PersisTables

A User Guide

Contents

Introduction 4

An End User Application 5

Nomenclature 6

Basic Tables 6

Create 6

Hierarchical Tables 13

Table Viewer 15

Integrating with Excel functions 17

Persistence 18

Save & Load 19

Monikers, Genres & Categories 21

Persistence Viewer 22

Arenas 26

Putting It Together 30

Managing Data 43

Monikers 43

Arenas 43

Genres & Categories 44

PersisTables & Editions 48

Protocols 50

Schemas and Revisions 53

Table Manipulation 56

PersisTables 56

AtXPath & XPath 58

Fill 61

Append and Filter 63

Arithmetic Functions 65

Select and Merge 67

Sort & Transpose 70

Advanced Topics 71

Using Range instead of Table Handle 71
Displaying Ranges instead of Table Handle 72
Loading a worksheet containing Tables 73
Table to Worksheet 73
PersisTables memory management 77
Table Handles 77
Mark & Sweep 78
Appendix 81
Array Functions 81
Excel functions that operate on PersisTable output 81
Table serialisation plug-ins 81
Errors 81
Setting Automatic Calculation Mode in Excel 82
Create an Update Macro 82
Conditional Formatting of Formula Overwrite 85
Creating a List from a Table 86
Copyright Acknowledgements 91
Glossary 92

Introduction

Excel is a great tool for end users to prototype and build effective business applications. It's fairly intuitive interface and the ability to create functional engines without the need to understand formal programming, make it an ideal tool for non-programmers to harness the power of modern computers.

Once built, Excel tools can become key business applications, aggregating data from multiple sources they often become the only place where strategic information can be accessed in its entirety. Simple spreadsheets can evolve into monsters as additional functionality is bolted on to the one tool that has access to the critical data.

At this point the urge to replicate the data becomes strong, this can lead to multiple copies of the data, none of which can be guaranteed to be definitive or multiple copies of functionality where changes are made on an ad-hoc basis to the different copies of the spreadsheet and the transformations performed by the spreadsheets are no longer comparable.

Recognising the danger of the degradation often requires such End User Computing (EUC) to be migrated to an IT platform, which inevitably leads to reduced flexibility, longer lead times for change and a general ossification of the tools.

The PersisTables Toolkit is designed to provide end users with the tools to secure and share their data both with other Excel users and with IT based applications. It can handle both simple tabulated data and create CSV[1] files or complex hierarchical information that requires the use of document storage technology based on XML or JSON[2] formats.

It can be applied to existing spreadsheets without affecting their current operation, allows non-programmers to manage the storage of their own data, is designed to be flexible in handling changes in the structure of data and provides a pathway for IT to migrate from an Excel prototype to a formal technology application while retaining the end user's ability to access the original data and modify the processing of that data as business requirements change.

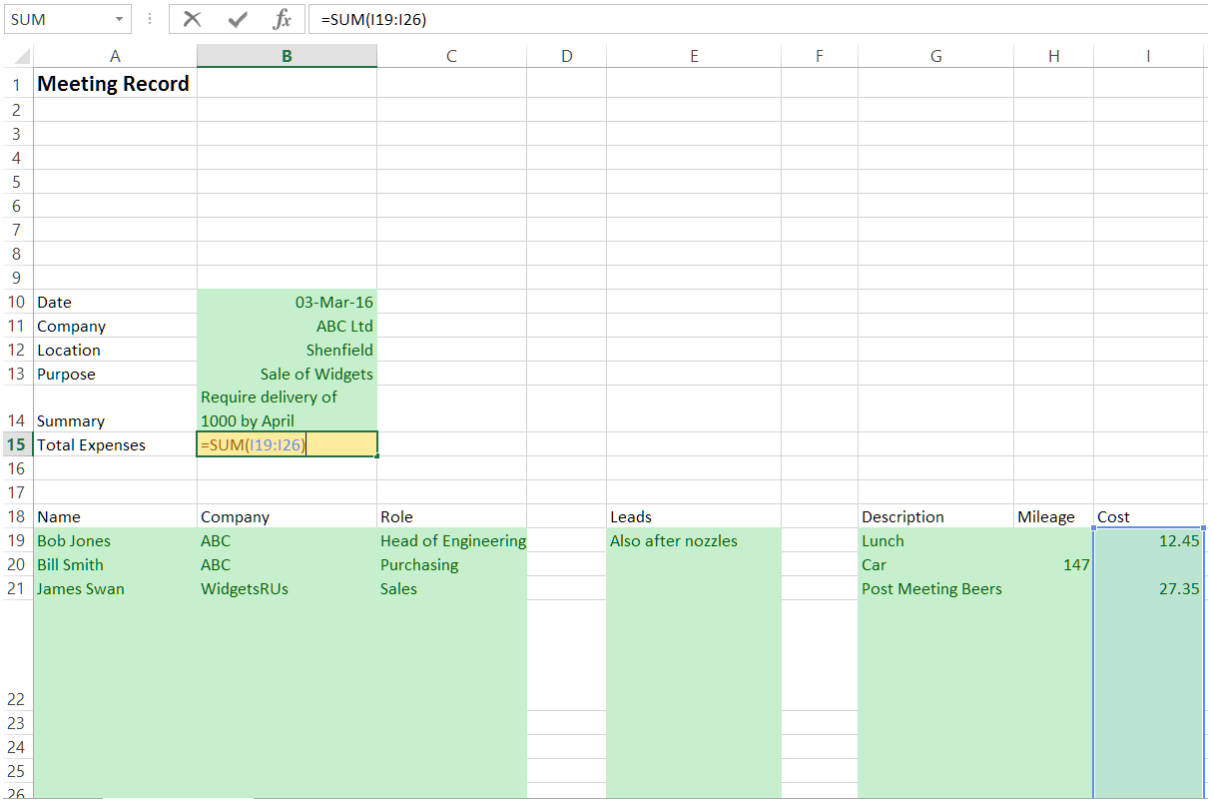
The PersisTable Toolkit is aimed at empowering end users in the design process of IT systems; create robust prototypes which apply good IT processes to the management of data without the need to become a programmer.

[1] CSV is a file format based on comma delimited data records, hence Comma Separated Values

[2] XML stands for eXtensible Markup Language which is a commonly used data format for hierarchical data structures, JSON is another format for describing hierarchical data.

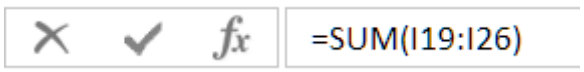
An End User Application

To demonstrate how we can use PersisTables to manage Excel based information the rest of this document will look at how we can implement a client contact reporting sheet. This is a deliberately trivial example but is a useful platform to demonstrate what PersisTables can do.



We will follow certain conventions in this document. First the colour scheme is green where data can be entered, yellow is a formula that may need to be changed and red should only be altered by the person who wrote the spreadsheet.

The yellow area includes formulae which when selected are highlighted on the function ribbon at the top of the picture. This is useful to see the function that is in the active cell. In this view the cell has been put into edit mode (Press {F2} Key) as well which shows the source of the arguments for the function.



The function ribbon can assist with understanding what functions are used on the worksheet and by pressing the *fx* button the function wizard is invoked which will identify the arguments and what they mean. We will use the function wizard to assist us in explaining the operation of various PersisTables functions.

Note that the range being summed is a guess, we do not yet know how many expenses we may have for a particular trip. If there are more than will fit in the highlighted region then the expenses will not be calculated correctly.

We will use this basic spreadsheet to show how we can adapt an existing spreadsheet and turn it into a more robust product using PersisTables. The spreadsheet itself is part of the installed package and can be used to familiarise yourself with the operation of PersisTables.

Nomenclature

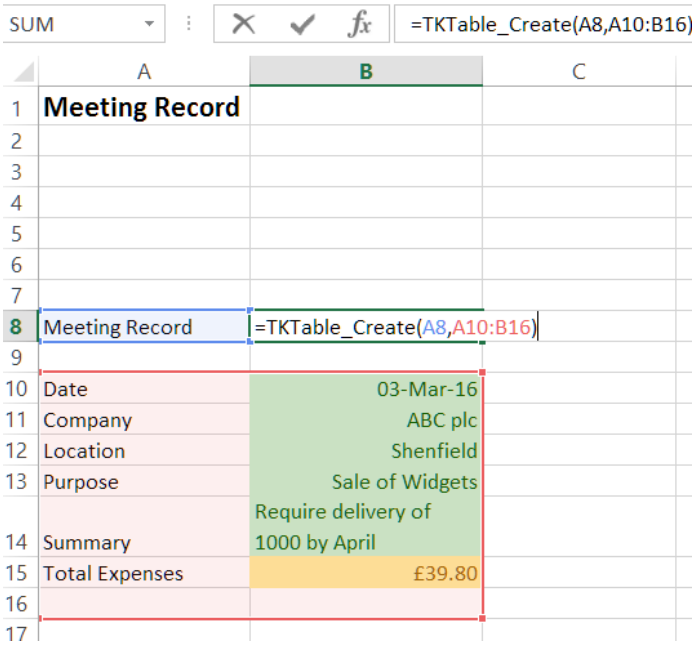
The PersisTables Toolkit introduces some new concepts which require new nomenclature. This may seem confusing initially but this document will introduce the concepts gradually. Terms with specific meaning within the PersisTables infrastructure will be given Initial Capitals. These are summarised in the Glossary.

Basic Tables

Create

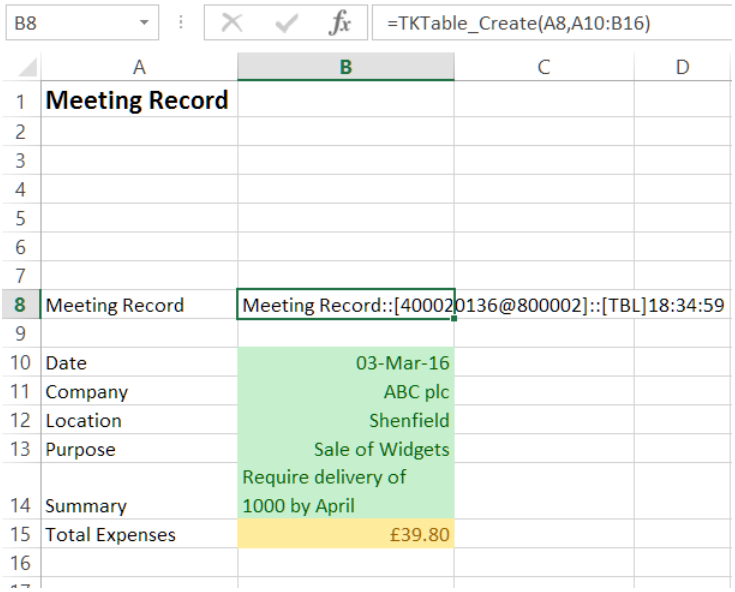
The basic component of the PersisTables Toolkit is the Table. A Table is a mechanism for capturing a range of data on the worksheet into a single cell, this can reduce clutter and allows the data to be manipulated within the PersisTables Toolkit.

A Table is created using the TKTable_Create() function. This function is part of the PersisTables Toolkit, it take a range of cells from the spreadsheet and returns a Handle to the Table which starts with the name given as the first argument in the call to the TKTable_Create() function and a unique TBL suffix. This Handle can be used to refer to the data in the range of cells in other Toolkit functions rather than having to include the original range.



Creating a Table from an excel range of data

Note the outlined rectangles refer to the cells or ranges that are used as inputs to the Excel function, the formula is displayed in the cell but also in the formula bar at the top of the picture. We will illustrate the use of the PersisTable functions either by showing the expanded function command in the cell or, where we want to show the results, with the function and arguments in the formula bar.

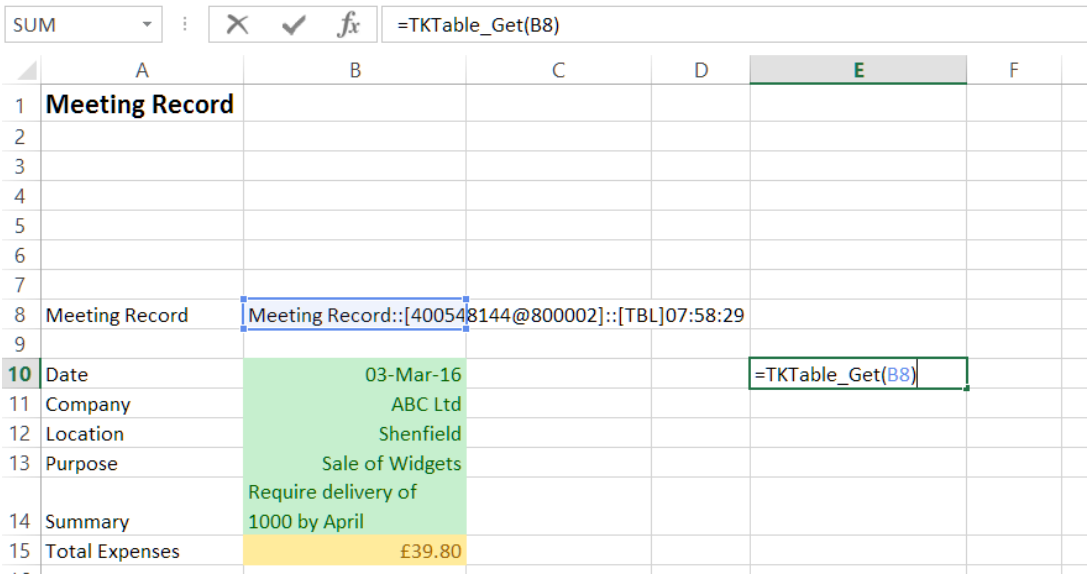


Here the executed function is shown with the results of the function execution displayed in the cell selected and the arguments in the formula bar.

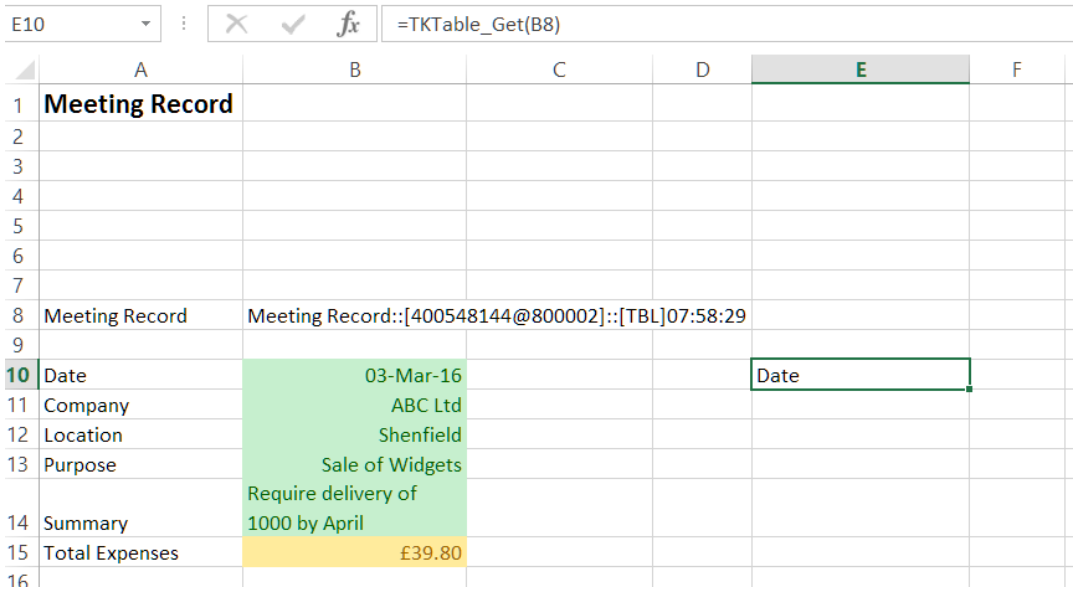
Note the Handle returned from the successful execution of this function, it contains the user supplied name 'Meeting Report' and some additional information which allows the PersisTable Toolkit to identify and manipulate the data linked to the Handle in other calls. From a user perspective the Handle can be treated as just a mangled name. For the curious the component details of the Handle are explained later. We call it a Handle because it is what allows us to hold onto the data in the Table.

Adding the TKTable_Create() function on the sheet has not affected the operation of the sheet. The original data is still present and can carry on being used in the same way as before. We have taken a copy of the data but it is not a snapshot, every time the data in the Range is changed, the Table will update itself.

We can see this behaviour if we try and extract the information currently held in the Table. The simplest way to access data referenced by a Table handle is to use the TKTable_Get() function. This extracts the data in the Table and displays it on the sheet. It takes a Table handle as input and displays the Table contents on the sheet.

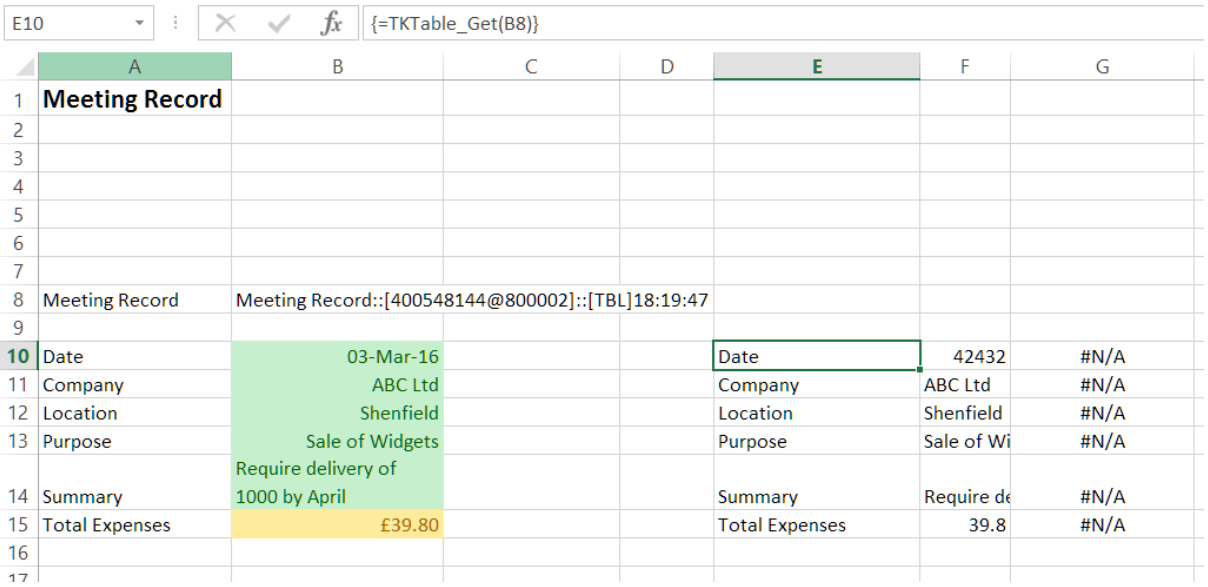


The Get Function as entered which refers to the Handle we previously created in Cell B8. When we press return on the formula we see



Where the TKTable_Get() Function displays the first cell in the Table we created earlier. That is a bit disappointing, how can we see the rest of the data held in the Table?

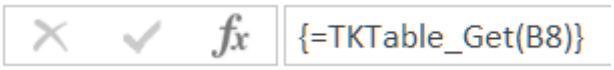
An Excel function normally fits its results into a single cell, we need a mechanism to allow the TKTable_Get() function to display all the data in the Table it is accessing. To achieve this we must execute the TKTable_Get() function as an Excel array formula (see Appendix 1 for details on creating array formulae) to display all the data on the sheet.



	A	B	C	D	E	F	G
1	**Meeting Record**						
2							
3							
4							
5							
6							
7							
8	Meeting Record	Meeting Record::[400548144@800002]::[TBL]18:19:47					
9							
10	Date	03-Mar-16			Date	42432	#N/A
11	Company	ABC Ltd			Company	ABC Ltd	#N/A
12	Location	Shenfield			Location	Shenfield	#N/A
13	Purpose	Sale of Widgets			Purpose	Sale of Wi	#N/A
14	Summary	Require delivery of 1000 by April			Summary	Require de	#N/A
15	Total Expenses	£39.80			Total Expenses	39.8	#N/A
16							

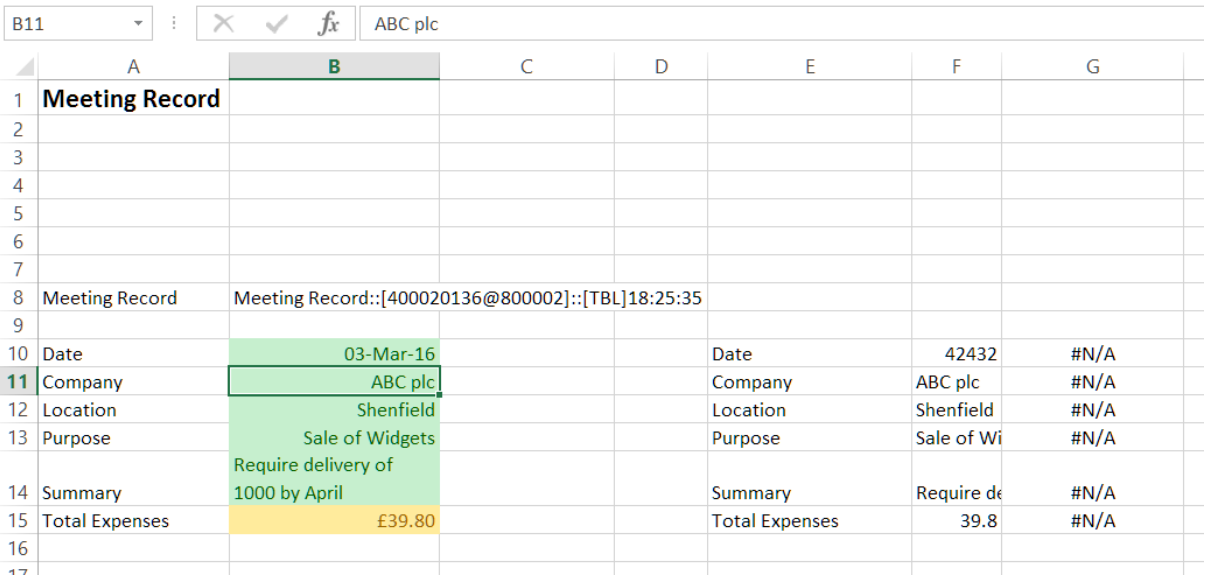
Now we have extracted all the data in the Table and displayed on the sheet.

Note that Excel indicates an array formula by placing {} round the macro when displaying it in the formula bar.



As the array function covers a greater number of cells than can be filled with the original data, the extra cells are filled with the #N/A value. This indicates areas where there is no Table data to display. The observant reader may have noticed that we have less data in the resultant Table than the Range we originally selected. This is because the Toolkit identifies rows or columns which are empty and does not add them to the Table when using TKTable_Create().

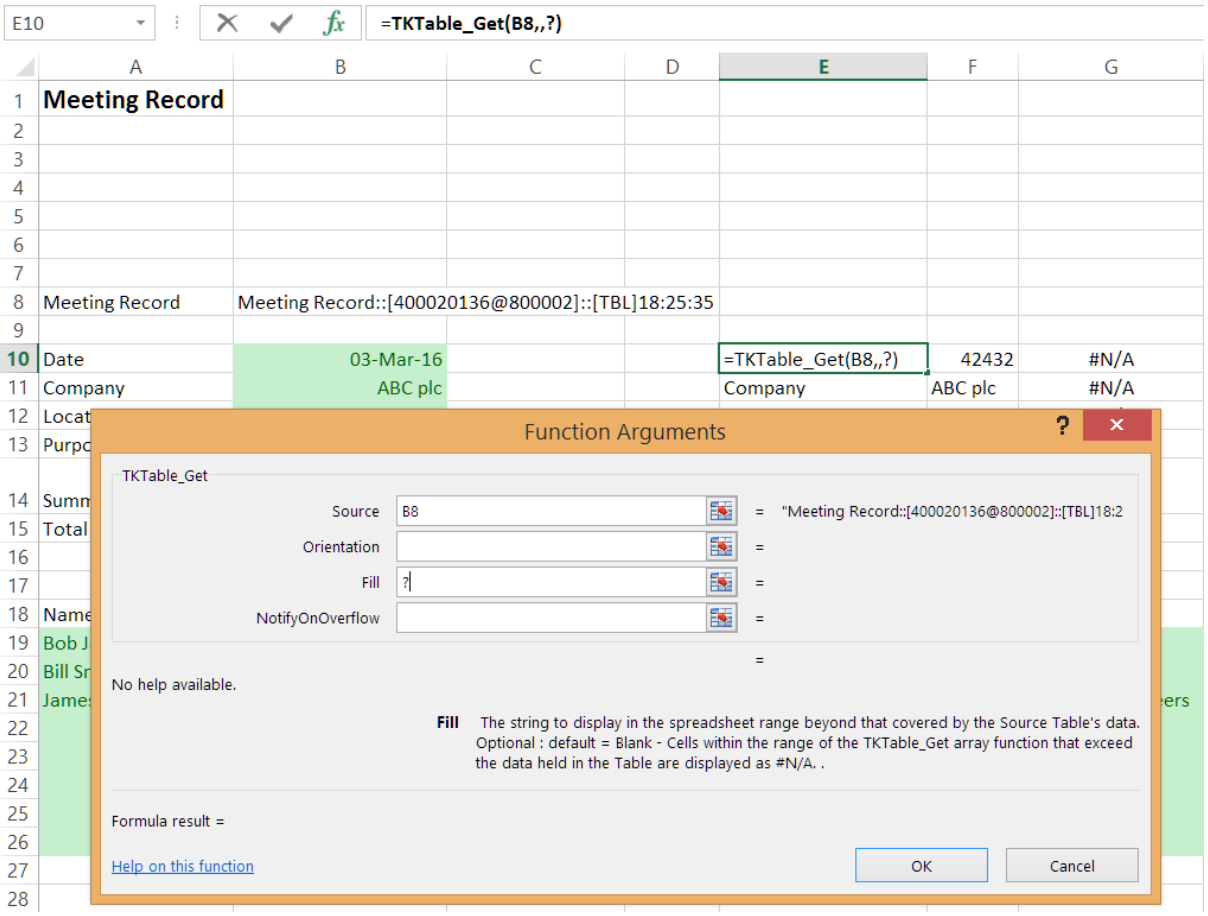
As soon as data is entered into the Range used for input to TKTable_Create() it is reflected in the data extracted from TKTable_Get(). If we change the company name, the results displayed from TKTable_Get() show the change.



	A	B	C	D	E	F	G
1	**Meeting Record**						
2							
3							
4							
5							
6							
7							
8	Meeting Record	Meeting Record::[400020136@800002]::[TBL]18:25:35					
9							
10	Date	03-Mar-16			Date	42432	#N/A
11	Company	ABC plc			Company	ABC plc	#N/A
12	Location	Shenfield			Location	Shenfield	#N/A
13	Purpose	Sale of Widgets			Purpose	Sale of Wi	#N/A
14	Summary	Require delivery of 1000 by April			Summary	Require de	#N/A
15	Total Expenses	£39.80			Total Expenses	39.8	#N/A
16							

Note for this to happen automagically, the Automatic Calculation option needs to be set within the Excel session (See Appendix to switch Setting Automatic Calculation Mode in Excel)

The #N/A displayed with the use of TKTable_Get() function can be a bit distracting, so the function allows the user to specify their own fill character or string. To understand how to use this feature we can access the function wizard. If a cell which is selected as part of the array function, it will highlight the TKTable_Get() function in the function toolbar. Clicking the *fx* button displays the function wizard.



The Function Wizard is a valuable tool in learning about the features of the PersisTables Toolkit. The user is encouraged to use the Function Wizard to learn about the different arguments, what they do and what values are applicable. If we set the Fill parameter to the ? character, this results in the Table being displayed as follows.

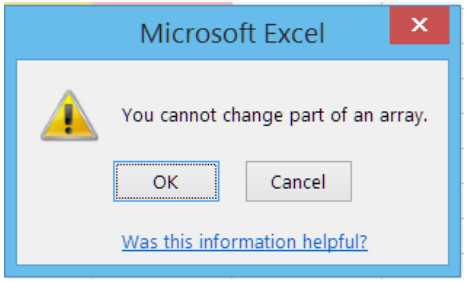
E10 {=TKTable_Get(B8,,"?")}

	A	B	C	D	E	F	G
1	**Meeting Record**						
2							
3							
4							
5							
6							
7							
8	Meeting Record	Meeting Record::[400020136@800002]::[TBL]18:25:35					
9							
10	Date	03-Mar-16			Date	42432	?
11	Company	ABC plc			Company	ABC plc	?
12	Location	Shenfield			Location	Shenfield	?
13	Purpose	Sale of Widgets			Purpose	Sale of Wi	?
14	Summary	Require delivery of 1000 by April			Summary	Require de	?
15	Total Expenses	£39.80			Total Expenses	39.8	?
16							

This is less intrusive than #N/A but still identifies the boundaries of the TKTable_Get() Array function. More usually the space character is used so empty areas appear blank.

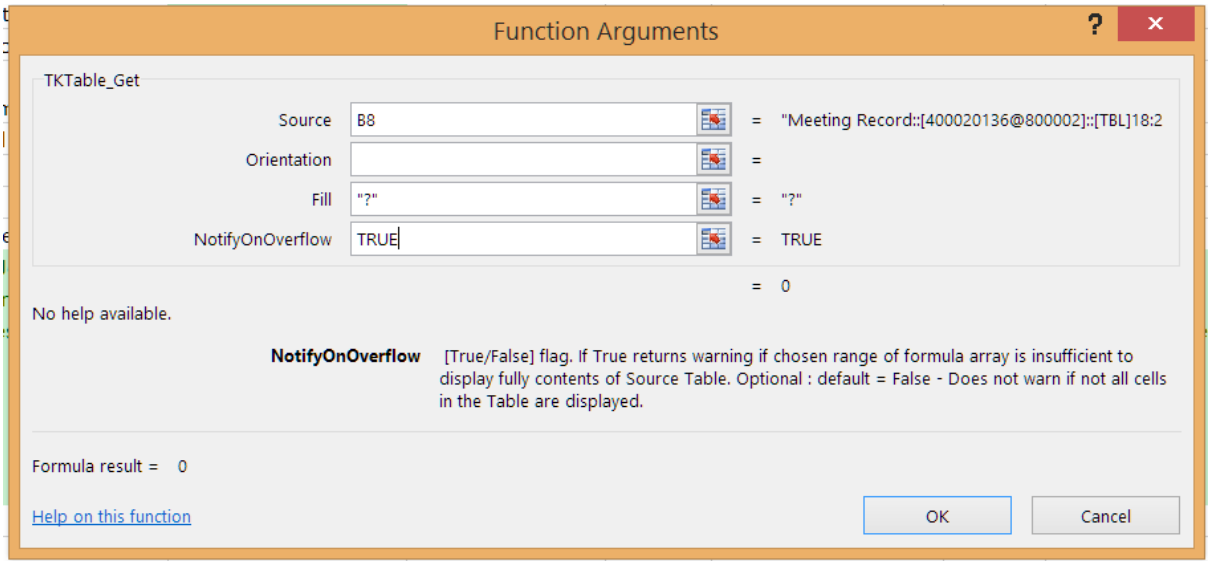
Note that by using the Wizard the optional 'Orientation' argument can be omitted but the formula has automatically included the additional commas for the missing optional argument.

Note also that when entering additional parameters directly on the sheet that the array formula must be re-entered otherwise the following error message is displayed.



If the array specified for the TKTable_Get() function is less than the size of the table then the table contents displayed is truncated to the given area. Care should be taken when relying on TKTable_Get() to display all the data that may be in a table. Where a Table is likely to grow in size either the array must be oversized to allow for that growth or the NotifyOnOverflow flag should be set to indicate where data maybe missing when the Table contents fills the given array.

Switching this parameter to true in the Function Wizard



And adding a further date to the input range to the TKTable_Create() function results in TKTable_Get() displaying a warning.

8	Meeting Record	Meeting Record::[400020136@800002]::[TBL]18:30:09			
9					
10	Date	03-Mar-16	#Warning - Table data e	#Warning	#Warning - Table data exceeds range supplied
11	Company	ABC plc	#Warning - Table data e	#Warning	#Warning - Table data exceeds range supplied
12	Location	Shenfield	#Warning - Table data e	#Warning	#Warning - Table data exceeds range supplied
13	Purpose	Sale of Widgets	#Warning - Table data e	#Warning	#Warning - Table data exceeds range supplied
14	Summary	Require delivery of 1000 by April	#Warning - Table data e	#Warning	#Warning - Table data exceeds range supplied
15	Total Expenses	£39.80	#Warning - Table data e	#Warning	#Warning - Table data exceeds range supplied
16	Meeting Time				

Note the PersisTables Toolkit returns Errors and Warning messages prefixing them with the '#' character.

The TKTable_Get() macro has another optional parameter before the fill string. This can be used to change the orientation of the output table. All Tables are assumed to have headings or Keys in the first row of data. By specifying "Row" orientation the keys are displayed in the first column rather than the first row. In this case PersisTables assumed that Date 03-Mar-16 are the Keys.

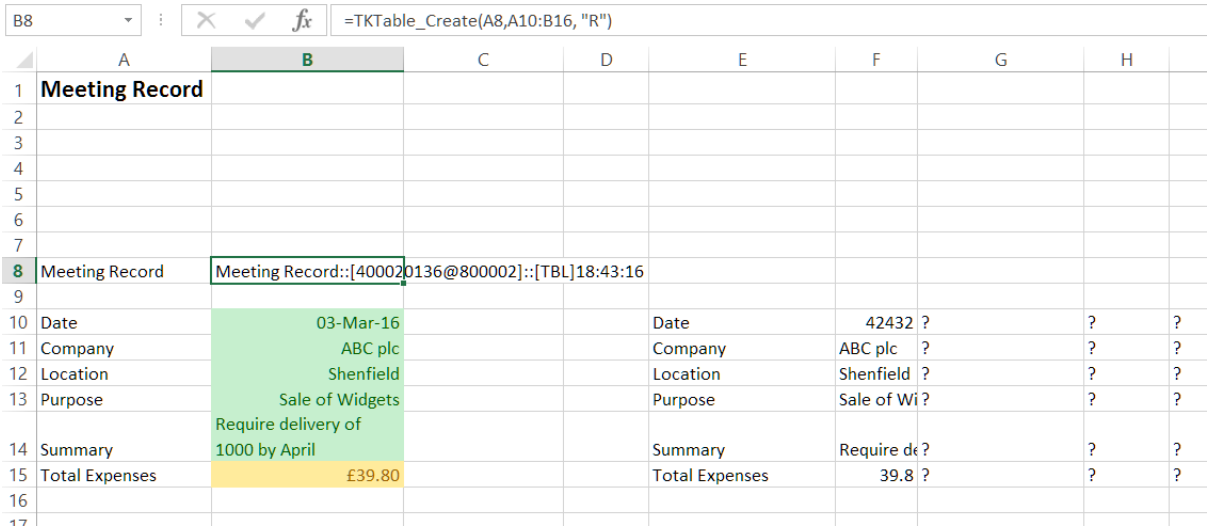
E10 {=TKTable_Get(B8,"Row","?",TRUE)}

	A	B	C	D	E	F	G	H	I	J	K
1	**Meeting Record**										
2											
3											
4											
5											
6											
7											
8	Meeting Record	Meeting Record::[400020136@800002]::[TBL]18:34:59									
9											
10	Date	03-Mar-16			Date	Company	Location	Purpose	Summary	Total Expe	?
11	Company	ABC plc			42432	ABC plc	Shenfield	Sale of Wi	Require deliver	39.8	?
12	Location	Shenfield			?	?	?	?	?	?	?
13	Purpose	Sale of Widgets			?	?	?	?	?	?	?
14	Summary	Require delivery of 1000 by April			?	?	?	?	?	?	?
15	Total Expenses	£39.80			?	?	?	?	?	?	?
16											

Output from TKTable_Get() function when orientation is specified as 'Row'.

We have had to extend the Range that TKTable_Get() occupies to ensure all the data is displayed.

The PersisTable Toolkit assumed that the first row represents the Keys to the data. However the original data is presented in Excel with keys in the left most column of the Range. We can use the Orientation parameter in the TKTable_Create() function to specify that the input is “Row” oriented.



B8 =TKTable_Create(A8,A10:B16, "R")

	A	B	C	D	E	F	G	H	
1	**Meeting Record**								
2									
3									
4									
5									
6									
7									
8	Meeting Record	Meeting Record::[400020136@800002]::[TBL]18:43:16							
9									
10	Date	03-Mar-16			Date	42432	?	?	?
11	Company	ABC plc			Company	ABC plc	?	?	?
12	Location	Shenfield			Location	Shenfield	?	?	?
13	Purpose	Sale of Widgets			Purpose	Sale of Wi	?	?	?
14	Summary	Require delivery of 1000 by April			Summary	Require de	?	?	?
15	Total Expenses	£39.80			Total Expenses	39.8	?	?	?
16									

The meeting details are presented with the Keys in the left column, if the TKTable_Create() macro is called with row orientation specified the PersisTables Toolkit knows that the Keys are in the left hand column. We can truncate the “Row” parameter to “R” and get the same effect.

When the Table data is extracted using TKTable_Get() it remembers that the input data was row oriented, the “Row” argument just confirms the fact that the data should be presented with the Keys in the left hand column. In fact the “Row” argument can be left out now and the PersisTable Toolkit will know that the Table was created form data in a Row orientation so will display it back in Row orientation.

So why might we choose to use orientation in TKTable_Get() when the orientation is specified in the TKTable_Create() function? Well we may have data created with the Keys as the top row and want to display the data in a Row oriented manner. This will happen whenever a PersisTable manipulation function is used which creates a new Table. The new Table will always be created with Keys as column headings.

Note all other operations on Tables which reference Keys or Items are agnostic to orientation. Wherever the Keys were located when the Table was created, these operations will operate on them in the same way.

Hierarchical Tables

Our Table currently only contains the top level details of the Meeting, Attendees, Leads and Expenses are not currently captured. We can use the TK Table_Create() function to create Tables for each of these elements.

