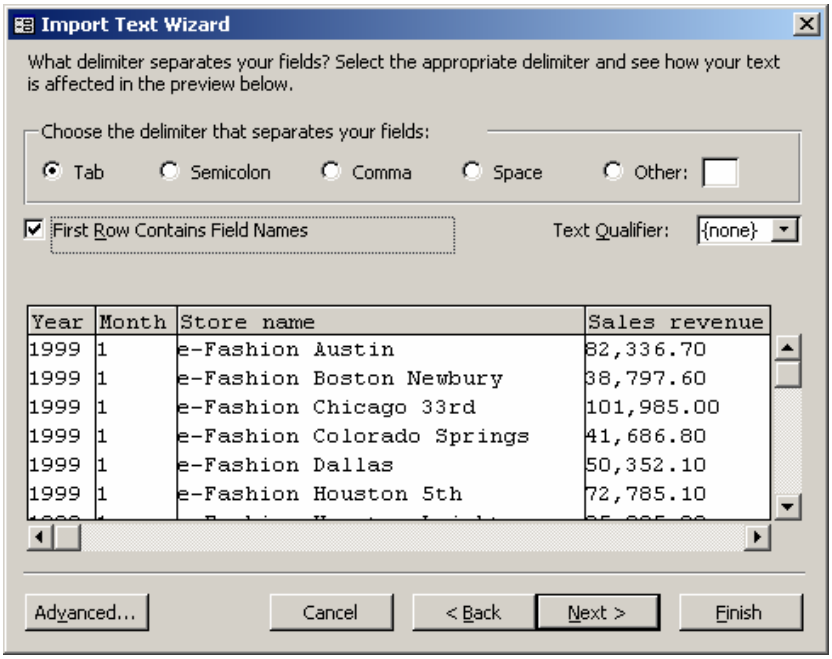
Import external data

This option is most useful when a "copy" of the data within the database is most useful. It also allows data validation, indexing for performance, and so forth to occur, as the data is being imported (though those techniques are beyond the scope of this document). This method will also easily accommodate "append" logic (as opposed to replace). Steps for importing external data are as follows:

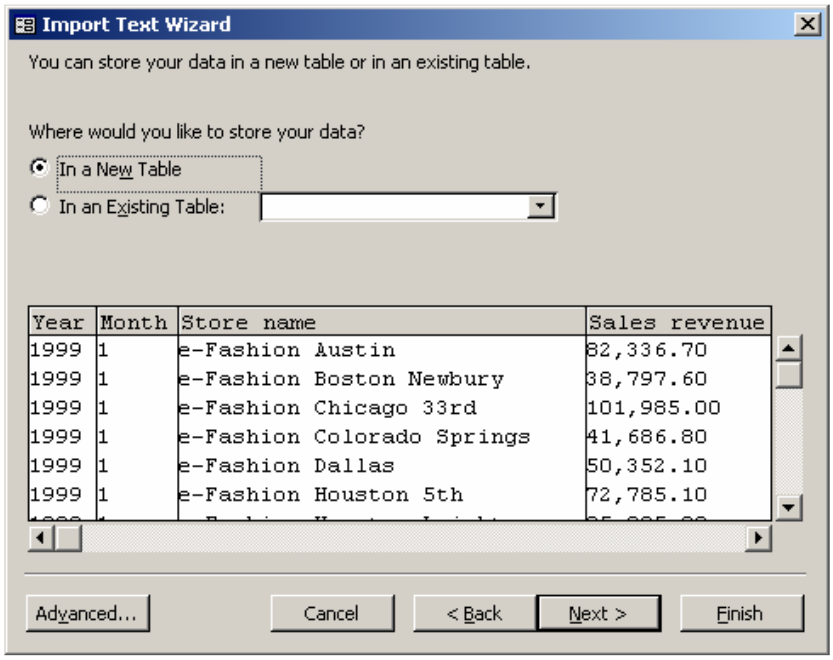
- From your database, choose File, Get External Data, Import...
- In the Files of Type drop-down, choose Text Files
- Browse to the appropriate directory, choose the text file, and click Import...
- This will start the Import Text Wizard:



Defaults are probably appropriate. Delimited is the file type. Click Next to continue with the next step in the wizard.

