MICROSOFT OFFICE 365 MIGRATION CASE STUDIES

As more and more business move to Office 365 a smooth migration process is essential to ensure business processes using Microsoft Office continue to work as intended and user confidence is maintained.

We have been tasked on many occasions with making the migration process run as smooth as possible.

Holding workshops for pilot scheme users is one way to proactively mitigate against issues occurring at the migration stage.

In this document we outline some of the issues users have faced as part of the migration process and our recommended solutions.

Excel workbook performance an issue

Where workbooks are saved in the traditional .xls, .xlsx and .xlsm formats performance gains can be made by saving in .xlsb format instead.

Case Study

Workbooks in the **xlsm** format were saved in **xlsb** format which resulted in performance gains when loading, saving and recalculating. This had no adverse effect on functionality.

Excel workbook bloating

Excel workbooks can suffer from bloating over time.

Case Study

A workbook in xls format was reduced in size when re-saved as xlsx format. By re-saving in xlsb format the size was further reduced with no adverse effects.

Excel workbooks where the host document renaming causes failures

In some cases renaming workbooks for performance gain resulted in the breaking of processes.

Case Study

A workbook with macros failed when saved in a different format. Windows were being activated by workbook name (including file extension) which will cause failure if filenames are changed. The following type of coding was found:

```
Workbooks.Open FileName:= “filename.xlsm”, UpdateLinks:=0
Windows( “filename.xlsm”).Activate
```

In VBA this could be changed to:

```
Dim wb as Workbook
Set wb = Workbooks.Open(FileName:= “filename.xlsm”, UpdateLinks:=0)
wb.Activate
```

This would resolve using explicit file naming except for the original place where it was opened and give a reference to the workbook via the reference **wb** which can then be used elsewhere without compromise.

A more streamlined approach would be to export data to a repository for **shared** use as **.csv** or **.txt** formats, to avoid sharing of workbooks between different process owners. The slightest change to one workbook could easily result in the failure of other workbooks without any kind of checks and balances.

Excel or Access macros fail with project library errors

An Office project may require object libraries which are no longer available or were installed separately into the host operating system.

There are also scenarios where an Active-X object may be removed in newer versions of the the same COM object library or replaced by a different but better version.

In such cases a macro may fail when it attempts to use the object with a **"Can't find project or library"** error.

Object library references can be checked by navigating to the **VBA IDE** (**Alt-F11**) and from the **Tools** -> **References** option. All available libraries are listed with ticked checkboxes for those in use. A lost reference is indicated by the prefix **MISSING** before the name of the library.

If the control is no longer required the object library may be removed by un-ticking the reference to it.

Where the control is required but the reference indicates that it is missing, the object library will need to be added to the host operating system often by just copying the required file to the **\ \Windows\SysWow64** folder.

Sometimes there is an additional step which involves registering the control with the operating system using the **regsvr32** command e.g. **regsvr32 /i vsflex3.ocx**

Active-X controls can be added to the Office suite and those known as **Windows Common Controls** included:

- TreeView and ListView
- ProgressBar
- Slider
- ImageList
- Toolbar
- TabStrip
- StatusBar
- ImageCombo
- Animation
- UpDown
- MonthView
- DateTimePicker
- FlatScrollBar

In Visual Basic 5.x the above controls were included in the **ComCtl32.ocx**, **ComCt232.ocx** and **ComCtl32.dll** files.

However, in Visual Basic 6 they were moved into two new files **MsComctl.ocx** and **MsComct2.ocx**.

Other controls commonly used include those from VideoSoft such as the **vsflex3.ocx** library of controls and the Microsoft **fm20.dll** forms 2.0 object library.

Case Study
An Access database failed with a spurious error relating to the use of the LEFT command which was not working. However, on further examination it was discovered that the OFFOWC.dll file (Microsoft Office XP Web Components) was missing. When the library reference to this component was removed the system then appeared to work correctly. It appeared that routines referencing the component were no longer in use but references to them still existed within the project.

Updating pivotTables using source data which has automatic subtotals via macro fails

In Excel you cannot use source data with automatic subtotals to be used for PivotTable creation and in the 2016 version an error is thrown. In previous versions this may have worked although this may not have been by design.

Case Study
A workbook which created a PivotTable used data from a worksheet containing sub-totals which were created within a macro.

A solution was found by removing the line within the macro used to create the sub-totals. It was then necessary to remove the actual existing row containing the sub-total created by the original macro.

If sub-totals are required, it is simply a case of replicating the worksheet and not using it as part of the pivotable update process i.e. remove the macro button.

Third party Excel add-ins not working

Third party add-ins may exhibit issues including:
- Add-ins not installed correctly
- Add-ins disappearing
- Functionality failure

There are different types of **Add-ins**. **COM** Add-Ins are automatically available from the **Add-ins** Menu and are installed normally as part of an installation process.