Name	Company	Role
Bob Jones	ABC	Head of Engineering
Bill Smith	ABC	Purchasing
James Swan	WidgetsRUs	Sales

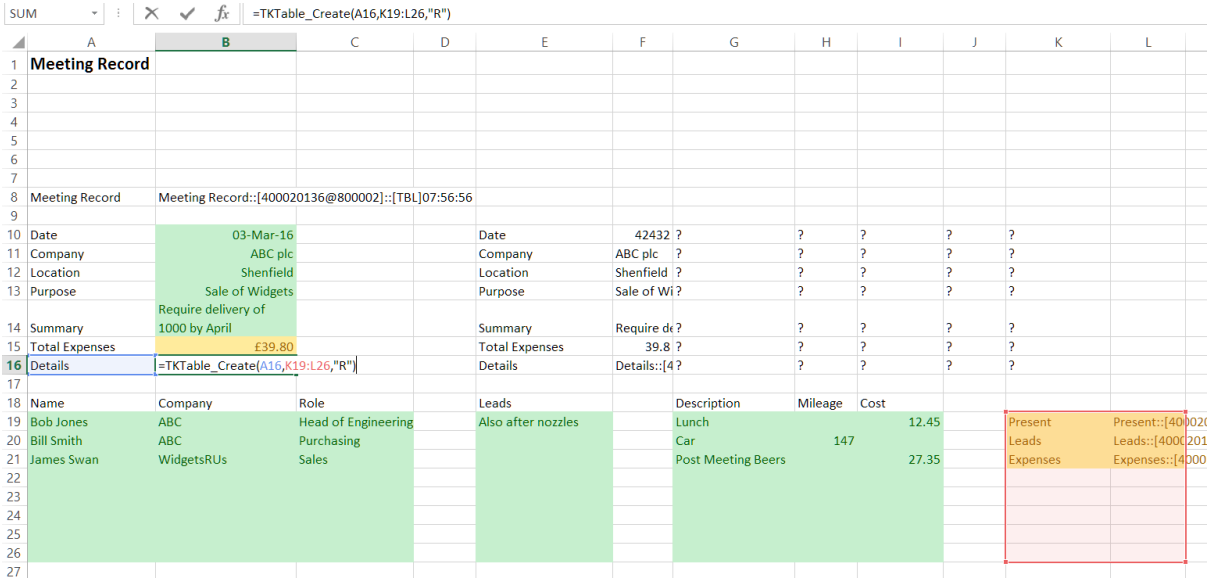
Leads
Also after nozzles

Description	Mileage	Cost
Lunch		12.45
Car	147	
Post Meeting Beers		27.35

Present	=TKTable_Create(K19,A18:C26)
Leads	Leads::[400548144@20
Expenses	Expenses::[400548144@

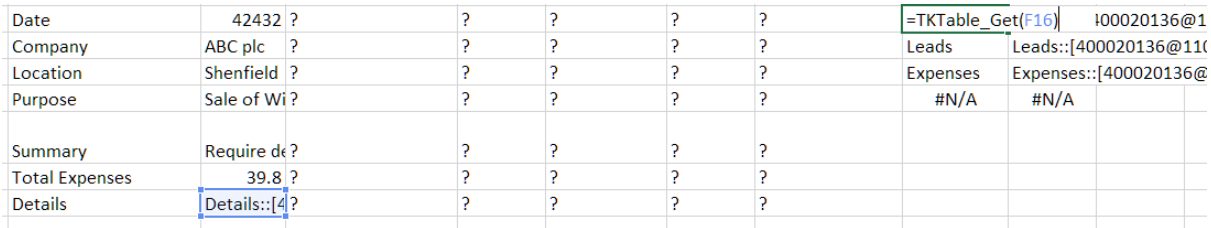
We now have 4 Tables each of which contains some of the information relating to the meeting but this is still of limited value.

We can use the TKable_Create() function to take a set of data found in a range on a spreadsheet which already contains Handles to Tables. In this case the new Table will embed the Table represented by the Handle inside itself. We can use this feature to create hierarchical data structures of arbitrary complexity (but note that we must not create a circular structure where Table A references Table B and Table B references Table A directly or indirectly).



The Details Key under the main Meeting Report contains a Handle to a Table which itself contains Table Handles. So the component Tables representing the Attendees, Leads and Expenses are now embedded within the Meeting Report. In the data extracted by TKTable_Get() we can see this also contains the Handle to the Details of the meeting.

Extracting the data using the TKTable_Get() function is more complicated on hierarchical Tables as Tables at each level have to be displayed in turn.



TKTable_Get() returns different Handles from the original data, the embedded Table displayed is a new copy of the original data. We can apply the TKTable_Get() function to the embedded Handles and extract the data from the component Tables. The orientation of the extracted Table is “Row” and is the same as the original Table. We could request TKTable_Get() to display in “Column” orientation to have the Keys at the top of the Table.

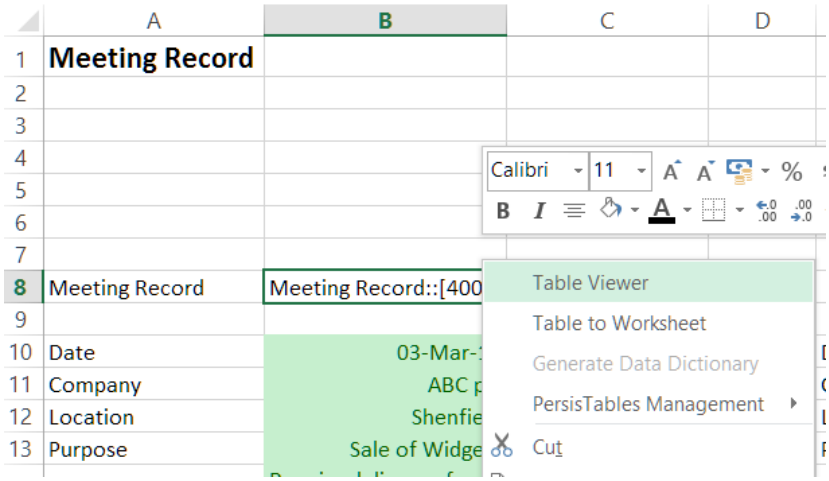
The Meeting Date is expressed as an integer when extracted from the Table. Excel converts date strings to a numeric value, these numbers are captured by the PersisTables Toolkit so are then displayed as plain numbers when extracted. It is easy to convert the cell to display the number as a date by formatting it using the {CTRL}# key combination.

The Table referenced in the extracted data are copies of the original data. Changing the data in these Tables will not affect the original data. Data changes flows from source to destination only.

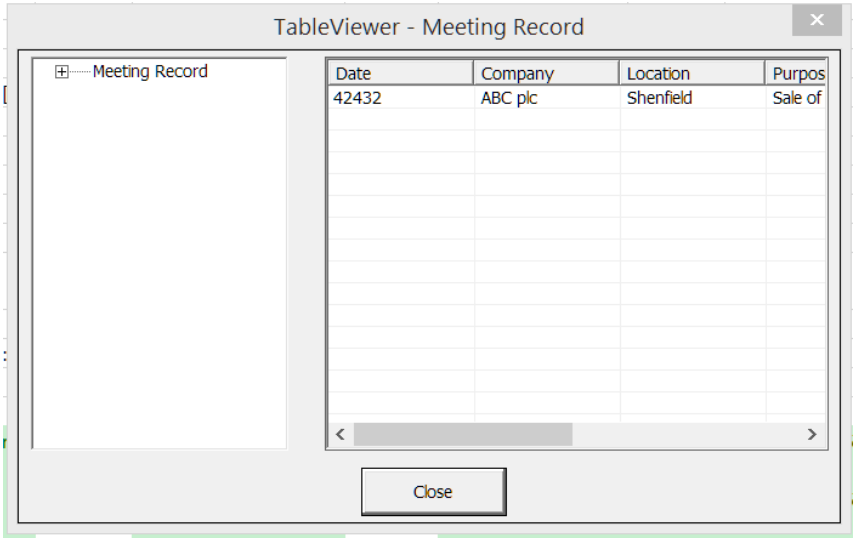
Table Viewer

While the TKTable_Get() function allows you to see the data held in a Table, it is awkward to use while analysing data in the process of creating an Excel worksheet. A simple visualiser is provided which allows you to view the Table contents associated with a Handle. This is invoked by selecting a cell which contains a valid Handle and right clicking on the cell. A context menu is displayed and near the top will be an option to invoke the Table Viewer.

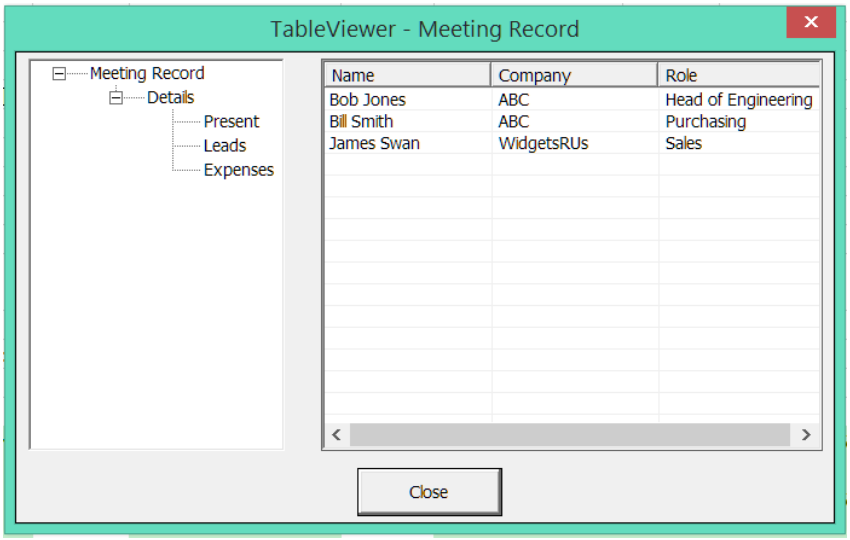
In the example below the user has right clicked on cell B8 to display the context menu.



Selecting the Table Viewer menu item will display a dialog box which shows the Table data represented by the Handle in the right hand window.



The Left hand pane shows the hierarchy of the object, clicking on the + sign opens up the lower levels of the hierarchy.

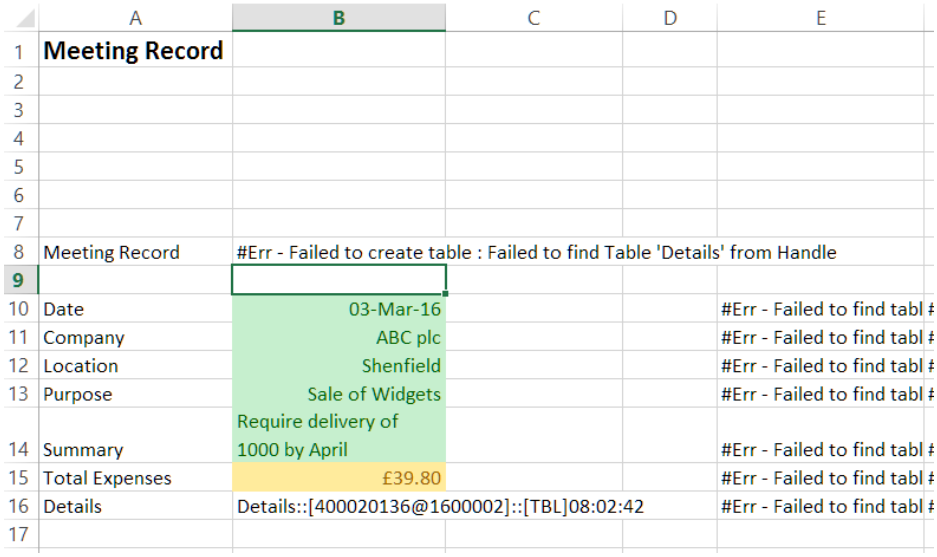


Clicking on different fields in the left pane will display the corresponding Table data in the right.

Note that when a spreadsheet containing PersisTables is reloaded the worksheet will look as if it contains Handles. Clicking on these may not produce the expected context menu, generally this is because the Handle does not correspond to a valid Table. Recalculating the cell (or worksheet) usually rectifies the situation[3].

Sometimes recalculating a single cell will generate an error.

[3] The data referenced by the TKTable_Create() function is only loaded when the function is executed. The function is only executed if a cell the function is dependent on is updated or the formula is recalculated manually using the [F2] key followed by [Enter].



	A	B	C	D	E
1	**Meeting Record**				
2					
3					
4					
5					
6					
7					
8	Meeting Record	#Err - Failed to create table : Failed to find Table 'Details' from Handle			
9					
10	Date	03-Mar-16			#Err - Failed to find tabl #
11	Company	ABC plc			#Err - Failed to find tabl #
12	Location	Shenfield			#Err - Failed to find tabl #
13	Purpose	Sale of Widgets			#Err - Failed to find tabl #
14	Summary	Require delivery of 1000 by April			#Err - Failed to find tabl #
15	Total Expenses	£39.80			#Err - Failed to find tabl #
16	Details	Details::[400020136@1600002]::[TBL]08:02:42			#Err - Failed to find tabl #
17					

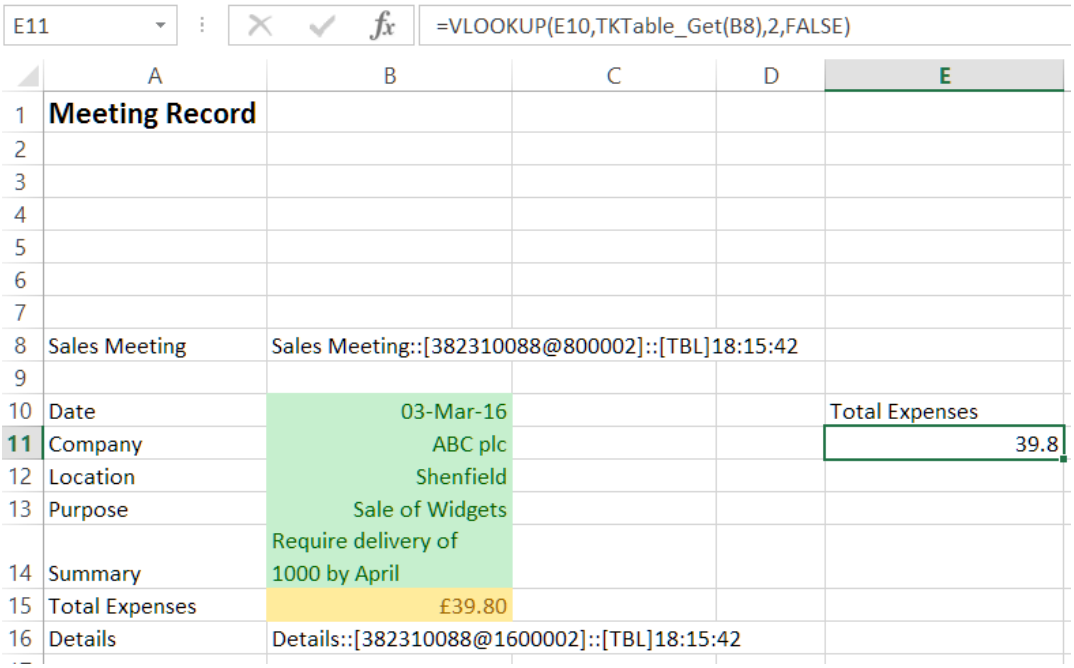
This is because the embedded Tables 'Details' has not yet been created. It can be recreated by recalculating the cell but then we will see the lower level Tables also need recalculating. One way to force all Tables to be regenerated is to recalculate the functions on the sheet (Press [SHIFT][F9] together) or recalculating the whole workbook (Press [SHIFT][CTRL][ALT][F9] together)[4].

Integrating with Excel functions

The TKTable_Get() function returns an array object representing the data in the Table. Certain standard Excel functions can accept the output from the TKTable_Get() function instead of selecting a range. Therefore it is possible to utilise existing Excel functions on Table data.

For example we may want to lookup data in a Table using the inbuilt Excel VLOOKUP function. This searches the first column of data in a range for a value and then returns the value in the specified column offset from that row.

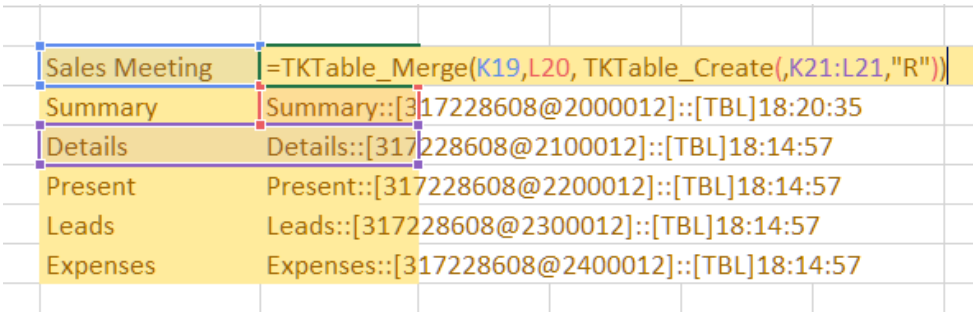
[4] See the FAQ for an explanation of why these Tables are not created when the workbook is loaded.



Here we have embedded the TKTable_Get() function within the VLOOKUP function (see the function bar) to find the Expense claim for the visit.

Other functions that can operate on the output from TKTable_Get() include mathematical functions that operate on ranges of data such as min(), sum(), average(), stddev()[5].

The presence of the Table Handles is a bit distracting we can move them away from the main data entry section of the spreadsheet.



To achieve this we have to create a Table representing the Meeting Summary (Cells A10:B15) and then merge these details with the Details Table created previously in Cell B16. TKTable_Merge() allows the Keys and data of one Table to be added onto the Keys of another Table. The resulting Table has the Keys for both Tables represented in it.

Persistence

Excel allows information used in a workbook to be saved for later use by saving the workbook. However when information is changed, saving the workbook will overwrite the original information unless the workbook is given a new name. This can result in the creation of multitudes of copies of

[5] More details are available in the Appendix

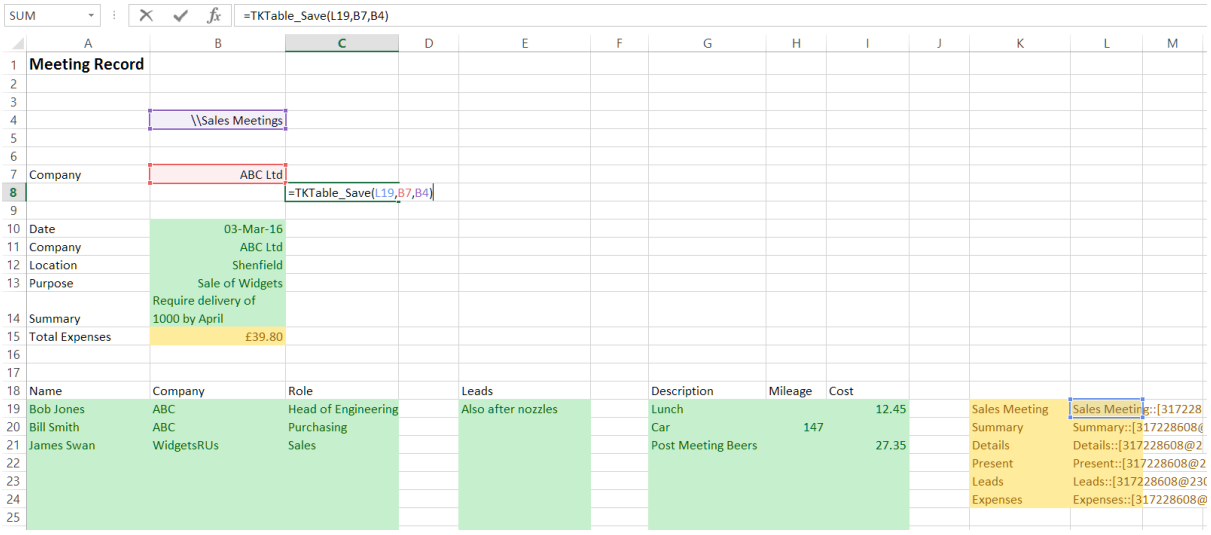
the original workbook, there is a risk of loss of important information and any changes required in a workbook has to be replicated across all the copies.

There are various workarounds to this, macro can be written to generate Comma Separated Value (CSV) files or the workbook can be integrated with a database. These general are fairly bespoke solutions, have limited flexibility to handle complex data sets and require significant technical input to setup and maintain.

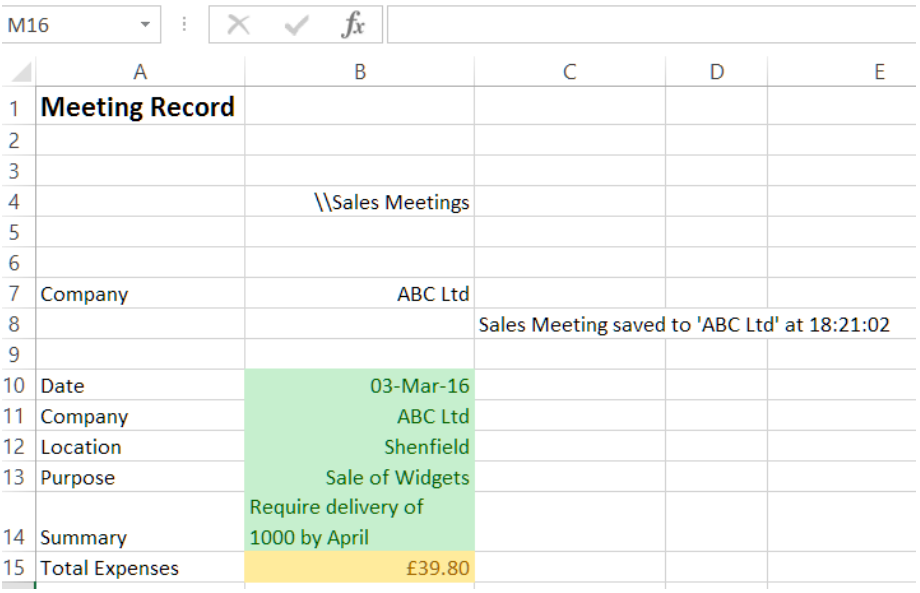
The PersisTables toolkit makes it very simple to save information (hence its name). Once data is captured into a Table it provides simple functions to save and reload data.

Save & Load

Saving data contained in a Table is done using the TKTable_Save() function. At its simplest this takes a Handle, a name for the Table to be saved and a Moniker which describes how and where the PersisTable should be saved. It saves the data held within the Table to a location based on the name and Moniker creating a PersisTable.

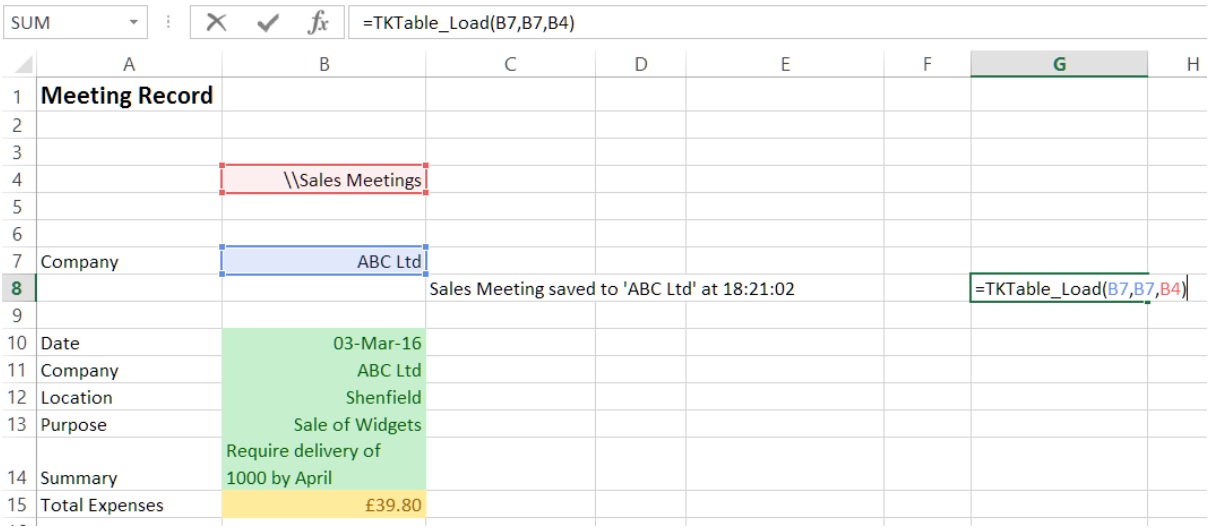


In this example the PersisTable takes the company name the meeting was held with and is saved using the Moniker "\\Sales Meeting". We will look at Monikers in more detail shortly. If the PersisTable is created the TKTable_Save() function will return a message indicating success.



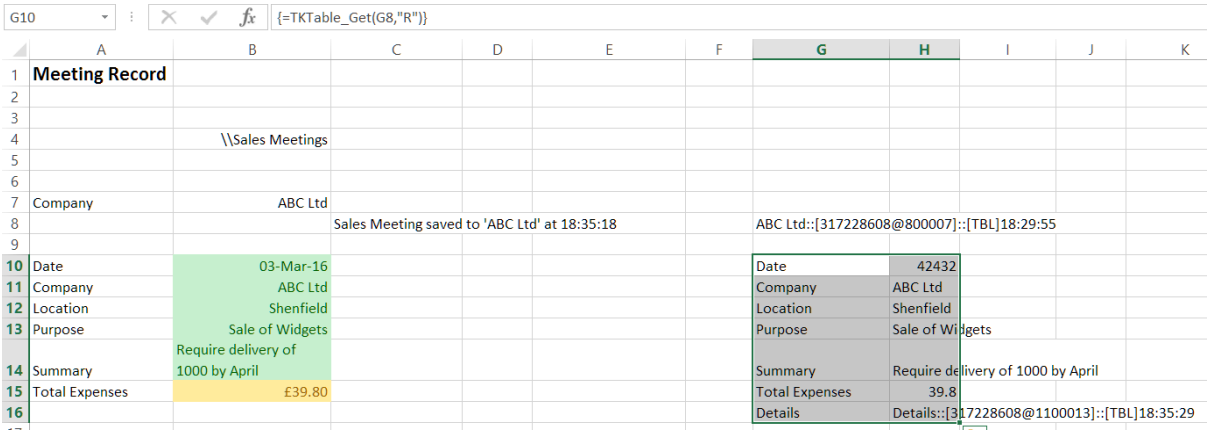
	A	B	C	D	E
1	**Meeting Record**				
2					
3					
4		\\Sales Meetings			
5					
6					
7	Company	ABC Ltd			
8			Sales Meeting saved to 'ABC Ltd' at 18:21:02		
9					
10	Date	03-Mar-16			
11	Company	ABC Ltd			
12	Location	Shenfield			
13	Purpose	Sale of Widgets			
14	Summary	Require delivery of 1000 by April			
15	Total Expenses	£39.80			

Of course, saving data is no good if it cannot be retrieved. The TKTable_Load() function reloads the PersisTable. It takes arguments giving the name of the Table to be created, the name of the PersisTable and the Moniker indicating where the PersisTable is stored.



	A	B	C	D	E	F	G	H
1	**Meeting Record**							
2								
3								
4		\\Sales Meetings						
5								
6								
7	Company	ABC Ltd						
8			Sales Meeting saved to 'ABC Ltd' at 18:21:02				=TKTable_Load(B7,B7,B4)	
9								
10	Date	03-Mar-16						
11	Company	ABC Ltd						
12	Location	Shenfield						
13	Purpose	Sale of Widgets						
14	Summary	Require delivery of 1000 by April						
15	Total Expenses	£39.80						

If successful the TKTable_Load() returns a Table handle which can be used in the same way that a handle created by TKTable_Create(). The data can be extracted using the TKTable_Get() function.



	A	B	C	D	E	F	G	H	I	J	K
1	**Meeting Record**										
2											
3											
4		\\Sales Meetings									
5											
6											
7	Company	ABC Ltd									
8			Sales Meeting saved to 'ABC Ltd' at 18:35:18				ABC Ltd::[317228608@800007]::[TBL]18:29:55				
9											
10	Date	03-Mar-16					Date	42432			
11	Company	ABC Ltd					Company	ABC Ltd			
12	Location	Shenfield					Location	Shenfield			
13	Purpose	Sale of Widgets					Purpose	Sale of Widgets			
14	Summary	Require delivery of 1000 by April					Summary	Require delivery of 1000 by April			
15	Total Expenses	£39.80					Total Expenses	39.8			
16							Details	Details::[317228608@1100013]::[TBL]18:35:29			

Here the data has been extracted from the loaded PersisTable and displayed on the worksheet. The Table Viewer can also be used to look at the data retrieved. Note that we use the ”Row” flag in the TKTable_Get() function to display the results in the same orientation as it was originally created[6].

We have successfully reloaded the data we saved onto the same sheet as it was created. It would be equally simple to load data in another worksheet by using the same Moniker and Name to retrieve it. On the other hand, if we specify a new Company Name then a new record of a meeting is created under the name of the new company.

Monikers, Genres & Categories

In our simple worksheet we have saved the Table using the Moniker ‘\\Sales Meetings’ and given the PersisTable created the name of the company the meeting was held with. If we change the name of the company we will save a separate record. We can then retrieve the different Meeting Reports into the same workbook just by modifying the company name.

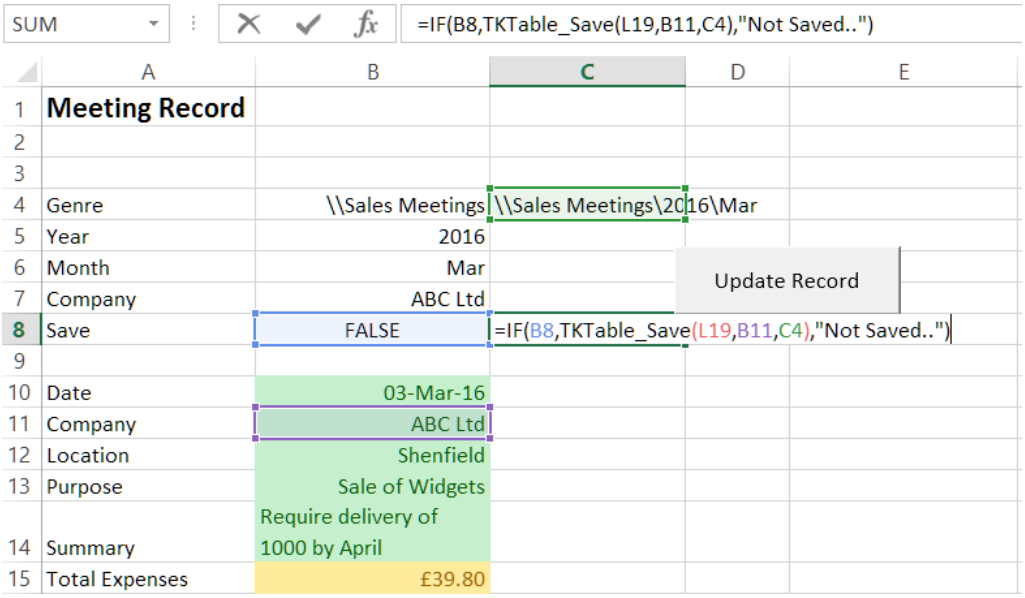
Where we have different types of information we can save them under different Monikers. It may be that we want to save Sales Targets for each sales person. This can be done by creating a spreadsheet that handles Sales Targets, creating the necessary Tables and saving it under the Moniker ‘\\Sales Targets’ using the Sales person’s name as the PersisTable name.

This basic Moniker is known as a Genre. It is always preceded by a ‘\\’ and allows us to divide our PersisTable data up by the type of information they contain. It is not necessary to define Genres before using them, the system will create Genres when a Moniker includes a new Genre name.

The obvious problem with our naming convention for Meeting Reports is that every time we have another meeting with the same company we will overwrite the previous record. This can be prevented by extending the Moniker to include one or more Categories that sub-divide the data held within the Genre.

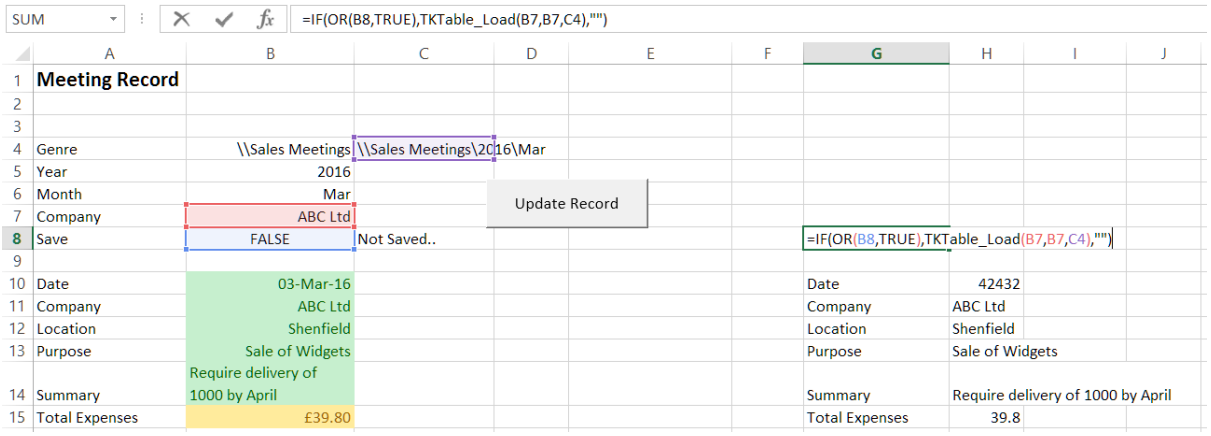
The choice of how to divide up the contents of a Genre into Categories depends on the most natural way to reference the PersisTables. For example, we could have a breakdown of Meeting Reports by company name and then hold all the records for one company under a Category named after the company. In this case we will specify Categories by Year and Month so we can easily identify the Sales Meetings performed each month.

[6] In fact, orientation is preserved even when a file is saved. In this case the Table to be saved had lost the orientation flag when the TKTable_Merge() function was used. It is not possible to provide orientation to this function; the resultant Table is always column oriented.



The Cell C4 contains the extended Moniker, each Category is preceded by a "\" character. The Year and Month element of the meeting has been extracted from the date and appended to the Genre.

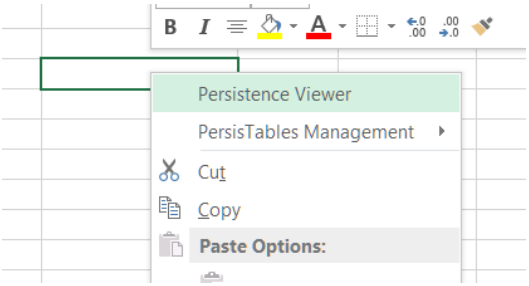
Of course the TKTable_Load() call must also be updated to use the revised Moniker if we want to access the same data set.



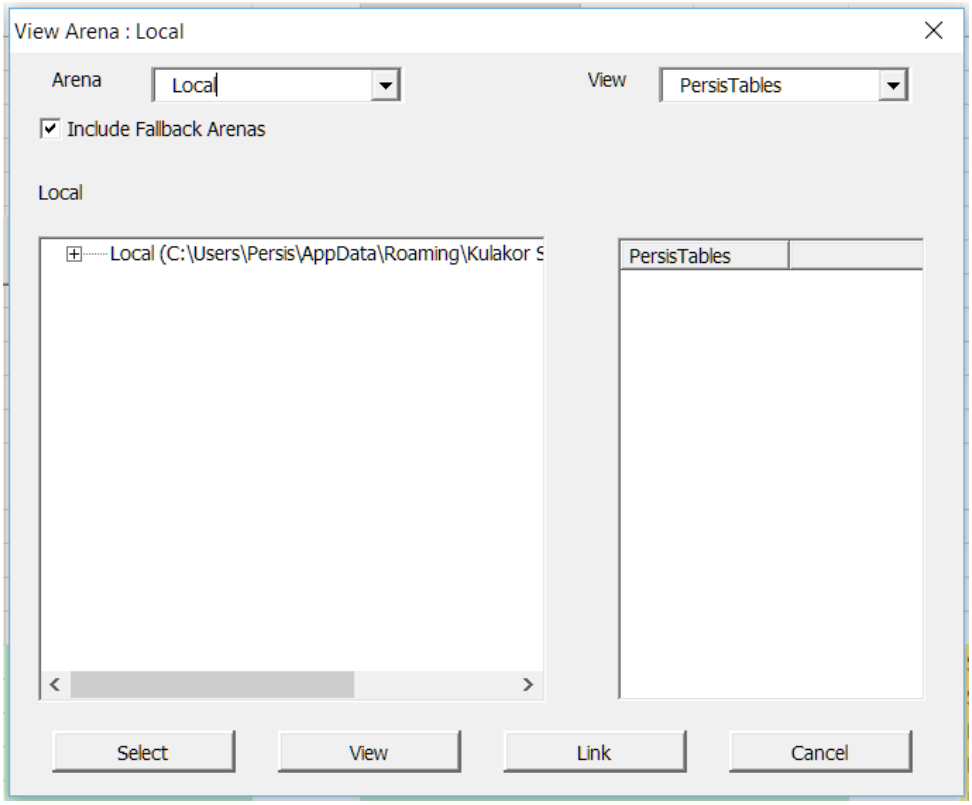
Persistence Viewer

We are now in a position to use the same workbook to save multiple meeting requests. Each meeting will be saved in the Sales Meeting Genre under the appropriate Category. However this creates the problem of knowing what information has been stored and retrieving the information to do further analysis.

The simplest way to do this is to invoke the Persistence Viewer. This is an application that can be launched from the context menu when the user Right Clicks on an empty cell.

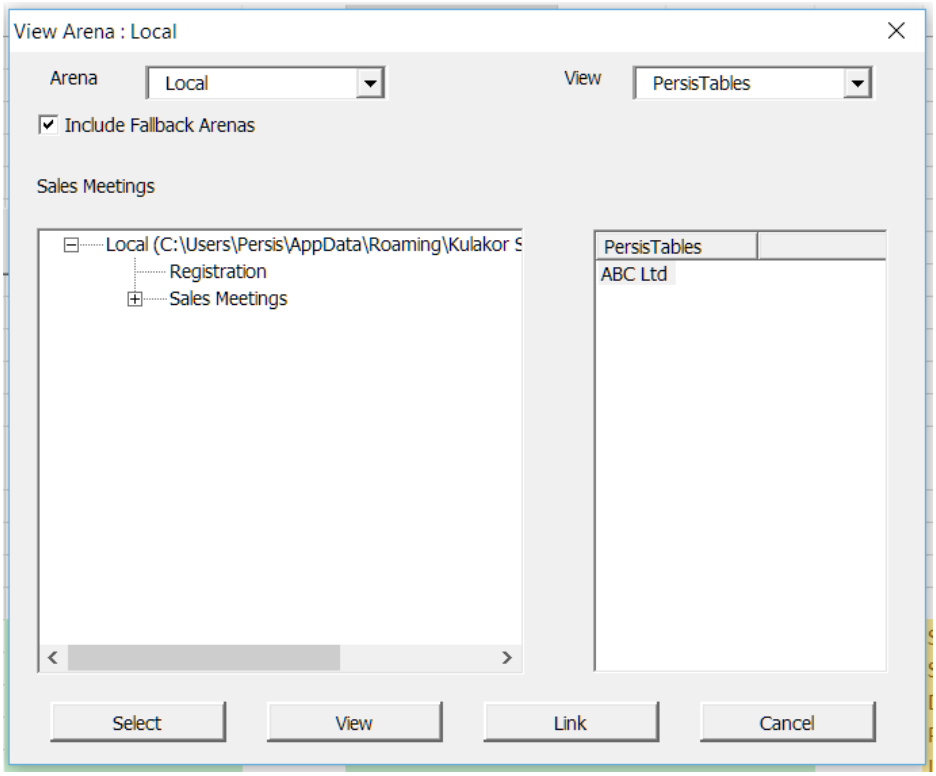


The Persistence Viewer displays the available PersisTable data. On opening the Viewer displays a top level view of the currently selected Arena. On initial installation there is just one Arena – named 'Local' which is located in the User's AppData area. The left-hand pane identifies the location of the Arena, the + next to it indicates that there are Genres beneath.