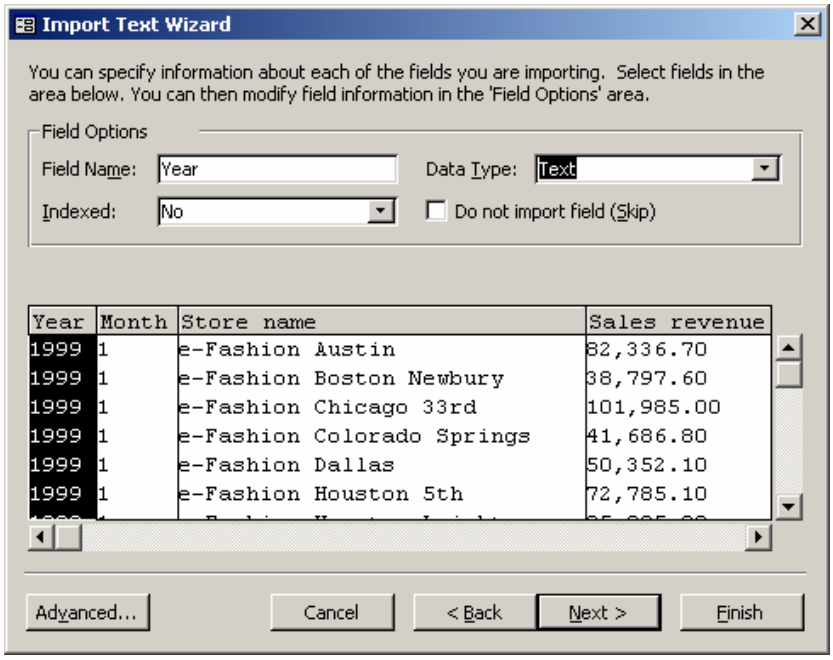


Tab is the appropriate delimiter. If the first row of the text file contains field names, indicate so by selecting the checkbox. Click Next to continue with the next step in the wizard.

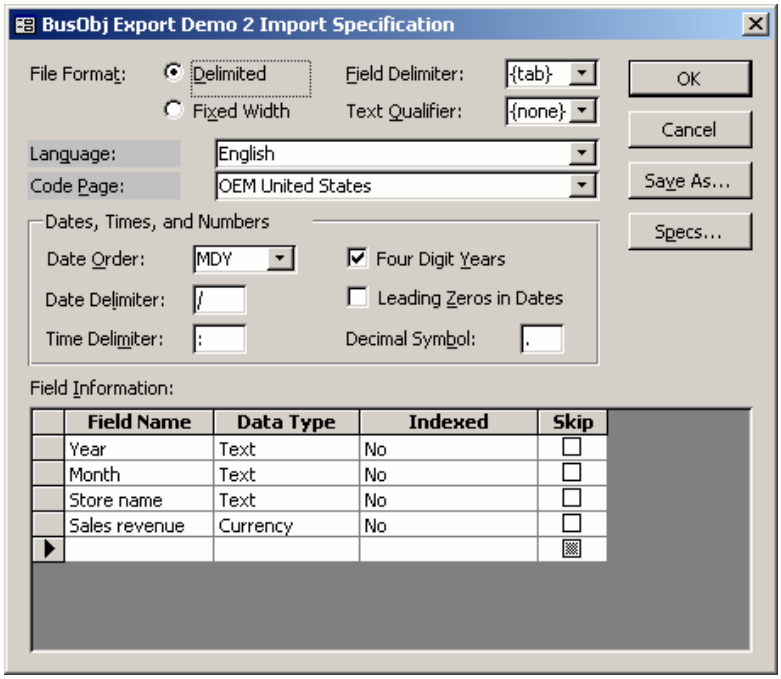


Since we are building this import for the first time, be sure In a New Table is selected. Click Next to continue with the next step in the wizard.