However, standard **Add-ins** MUST be installed separately using the standard Excel Add-ins dialog.

Note: When browsing to the folder and selecting an add-in MAKE SURE you do not SAVE the add-in to your local user folder (always answer **No** to the prompt).

Always check that the add-in is compatible with the current version of Excel. Functionality may change from one version to another, so this may need testing.

Case Study 1
The IBM i Series data transfer Add-in for Excel 2007/2010 was no longer available for Excel 2016. A new method using ODBC data transfer was configured for users to negate the requirement for an add-in.

Case Study 2
It has been noticed that some Excel 2007 workbooks which extract balances data using the CODA XL Integration package when run in Excel 2016 showed zeroes instead of the correct values. This suggests that there are some differences between the versions of the CODA XL Integration add-ins for Excel 2007 and 2016. It was noticed in CODA that a checkbox option when ticked was being ignored in the 2007 version of the add-in but was being utilised in the 2016 version. Simply unticking the option resolved the issue.

Macro running from standard Excel button does not work correctly in Excel 2016

When a macro is run from a button in earlier versions of Excel one would expect it to work the same in Excel 2016 although in some instances this was not the case.

Case Study
An Excel workbook macro failed to work (verified by data was not being populated correctly) when run from a **standard command button**. However, when the macro was executed directly the data was populated correctly, so the issue appeared to be with the common control being used.

It was found that the equivalent **Active-X command button** control when programmed with the macro did work, so this was adopted as the workaround.

Further investigation revealed that there was heavy dependence of the selection of and copying of data between multiple worksheets. A re-write of the VBA code resolved the issue.

Further investigation found that the issue was resolved by replacing the following line:

```
Range("H6:H" & LastRow).SpecialCells(xlCellTypeVisible).Copy
```

with the the following line (which is identical in functionality):

```
Sheets("Invoice").Range("H6:H" & LastRow).SpecialCells(xlCellTypeVisible).Copy
```

The original line **depends** on Excel **remembering** which sheet it is on whereas the replacement line explicitly defines the sheet to copy from.

If we were to select another sheet mid-stream in VBA code the original line of code will produce undesirable results. There are a number of places where the **select** statement is being used in the example workbook.

Note: It is recommended that you **do no**t use the **select** statement unless absolutely necessary i.e. unless you want the user to end up on the actual selected sheet. Unfortunately the macro recorder will create **select** statements when the user moves between sheets.

A preferred method is to refer directly to the sheet via the **Sheets** statement as per the example above.

There are other ways which may be considered better i.e. store the sheet object and then re-use it as follows:

```
Dim sh As Worksheet
Set sh = Sheets("Invoice")
sh.Range("H6:H" & LastRow).SpecialCells(xlCellTypeVisible).Copy
```

By using the above code you can refer to the **Invoice** worksheet at any point in the code by using **sh.**

ODBC data transfers not working as before

Some users have reported issues whereby the data being extracted via ODBC is different to that from previous extraction methods.

There have also been reports that the queries do not work exactly the same as before. This is to be expected as the query engines now being used might be different than those being used before.

Some users were previously using the IBM Client Access Tools to extract data to an intermediate file which was then imported to Excel. With ODBC a new statement can be created to extract and populate the workbook without an interim step.

Case Study 1

A lookup was found to not work in Excel 2016 and further investigation revealed that the data extraction using ODBC was slightly different to that of the previous method. Spaces were being added to the end of a string which was being used to lookup on post code.

It is important to be aware that data extraction can sometimes differ with strings whereby spaces are added to the end of data. For example, the post code might be 10 characters in length but most post codes are less than the maximum allowed. This may add spaces to the end of a post code.

To resolve such issues make sure that data is TRIMMED before extracting into Excel 2016. This will ensure compatibility between the old and new methods of data import.

Case Study 2

An issue was found where a table query copied down existing adjacent formulae with incorrect cell references (in particular after the 100th inserted row). Query normally previews the first 100 rows, so this explains why it works for only these rows.

The resolution was to create just one row of formula in adjacent columns, insert the query and then include all used rows and columns as part of the inserted query table (by dragging the data table boundary marker to include the existing columns), finally refreshing the data to copy down and re-calculate the formulae.

Checking for errors in macro code between versions

A quick method to check for potential errors is to go to the VBA Editor (Alt-F11) and use the **Debug - Compile** option to make sure that there are no coding errors not yet discovered.

Case Study

The Access Purchase Orders system was checked for errors and many were found which had not been previously identified. The cause was simply the use of the Copy and Paste feature to make duplicates of forms which were then amended by removal of fields from the form. Code which used the removed fields then failed when checked via the **Compile** option. It was simply a matter of removal of lines not used to correct the errors.