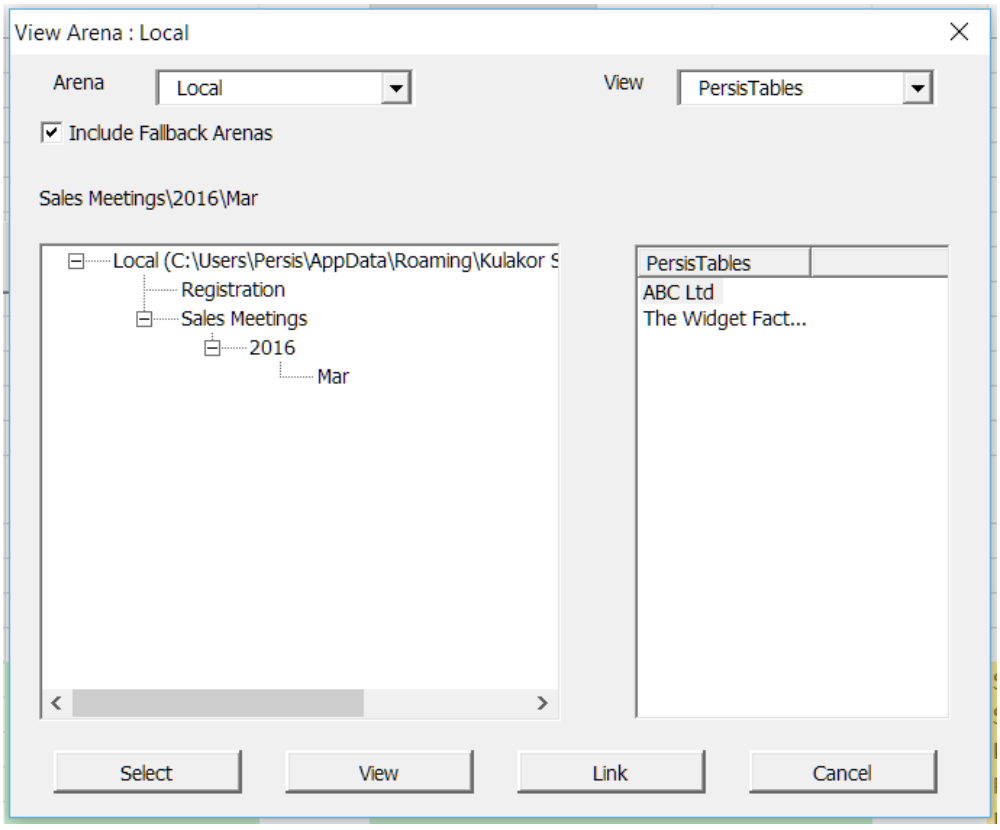


Clicking on the + opens up the Arena to show a list of Genres available within this Arena. We can see the 'Sales Meetings' Genre and the Right pane shows that our original Meeting Report we saved directly in the Genre is still present[7].

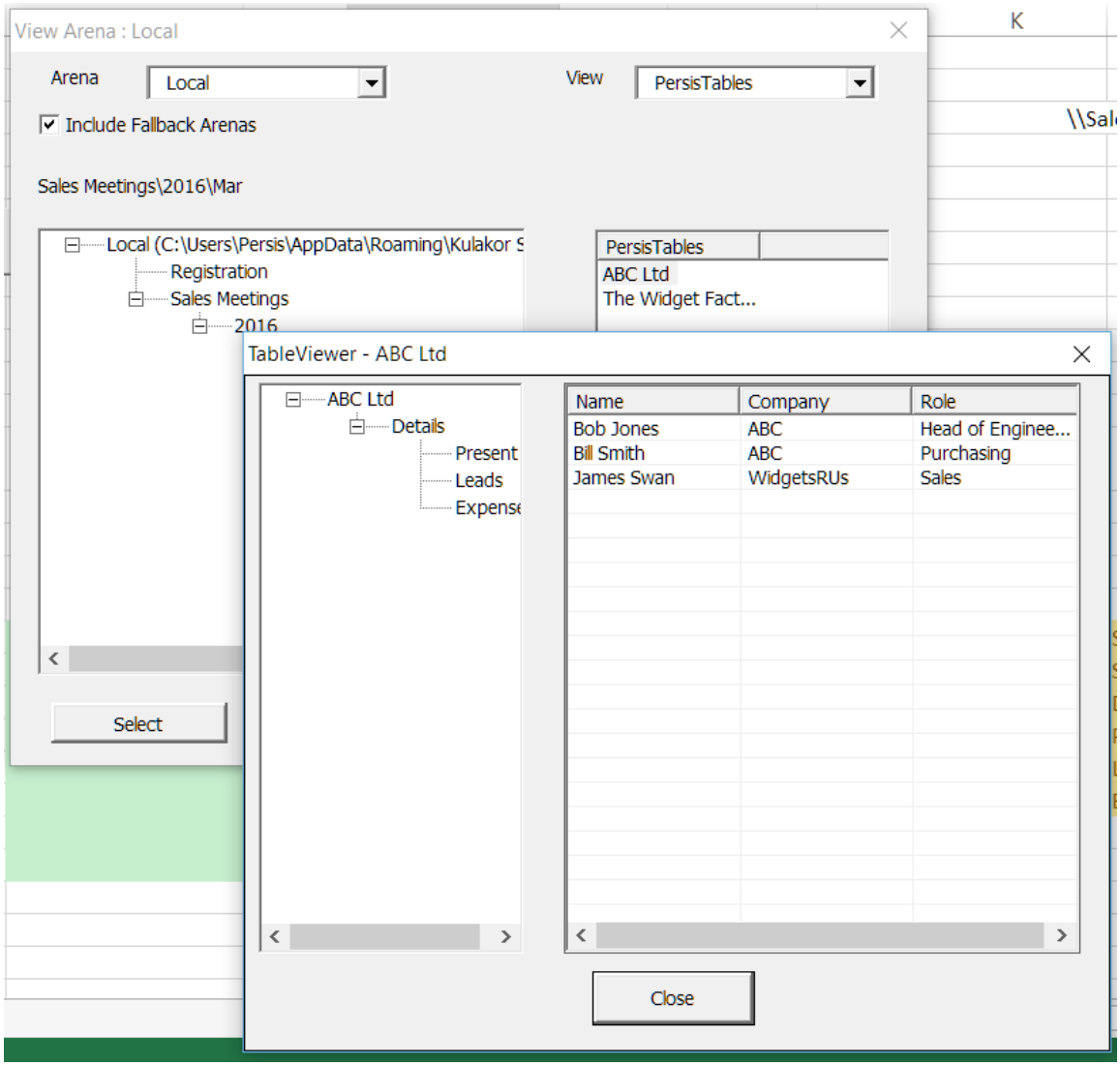
[7] The PersisTable toolkit does not provide a mechanism to delete PersisTables directly, however any PersisTable can be deleted using the File Explorer if the user has permission to delete the file. This is a feature of PersisTables to safe guard against accidentally deleting important data from Excel.



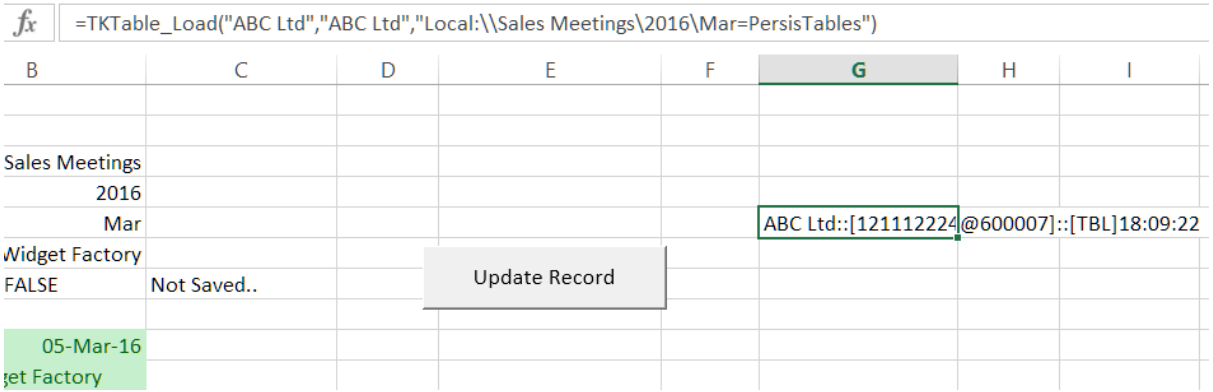
We can see a + next to the 'Sales Meetings' Genre which means we can drill down and see the Categories under the Genre and eventually we get to the lowest Category and can see the PersisTables we saved previously.



If we select one of the PersisTables in the right-hand pane we can View it and see its contents displayed in the Table Viewer.



Alternatively, we can 'Select' the PersisTable, this will paste the appropriate formula to load the PersisTable into the empty cell that the Persistence Viewer was invoked from.



The Persistence Viewer provides a mechanism for searching for and loading PersisTables that have been created over time. As the number of PersisTables saved within a Genre increases, it becomes

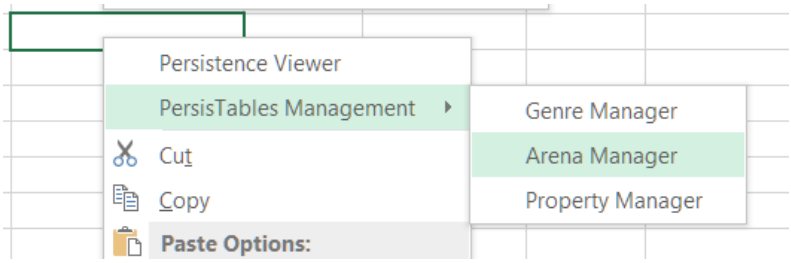
more obvious that a sensible method of dividing the PersisTables across Categories can make it easier to find a particular PersisTable.

Arenas

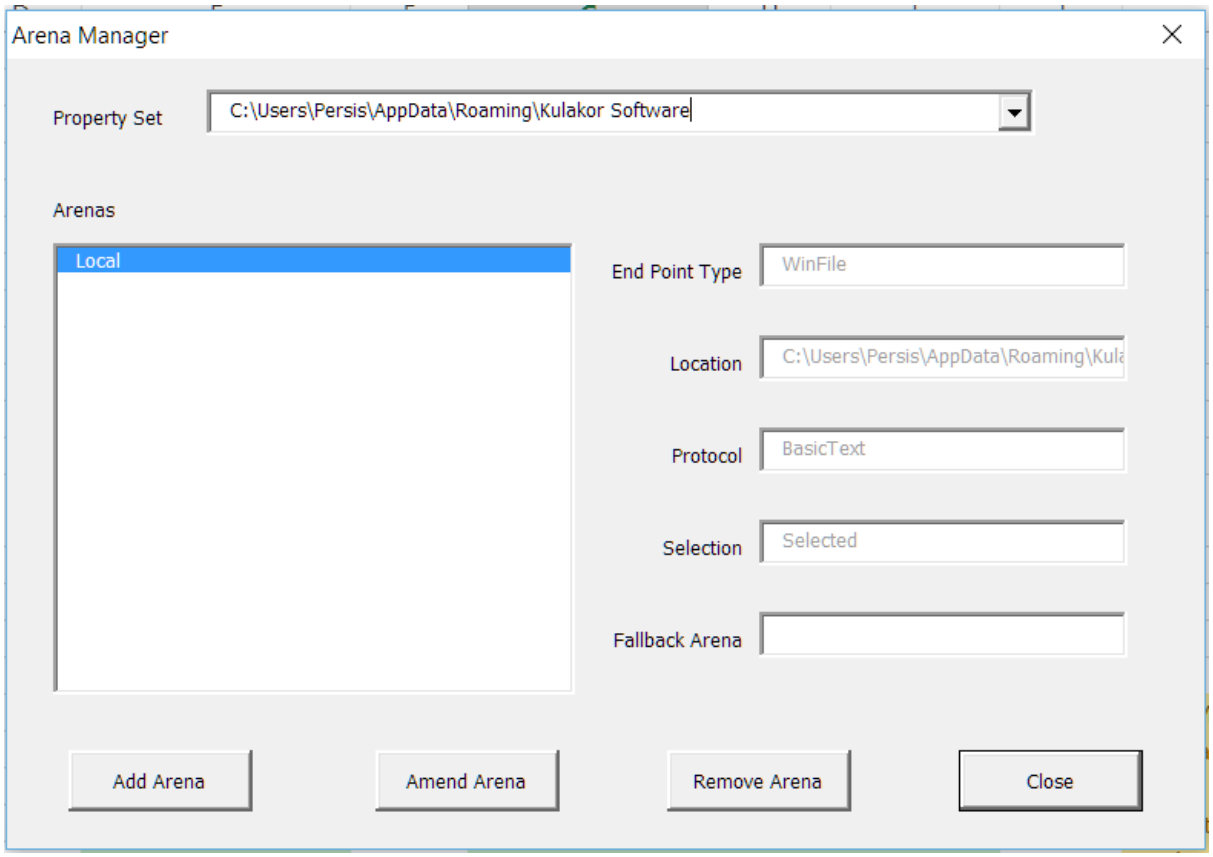
The Persistence Viewer introduces the concept of Arenas. Genres and Categories provide a mechanism for logically dividing up PersisTables however we need a place where the PersisTables exist across sessions. An Arena is a place where we can store PersisTables.

When the PersisTable toolkit is installed each user is provided with their own Local Arena. Since this is in the user's AppData directory it is only accessible to the User. This is not ideal if we want to share data. Fortunately, we can easily set up an Arena which allows data to be shared between Users.

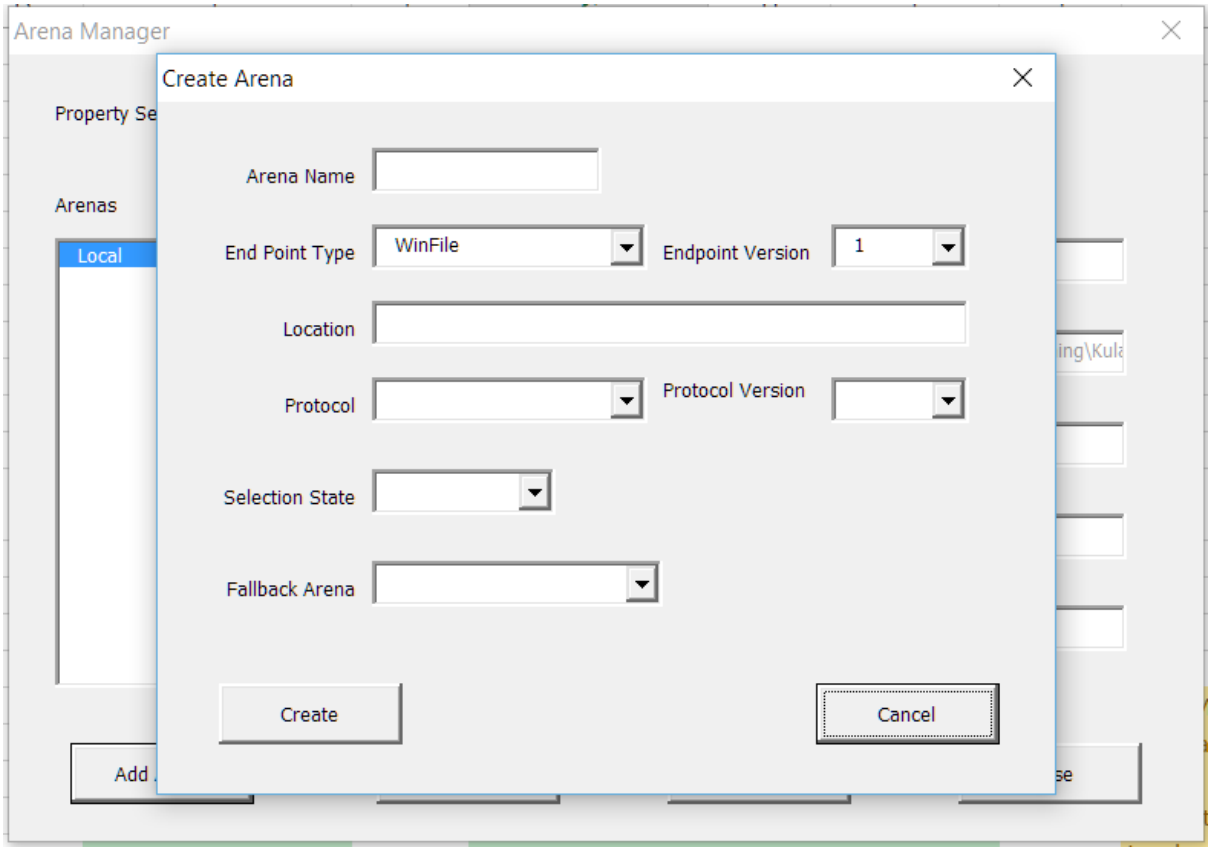
To do this we use the Arena Manager accessed through the context menu



This will launch the Arena Manager application.



Here we can see the Current list of Arenas, as we only have one, it is selected and the details are displayed on the right. It is straight forward to add a new Arena by clicking on the Add Arena button.



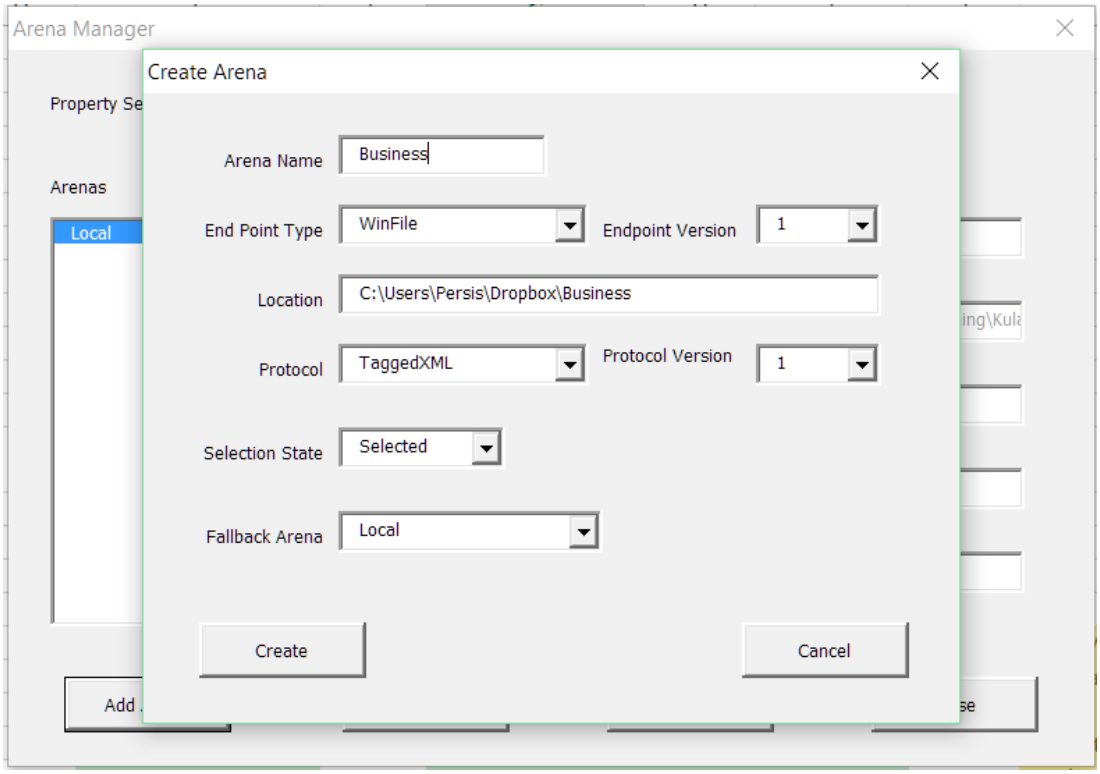
We need to provide a name for the Arena and the location where we want it to save PersisTables plus we need to select a Protocol. The Protocol defines the way that the data is stored in a PersisTable. For our new Arena we will use an XML Protocol but this really is not a critical issue, whatever Protocol is used we will get back the same Table information when we reload it.

A couple of other points, the Selection State allows the user to nominate the Arena that is used by default. If it is set to Selected then the Arena is used when saving a PersisTable without specifying the Arena in the Moniker. Defining a Fallback Arena means that PersisTables present in the Fallback Arena can be loaded if a PersisTable is not found in the current Arena, we will look at this in more detail shortly.

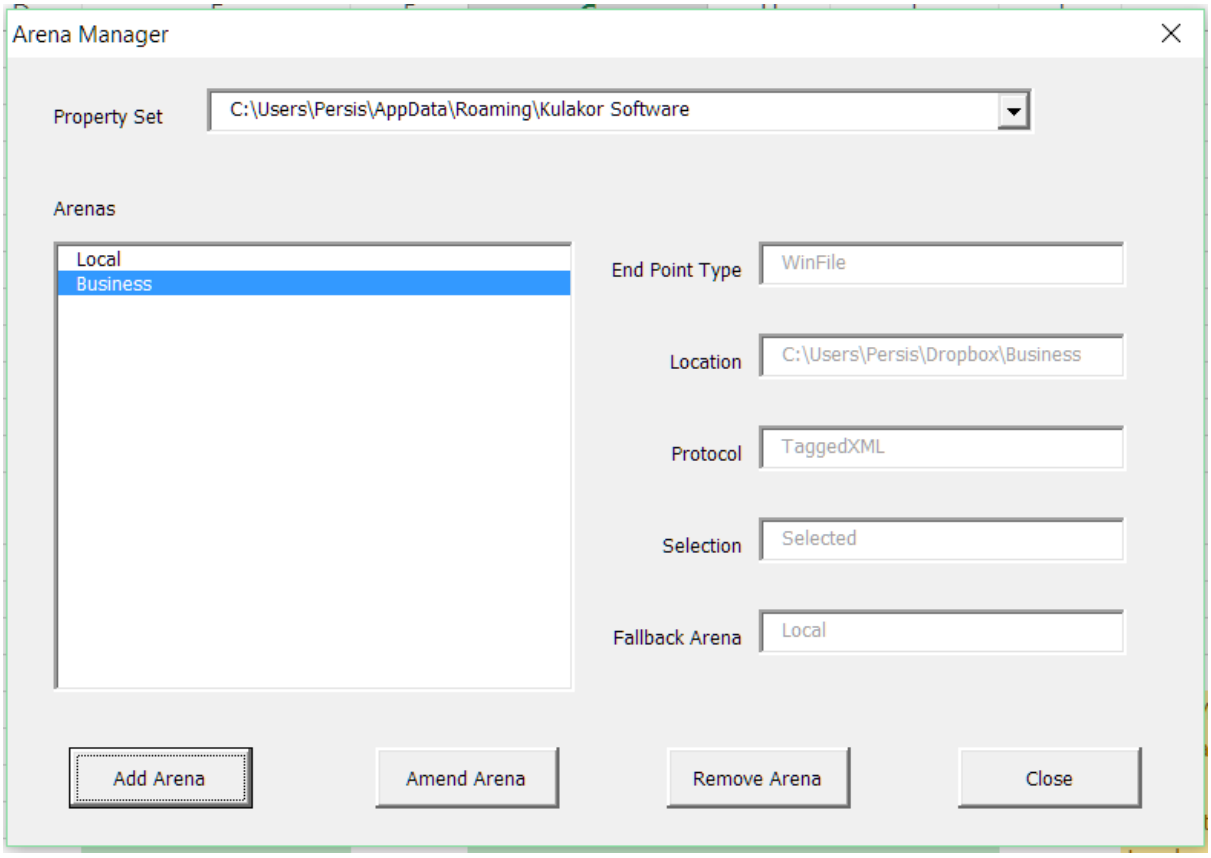
The completed Arena definition is shown below. We are creating a new Business Arena which will be used by default and it is located in a Dropbox account so anyone who has access to that account can retrieve the PersisTables saved there. The Arena is defined to fallback to the Local Arena so any PersisTables that have been saved in the Local Arena are visible in this Arena.

Note that for other users to access this Arena they will have to set up a similar Arena definition within their own Excel PersisTables installation. There are ways of sharing definitions using shared Property Files[8].

[8] For more details on defining Arenas and Property Files see the Configuration Guide



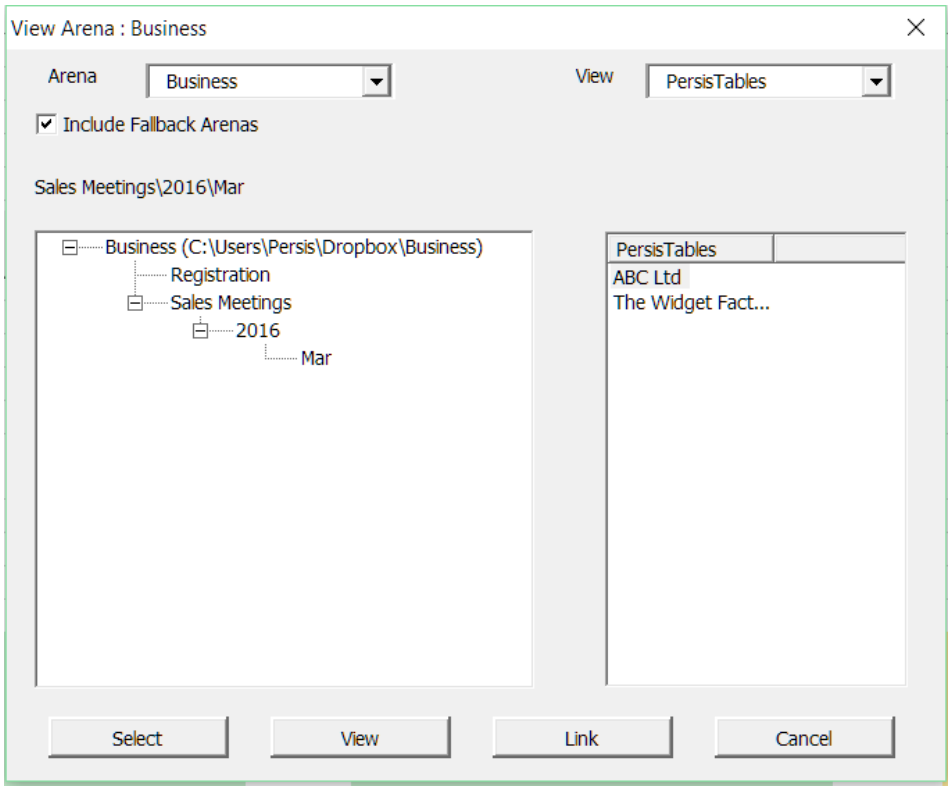
Once set up and we Create the Arena we can view the new Arena details in the main Arena Manager screen



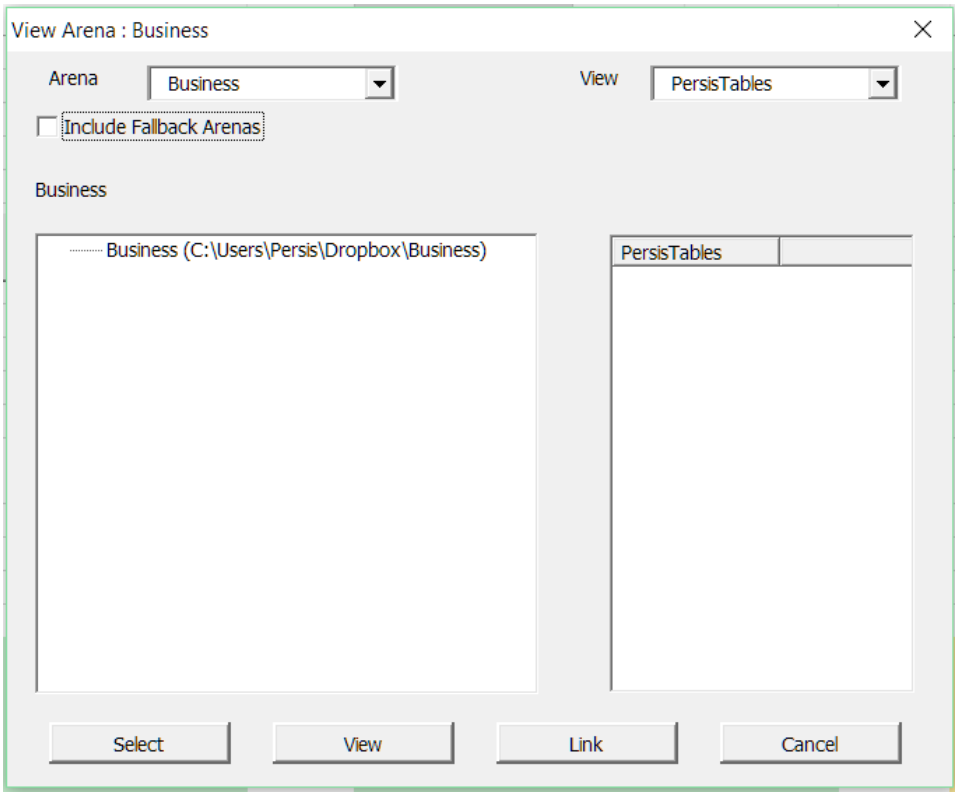
Since the new Arena is Selected, our Meeting Reports are automatically saved to the Business Arena. This does not mean we cannot access PersisTables held in the Local Arena. To access data in another Arena we need to modify our Moniker to specify the Arena we want to access. This is done by prefixing the Genre with the Arena name followed by a colon. So to amend the Sales Meeting Reports in the Local Arena we would specify the moniker as

Local: \\Sales Meetings\2016\Mar

However in our Business Arena we can automatically see PersisTables saved in the Local Arena as it is the Fallback Arena of the Business Arena. We can observe this behaviour in the Persistence Viewer.



Although we have not yet saved any data to the Business Arena we can see our original PersisTables in the Sales Meeting Genre. Clicking on the Include Fallback Arenas checkbox we see that these disappear.



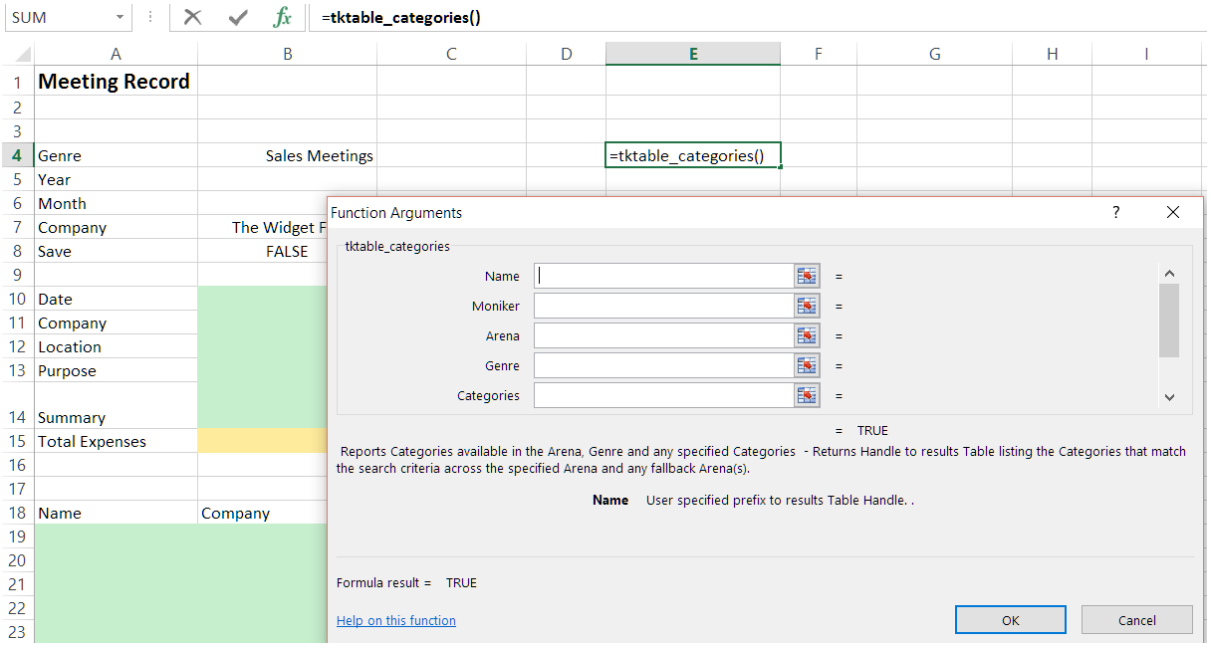
This feature can be useful as it allows read only access to important business information held in the Fallback Arena while new business functionality is being developed. A 'Test' Arena can be set up as the Selected Arena and this has the Business Arena as its fallback. When there is a need to access existing PersisTables held in the Business Arena it is accessible but any PersisTables created as part of testing will only be written into the Test Arena. Once a PersisTable exists in the Test Arena it will be loaded in preference to data in the Fallback Arena which has the same Genre/Category/Name.

Putting It Together

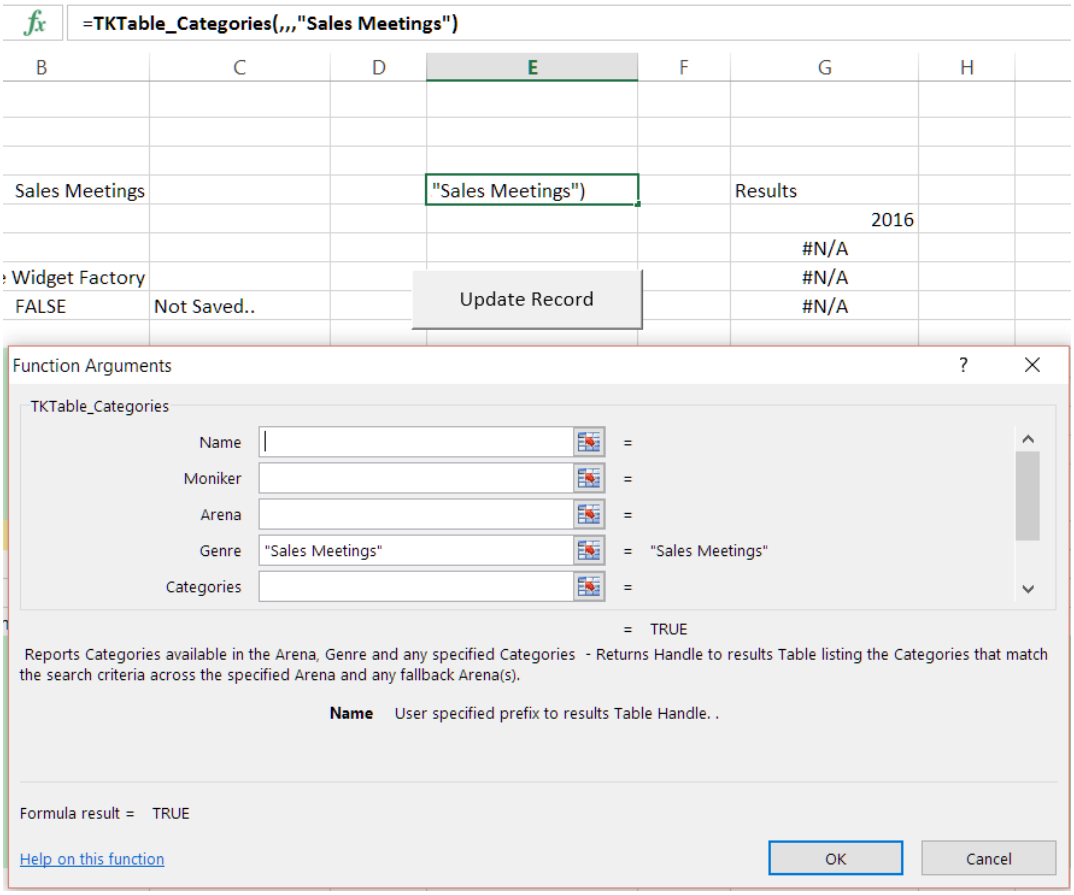
We now have the tools to capture complex data sets and reload them at a later date. We can develop a robust application in Excel where we can manage multiple sets of data from a single spreadsheet.

The first element is to be able to find an existing Meeting Report. While the Persistence Viewer allows us to explore the PersisTables in an Arena, we would really like to be able to directly select the category to locate the Meeting Report we are interested in. The tools used to create the Persistence Viewer are also directly available as worksheet functions.

We know that the Meeting Reports PersisTables are held in the Sales Meetings Genre but we will be adding new Categories as we cycle through the Months and Years. The PersisTables toolkit provides the TKTable_Categories() function to list the Categories found within a Genre.

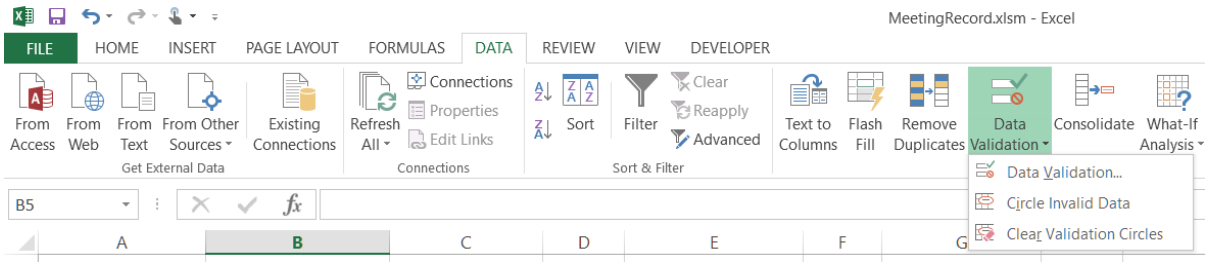


The function returns a Table of results, so the first argument allows the user to name the Table. The second argument can be a partial Moniker for example 'Business:\\Sales Meeting' and the function returns the list of Categories found immediately under the 'Sales Meeting' Genre. Alternatively the Arena and Genre can be specified separately. The final argument allows sub-categories to be listed by providing the Category path that has been explored so far.

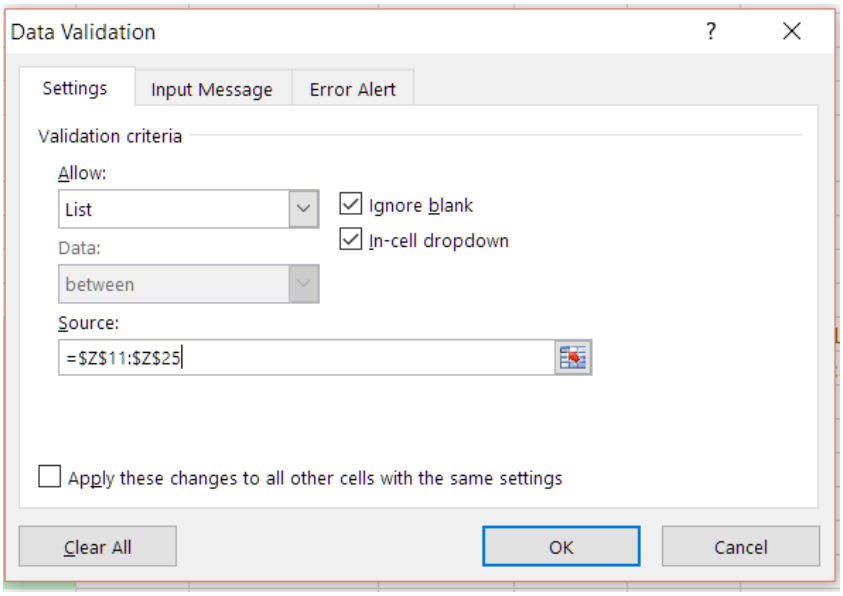


As the Selected Arena is Business we do not need to specify it to search it. The Genre name is given without its "\\" prefix. The results are show next to the function call and of course need to be extracted using the TKTable_Get() function.

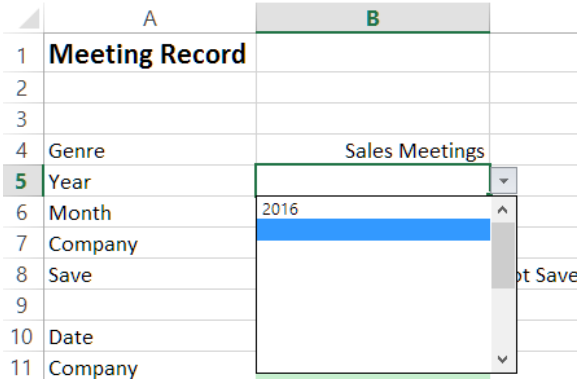
We can use the results to generate a drop listbox on the sheet from which we can select the Category we are interested in. This is done using the Data Validation Tool on the Data ribbon.



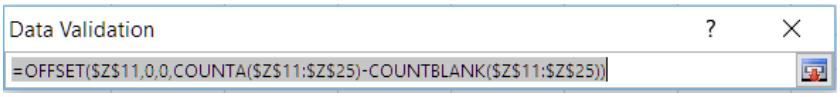
Selecting the Cell B5 and then clicking on Data Validation… will open a dialog which allows the type of validation to be selected. By selecting List form the Allow drop down and entering the range of cells containing the results extracted from the TKTable_Categories() function we can create a drop down list containing the available Categories.