Note: If you select In an Existing Table, the next two steps will be skipped and data will be appended to the existing table with all previously defined data types, formatting, validation rules, and so forth enforced.

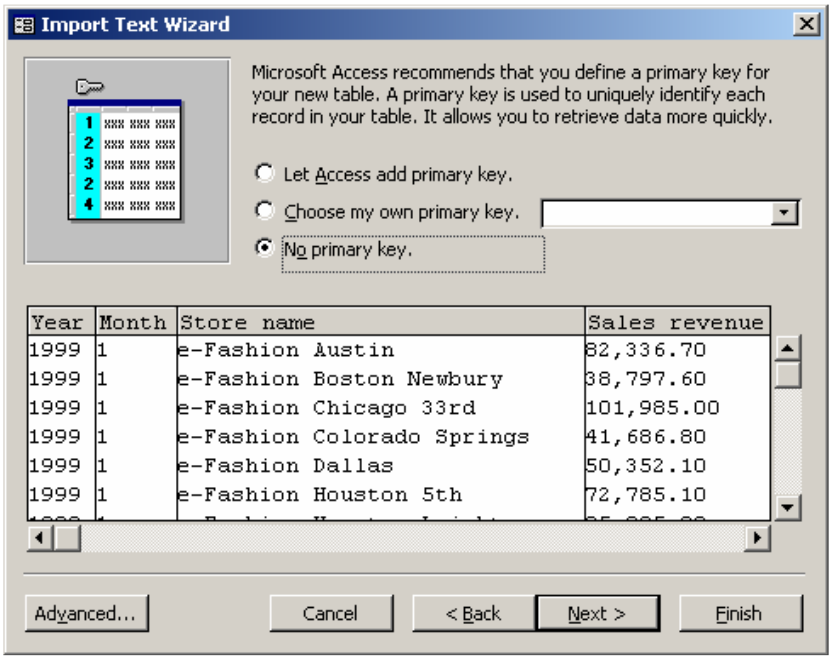


By default, Access does its best to interpret the appropriate data types. This is often a problem where columns that contain only digits (like account number in the example above) are erroneously interpreted as numeric fields. These columns can be set explicitly to text by selecting each column, and setting the Data Type to text. You can even set numeric columns to other data types if you like. Now that you've gone through all the trouble of creating this custom import specification, let's save the definition so it can be re-used. Click the Advanced... button.

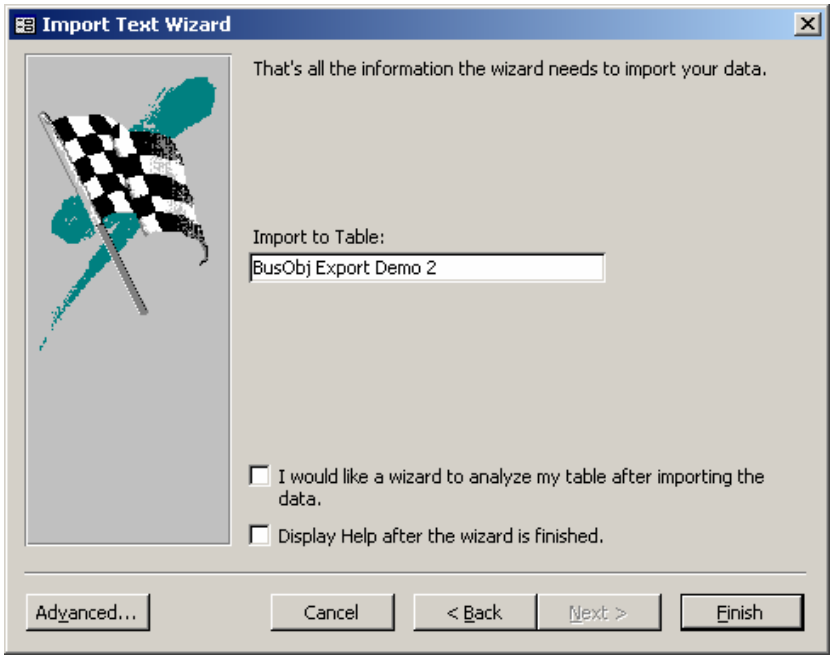


This is basically a one-screen summary of all of the import specification choices you made elsewhere in the wizard. Click the Save As... button to continue.