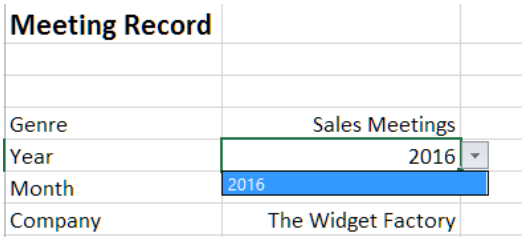
Ensure that when extracting the values from the results Table that empty Cells are set to Blank.



Since we do not necessarily know how many Categories will be returned we need to extract multiple rows from the Categories result Table, this leaves us with many empty lines in our drop down listbox. We can be a bit cleverer than this by creating a dynamic range for the Data Validation Source data. Entering the Excel Offset formula allows us to dynamically size the array used to populate the list box.

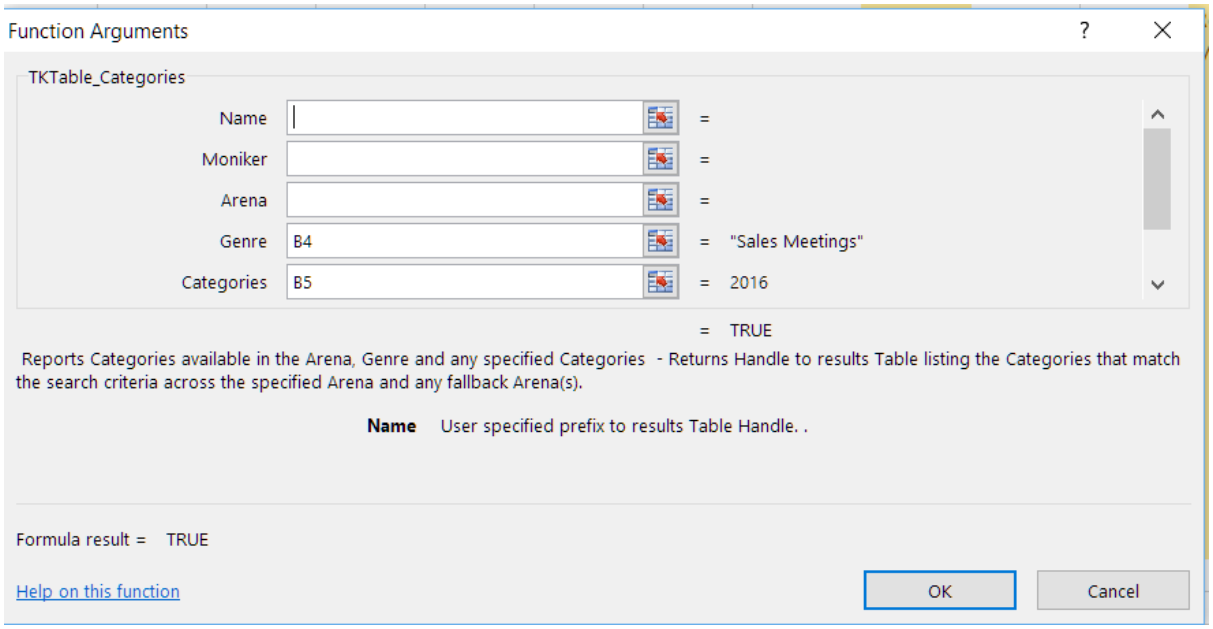


The first Category appears in Cell Z11 and the range where Categories can be extracted to is from Z11 to Z25.



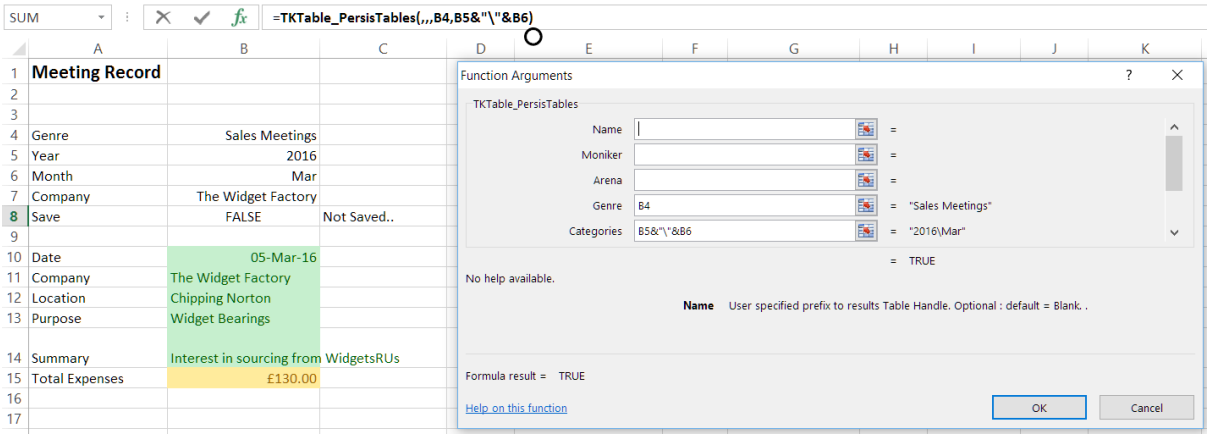
The list now contains only the values returned by the TKTable_Category() function.

Adding the selected Category to the Category argument in the TKTable_Category() function allows us to get a list of Months that have Meeting Reports.

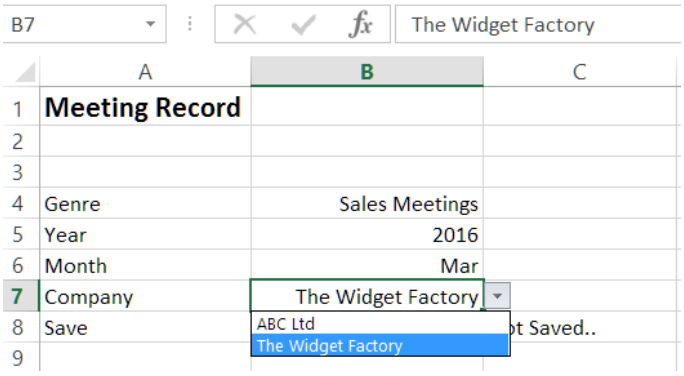


Again the list of available Categories can be listed using the data validation tool.

In order to complete the process of selecting a specific PersisTabe to load we use the TKTable_PersisTable() function which takes similar arguments to the TKTable_Categories() function but returns a list of PersisTables found at the given location.



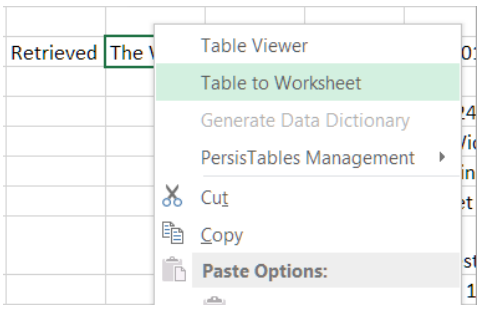
We concatenate the two Category levels together using the "\" character to separate them. The results are again displayed in a listbox.



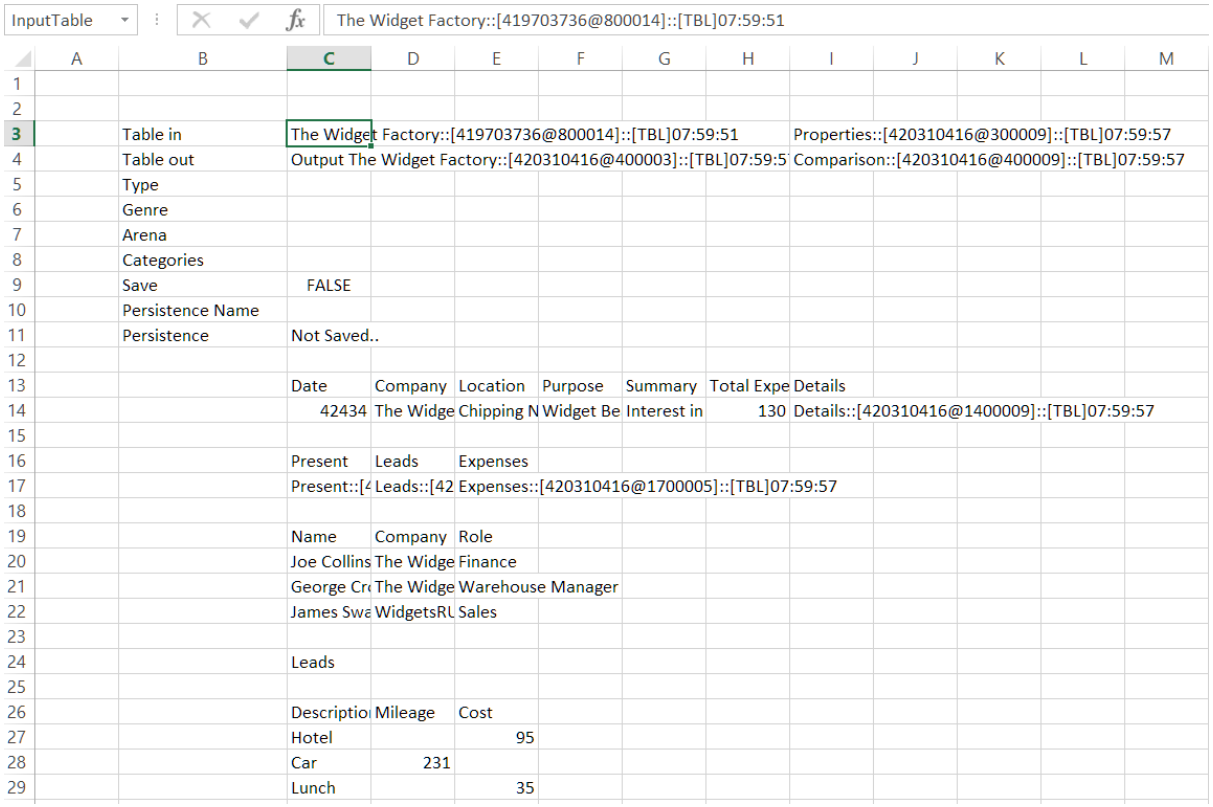
We can now select any Meeting Report by selecting the Year and Month of the Meeting and finding the PersisTable named after the Company we are interested in.

There are similar functions that allow us to list the available Arenas – TKTable_Arenas() – and enumerate the Genres - TKTable_Genres() - within an Arena. These are described in the PersisTables Reference Guide.

Having retrieved the PersisTable we can view it in the Table Viewer however this does not allow us to modify the information within it. A quick way to access the data in any Table is to use the 'Table to Worksheet' command found when right clicking on a cell containing a valid Table Handle.



This option will create a new worksheet at the back of the current workbook with the contents of the Table laid out on the new sheet.



As can be seen it is not elegant but the data can be modified directly on the sheet, a new Table is created with the changes and a comparison of the original and current version is shown in the cells below Cell I6.

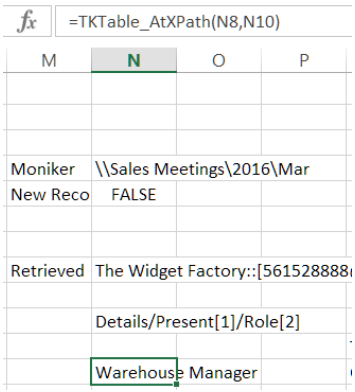
Table in	The Widget Factory::[419703736@800014]::[TBL]07:59:51						Properties::[420310416@300009		
Table out	Output The Widget Factory::[561540488@400003]::[TBL]18:04:0						Comparison::[561540488@40000		
Type									
Genre							Xpath	Left	Right
Arena							/Details[1]	231	245
Categories									
Save	FALSE								
Persistence Name									
Persistence	Not Saved..								
	Date	Company	Location	Purpose	Summary	Total Expe	Details		
	42434	The Widge	Chipping N	Widget Be	Interest in	130	Details::[561540488@1400009]::		
	Present	Leads	Expenses						
	Present::[4	Leads::[42	Expenses::[561540488@1700005]::[TBL]18:04:06						
	Name	Company	Role						
	Joe Collins	The Widge	Finance						
	George Cr	The Widge	Warehouse Manager						
	James Swa	WidgetsRU	Sales						
	Leads								
	Descriptio	Mileage	Cost						
	Hotel		95						
	Car	245							
	Lunch		35						

If the appropriate Arena/Genre/Category and Name are entered the Table can be saved by setting the Save value to TRUE.

While a feasible way to maintain Meeting Reports it is not really effective. Since the format of the Meeting Report is known we can utilise the worksheet we created it with to also edit a Meeting Report.

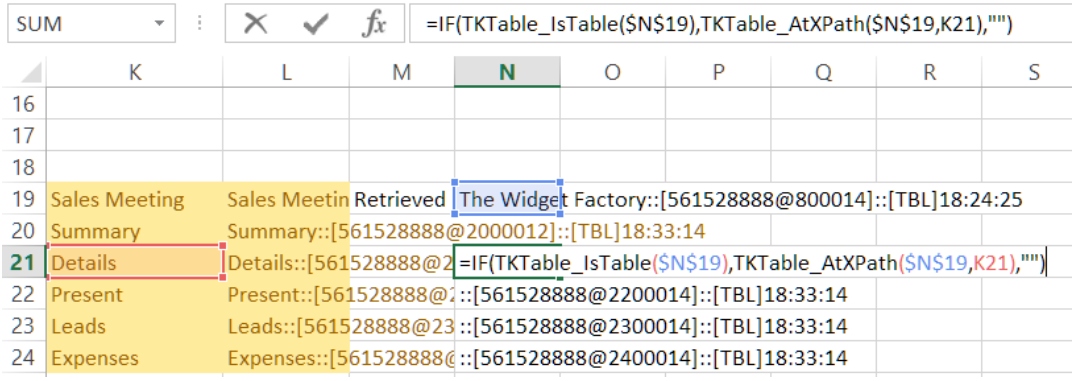
The first step is to unpack the Tables embedded within the Meeting Report Table. This allows us to extract the data from each Table separately. To do this we introduce a new function, TKTable_AtXPath(). This function allows us to extract a value from within a Table by specifying its 'XPath'. An XPath describes a location within a Table or an embedded Table. It consists of a series of Key:Offset pairs, each Key:Offset accesses an element of a Table. If the value found by a Key:Offset pair is a Table then the next Key:Offset pair will access a value within this embedded Table.

Each Key:Offset pair is delimited by the '/' character. The Offset is optional if the first value below a Key is required. The offset is wrapped by the '[' and ']' characters.



Here we have extracted the Role of the second person listed in the Table of those who are present which is embedded in the Details Table which itself is itself embedded in the Main Table under the Details Key. Note that the first row is automatically extracted when we do not specify an Offset.

We will restructure the worksheet so that the extraction of data from a retrieved PersisTable reflects the way the Tables used to construct the PersisTable in the first place.



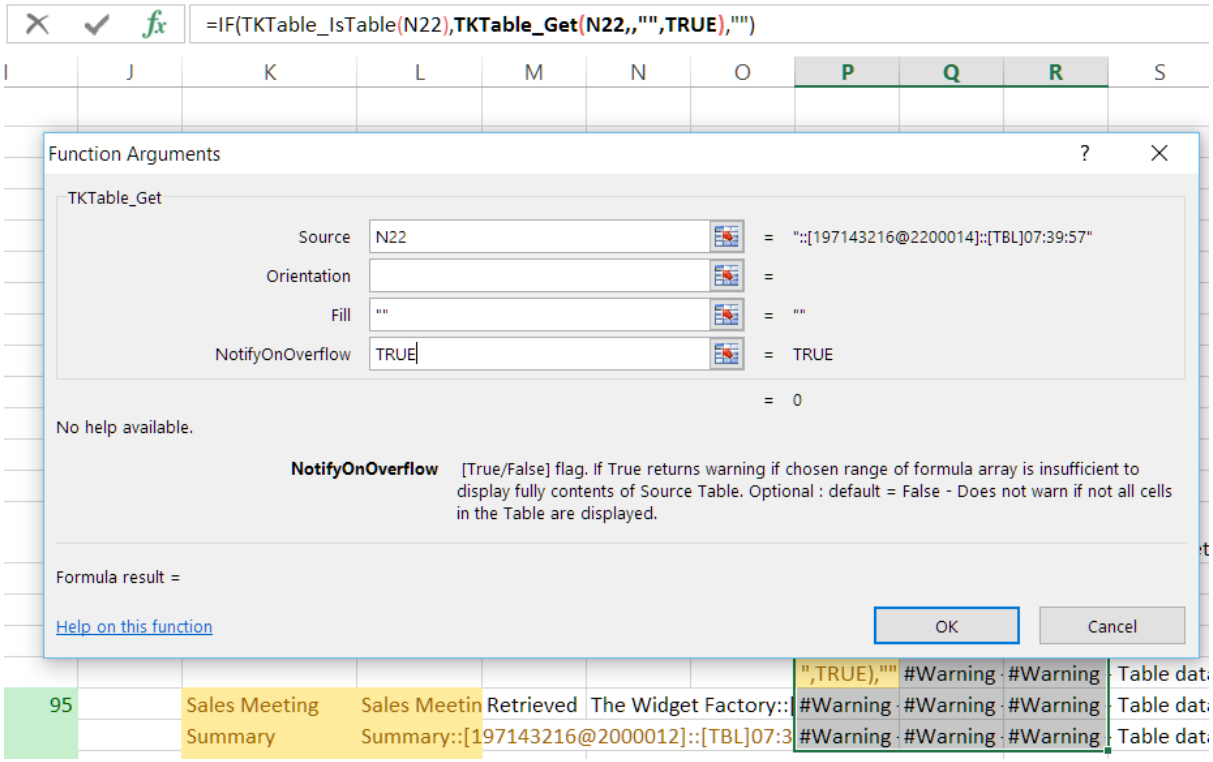
First we extract the Details Table using the same tag as was used to create the original Table. Note that we have used the TKTable_IsTable() function to decide if we have successfully retrieved a Table in the first place. This function lives up to its name, returning the Boolean value TRUE if the cell contains a valid Table Handle[9]. The next 3 cells down extract the component Tables from the Details Table using the Keys used to label them in the construction of the original Table.

We could have accessed the component Tables by using the TKTable_Get() function in the usual way, this method guarantees the order we extract data so we always get the Table Handles for the Present, Lead and Expenses Tables in that order. It is possible that an earlier version of the Meeting Report worksheet used a different ordering when creating the original Table.

The contents of each Table is extracted using the TKTable_Get() function. The range of cells allocated to the extracted data should be at least as large as the Range used to capture that data in the first place otherwise some data will not be displayed. The NotifyOnOverflow argument with

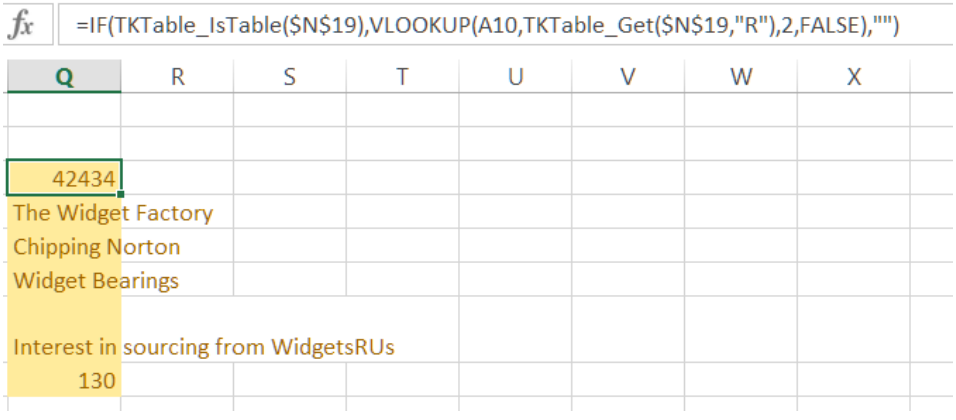
[9] A valid Table Handle is one which is associated with an active Table. When reloading a spreadsheet which contained Tables, the Handles will be stale until they have been recalculated. Stale Handles will return FALSE with TKTable_IsTable() even though they appear to be validly formed Handles.

TKTable_Get() will ensure that if there is insufficient space to display all the data held in the Table then a Warning is generated.



Here we have not allocated sufficient space for the data in the Present Table so all data has been flagged as a Warning message.

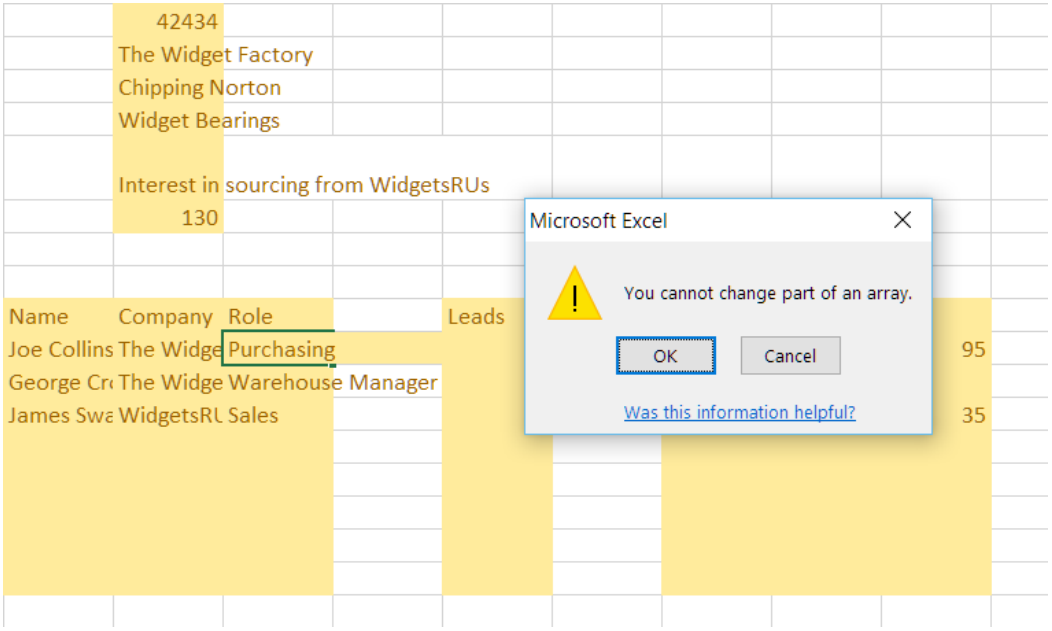
For the primary Table a slightly different approach has been used, the data is extracted using the Excel VLOOKUP() function. It is operating on an array that is specified by the output from TKTable_Get().



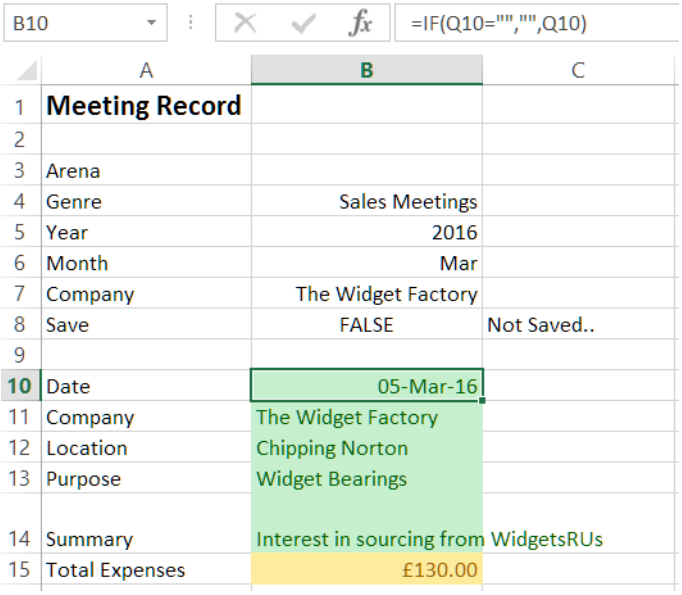
We use the Heading from the original Table to perform the lookup on the data extracted from the TKTable_Get() function and take the second column value where there is a match. This technique is useful on large data sets where individual values need to be extracted[10].

[10] The observant will have noticed the need to specify the use of the Row Oriented flag when extracting the data. Even though the original data was saved ass Row Oriented and the TKTable_Get() function respects this

Having laid out the data retrieved from a PersisTable we may think we are able to amend the data. Unfortunately as it consists mostly of data extracted through Array functions any attempts to change directly will fail.



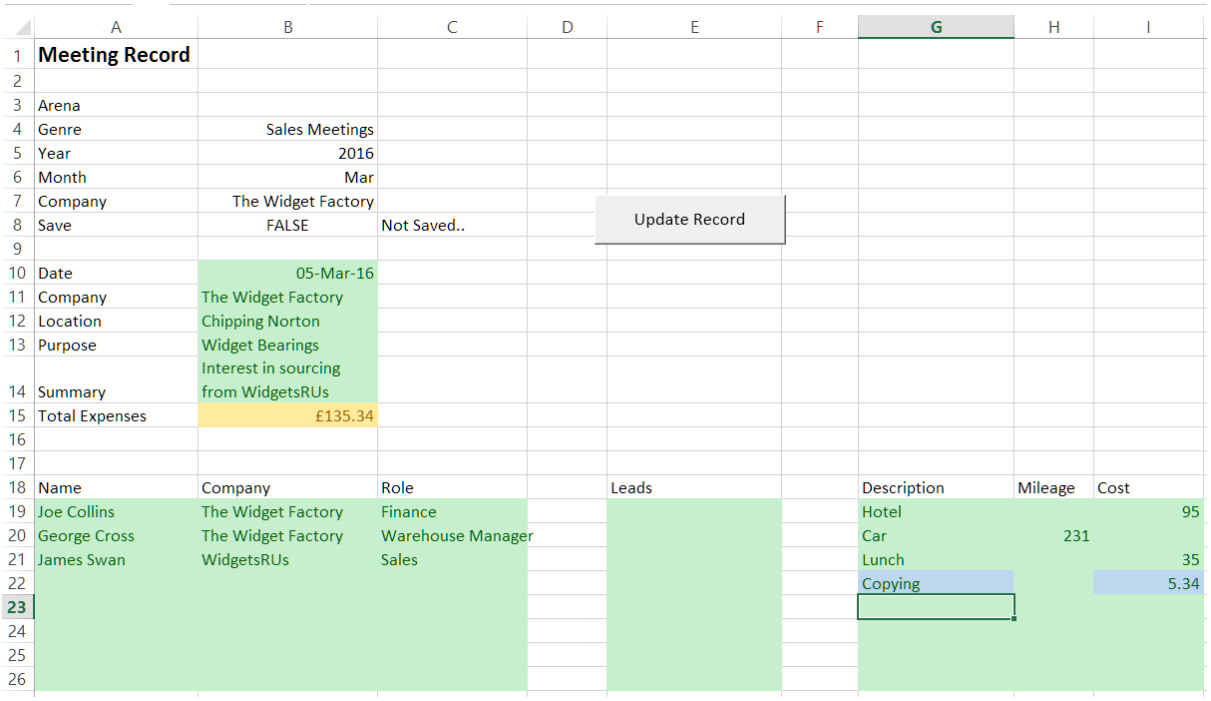
To overcome this we copy the data back to the original input area using formula in the input cells. Typically this will be of the form



The reason the test is in place is that Excel interprets a blank cell that is referenced by another as having the value of 0. To avoid having the blank cells represented by 0 we test explicitly for a blank and display blank if it is true.

when extracting data to the sheet, it does not do this automatically when data is extracted internally to an Excel function. Alternatively we could use HLOOKUP() instead.

We can enter a similar formula in all of the green data entry cells and we can now reload, view and amend an existing Meeting Report. We have arranged the worksheet so the mechanics of retrieving the PersisTable data is held on the right-hand side of the worksheet so the user just sees the data entry area. Note we still retain the formula to sum up the expenses since if we alter the expense amounts we want the new amount captured in the updated Table.



	A	B	C	D	E	F	G	H	I
1	**Meeting Record**								
2									
3	Arena								
4	Genre	Sales Meetings							
5	Year	2016							
6	Month	Mar							
7	Company	The Widget Factory							
8	Save	FALSE	Not Saved..		Update Record				
9									
10	Date	05-Mar-16							
11	Company	The Widget Factory							
12	Location	Chipping Norton							
13	Purpose	Widget Bearings							
14	Summary	Interest in sourcing from WidgetsRUs							
15	Total Expenses	£135.34							
16									
17									
18	Name	Company	Role		Leads		Description	Mileage	Cost
19	Joe Collins	The Widget Factory	Finance				Hotel		95
20	George Cross	The Widget Factory	Warehouse Manager				Car	231	
21	James Swan	WidgetsRUs	Sales				Lunch		35
22							Copying		5.34
23									
24									
25									
26									

Here we have some additional expenses and added it to the list of expenses. This has automatically updated the total expenses amount. Note also that the background of the cells he has changed is different. We have added conditional formatting to change the colour of a cell where the formula in the cell is overtyped.

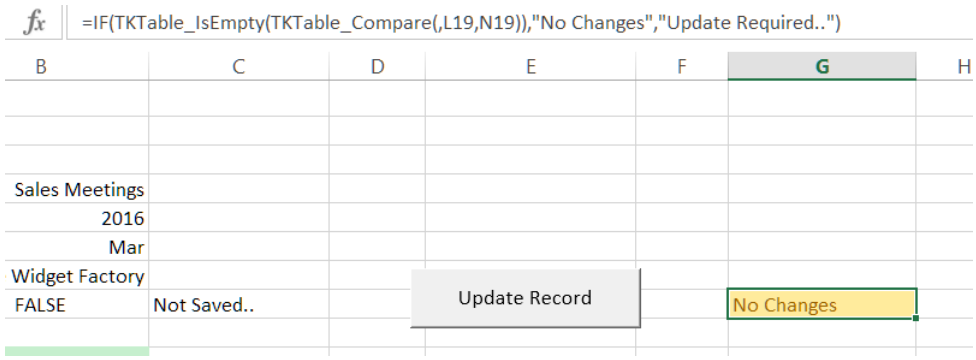
A common problem with Excel is to lose a formula by overtyping it and then saving the spreadsheet with the formula missing. Highlighting the cell where the formula has been overtyped reduces the risk of the lost formula becoming a permanent bug on the spreadsheet. The appendix Conditional Formatting of Formula Overwrite describes how to do this.

Where a formula has been over-typed it is possible to replace the lost formula by copying it from an adjacent cell. However a more robust solution is to make the workbook a read-only workbook[11] so it is not possible to save the workbook with an overwritten formula. We can do this because no information is retained in the workbook when it is saved. The data is all held in PersisTables in the selected Arena.

Of course we need to make sure that the Table is persisted before we close the worksheet, the TKTable_Compare() function will return a Table of the differences between two Tables. We can use

[11] The simplest way to do this is to use Explorer to find the workbook, set the properties on the file so it is read-only.

this and the utility function TKTable_isEmpty() to identify when the Table created on the sheet does not match the PersisTable retrieved.



Here we check the results returned from TKTable_Compare() to see if the Table contains any data, if it has then we need to update the PersisTable store with the new Table if we wish to capture those changes.

	A	B	C	D	E	F	G	H	I
1	**Meeting Record**								
2									
3	Arena								
4	Genre	Sales Meetings							
5	Year	2016							
6	Month	Mar							
7	Company	The Widget Factory							
8	Save	FALSE	Not Saved..		Update Record		Update Required..		
9									
10	Date	05-Mar-16							
11	Company	The Widget Factory							
12	Location	Chipping Norton							
13	Purpose	Widget Bearings							
14	Summary	Interest in sourcing from WidgetsRUs							
15	Total Expenses	£136.34							
16									
17									
18	Name	Company	Role		Leads		Description	Mileage	Cost
19	Joe Collins	The Widget Factory	Finance				Hotel		95
20	George Cross	The Widget Factory	Warehouse Manager				Car	231	
21	James Swan	WidgetsRUs	Sales				Lunch		35
22							Copying		6.34
23									

Modifying the expenses entry for copying, flags the record as modified and that the PersisTable requires updating to retain the changes. Pressing 'Update Record' will save the change and reload the modified record. The worksheet will update and the flag will indicate "No Change". Note the Copying expense is still a directly entered value so the formula must be copied from the cell above or below using Copy and Paste commands.

We now have a Worksheet that allows records to be amended but if we want to create a new record we do not want to see the information from the last record loaded. We can overcome this by clearing any of the Year/Month/Company cells in B5:B7. Another VBA function could be defined to do this, attached to a button to create a New Record.

An alternative is to identify when the user has entered new details in the date or company fields and to hide the retrieved data based on the data entered into Cells B5:B7. This is a bit more complicated because any attempt to make the loading of the record dependent on anything that identifies the difference will result in a circular reference error.

The way forward is to make the display of this data in the green cells dependent on whether the retrieved record matches the details found in the Company and Date cells B10:B11. This is a bit more complicated to manage and means that the formulae in Cells B10:B11 is different from the other green cells.

H3 =IF(OR(B10="",B11=""),TRUE,NOT(AND(YEAR(B10)=B5,TEXT(B10,"mmm")=B6,B7=B11)))

	A	B	C	D	E	F	G	H
1	**Meeting Record**							
2								
3	Arena						New Record	FALSE
4	Genre	Sales Meetings						
5	Year	2016						
6	Month	Mar						
7	Company	The Widget Factory						
8	Save	FALSE	Not Saved..		Update Record		No Changes	
9								
10	Date	05-Mar-16						
11	Company	The Widget Factory						
12	Location	Chipping Norton						

The New Record flag is set to True when either the Date or Company details do not match what is displayed in the Year/Month/Company cells. All the green cells except the Date and Company include the "New Record" flag in their formulae which display the retrieved data

B12 =IF(OR(Q12="",H3),"",Q12)

	A	B	C
1	**Meeting Record**		
2			
3	Arena		
4	Genre	Sales Meetings	
5	Year	2016	
6	Month	Mar	
7	Company	The Widget Factory	
8	Save	FALSE	Not Saved..
9			
10	Date	05-Mar-16	
11	Company	The Widget Factory	
12	Location	Chipping Norton	
13	Purpose	Widget Bearings	
14	Summary	Interest in sourcing from WidgetsRUs	
15	Total Expenses	£136.34	

Retrieved data is only displayed if the retrieved record matches the currently displayed Date and Company.

The Meeting Report worksheet is now more or less complete. It allows the Sales team to capture the details of meetings, review them at a later date and amend them as necessary. The information captured is easily shared between members of the Sales team through the use of a common Arena. No data is retained in spreadsheets.

There are enhancements that can be made, the list of categories and PersisTables is only updated when the worksheet is recalculated, making the TKTable_Category() functions that populate the

drop down list box depend on the Update flag allows these to be repopulated with new Categories when a new record creates them.

Managing Data

Of course capturing information is only useful if we can do something with that information at a later date. In this section we will look at various ways the data captured can be analysed.

Monikers

We have seen Monikers in use when saving and loading PersisTables. A Moniker coupled with a Name is effectively the index to a PersisTable. This has to be unique for each PersisTable, reusing a Moniker and Name to save a PersisTable will overwrite the existing PersisTable associated with that Moniker. Of course if the information within a PersisTable changes then it is reasonable to update an existing PersisTable.

A Moniker consists of 4 parts, an optional Arena name, a mandatory Genre, an optional set of Categories and an optional set of control parameters. If o Arena is specified then the default Arena (as defined in the Properties) is used. If no Categories are specified then the Moniker will point directly at the Genre. The control parameters allow the user to modify the behaviour of the Load and Save functions.

The format of a moniker is

[<Arena>:]\\<Genre Name>[\<Category>]^{N}[<control parameters>]

Where

[] indicates an optional element
<> indicates a value that needs to be entered
N indicates a parameter that may be repeated

Arenas

The Arena must have been previously defined in the user's Property Set[12]. This can be done using the Arena Manager described above. Each user will need to define the Arenas they want to access individually or share a common Property File which includes a set of shared Arenas.

An Arena is a physical location that will be used to store PersisTables. It must exist prior to setting up a reference in the Property Set. At present all Arenas are located on the file system accessible from the user's computer.[13]

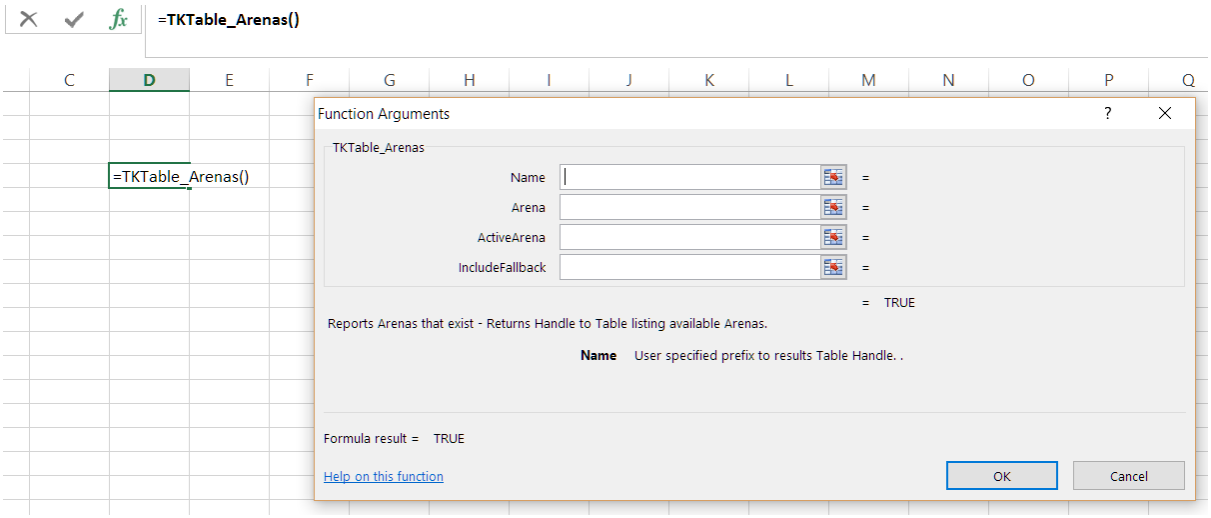
By marking an Arena as 'Selected' in the Arena Manager, the Arena will be used when no Arena is specified in a Moniker or where the name of an Arena is an optional parameter. Only one Arena

[12] The Property Set is the combination of Properties held in the user's local Property File and any other Property Files that the local Property File references either directly or indirectly (through further references to Property Files in referenced Property Files). Where duplicate definition of Properties are found in the Property Set the value in the Property File closest to the user's local Property File is used.
[13] The PersisTable toolkit uses plug-in code components to extend its functionality. Alternative types of 'End Points' can be handled with other EndPoint plug-ins. These would allow PersisTable data to be saved in relational databases or XML storage.

within a Property File can be marked as Selected and if multiple Property Files contain Selected Arenas then the Selected Arena is the one closest to the user's local Property File.

An Arena can be configured to have a nominated Fallback Arena. The Fallback Arena is searched if a requested PersisTable cannot be found in the Arena when attempting to load it into Excel. PersisTables are never written to the Fallback Arena. This allows a simplified testing process for changes to Excel applications. Instead of copying all information from a Production Arena to a test Arena and then checking the operation of the application, the user can set up a test Arena (as the selected Arena) and define the production Arena as the Fallback Arena. This will allow the Excel application to read any data required from the Production data set but only write back modified data to the test Arena.

The list of available Arenas can be found using the TKTable_Arenas() function.



This returns a Table containing a list of Arenas known to the current Excel session. If an Arena name is given, then the list will be limited to that Arena and all of its Fallback Arenas. Specifying the ActiveArena as "Selected" (and Arena is left blank) will list the currently Selected Arena.

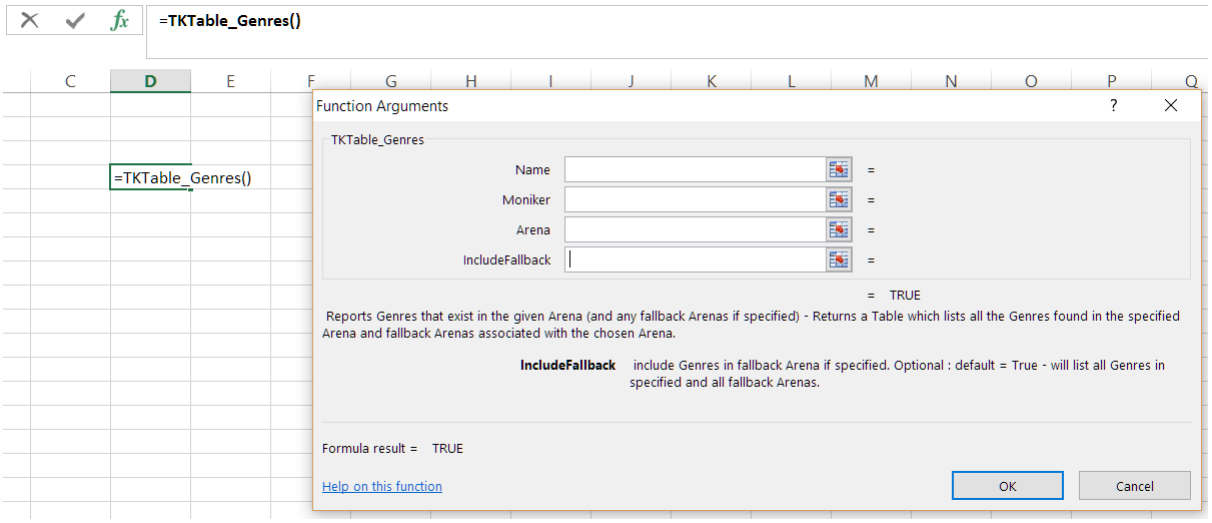
Genres & Categories

A Genre allows PersisTable data to be divided into the types of information they contain. We can think of this as a logical division of business information into different segments. For example we would probably want to keep our Meeting Reports in a separate Genre to holiday calendars or office bookings.

A Genre does not have to be pre-defined. Specifying a new Genre name in a Moniker when saving a PersisTable will automatically create the Genre in the Arena specified as long as the user has the right to create data in the Arena. Where data is stored in the file system this requires that the user has write permissions on the directory where the Arena is located. Of course if the user does not have write permission then they will only be able to read PersisTable data[14]. This allows a network administrator to provide security for sensitivity data created using the PersisTables Toolkit.

[14] Assuming the user has permission to view the directory the Arena is located in

The division of information by type makes it more intuitive to know where to find specific PersisTables containing information of a particular type. The TKTable_Genres() function provides a list of Genres available within an Arena.

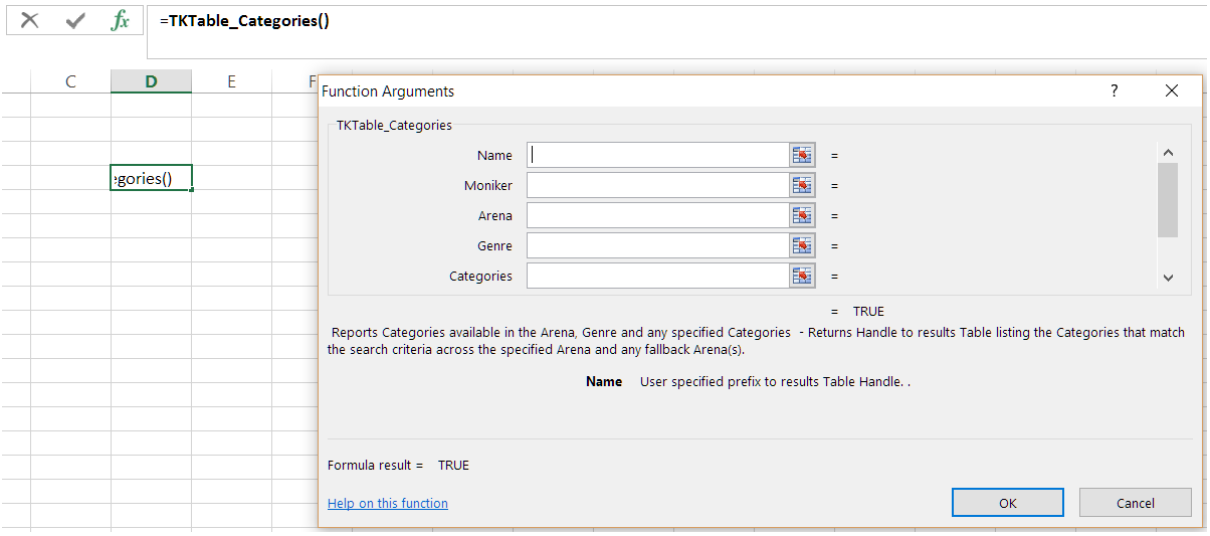


This function will generate a Table listing the Genres accessible from the specified Arena. If no Arena is given then the Selected Arena is used. If a Moniker is provided it will use the Arena element of the Moniker to identify he Arena to search. Where the IncludeFallback parameter is set to True, the list of Genres includes the union of Genres in the specified Arena and its Fallback Arenas.

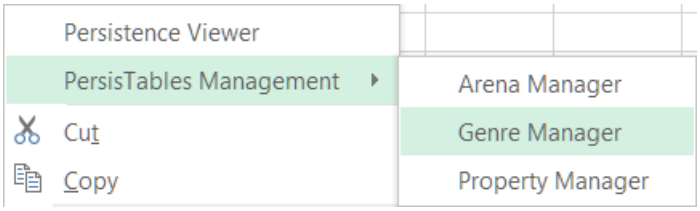
The Persistence Viewer allows the contents of a Genre to be viewed and a PersisTable loaded into a workbook, however if a large number of PersisTables are saved into a Genre then it may become difficult to locate the specific PersisTable that is required. The Genre can be divided into Categories to sub-divide the data into more easily queried chunks.

We have seen that Meeting Reports have been divided by year and month that they took place in. Again as long as the user has write permission on the Arena, new Categories will be created when required.

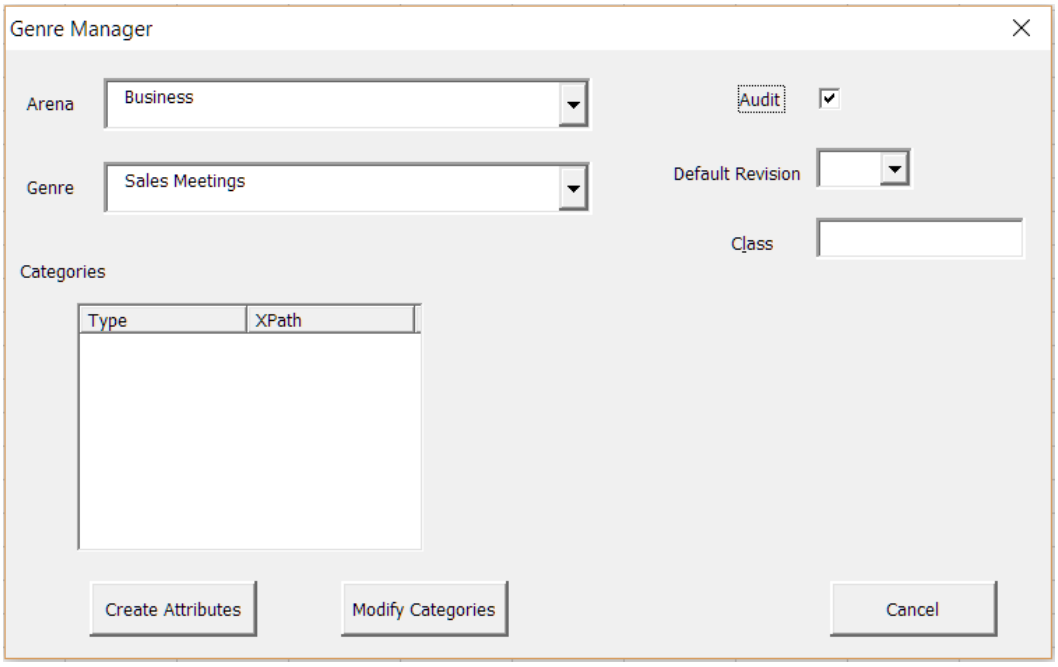
The TKTable_Categories() function allows the user to list the Categories within a Genre or the sub-categories within a Category. It allows the user to specify a Moniker from which it will extract the Arena and Genre details and if the moniker includes Category information will use this as the base Category from which further sub-categories are enumerated. Alternatively the Arena/Genre/Category information can be specified separately. Note that sub Categories are delimited by the use of the '\' character.



A Genre within an Arena allows some control to be provided over the PersisTables stored within it. The Genre Manager is a utility that manages this control.

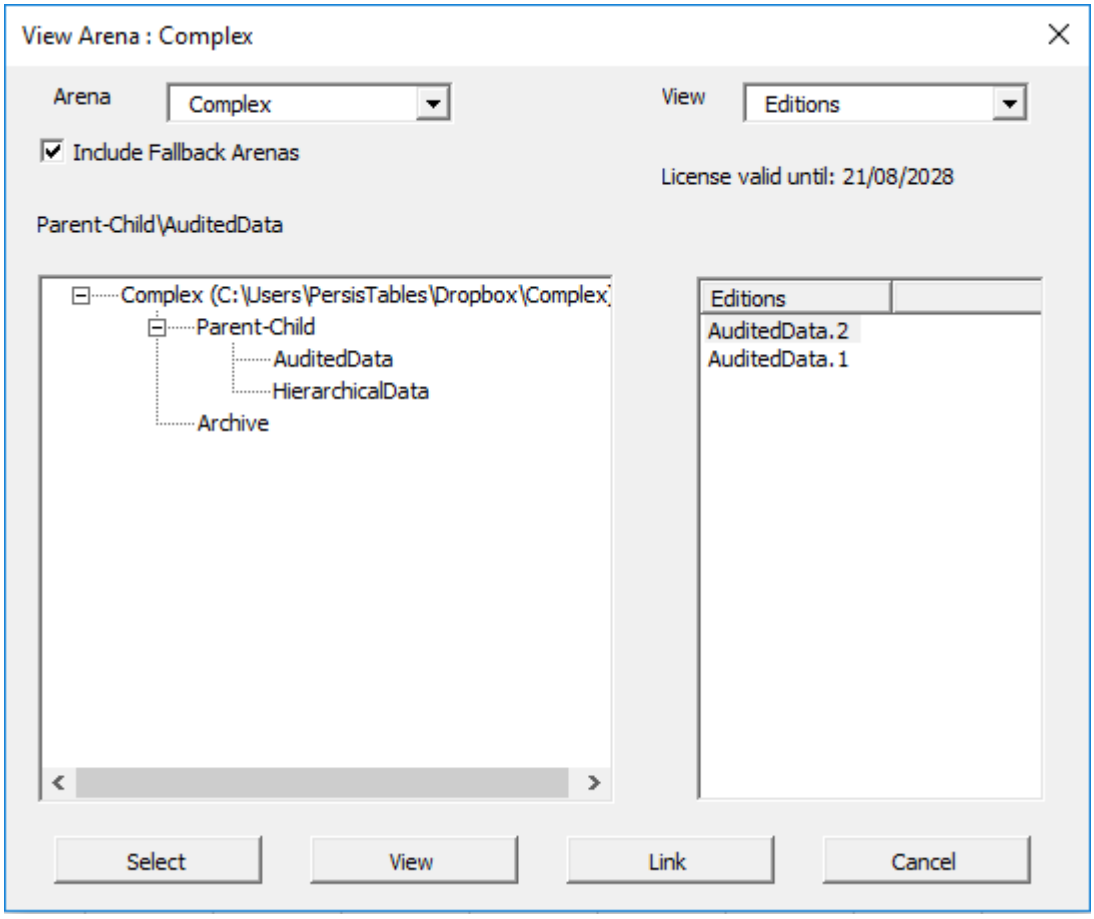


For example a Genre can be flagged to support auditing of changes to PersisTables. In this case whenever a PersisTable is amended, a copy of the original PersisTable is maintained in an audit file.

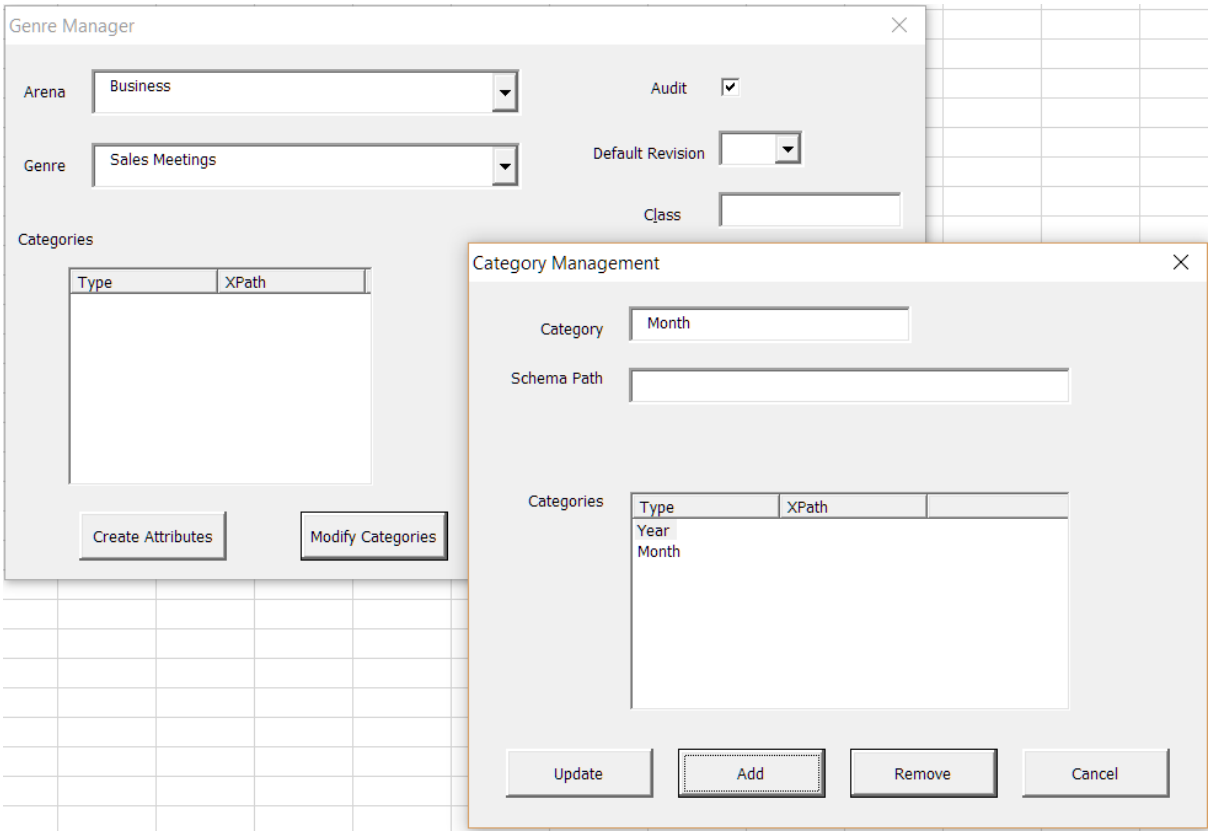


Here we have switched on Auditing for the Sales Meetings Genre. Any amendment to an existing PersisTable within this Genre will automatically be audited.

These audit copies can be accessed through the Persistence Viewer by selecting the ‘Editions’ option form the View drop down menu. This allows a previous version of the PersisTable to be retrieved to the spreadsheet. Here the Parent-Child genre is audited and selecting the PersisTbale named AuditedData in the left hand pane we can see there are 2 editions of the PersisTable in the right hand pane.



We can also ensure that the appropriate number of Categories are specified when saving a PersisTable within a Genre by specifying the Categories required.

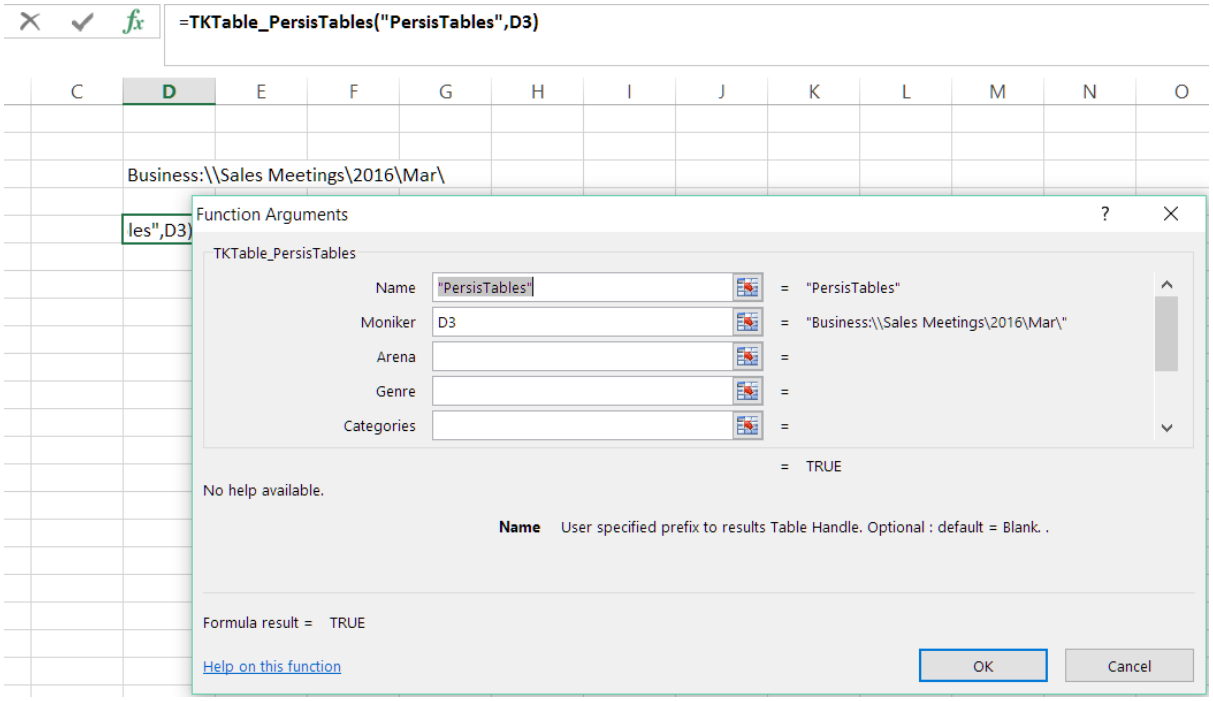


Here two Categories must be specified in the Moniker representing the Year and Month. There is no validation of these fields so 'qwerty' would be a valid Category for Year. If a field is present within the PersisTable that matches a Category, then the Schema Path can be set to the XPath that would extract the value from the PersisTable. Given such a path the Category provided will be matched against the value in the PersisTable and the PersisTable will fail to save unless they match. This allows the Genre administrator to ensure the PersisTable is saved into the appropriate Categories.

PersisTables & Editions

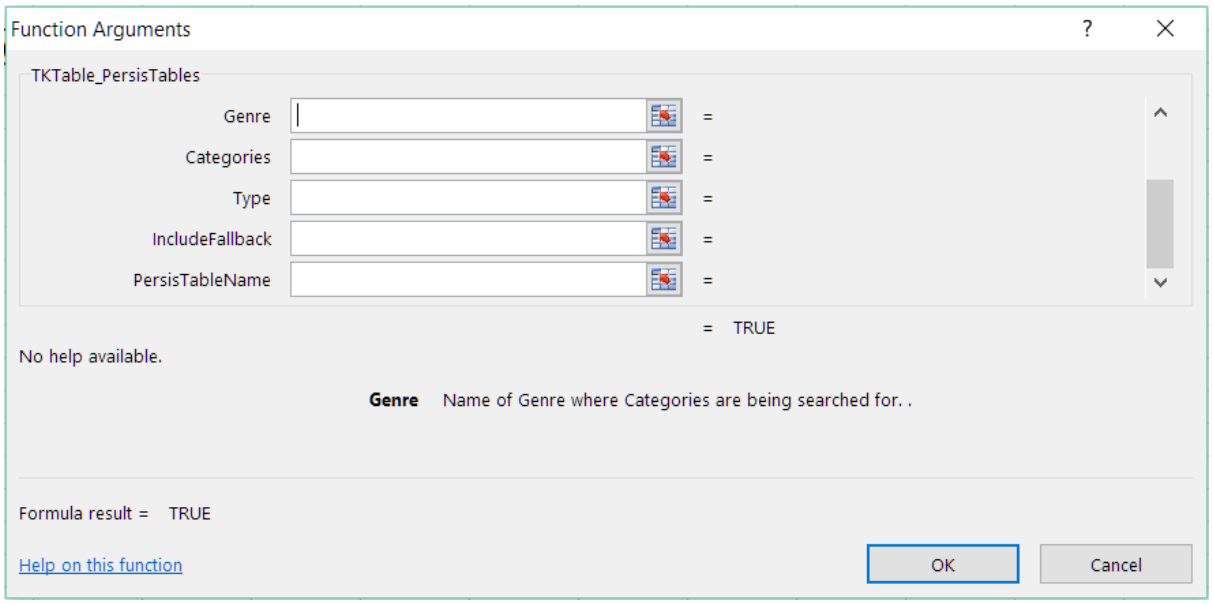
We have seen that a PersisTable is a Table that has been saved so that it can be accessed in other workbooks or at a later point in time. It is a persistent version of the Table. As we can select any sub set of data from a spreadsheet we can create a PersisTable in a shared Arena which allows other users to access that data without needing to close the original workbook or even access a menu item, the PersisTable is updated as soon as the data within the Table is changed (assuming Auto Calculation mode is switched on).

Just as with Arenas, Genres and Categories, it is possible to list the PersisTables found in a specific location using the TKTable_PersisTable() function.



In this case we have supplied a Moniker that points to the Sales Meeting Reports held in March 2016. We could have specified the Arena, Genre and Categories separately. The results are a Table containing all the PersisTables found in this location.

There are additional parameters which can be specified



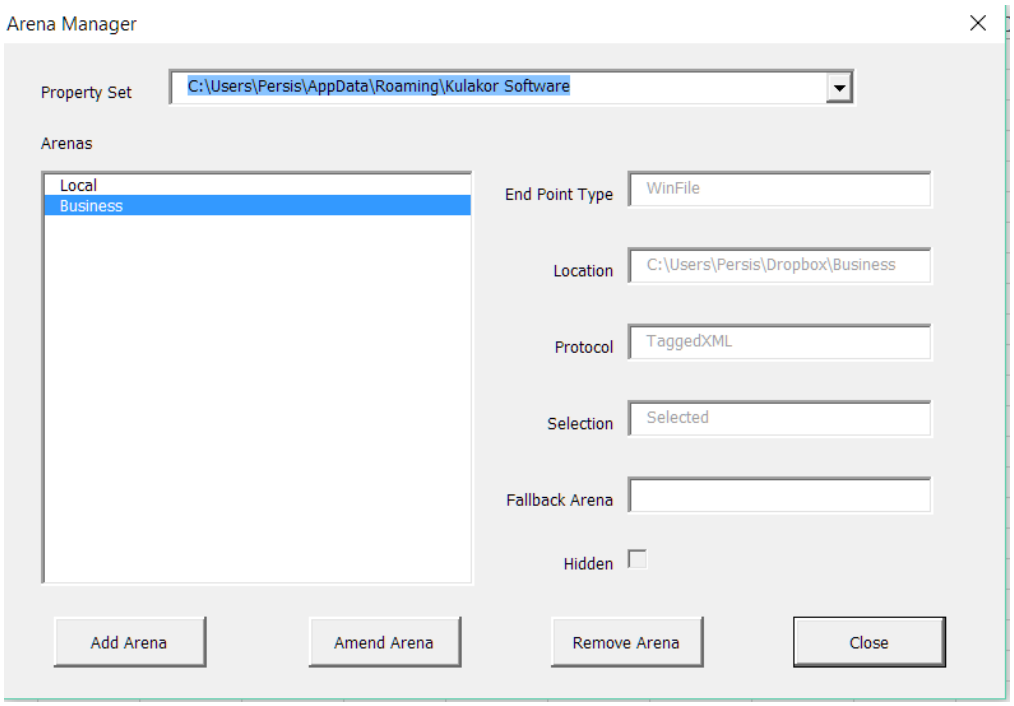
The Type parameter identifies whether the list contains PersisTables, Editions or Attributes. When Auditing is switched on copies of the PersisTable are made whenever it is updated. Each copy is known as an Edition. They are given the same name as the PersisTable qualified by an Edition number (the higher the number the more recent the copy). Often we would like to list only the Editions of a specified PersisTable, in this case we should provide the PersisTable Name to filter out any other Editions.

The ‘Include Fallback’ option will list those PersisTables found in the same Genre, Category location in the Fallback Arena (or its Fallback Arena if specified). Note that while these PersisTables can be read into Excel they cannot be updated, a copy will be saved in the specified Arena if an update is made. After that only the version of the PersisTable in the active Arena can be reloaded.

Attributes are used to specify the behaviour of a Genre. These are always held in the Genre itself (no Category is used). While these may be viewed using the spreadsheet formulae provided it is advised they are only updated through the Genre Manager applet.

Protocols

Protocols are the mechanism whereby the information in a Table is transformed into a PersisTable. There are multiple Protocols available but the default behaviour is to us the Protocol defined for an Arena when setting up the Arena. This can be viewed using the Arena manager applet.



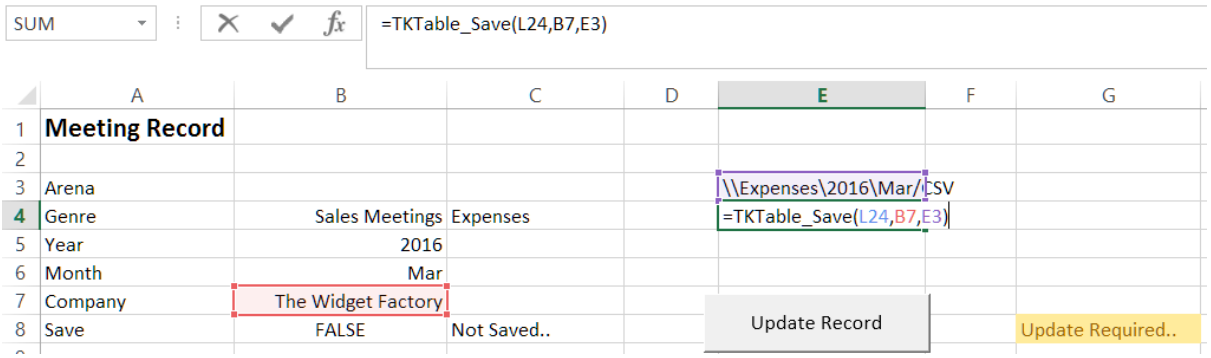
In this case the Business Arena is configured to use the TaggedXML Protocol. This translates the Table into an XML document[15] where the Table column headings are used as the Tags in the XML document.

For the purposes of saving and retrieving data into a spreadsheet, the choice of Protocol is largely irrelevant as what is saved is what is retrieved. The provision of multiple Protocols is really only relevant where other applications are looking to access the data held in the PersisTable. XML is a useful choice where It systems are used to extracting information from XML documents.

[15] XML is a standardised syntax for specifying hierarchical data used widely within computer applications.

One particularly useful Protocol is the CSV Protocol. This allows the contents of a Table to be delivered in a csv[16] file format. Note it can only represent a single Table of data (no embedded Tables). Saving a CSV file from Excel can be a bit awkward as the data to save must be on a single sheet, all formatting is lost and various adjustments must be made to the system environment to use a separator other than comma. However many systems can consume CSV files as input so the ability to create them is valuable.

We could change the Protocol associated with the Business Arena to CSV and save information in csv format, however this would prevent us from saving the structured Sales Records we already have created.



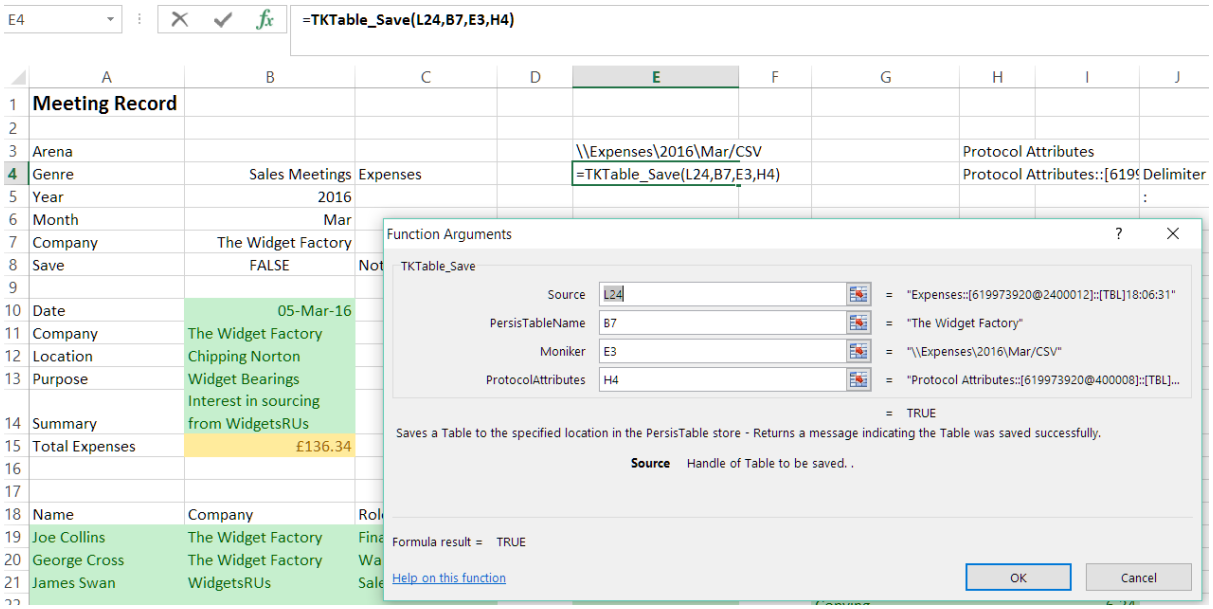
Here we have saved the Expenses reported as a csv file. Note the Moniker ends with "/CSV". This is how we can override the default Protocol when saving a PersisTable. The text after the '/' character defines the name of the Protocol to use (which of course must exist).

Normally once we have defined the Protocol used to create a PersisTable it is not necessary to specify the Protocol again when reloading the data. This is because most protocols used within the Toolkit are self-describing, they allow the Toolkit to work out which Protocol was used and how to interpret it. Unfortunately the CSV Protocol does not support self-description so the process of loading a CSV file requires that the Moniker includes the protocol description in the same way it used when saving the PersisTable.

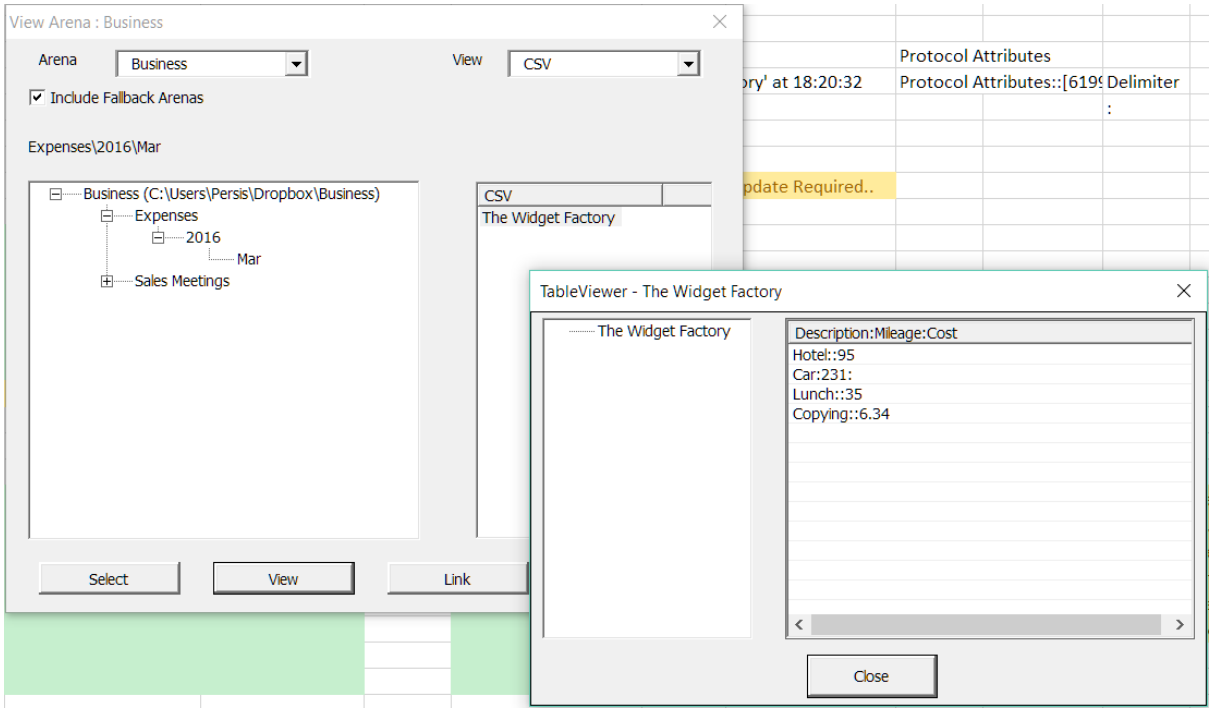
One advantage is that any CSV file can be loaded into Excel using this Protocol as long as its file suffix is ".csv". We are not limited to only reloading files created by the Toolkit. Again this improves the ability of Excel to interact with other applications.

If the CSV file needs to use another character as a delimiter it is easily done by providing an additional Table specifying the delimiter as the Protocol Attributes.

[16] CSV stands for "comma separated value" and is a commonly used format to save a table of data from Excel but also used when applications want a simple to define format to share regular data. Confusingly it is often used where the delimiter between fields is a character other than a comma.

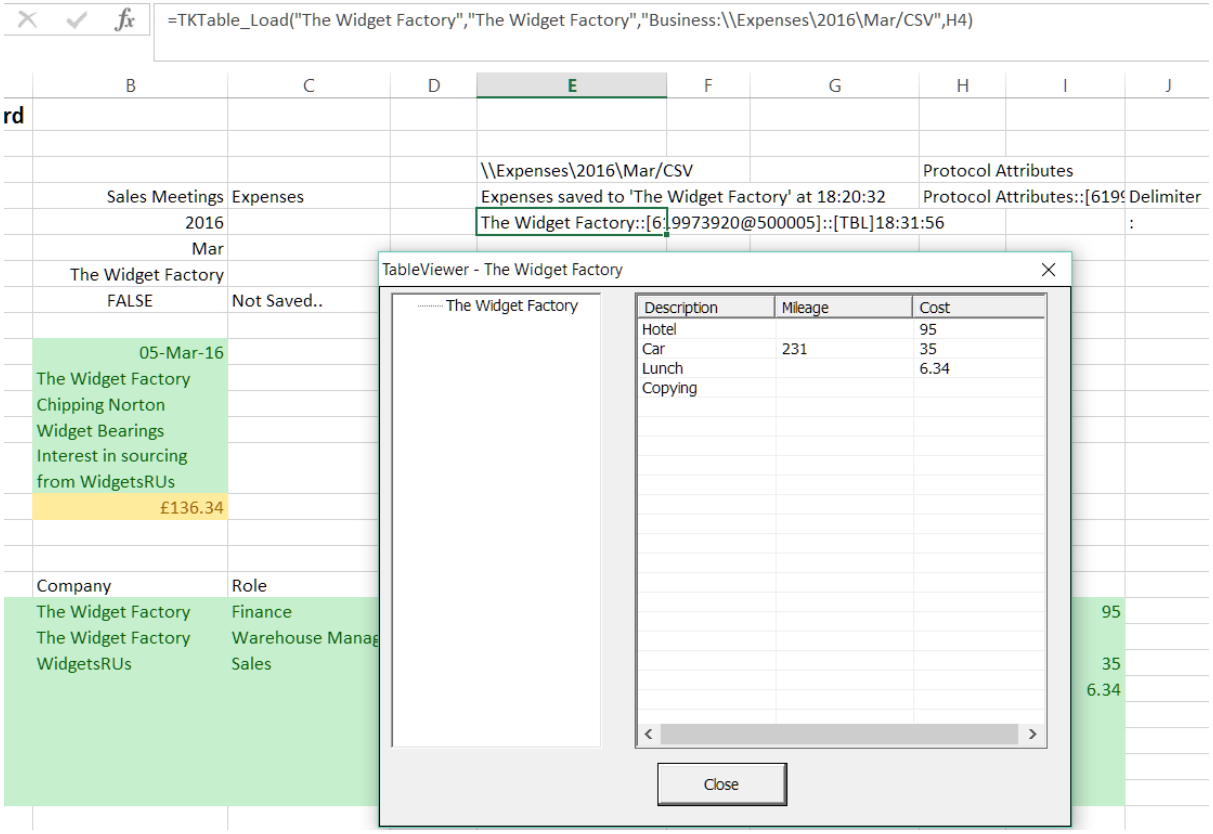


The Protocol Attributes define the delimiter as a : we can observe this if we try to view the PersisTable in the Persistence Viewer.



The data within the PersisTable appears to consist of one column with colons embedded in the data. We can only view this PersisTable properly by loading it with the Protocol Attributes specifying the delimiter[17].

[17] Note the View selected in the Persistence Viewer is CSV. We can only see csv files this way.



The Table Viewer shows us that we have now correctly interpreted the PersisTable data.

Schemas and Revisions

Tables can capture information in any form or shape. If we wanted we can easily add additional Keys and associated data to a Table. This makes it easy to prototype behaviour to create the optimal structure to meet business requirements. For example we might decide we should add phone and email addresses to the list of those present at meetings within the Sales Meeting Report. We can simply change the spreadsheet to contain the additional information and ensure it is captured by the appropriate TKTable_Create() function.

At some point we will settle on the form of the data[18] we want to capture and share with others. We can create a worksheet to display the information and other users can extract subsets[19] of the information to perform their own analysis.