Name the specification, as you like. Click OK to save the specification, and OK again to return to the wizard. At any time in the future, you can click the Advanced... button anywhere in the wizard, and click the Specs... button to re-use the specification you just created. This will also be useful in the automation steps described later. Click Next to continue with the next step in the wizard.



If your database makes use of primary keys to enforce referential integrity, you can select that primary key. If not, select No primary key. Click Next to continue with the next step in the wizard.



By default, the table name is the same as the text file name (without the extension). Change the name if desired, and click Finish to complete the Import Text Import Wizard.

The imported table can now be referenced as needed by other steps in the Access process. Because an imported table exists within the database, you can add indexes, validation rules, default formatting, and so forth as needed.

Automation Options

The automation options discussed below assume a basic familiarity with Visual Basic for Applications (VBA), including use of the VB Editor for such tasks as creating and importing modules, commenting out lines of code, and basic debugging techniques.

BusObj automation

The creation of the tab-delimited text file(s) can be accomplished by using a VBA procedure. This task can also be triggered automatically each time the BusObj document is refreshed. If copied into the ThisDocument module of a BusObj document, the following example code will accomplish those tasks.

```
Option Explicit

Private Sub Document_AfterRefresh()
' The following lines can be commented out as appropriate.  In most cases,
' one or the other would be used but both can be used if needed.  If both
' are used, the data provider and report tab names must be globally unique.

    Call ExportDataProvs
    Call ExportReportTabs

End Sub

Sub ExportDataProvs()
' Each data provider is exported to the same directory as this document, in
' tab-delimited text format, using the same filename as the data provider.

    Dim DataProv As DataProvider
    On Error GoTo ErrorHandler

    For Each DataProv In ThisDocument.DataProviders
        Call DataProv.ConvertTo(boExpAsciiTab, 0, ThisDocument.Path _
            & "\" & DataProv.Name)
    Next DataProv

Exit Sub
ErrorHandler:
    MsgBox "Error " & Err.Number & ":  " & Err.Description, _
        vbCritical, "Failure exporting data provider from Business Objects"

End Sub

Sub ExportReportTabs()
' Each report tab is exported to the same directory as this document, in
' tab-delimited text format, using the same filename as the report tab.

    Dim Rpt As Report
    On Error GoTo ErrorHandler

    For Each Rpt In ThisDocument.Reports
        Call Rpt.ExportAsText(ThisDocument.Path & "\" & Rpt.Name)
    Next Rpt

Exit Sub
ErrorHandler:
    MsgBox "Error " & Err.Number & ":  " & Err.Description, _
        vbCritical, "Failure exporting report tab from Business Objects"

End Sub
```

The Document_AfterRefresh procedure is the code that runs automatically after the document refresh event occurs. The Document_AfterRefresh procedure will call the ExportDataProvs and/or ExportReportTabs procedure(s) as needed. As noted above in the comments, you can run one, the other, or both by commenting out the appropriate lines. The ExportDataProvs and ExportReportTabs procedures iteratively loop through each data provider or report tab and export each to a tab-delimited text file in the same directory as the document itself.

Application automation

Once the Get External Data links have been established as described above for Excel, or the linking / importing is done for Access, it is possible to add automation to the application (Excel or Access) so that refreshing the data is controlled by a VBA procedure. Generically, the automation steps are as follows:

- Add a VBA project reference to the BusObj object model.
- Start a session of BusObj, and log in to BusObj via ZABO connection, open and refresh the BusObj document containing the data providers / report tabs to be exported.