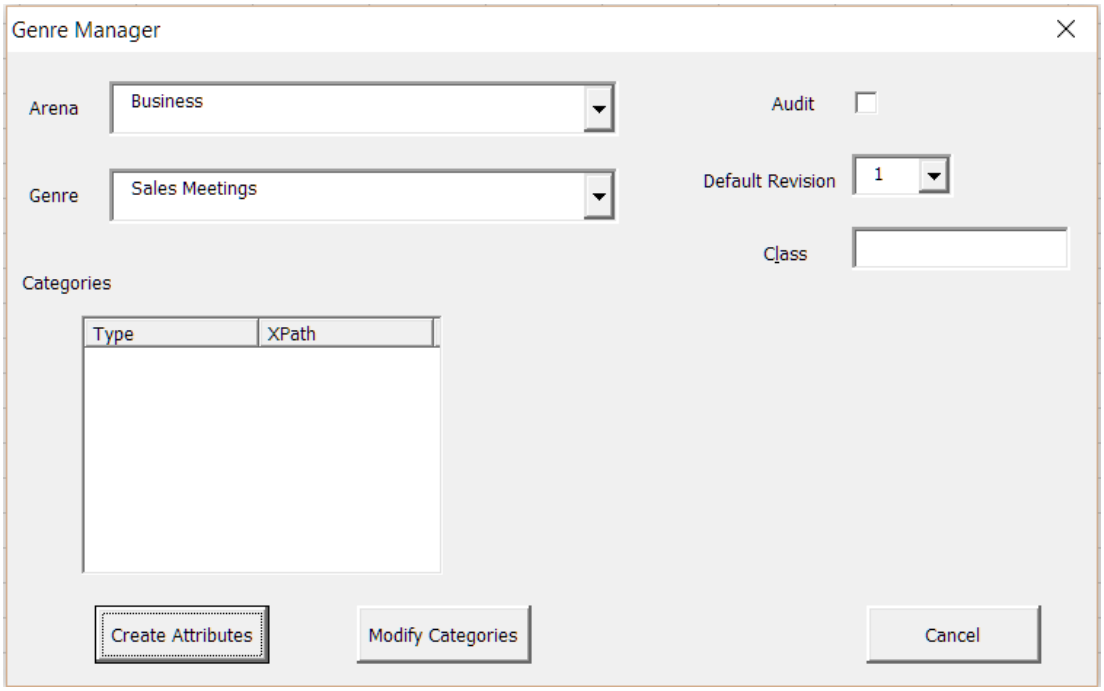
Over time we may find that we need to amend the schema because new business requirements have turned up. Changing the form of the data is simple enough, but we need to ensure that these changes do not affect the current business operation. One method is to save the new information in a new Genre however that requires modifying the location searched when looking at information over time.

We can continue to save PersisTables in the same Genre but there is a danger that the new format of data may cause problems to applications that utilise the information. How does an application

[18] We will refer to the format or shape of the data as the schema. The schema essentially defines the Keys for each Table, the type of the data held under the Key and the relationship between Tables.
[19] We will examine how to do this in the next section.

reading the PersisTable know which shape of data to expect when they unwrap the data? To handle this problem, PersisTables are marked with a schema Revision. This allows an application to determine the expected shape of the information within the PersisTable.

There are two mechanisms to set the schema Revision. The first is to add the Revision to the moniker when saving the PersisTable. This is done by appending the string !<revision number> to the moniker. Where <revision number> is the appropriate revision of the schema. More usefully it is possible to configure a Genre to have a default Revision and all PersisTables created without the revision being supplied will use the default revision.

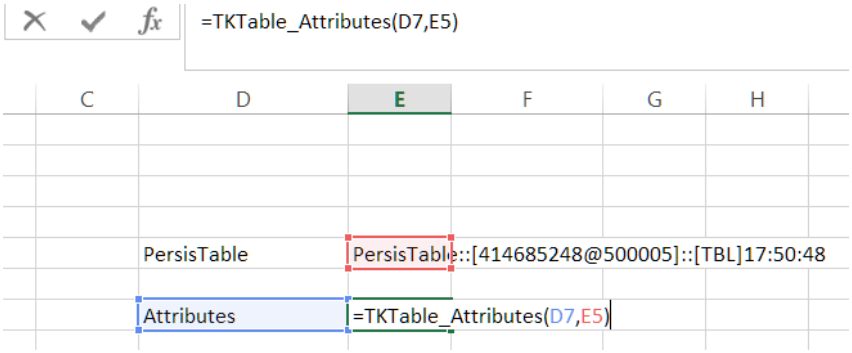


Here the Sales Meetings Genre is set to mark PersisTables saved as belonging to Revision 1. By convention Revision 0 is used to indicate no schema has been assigned, it can be used when prototyping[20].

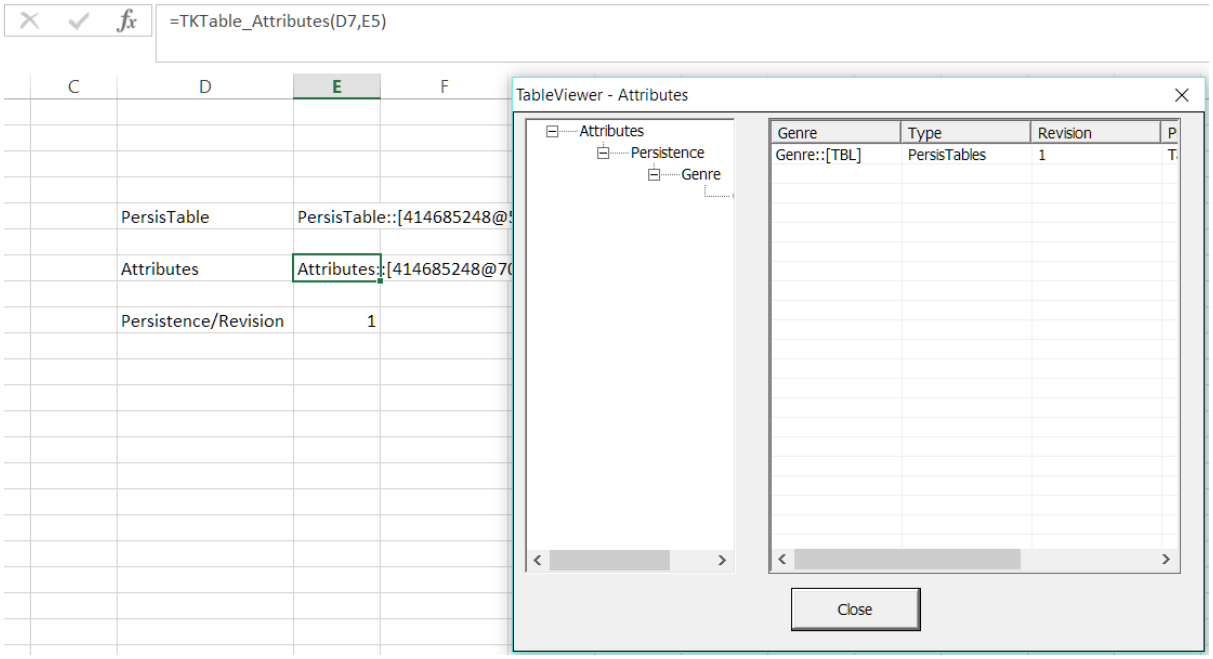
The Revision of a PersisTable is stored as part of the PersisTable Attributes. Table Attributes represent additional information stored alongside the Table data that can describe the context of that data but is not part of the data. For example the orientation of a Table is saved as a Table Attribute allowing the data to be redisplayed in the appropriate manner.

To identify the revision of a PersisTable we use the TKTable_Attributes() function to retrieve the attributes associated with the PersisTable.

[20] The premium version of the PersisTables toolkit provides tools to allow validation of PersisTables against schemas preventing data being saved in an incorrect format.



We can view all the Table Attributes using the Table Viewer.



We can see that the Revision information is held in the Persistence Table so we can access it using the TKTable_AtXPath() function with an XPath argument of “Persistence/Revision.

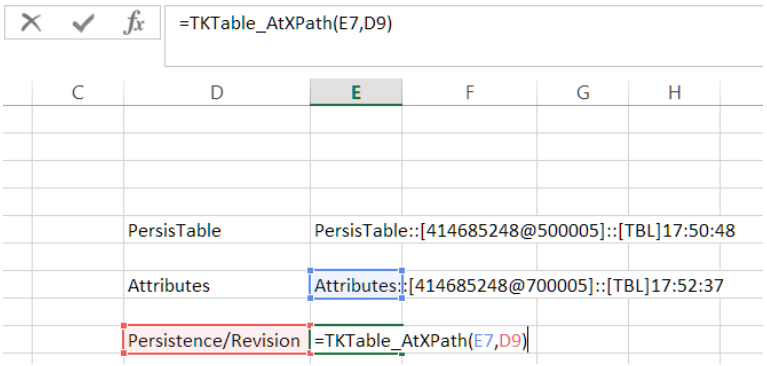
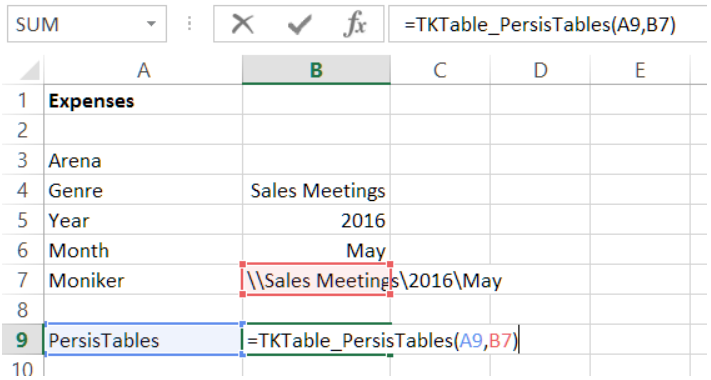


Table Manipulation

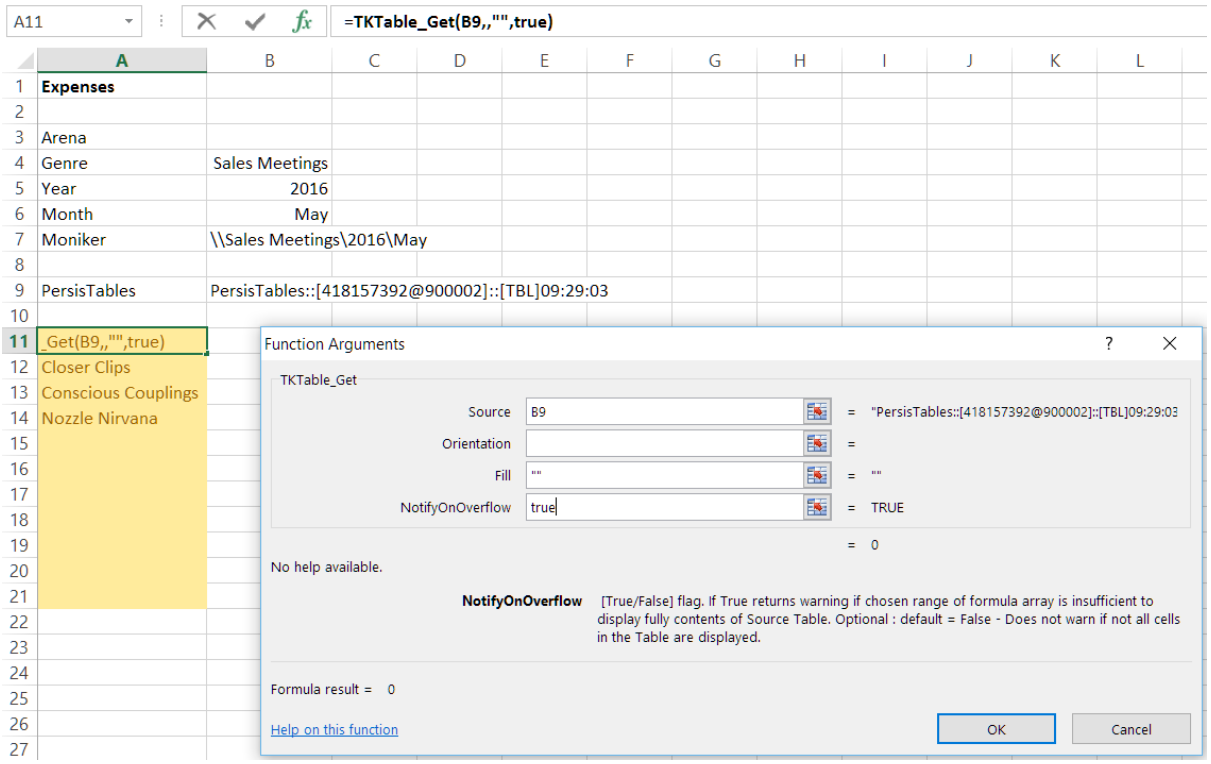
So far we have looked at the way PersisTables can be used to capture arbitrary sets of information, save them and restore them at a later date. The Toolkit provides a number of functions that allow the data within a PersisTable to be processed. To illustrate this we will look at how we might consolidate the expenses from a number of different meetings and generate a report allocating expenses to Salesmen.

PersisTables

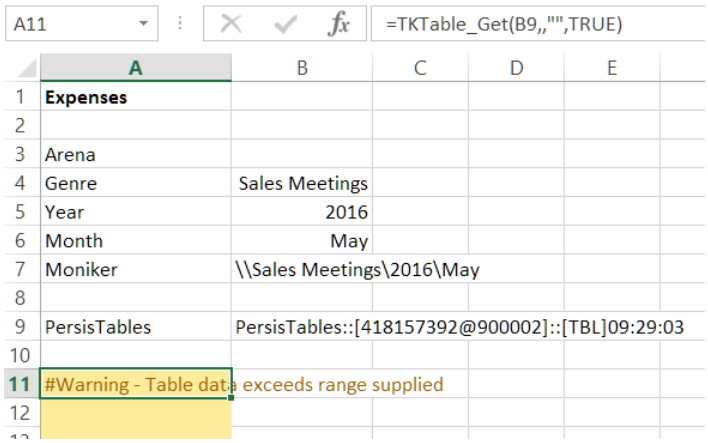
In order to assign the expenses back to individual Salesmen we need to look at all of the Meeting Reports. We will perform the analysis month by month so we can use the same format sheet to specify the year and month we are interested in as we did when retrieving the original Meeting Reports. As before we allow the user to select from a list of dropdown menus to choose the Year and Month and we use this to create a Moniker describing the Category to search for PersisTables.



Instead of using the list of PersisTables to populate a list box we extract them to the worksheet.

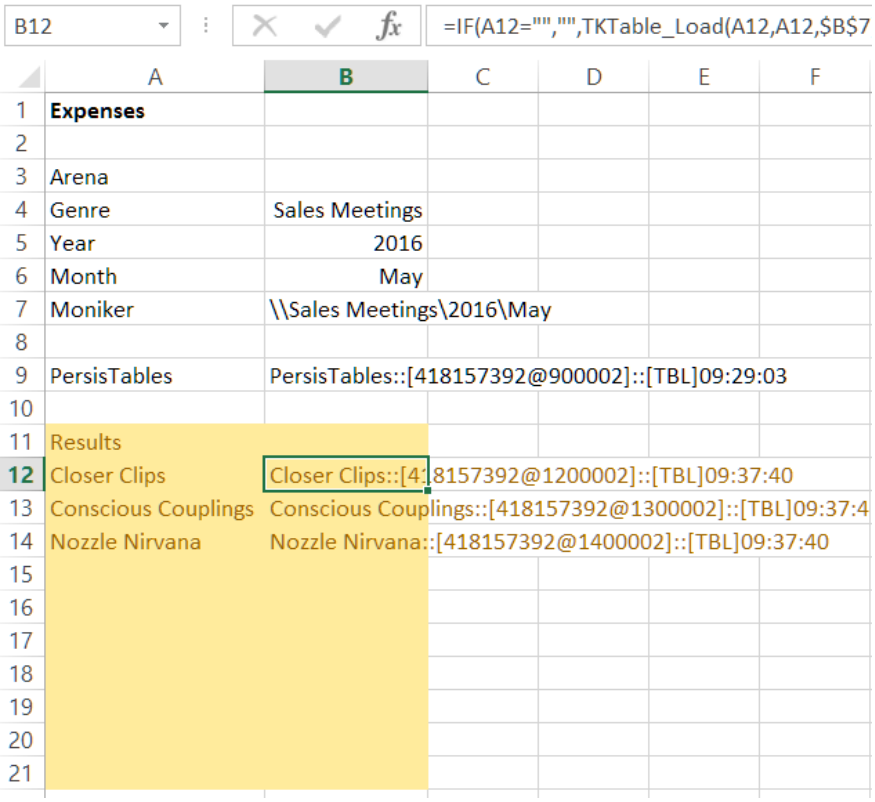


The area where the TKTable_Get() function will extract data to has been highlighted in yellow. It is feasible that we will have more reports in a month than will fit into the available space. To avoid the consequence of 'losing' data we switch on the NotifyOnOvervlow flag which will display a warning message in the range used by TKTable_Get() if that Range is insufficient to display all the data in the Table it is extracting data from. This rapidly highlights problems caused by allocating insufficient cells to the output array.



Here the TKTable_Get() has been applied to a single cell rather than to an array and a clear warning is presented that there is insufficient space to display all the data from the underlying Table. The short term mechanism to alleviate the problem is to extend the Range that is used to extract the list of PersisTables into. Alternatively it may be appropriate to provide more Categories so the PersisTables are spread over more Categories

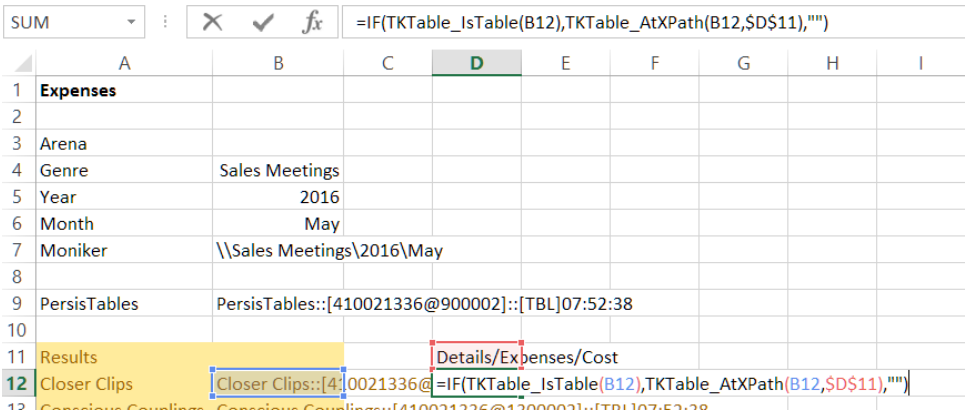
We can now load the individual Meeting Reports generated during the selected month using the Moniker created to list the PersisTables and each PersisTable name retrieved.



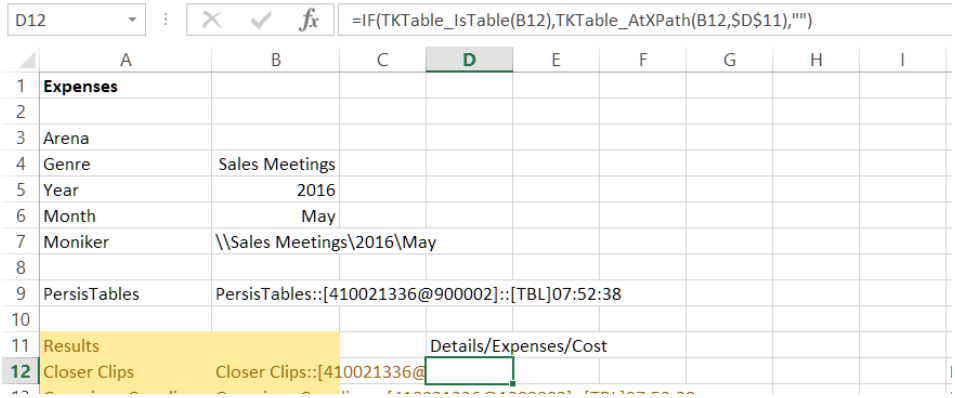
Note we check to see if there is a valid name in column A before attempting to load the PersisTables.

AtXPath & XPath

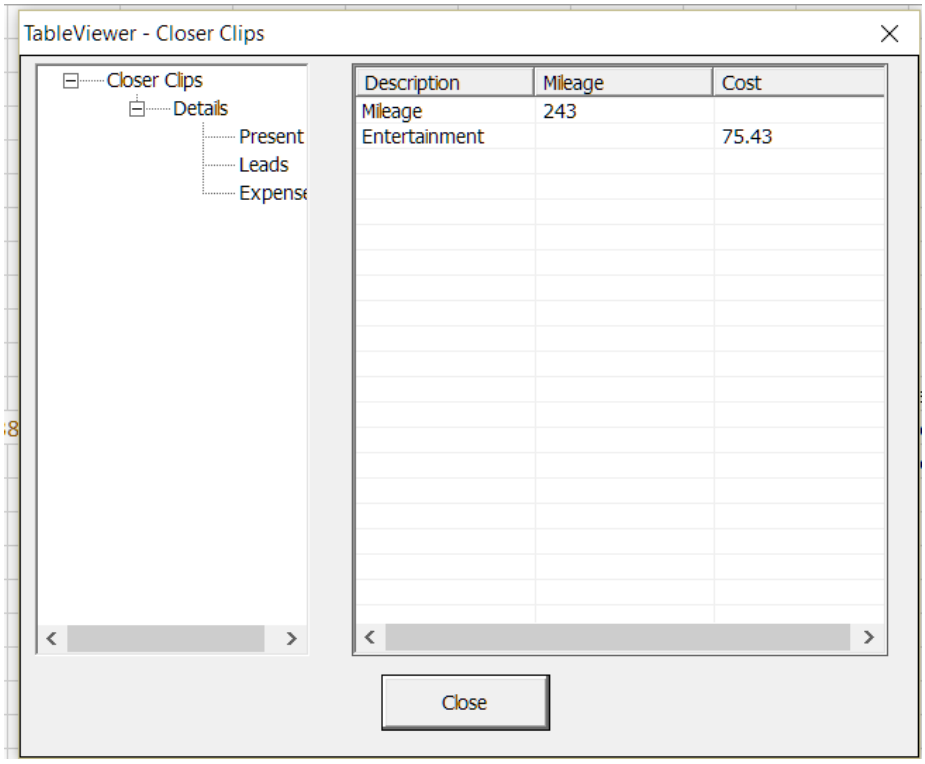
We have already seen that we can extract data from a Table using the TKTable_AtXPath() function. This allows us to go direct to an item within a Table and get its value (which maybe a Handle to a Table). We specify the XPath as a series of Keys separated by the '/' character as shown in the example below



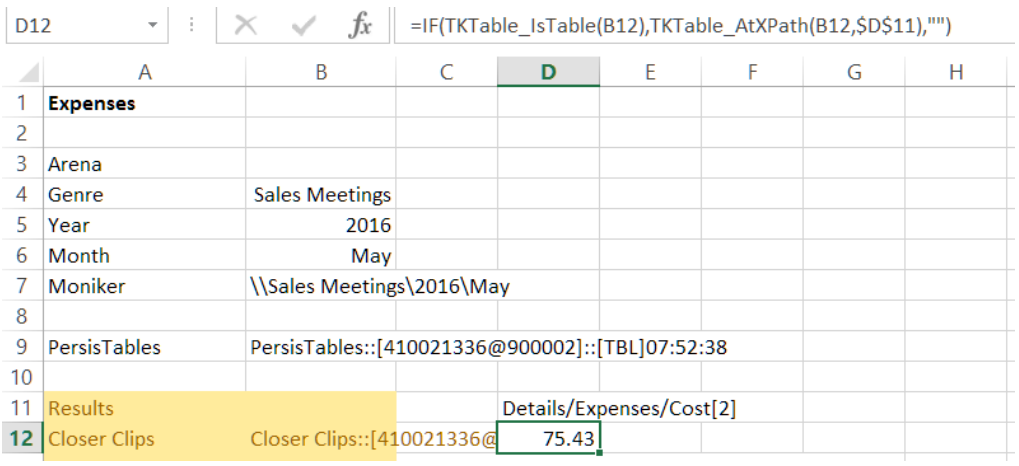
We might expect this to return the Costs associated with this meeting. Unfortunately it does not.



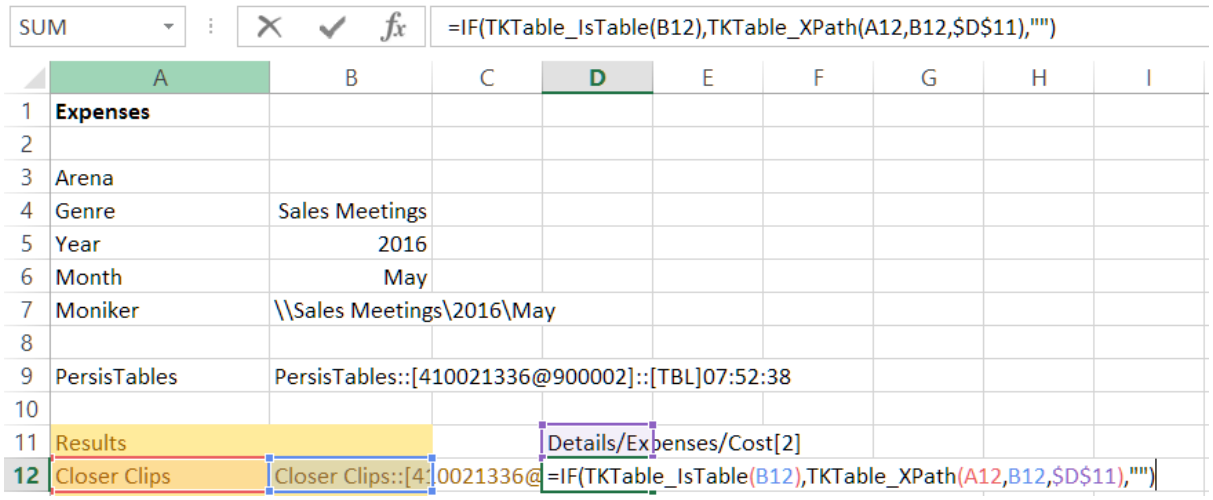
This function only returns a single Item from a Key, if there are many items under a Key it picks the first entry unless we specify an Item offset. In this case the first value is empty.



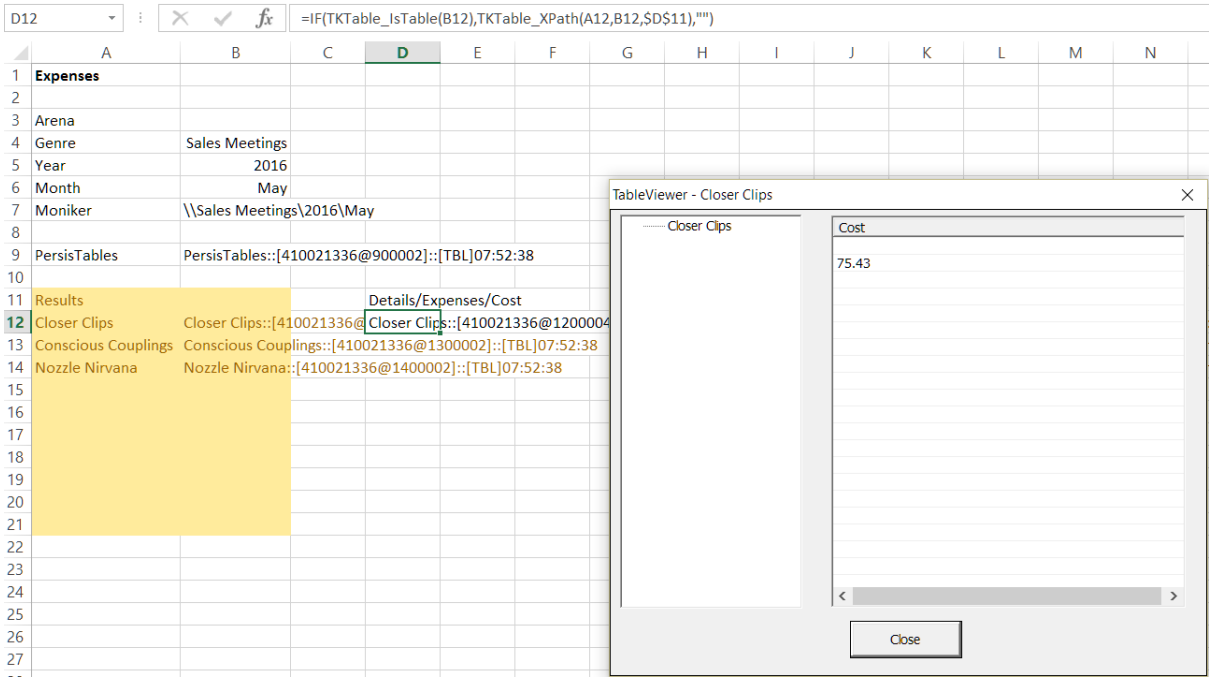
By amending the XPath to pick the second Item under the "Cost" Key we do get a result



However as we can see the TKTable_AtXPath() function would never return all of the Costs, it can only return a single value. The alternative TKTable_XPath() function returns all of the values found at an XPath.

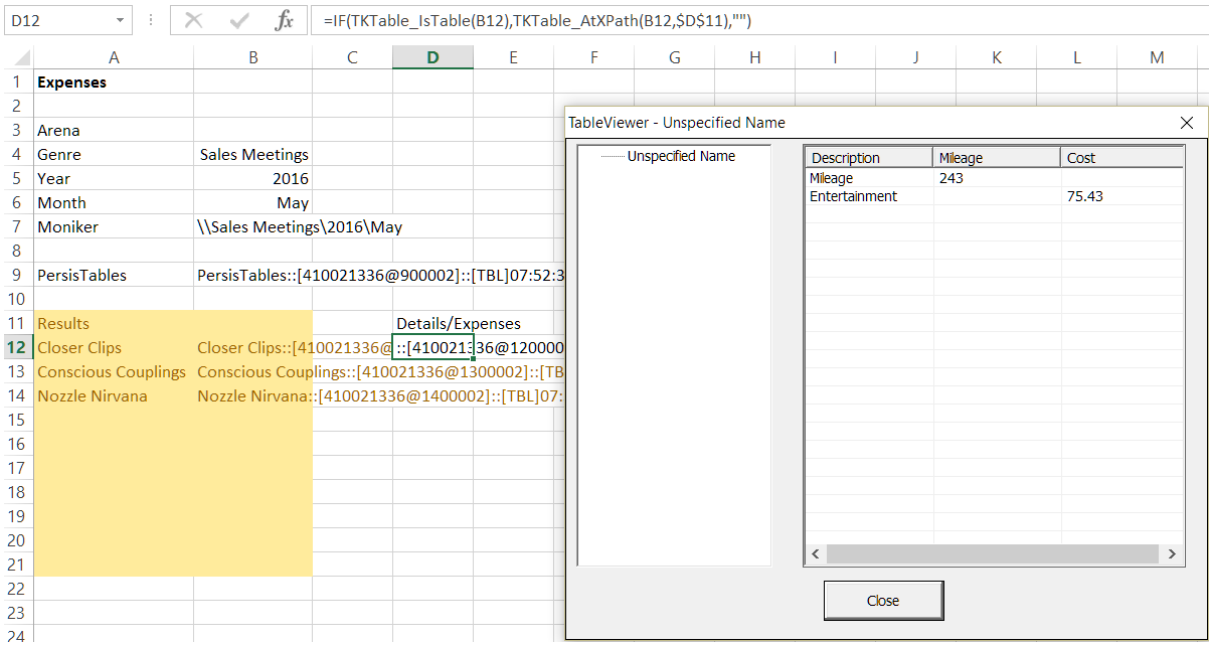


As we have selected a single Item we get the same results as before but with the result returned in a Table. Removing the qualifying Item from the XPath returns all the Costs found under the given Key.



We can see that we have now retrieved all of the values under the Key ‘Cost’ including the empty value. The TKTable_XPath() function always returns a Table even if the Table is empty. Changing the XPath to an invalid value leads to an empty result Table unlike TKTable_AtXPath() which returns an error.

In fact getting the costs alone is not really helpful as we cannot see what generated the costs so we will use the TKTable_AtXPath() function to access the Table holding all of the costs.

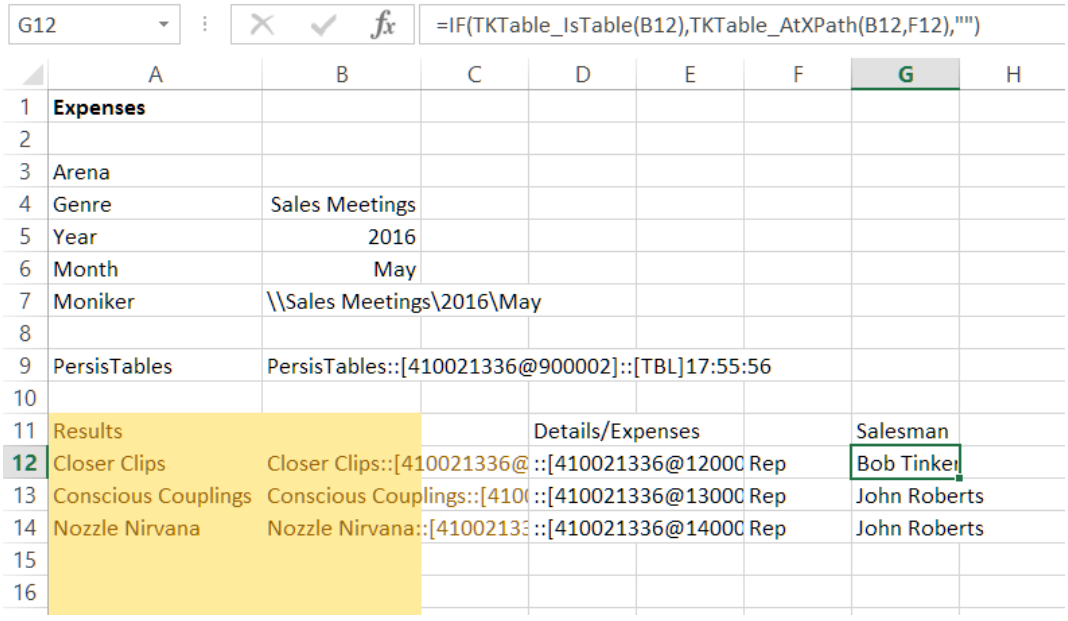


We have retrieved all of the information in the Expenses Table for the use in the next step.

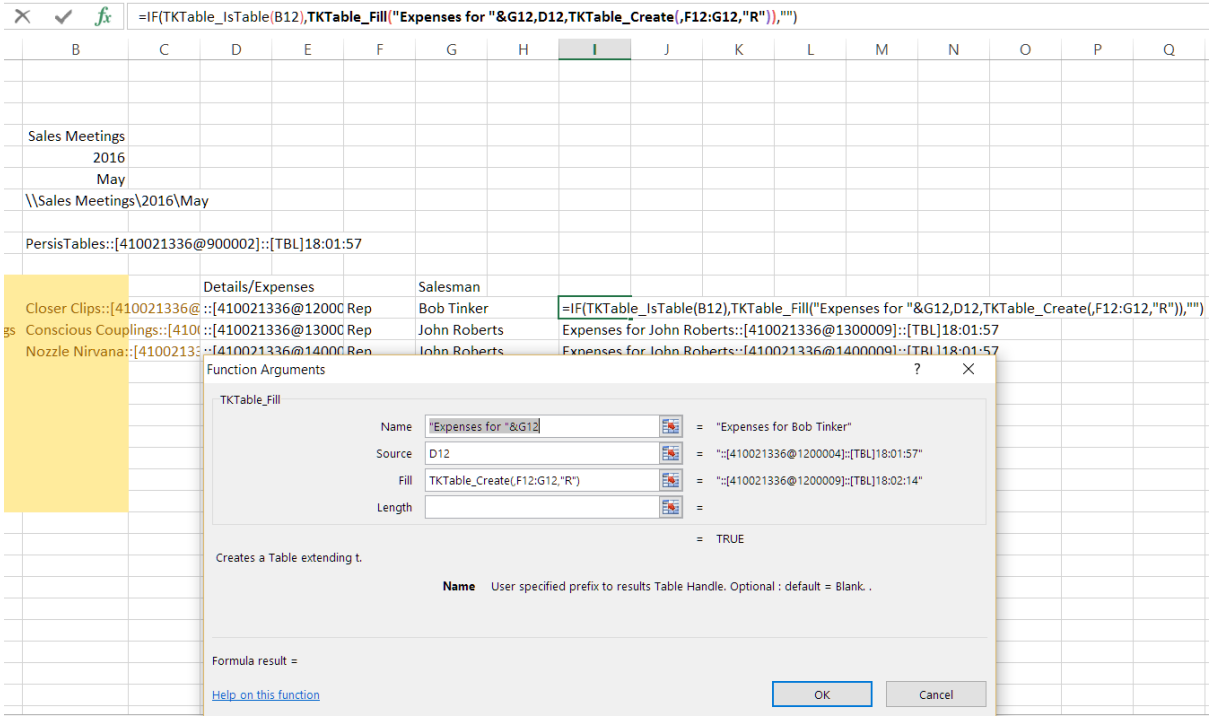
Note we have checked to see if we have successfully loaded a Table before we run the TKTabe_AtXPath() function by using the TKTable_IsTable() function. This is a useful function for removing unwanted error messages from spreadsheets but be warned that if it is recalculated immediately after a workbook is reloaded then the Handles will be invalid and so this function will return FALSE.

Fill

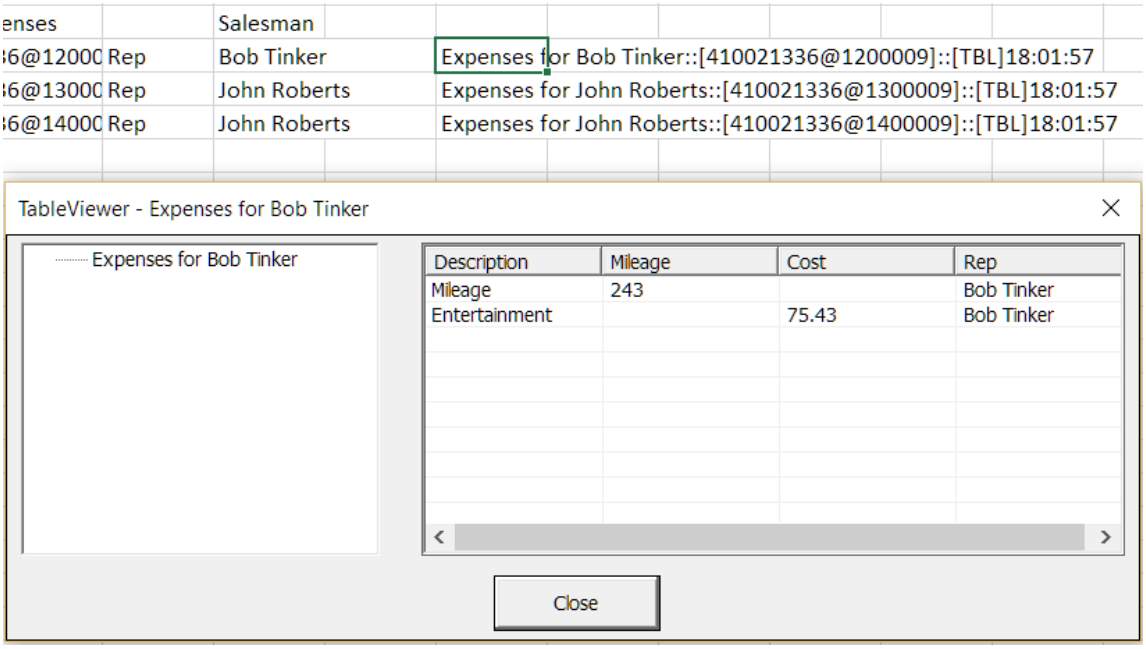
Now that we have the individual expenses we need to identify which Salesman raised them. We can easily extract the Salesman using TKTable_AtXPath() from the Summary Table.



We would like to add a column to the expenses Table containing the name of the Salesman. We can do this with the TKTable_Fill() function. This appends a set of Keys specified in a Table to an existing Table and fills the Items under those Keys with the values in the first Item under each Key.