To add a VBA project reference, from the VB Editor (VBE), choose Tools, References... and select the BusinessObjects Object Library from the list. You can now use VBA to control BusObj from Excel or Access.

Now let's look at an example of VBA code that logs into BusObj and refreshes a document. Most of this code provides error handling. The "bare minimum" lines are shaded gray:

```
Function RefreshBusObjReport() As Boolean

    Dim BOApp As busobj.Application
    Dim Doc As busobj.Document
    Application.DisplayAlerts = False               'suppress Excel alert messages
    On Error GoTo ErrorHandler

    '1. start BusObj and log in
    Set BOApp = New busobj.Application
    Do Until BOApp.Variables("BOUSER").Value = ""   'be sure not to
        Set BOApp = New busobj.Application          'grab existing
    Loop                                            'BusObj session
    Call BOApp.LoginAs("userid1", "password", False, "busobjreporting.plano.esc.eds.com")

    '2. open and refresh the document
    Set Doc = BOApp.Documents.Open("BusObj Export Demo.rep", True)
    Call Doc.Refresh
    RefreshBusObjReport = True

    '3. clean up and exit BusObj
CleanUp:
    If Not Doc Is Nothing Then
        Call Doc.Close(boDontSave)
        Set Doc = Nothing
    End If
    If Not BOApp Is Nothing Then
        Call BOApp.Quit
        Set BOApp = Nothing
    End If
    Application.DisplayAlerts = True

Exit Function
ErrorHandler:
    RefreshBusObjReport = False
    MsgBox "Error " & Err.Number & ":  " & Err.Description, _
        vbCritical, "Failure getting data from Business Objects"
    Resume CleanUp

End Function
```

Referencing the numbers in the code comments above:

1. Change the userid and password as appropriate. Alternatively, you can leave off all of the parameters and the parentheses from the .LoginAs line of code, and the regular Business Objects login screen will appear.
2. Change the filename as appropriate. If the file is not in the ...\UserDocs directory, the filename needs to include the full pathname (like "D:\Documents and Settings\username\My Documents\BusObj Export Demo.rep"). Alternatively, you can leave off all of the parameters and the parentheses from the .Open line of code, and the regular Business Objects open dialog box will appear. Assuming that the BusObj automation above is included in the document, the text files will automatically be created.
3. The cleanup items ensure a "graceful" exit of BusObj. The refreshed report file is not saved with the boDontSave option on the .Close line of code. Other options are boSaveIfModified (saves without prompting) and boPromptUser (always prompts user).

Remember that if the BusObj automation described above is included in the report (.rep file) itself, the text files will automatically be created.

Excel Automation

Once the above automation (BusObj and Application) is complete, the final step for Excel is to refresh the external links in the Excel model, using the newly created text files. The following procedure will cycle through each worksheet and refresh each link. Again, the "bare minimum" lines of code are shaded gray.