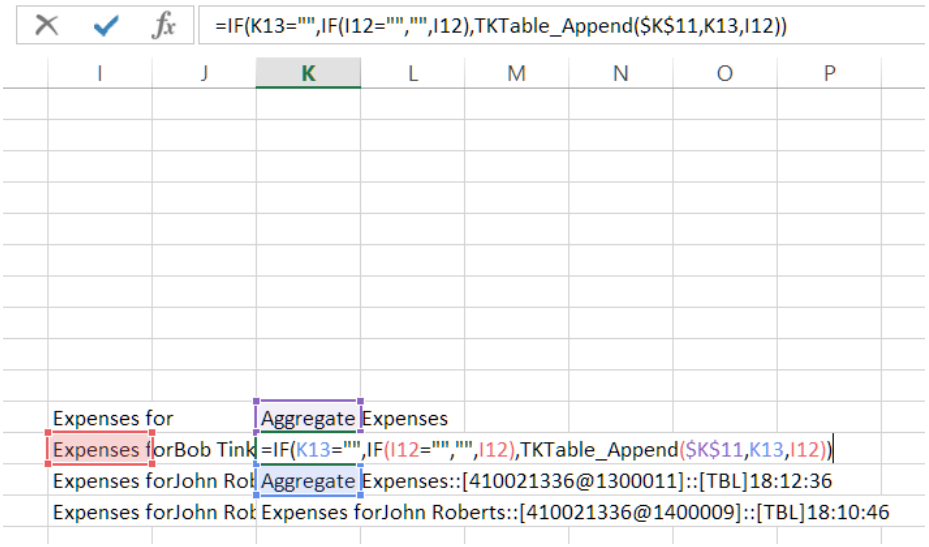
The TKTable_Create() function is required to generate a Table which has a Key named “Rep” and the name of the Rep as its only Item. This Table provides the template for the additional columns to be added to the Expenses Table using the TKTable_Fill() function.



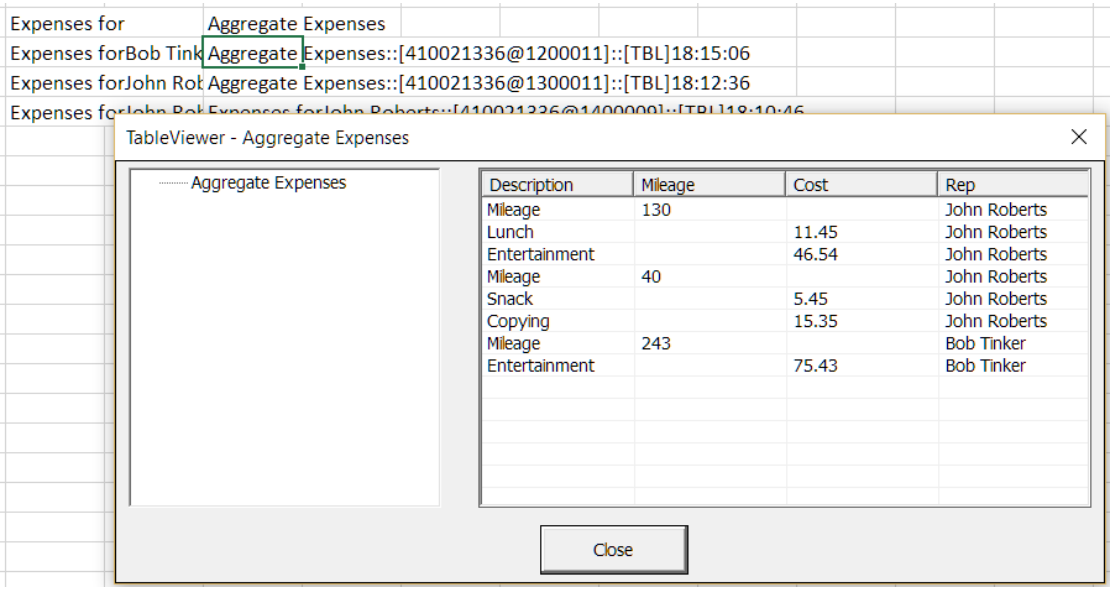
The results are an extended Table which contains the Salesman’s name against each item.

Append and Filter

The next step is to create a single Table from all of the Expense Tables. Where Tables have common Keys then they can be simply appended to each other using the TKTable_Append() function.

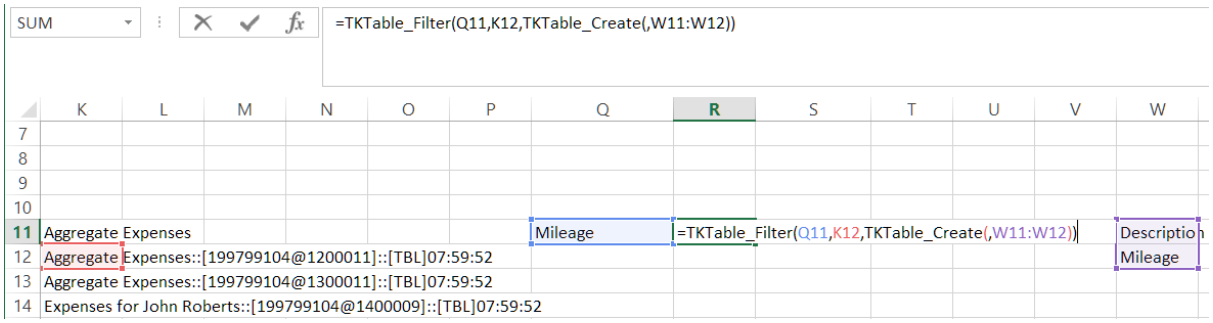


The logic is to check if we already have a partially generated aggregate in the cell below, if not then we use the Expense Table to the left of the formula. If we have an aggregate then we append the Expense Table to the left with the current aggregate.

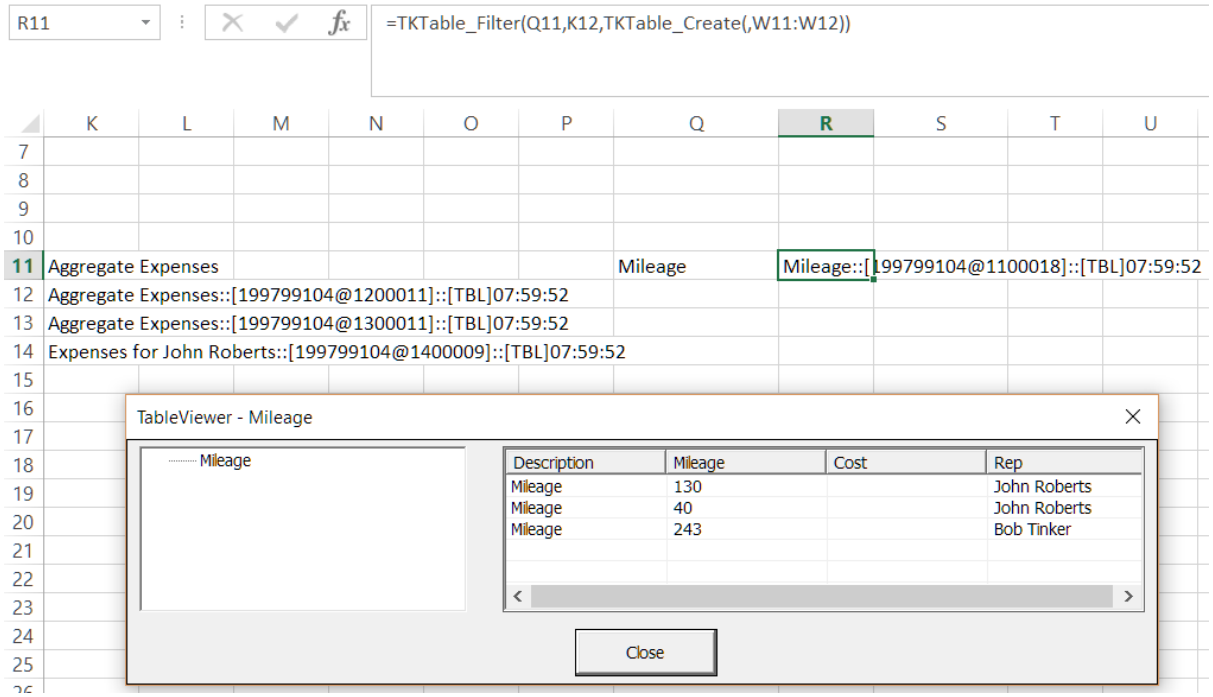


We can see that we now have a single Expense Table with all claims in it. Unfortunately we do not have value for mileage claims. The cost per mile is set by Finance and changes regularly. The Salesmen just enter the mileage and then Finance want to assign a cost per mile at month end.

We need to separate the Mileage claims so they can be processed. We do this with the TKTable_Filter() function. This takes a source Table of data, and a filter Table, the filter Table contains a Key and a list of Items. The Key is matched against the source Data set and then the Items are used to select the rows where there is a matching Item under that Key in the source Table.

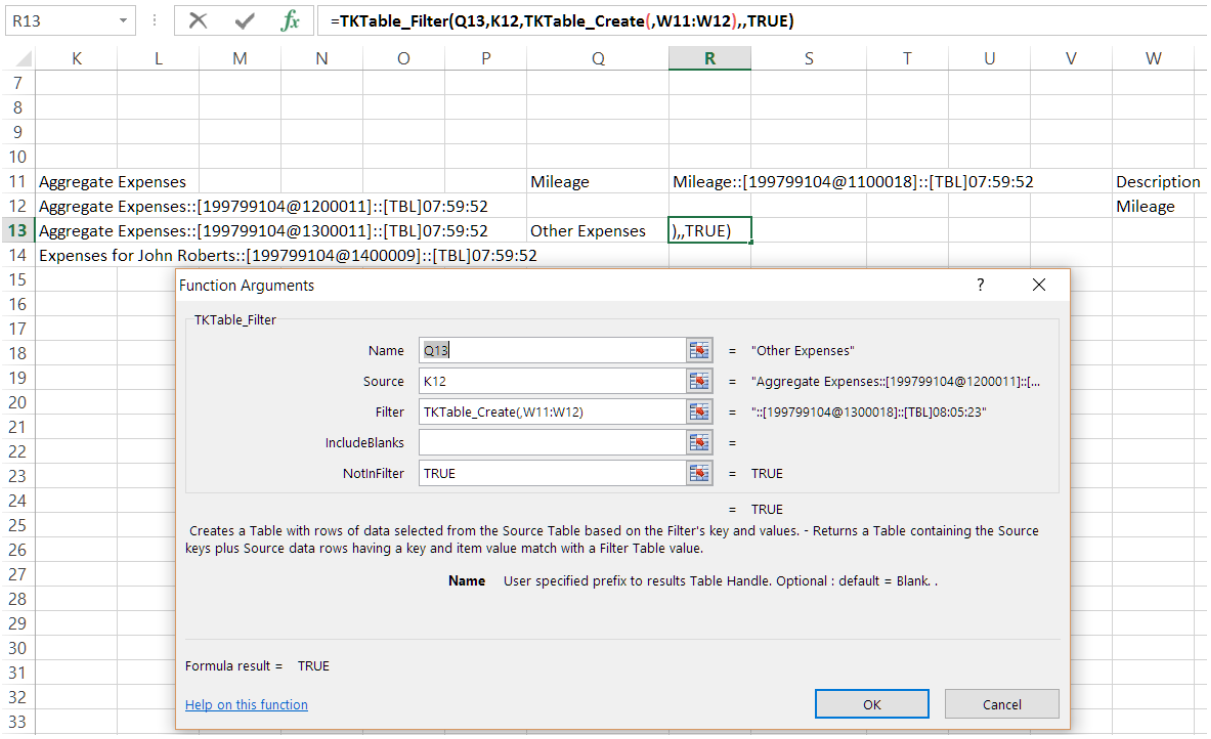


We have taken the aggregated set of Expenses and applied the Filter described by the pair of Cells at W10:W11. We create the Filter Table in the formula itself so there is no need to provide it with a user friendly name[21]. Based on this Filter Table the function will search for the Key 'Description' and copy any row where there is an Item containing the text 'Mileage'. Note that the PersisTable Toolkit is case sensitive so while this Filter will filter Descriptions specified as 'Mileage', it will not filter those specified as 'mileage'.

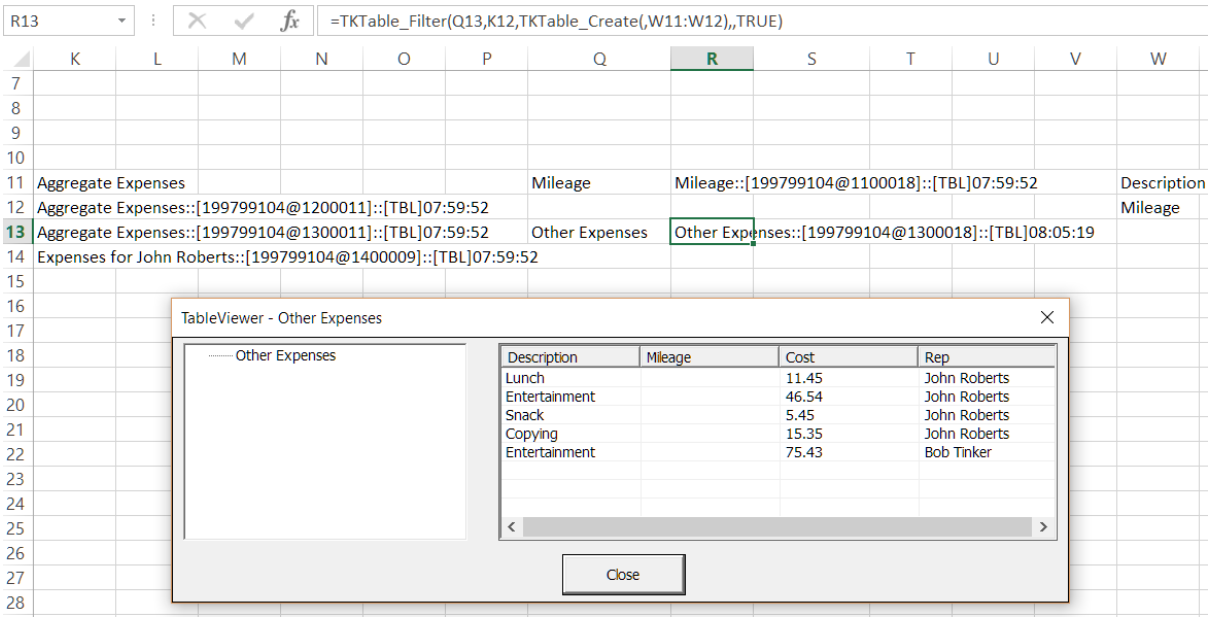


The results show that only the Mileage details are in the resulting Table. By setting the NotInFilter to TRUE we can select all Items that do not match the filter.

[21] It is always possible to create the 'anonymous' Tables but providing the user name makes it easier to understand the information represented by the Handle and so easier to understand the operation of the spreadsheet.



The results of the filter process is a list of expenses excluding Mileage claims

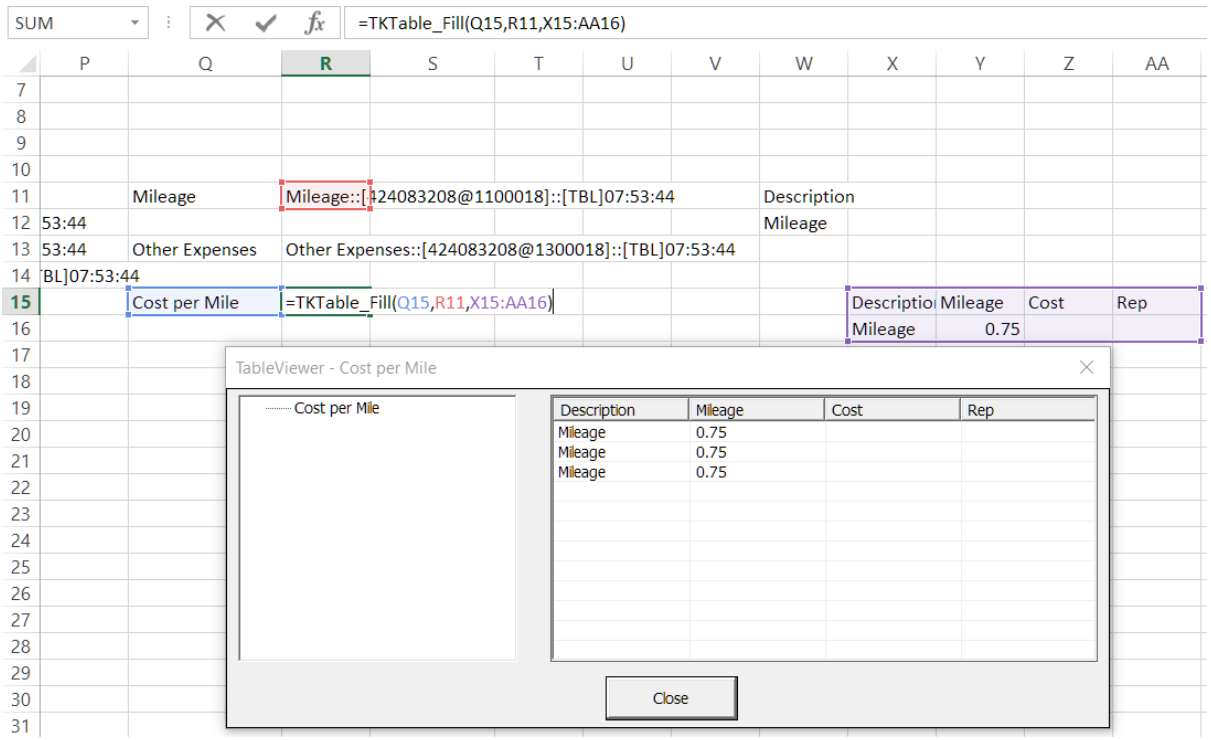


We just need to multiply mileage by the mileage rate and we have the total costs. While this is straight forward within Excel we will demonstrate how this can be achieved directly with PersisTables.

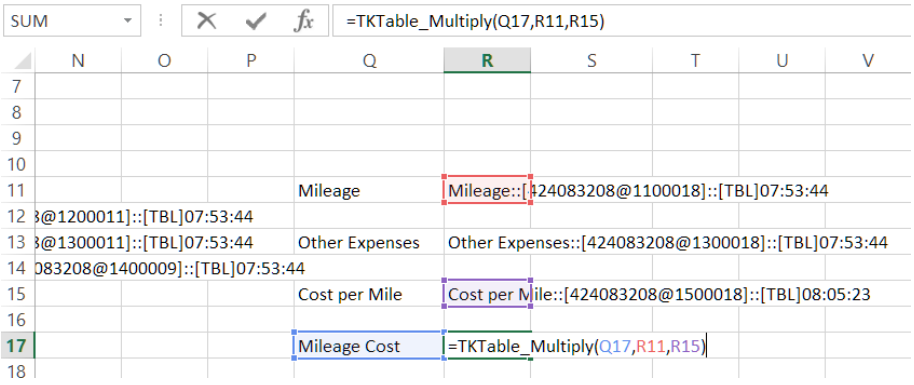
Arithmetic Functions

The Arithmetic functions allow data within identically shaped Tables to be added, subtracted, multiplied and divided. The arithmetic operation operates on numeric fields which have matching positions across the input Tables. The default operation is to perform the arithmetic operation on Cells within the Tables where both the input Tables have numeric values on that Cell location.

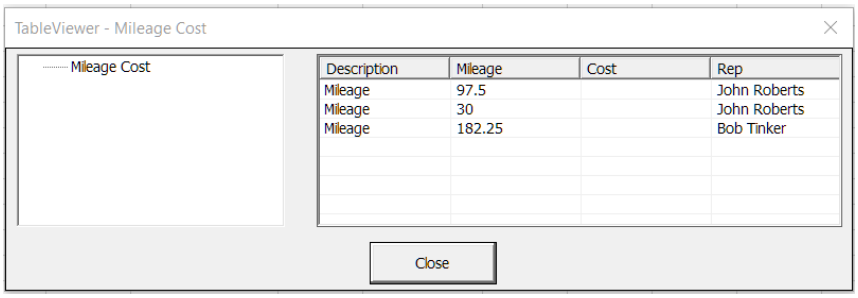
So we want to create a Table that aligns the payment per mile with the Mileage travelled in our Mileage Expense Table. The easiest way to do this is to create a Table which has a Mileage column filled with the cost per mile. We achieve this using the TKTable_Fill() function again.



We can now use the TKTable_Multiply() function to multiply the cells containing numbers in the Mileage and Cost per Mile Tables.



This gives the mileage costs by multiplying the matching cells under the Mileage Key and copying the non-numeric information from the 'Left' or first Table provided to the results Table.



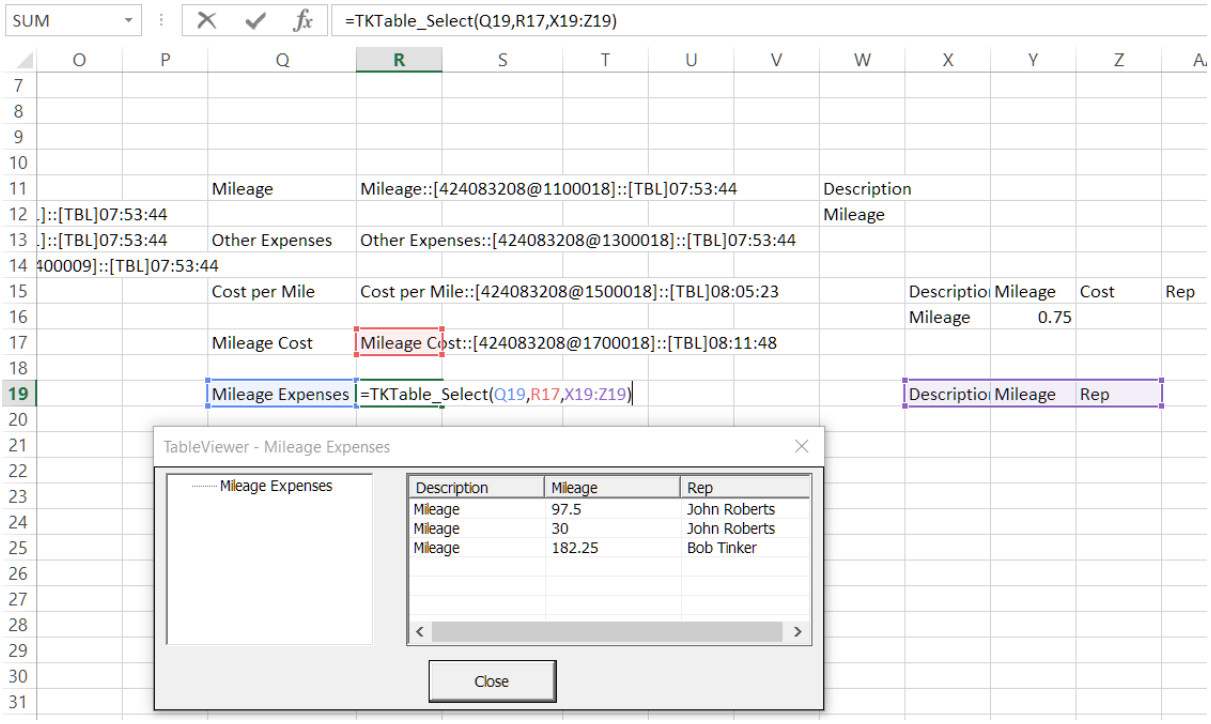
The arithmetic operations will operate across embedded Tables as well, as long as these are of matching dimensions in the input Tables.

The default behaviour is relatively indiscriminate. If a Key is numeric, say a Date, then it will apply he arithmetic function to the Key. It is possible to limit where the arithmetic operation is applied by providing a Template Table which matches the shape of the input Tables but with the string "%D" in any Cell that the arithmetic operation is to be applied to.

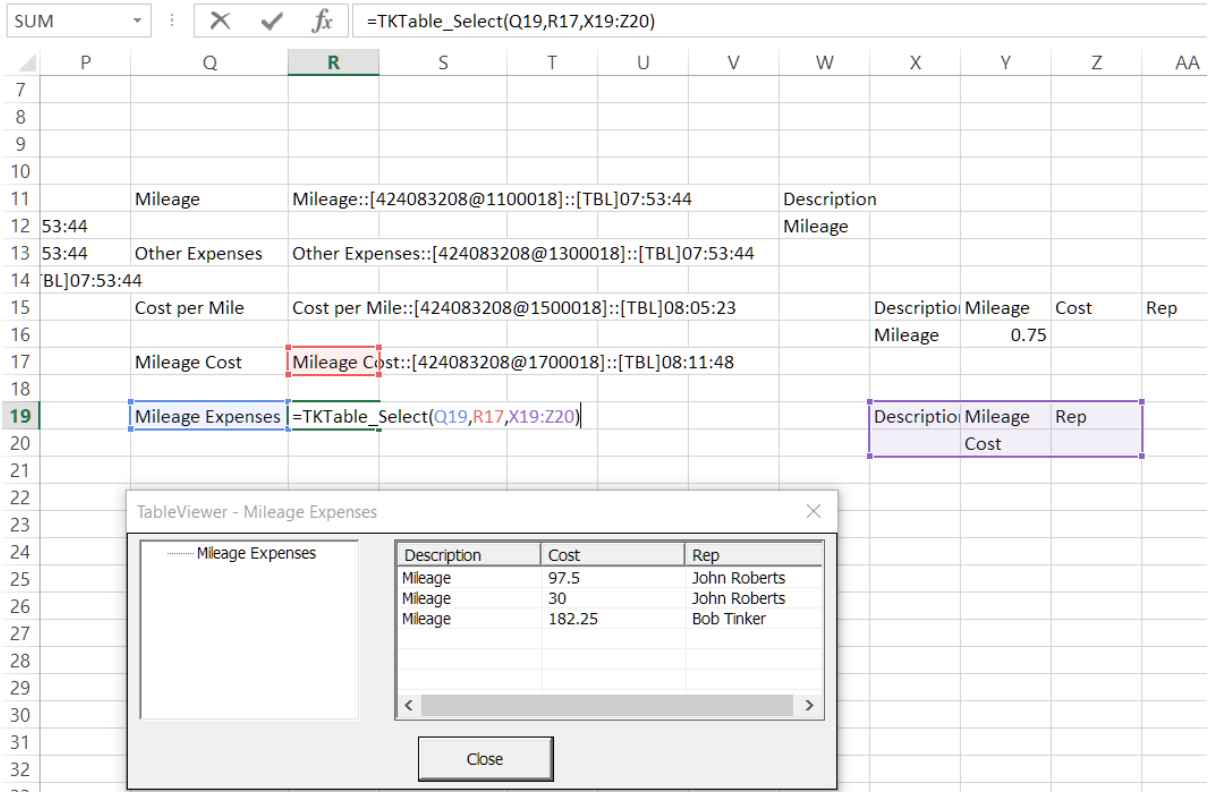
Select and Merge

We have now calculated the Mileage Cost but it is under the wrong heading. We need to rename the Key 'Mileage' to Cost and restore the Mileage data to record the mileage claimed.

The TKTable_Select() function allows a subset of data by Key to be extracted from a Table. Given a Table containing a list of Keys it will copy the data under those Keys to the results Table. We can use this function to copy over the data we want to preserve from the results to the call to the TKTable_Multiply() function. Note we create the Table containing the List of Keys on the fly by specifying the Range rather than using TKTable_Create() here.

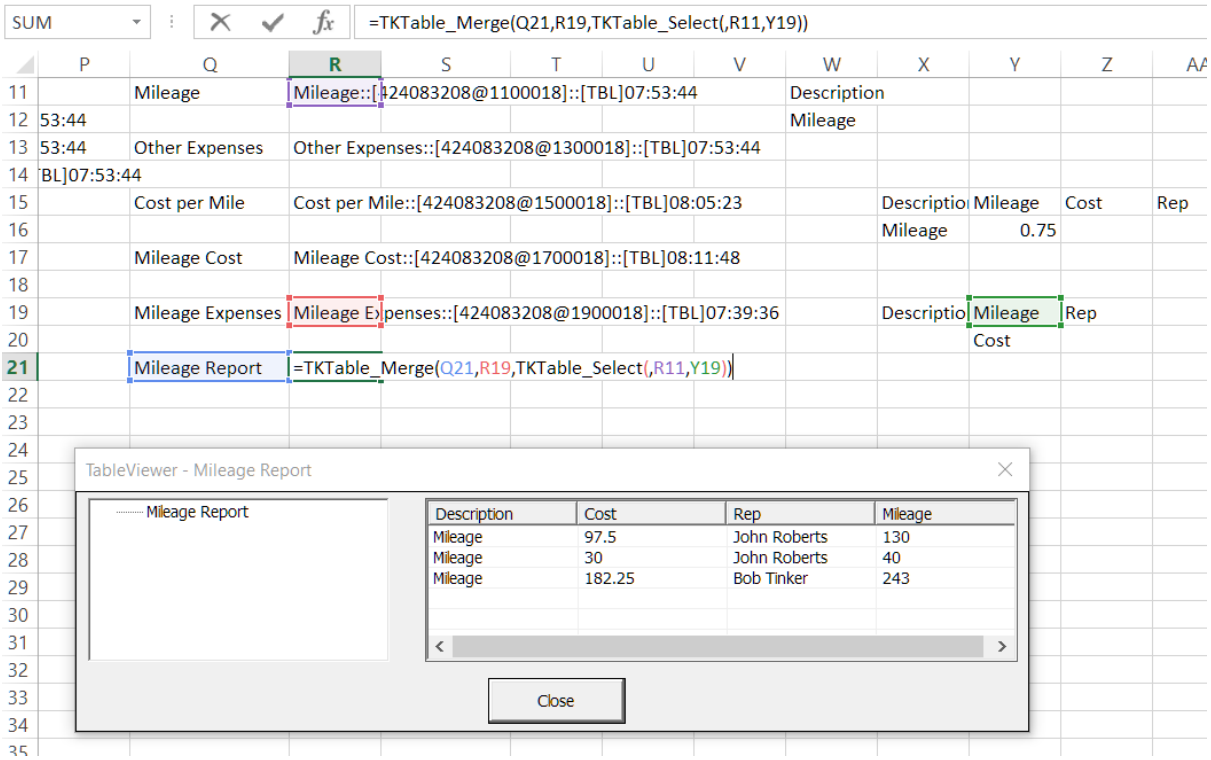


This has removed the empty Cost column but left us with the Costs under Mileage. We can use the TKTable_Select() function to rename columns by entering an Alias for the Key underneath the Key in the list supplied.



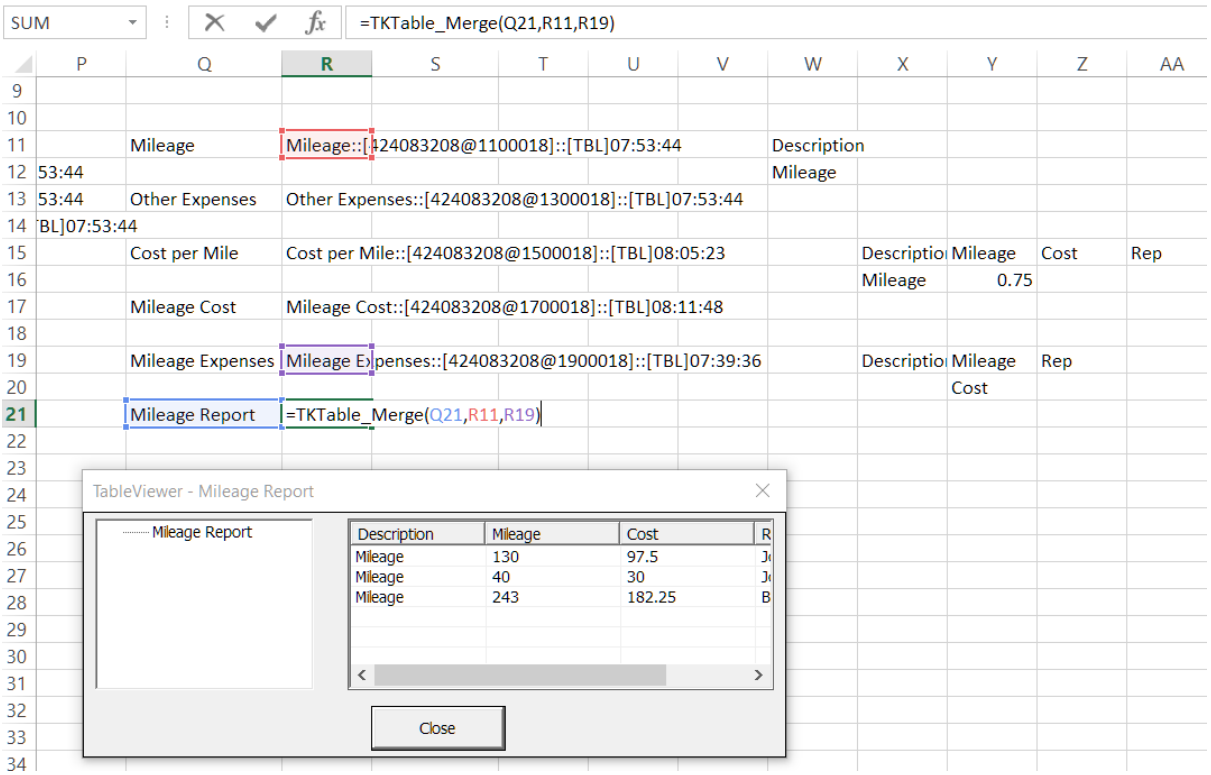
By specifying the Alias we have now been able to rename the Mileage Column as the Cost column. Now we are just missing the original Mileage Column.

The TKTable_Merge() function allows two Tables to be added together so the Results Table contains Keys (and Data) from both of the source Tables. If we select the Mileage data from the original Mileage Table we can merge it with the Mileage Costs to create the full Mileage Report.

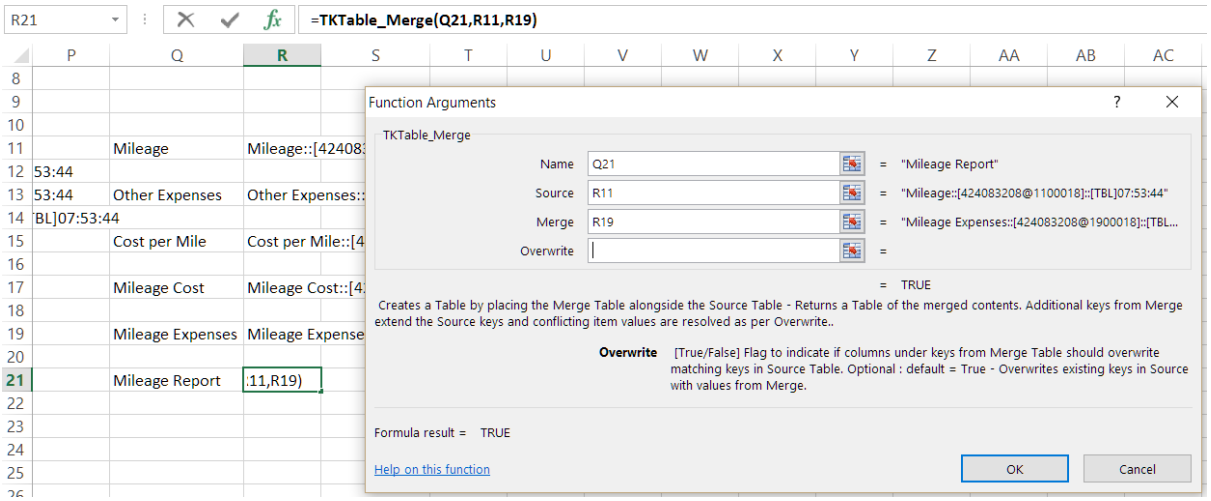


To keep the spreadsheet short we have combined the select and merge operation into one, the TKTable_Select() function returns a Table with the Mileage information and then this is joined onto the results of the Cost calculation. While embedding multiple functions in one cell tends to reduce the overall congestion on a spreadsheet it can make it more difficult to diagnose problems.

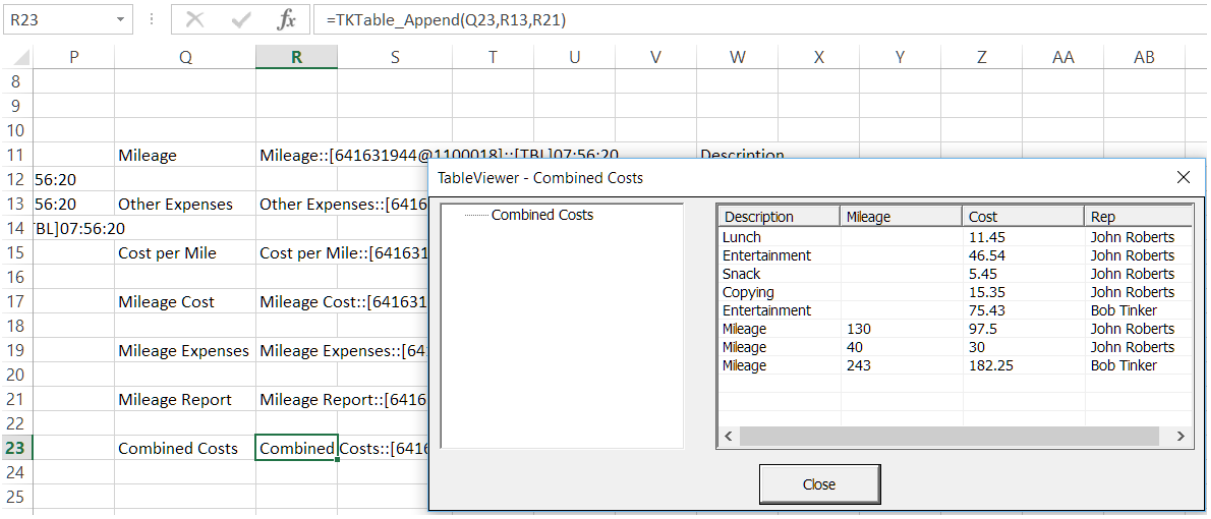
An alternative approach to reaching the same end is to use the ability of TKTable_Merge() to replace data in the first Table with data from the second where the Keys match.



This is a simpler approach which removes the need for the embedded TKTable_Select() call. If data in the First Table needs to be preserved then the Overwrite argument is set to FALSE.

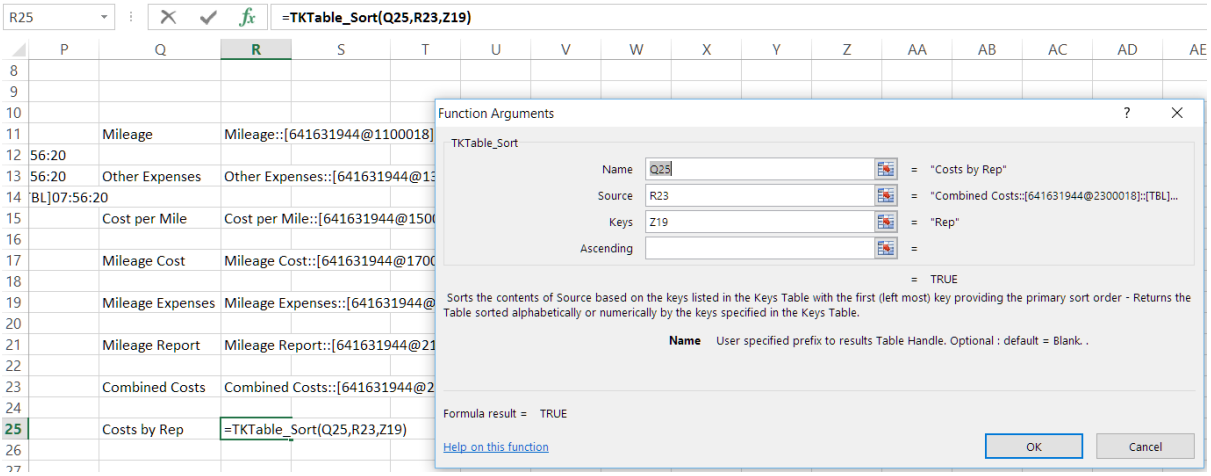


To complete the generation of the expenses report we can append the mileage details with the non-mileage expenses by appending one Table to the other.