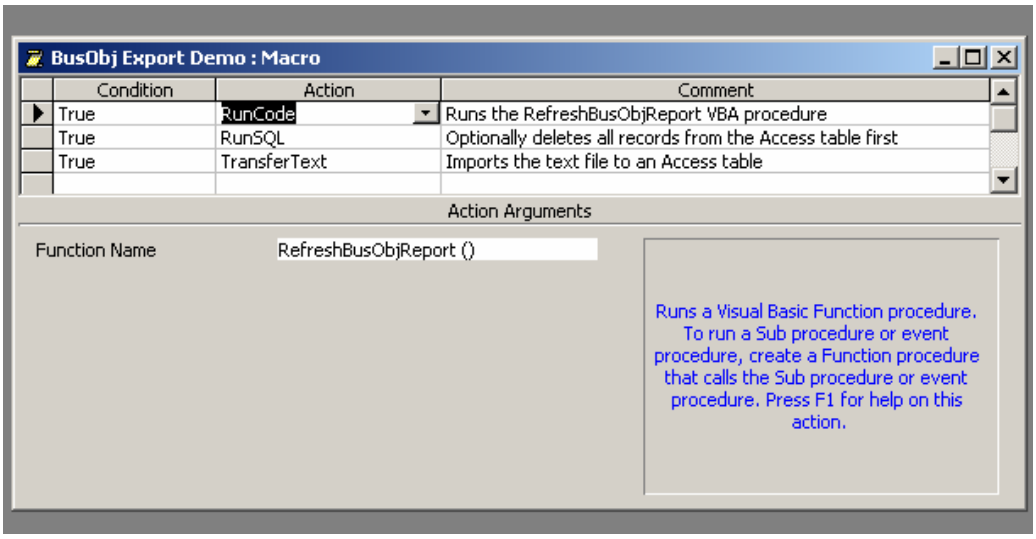
```
Function RefreshXLLinks() As Boolean

    Dim WkSht As Excel.Worksheet
    Dim QryTbl As Excel.QueryTable
    On Error GoTo ErrorHandler

    For Each WkSht In Worksheets
        For Each QryTbl In WkSht.QueryTables
            Call QryTbl.Refresh
        Next QryTbl
    Next WkSht

    RefreshXLLinks = True

Exit Function
ErrorHandler:
    RefreshXLLinks = False
    MsgBox "Error " & Err.Number & ":  " & Err.Description, _
        vbCritical, "Failure getting data from Business Objects"

End Function
```

The two steps (BusObj refresh and Excel refresh) can be tied together with some error checking as follows. Note that the two procedures (RefreshBusObjReport and RefreshXLLinks) are functions, and are coded to return false if they were unsuccessful for any reason.

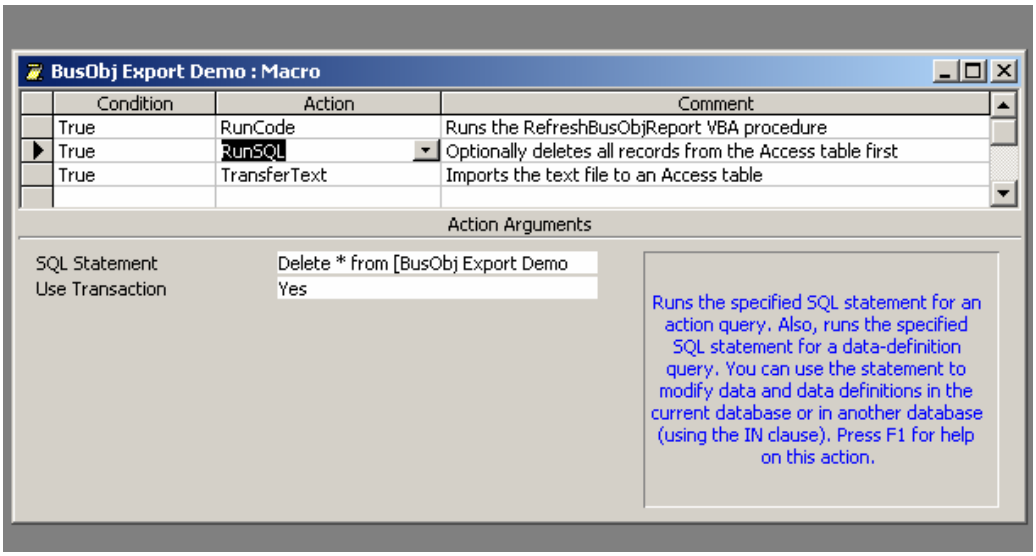
```
Sub UpdateFromBusObj()

    If RefreshBusObjReport = False Then GoTo NotSuccessful
    If RefreshXLLinks = False Then GoTo NotSuccessful

Exit Sub
NotSuccessful:
    MsgBox "Update from Business Objects was not successful"

End Sub
```

Access automation

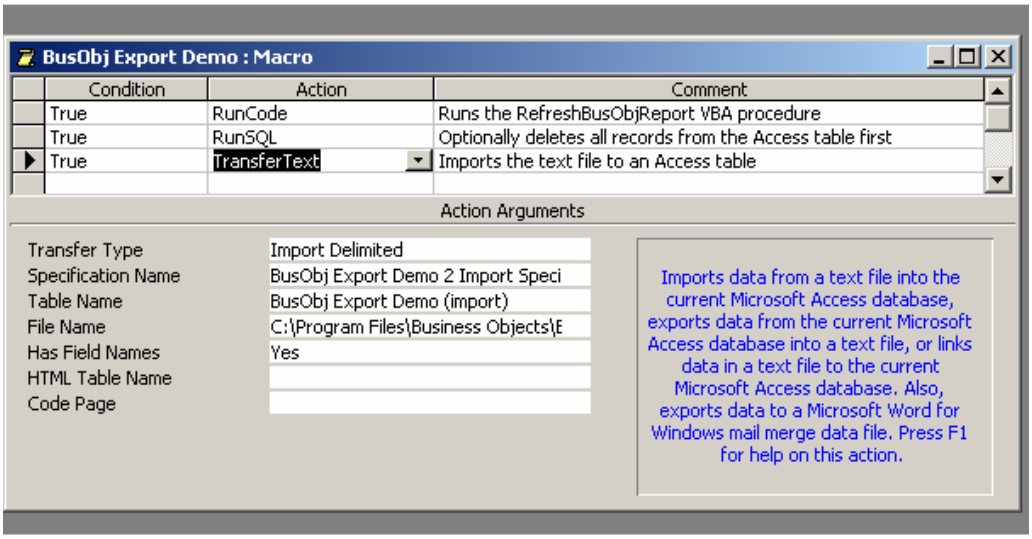
Once the above automation (BusObj and Application) is complete, the final step for Access is to refresh the database, using the newly created text files. Automation in Access is often accomplished using macros. The following macro has three steps:



The first step executes a RunCode action that calls the RefreshBusObjReport procedure that was outlined above. This will cause the BusObj report to be refreshed, and assuming the BusObj automation described above is included in the Busobj report, the text files will also be created. At this point, the remaining two steps are needed only if you've chosen the import option for Access. The linking option requires no further steps.



The second step in the macro is optional. If you want to re-use an existing table name, and do not want data appended to that table, it is necessary to empty the table first. This step executes a RunSQL action with the following SQL statement ... Delete * from [table name].



The third step executes a TransferText action. The Action arguments are self-explanatory for the most part. The Specification name is the name that we saved for "re-use" back when we created the import the first time manually. Remember that if the Table Name already exists, the data will be appended. Otherwise, the table will be created. If you want to re-use a table name, but want a full replacement rather than append, use the second step above to empty the existing table first.

Summary

While we encourage the use of BusObj as much as possible for reporting and querying, there are instances when integration with other applications is necessary. This document outlines the steps for integration of BusObj with Excel and Access, although integration with other custom applications is also possible.

The key to the integration is to utilize the strengths of each software component. Let BusObj do the queries (and optionally some calculations), export those results to tab-delimited text files, then use the "Get External Data" strengths of Excel and Access to get at the exported data.

This document outlines the manual steps to achieve this integration. It also includes a discussion and sample code for automation of that integration. The automation options illustrated in this document offer complete solutions, but there are more advanced approaches that are possible, once the approaches illustrated here have been mastered.

In closing ....

This document has ended up being quite lengthy. Hopefully you will find the content and step-by-step examples useful. You might even need to re-read portions of it, because the steps are sometimes subtle.

While it is believed that the contents of this document are accurate, the need to get this document published did not allow for a lot of testing outside of my own. Hey, it all worked on my machine **smile**. If there are any errors, or have ideas for additional content, please address those to:
Author: dwayne.hoffpauir@eds.com

If you have questions on usage or general questions on BusObj, please utilize the following resources:
External: http://busobj.forumtopics.com