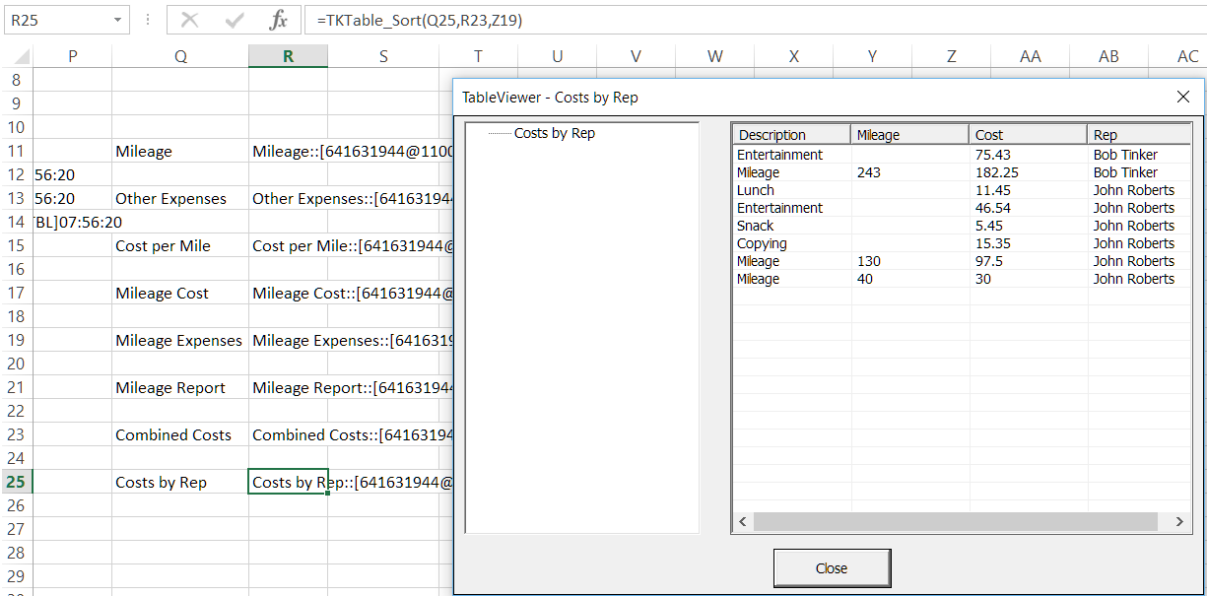


Sort & Transpose

The final step is to rearrange the data by the Sales Representative who had claimed the Expenses. The TKTable_Sort() function provides the ability to sort data on one or more Keys. Similar to the Filter and Select functions, a Table containing the list of Keys to sort on is provided alongside the Table to be sorted.



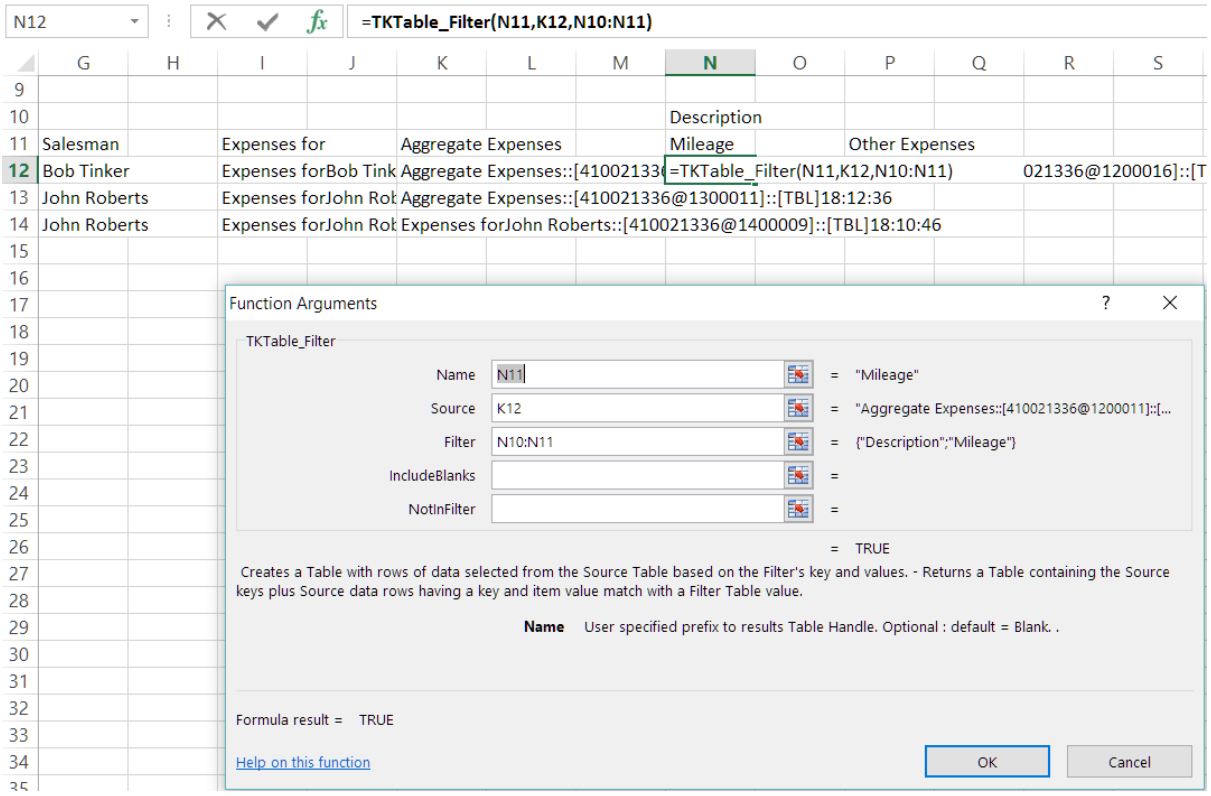
The Results can be sorted in Ascending or Descending order.



Advanced Topics

Using Range instead of Table Handle

Many of the PersisTable functions take a Table handle as input. Where the input data is a simple range of cells, rather than creating a Table and then passing that as an argument it is possible to select the range of data required as the input and pass that directly to the function. The Table constructed is not named and cannot be accessed elsewhere but such an approach can simplify the construction of sheets. Note that in these cases the Keys are always assumed to be in the first row.

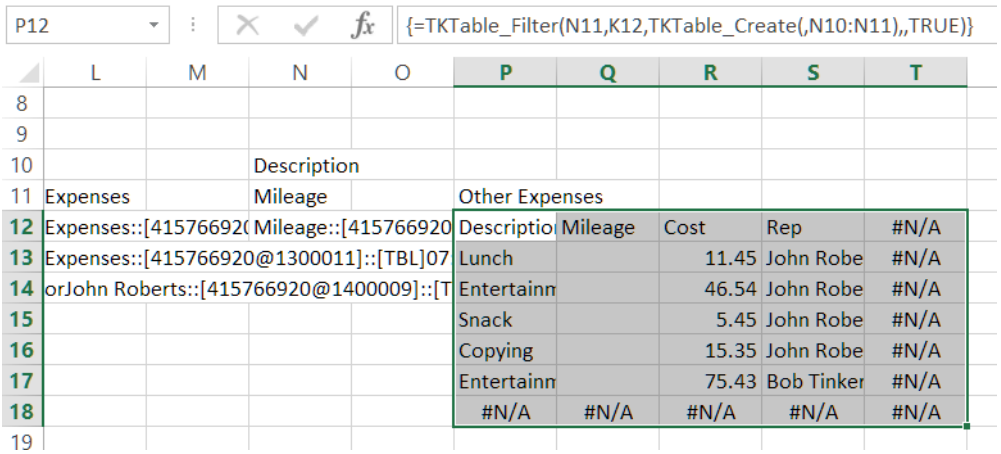


Here the filter table is defined as an input range rather than as a handle for a previously defined table.

Displaying Ranges instead of Table Handle

A function that returns a Table Handle can have the data directly displayed on the output sheet using the array function capability of Excel. When an array function is entered for a function that normally returns a handle, the results of the function are displayed in the range as if the TKTable_Get() function had been called.

This allows instant access to the results but the flexibility provided by the TKTable_Get() function is lost. The results are always displayed in a column oriented manner, there is no ability to define the fill character and no warning of data not displayed.



Here the non-mileage expenses have been displayed directly onto the sheet by using an array formula on the results of the TKTable_Filter() function. It has the Keys at the top and no fill character specified.

Loading a worksheet containing Tables

When a Table is created on a spreadsheet, it is held in memory while the Excel session is open. Once a spreadsheet is saved and then the Excel session is closed, the Tables are discarded. When the spreadsheet is reopened the Table Handles visible on the worksheet will not reflect Table data until they have been recalculated.

Unfortunately Excel does not recognise that the Handles are inactive when they are first loaded so they must be manually recalculated. This is most easily done using the global recalculation command by pressing [Ctrl][Alt][Shift][f9] together. This ensures that Excel will recalculate the whole sheet.

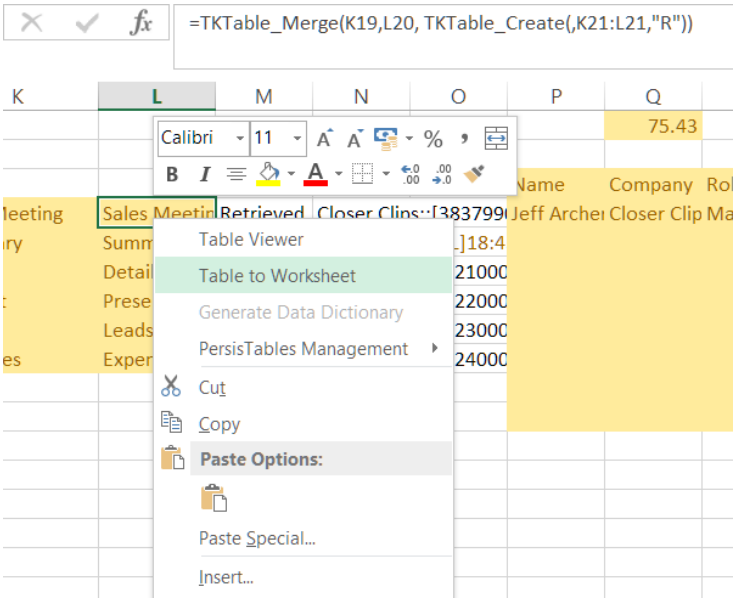
The problem is that as far as Excel is concerned Tables are created as a side effect of the PersisTable function, so TKTable_Create() generates a Handle and when Excel examines the cell it considers the cell is up to date, it does not know that the Table is not yet in existence. The forced recalculation will ensure the Tables are regenerated when a worksheet is reloaded.

It is generally quite obvious when Tables have not yet been calculated, if a cell is changed which requires a PersisTable function to be recalculated where the inputs include an uncalculated handle, the results for that function and all dependent function will report an Error “#Err – ...” with the message including the text “need to recalculate precedents”.

If an expensive calculation operation is associated with the output from PersisTable function then it is advisable to include some guard around the calculation so it is not unnecessarily calculated when the sheet is loaded.

Table to Worksheet

When looking at the Table Viewer in the Excel context menu you may have noticed another related command. The ‘Table to Worksheet’ menu will output the full contents of the table onto a new worksheet at the end of the workbook. As with the Table Viewer this menu option is only available if the selected cell contains a valid table handle. If a cell containing what looks like a table handle does not give this menu option then the cell needs to be recalculated.



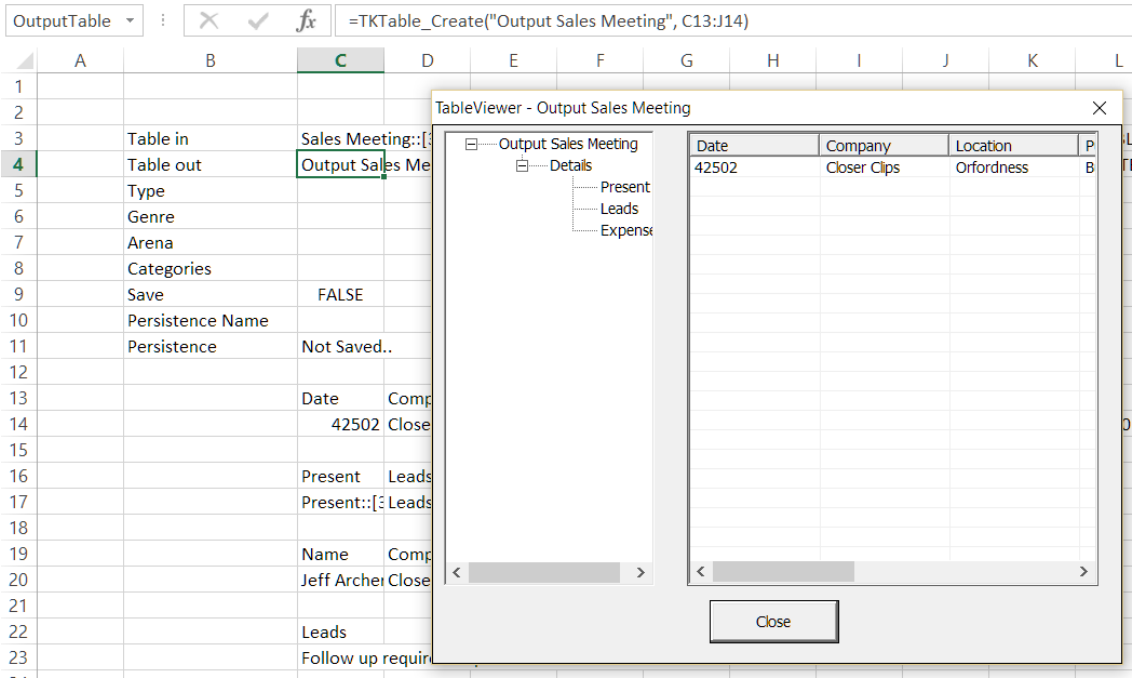
The “Table to Worksheet” command loads a standard template sheet into the workbook and pastes the Table Handle and its content onto that sheet. It then dumps the contents of the Tables onto the worksheet. The data is captured in a series of TKTable_Create() functions recreating the original hierarchy constructing an updated version of the Table.

As the raw data is output onto the sheet in the layout of the original table (rather than being extracted through the TKTable_Get() array function), it is immediately editable and the changes are automatically added into the copy of the original Table.

The template sheet provides a comparison between the original Table and the new Table so changes can be audited. The template sheet also allows the object to be saved if required.

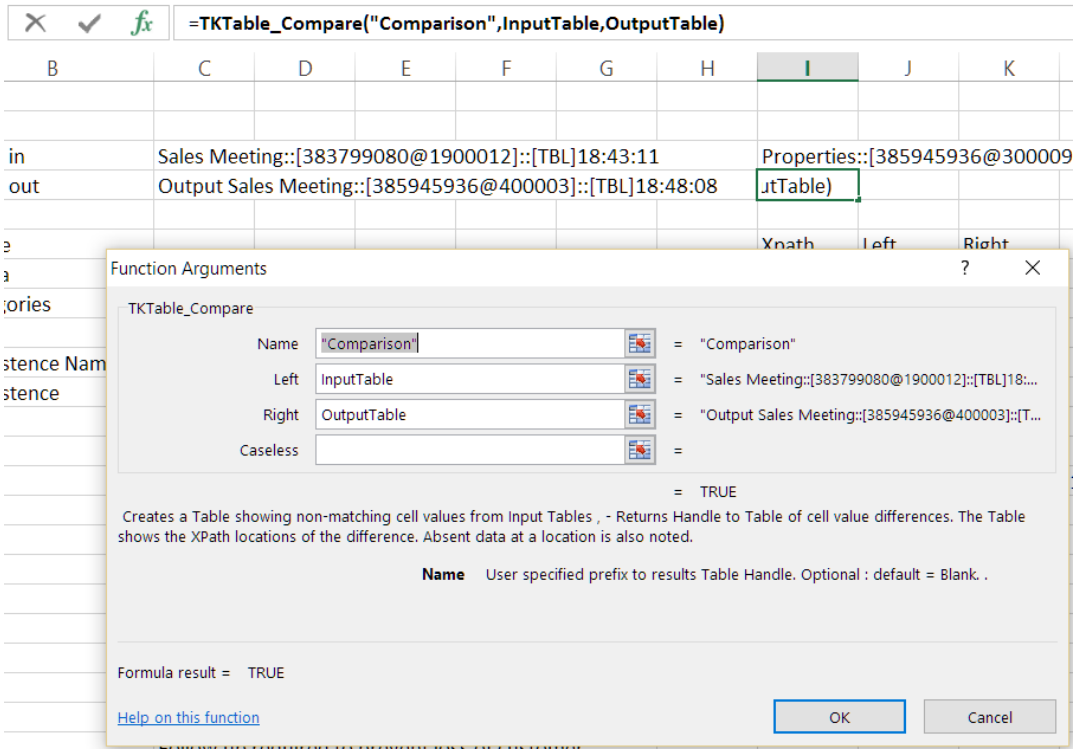
	A	B	C	D	E	F	G	H	I	J	K	L	M	N
1														
2														
3		Table in	Sales Meeting::[383799080@1900012]::[TBL]18:43:11						Properties::[385945936@300009]::[TBL]18:46:19					
4		Table out	Output Sales Meeting::[385945936@400003]::[TBL]18:46:20						Comparison::[385945936@400009]::[TBL]18:46:20					
5		Type												
6		Genre												
7		Arena												
8		Categories												
9		Save	FALSE											
10		Persistence Name												
11		Persistence	Not Saved..											
12														
13			Date	Company	Location	Purpose	Summary	Rep	Total Expe	Details				
14			42502	Closer Clip	Orfordnes	Bulk Clips	5mm and (	John Robe	75.43	Details::[385945936@1400010]::[TBL]18:46:20				
15														
16			Present	Leads	Expenses									
17			Present::[3	Leads::[38	Expenses::[385945936@1700005]::[TBL]18:46:20									
18														
19			Name	Company	Role									
20			Jeff Archer	Closer Clip	Manufacturing Manager									
21														
22			Leads											
23			Follow up required to prevent loss of customer											
24														
25			Descriptio	Mileage	Cost									
26			Mileage	243										
27			Entertainment		75.43									

Here the Meeting Report has been output on the template sheet. Each Table of data is laid out with the Keys on a row and the values displayed below. The Table is reconstructed in the 'Table out' cell at C4. This can of course be viewed in the Table Viewer



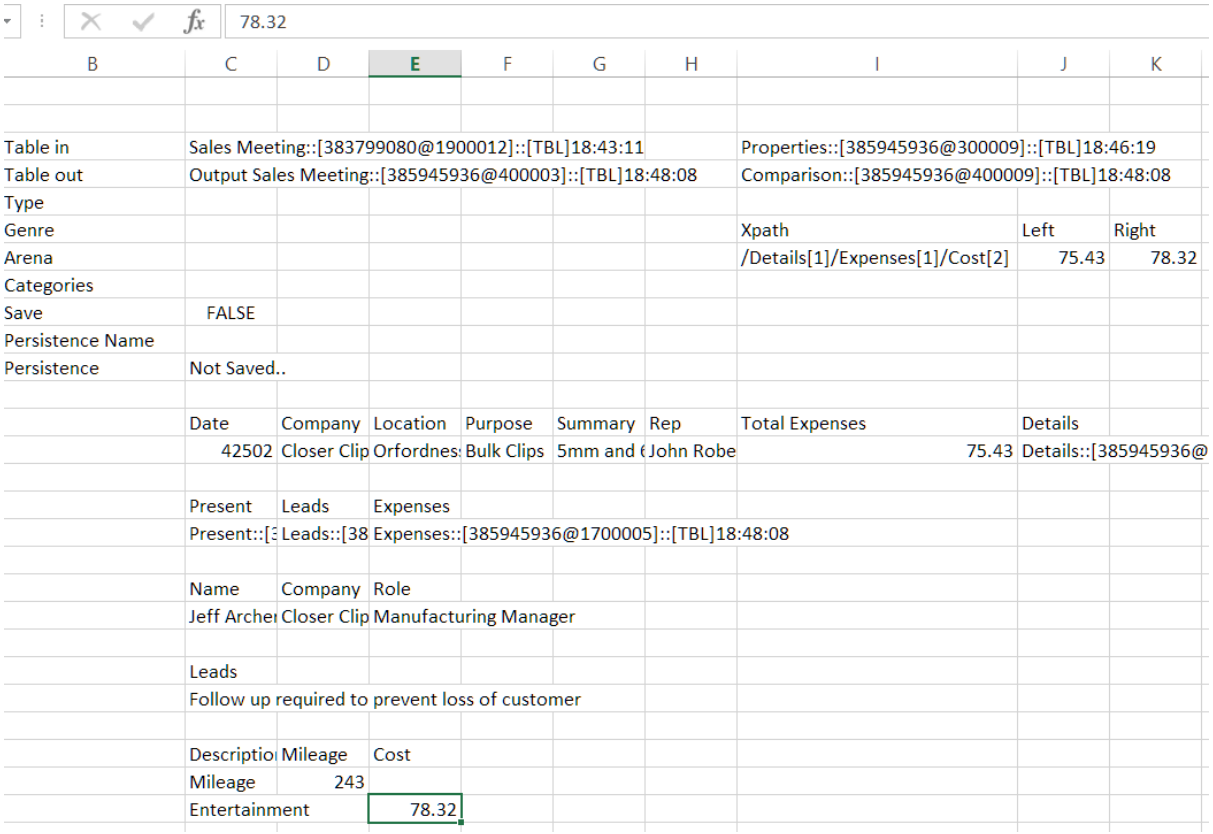
By filling in the Arena, Genre, Category and Name the Table can be saved to the appropriate location in the persistence Store. This will only be done if the Save Cell at C9 is set to TRUE.

Changes to the data in the Table can be tracked in the range I6 to K11. We use the TTKTable_Compare() function to identify differences between the original Table and the Output Table.



The Compare function takes two Tables and describes the differences between them ass a set of XPaths and the different values found at those XPaths. If an XPath does not exist in one Table then it is reported as Missing.

Changing the Amount paid for Entertainment by overtyping the original value creates a new Table with the altered amount.



	B	C	D	E	F	G	H	I	J	K
	Table in	Sales Meeting::[383799080@1900012]::[TBL]18:43:11						Properties::[385945936@300009]::[TBL]18:46:19		
	Table out	Output Sales Meeting::[385945936@400003]::[TBL]18:48:08						Comparison::[385945936@400009]::[TBL]18:48:08		
	Type									
	Genre							Xpath	Left	Right
	Arena							/Details[1]/Expenses[1]/Cost[2]	75.43	78.32
	Categories									
	Save	FALSE								
	Persistence Name									
	Persistence	Not Saved..								
		Date	Company	Location	Purpose	Summary	Rep	Total Expenses	Details	
		42502	Closer Clip	Orfordnes	Bulk Clips	5mm and (	John Robe	75.43	Details::[385945936@	
		Present	Leads	Expenses						
		Present::[	Leads::[38	Expenses::[385945936@1700005]::[TBL]18:48:08						
		Name	Company	Role						
		Jeff Archer	Closer Clip	Manufacturing Manager						
		Leads								
		Follow up required to prevent loss of customer								
		Descriptio	Mileage	Cost						
		Mileage	243							
		Entertainment		78.32						

The change is reflected in the comparison area with the full XPath to the modified value[22].

PersisTables memory management

The creation of a Table consumes computer memory. The operation of Excel makes it difficult to allow for a sensible automated garbage collection so tools are provided to allow a user driven manual process to free memory when necessary. It should be noted that the amount of memory consumed by a Table is generally small so it is only after extensive use of Excel with large Tables that any garbage collection operation is required.

Table Handles

A Table is referenced through its Handle that is returned when the Table is created through a toolkit function. The Handle is constructed from the user specified name, a unique identifier which links it to a domain and a suffix that consists of a Table indicator [TBL] and the time of creation.

Handle = [User Name]::<domain>@<identifier>::TBL<time format hh::mm:ss>

User Name is an optional element and will be left out if not supplied. It is used to provide a description of the Table contents.

Domain describes the source of the Table– typically it is the Excel worksheet identifier of the sheet it was first created in.

[22] Note the Expenses total is not automatically recalculated as we no longer have the formula which calculated this in the original sheet.

Identifier is used within a domain to establish whether the Table can safely replace a previous version.

Time is the time of creation of the Table

This handle can then be referenced from any other PersisTable function used during the current Excel session. Typically this is done through a cell reference but the Handle can be copied as a string and still remain valid. It is for this reason that it is non-trivial to provide an automated garbage collection process. This reference remains active even after the spreadsheet creating the Table has been closed.

On a spreadsheet the domain is the Excel worksheet identifier with the cell location being provided to ensure a unique reference within that domain. When a cell is recalculated and as long as the user name does not change, the new Table will replace the existing Table associated with that cell. Therefore there will be no leakage of memory caused by repeated calculation of an unmodified worksheet.

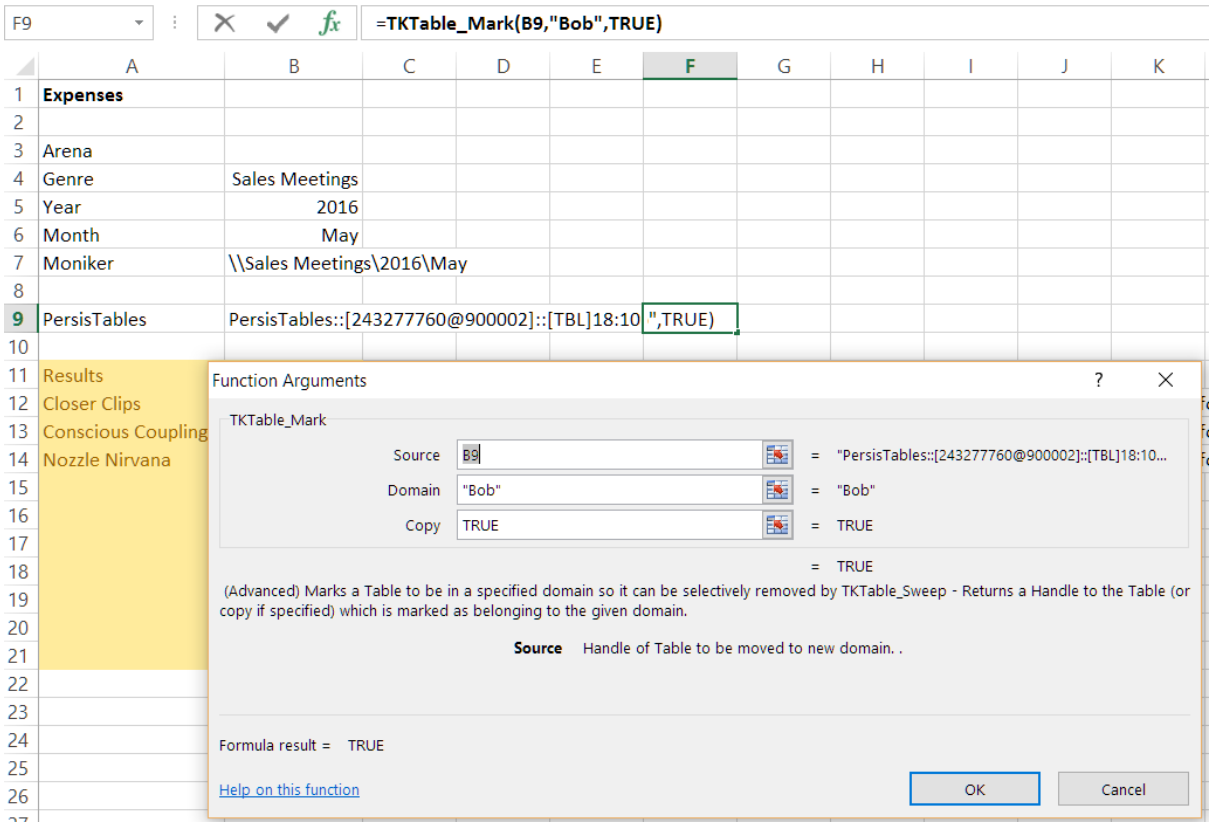
Mark & Sweep

If a workbook containing Handles is closed within an Excel session, it is not possible to guarantee

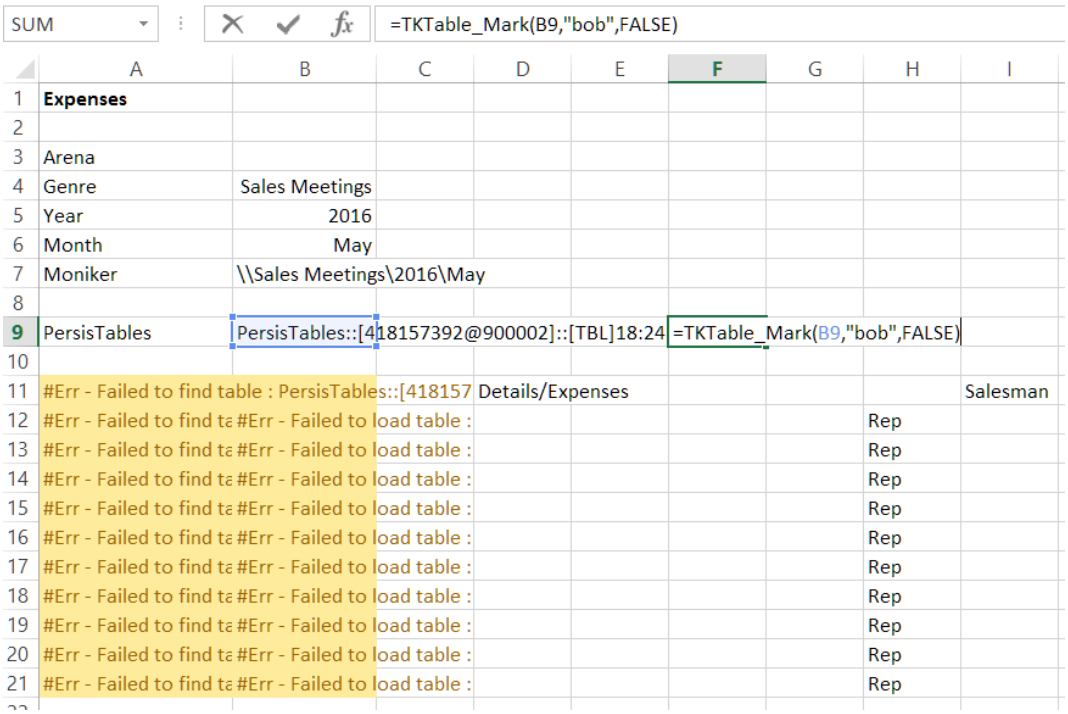
that there are no copies of Handles referencing a Table created within that workbook so the Tables are retained in memory until the Excel session closes. Over time this can create a memory footprint that is significant. The Mark/Sweep/Flush functions provide user control over the consumption of memory.

The TKTable_Sweep() function will remove all Tables from a specified domain. As all Tables on a worksheet belong to a domain given by the worksheet id it is possible to implement a VBA macro which is run as a workbook closes that sweeps all Tables out of memory that belonging to that workbook.

If a Table is required to exist after the worksheet is closed it can be moved or copied to a separate domain by using the TKTable_Mark() macro. This creates a copy of the table in the domain specified and this will continue to be available until TKTable_Sweep() operates on that domain. The domain name used in the TKTable_Mark() function can be any string.



The Table created when loading the list of PersisTables is copied into the Domain named ‘Bob’. A copy is made because otherwise the original Handle becomes invalid.



In this case the data within the Table produced by the request for a list of PersisTables is no longer valid as the TKTable_Mark() function has moved the Table into domain Bob rather that 418157392, the Excel id for the sheet being calculated.

The domain parameter of the TKTable_Create() function assigns a newly created Table to a specific domain. This removes the need to mark it explicitly.

However tables created as a result of other toolkit operation will always be allocated to the default domain so these will need to be transferred to the appropriate domain using the TKTable_Mark() function.

If Tables are created through the use of VBA macros and no domain is specified, the handles are given the domain id of “vba”. The objects are provided with a unique id which is just an incremented number. There is no specific reuse of such tables and there is a danger of memory leaks unless the user ensures these objects are deleted.

The TKTable_Flush() function will remove all objects from memory. If this is executed any function that relies on a Handle to access Table data will fail when next executed. Generally the solution is to force a full recalculation to recreate the Tables using [Shift][Ctrl][Alt][F9]

Appendix

Array Functions

This is an Excel technique that allows a number of cells to perform the same function operation or to return an array of data to a sheet from a single call to the function. Essentially the function is entered into a single cell. The surrounding area that represents the array of cells where the results are to be displayed is highlighted with the focus on the cell containing the macro. The F2 key is pressed on the highlighted cell followed by [Ctrl][Shift][Enter} keys which are hit simultaneously. The formula will appear in the formula bar surrounded by {} when this has been successfully done.

While using array function is a powerful way to execute Excel macros the cells making up the array cannot be individually modified. The formula can be modified in the formula bar and the [Ctrl][Shift][Enter] key combination re-entered to apply the change to all cells. Modification otherwise requires the array to be deleted and replaced. Note it is possible to extend an existing array macro to cover more cells as long as it includes the entire original array in the new array.

Excel functions that operate on PersisTable output

Excel functions that operate on arrays can accept the output of TKTable_Get() as input. This includes statistical functions such as AVERAGE, STDDEV etc and lookup functions such as VLOOKUP and MATCH.

Simply providing the TKTable_Get() function where an 'array' argument is allowed will allow the Excel function to operate on the data held in the Table.

Table serialisation plug-ins

The PersisTables toolkit allows a developer to define their own format of data when persisting or loading data. This is done by building a "plug-in" component that translates the input data into Table format and reverses the procedure when saving the data.

This feature may be used where there is a desire to load data from existing sources which do not conform to the available PersisTable data formats or to save processed information in a form that can be directly accepted by other systems.

Plug-ins need to be loaded from a known location. This location is defined in the Library details of the Properties settings (See Installation and Configuration Guide for more details). New plug-ins can be saved alongside the existing plug-ins shipped with toolkit. More adventurous users may add plug-ins in alternative locations but must update properties to indicate where the plug-in is located.

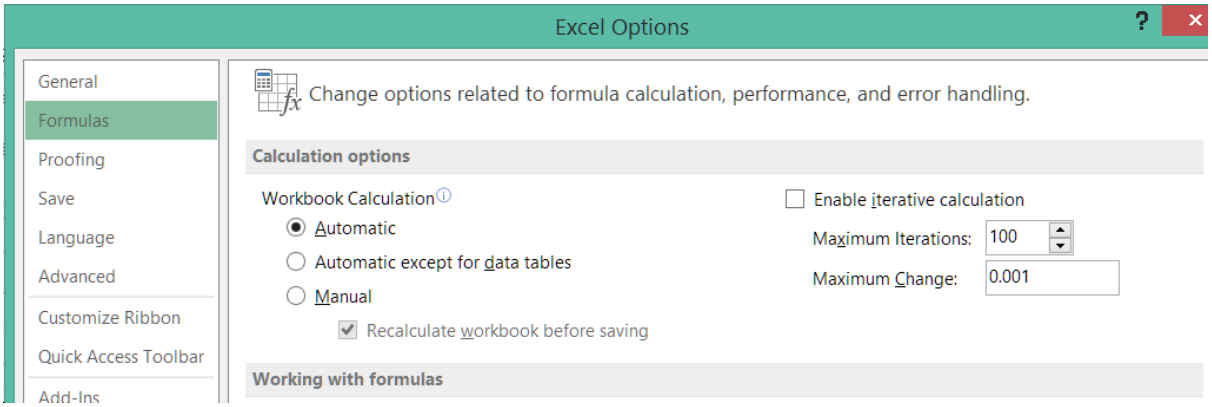
Plug-ins can be built using the PersisTable SDK. The installation and configuration of plug-ins is covered more fully in the SDK documentation that accompanies the SDK.

Errors

PersisTable Toolkit errors are prefixed with "#Err –". There is no definitive list of error messages at present. While every attempt is made to ensure the error messages are meaningful, this is an ongoing process and changes will be made to enhance their diagnostic contribution.

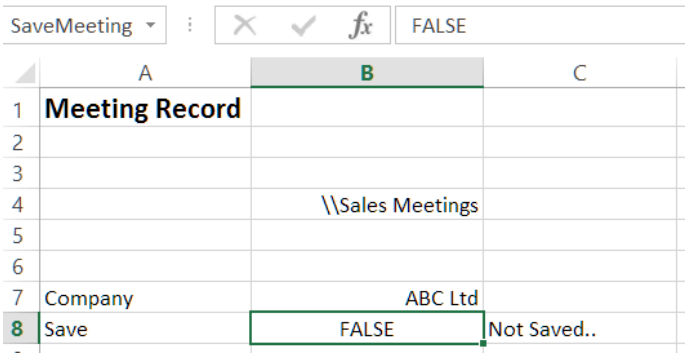
Setting Automatic Calculation Mode in Excel

In the File->Options menu (Excel 2013 – may be elsewhere in other versions of Excel) the Formulas option gives access to the Calculation Mode settings



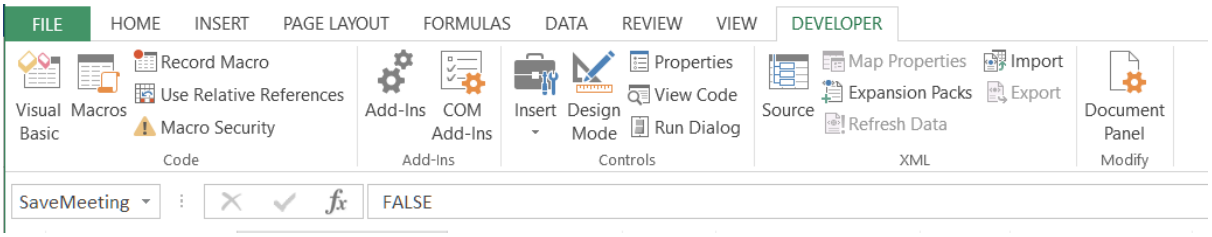
Create an Update Macro

First give the Guard Cell a Name. This can be done in the Name Manager Dialog in the Formula Menu Ribbon but is easily done by selecting the Guard Cell and typing the Name into the box to the left of the formula Ribbon.

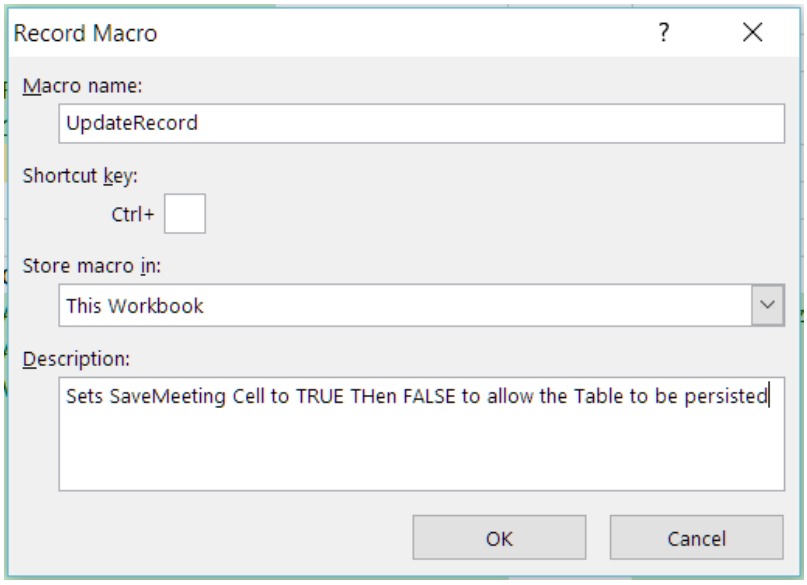


Here we have named the cell a SaveMeeting.

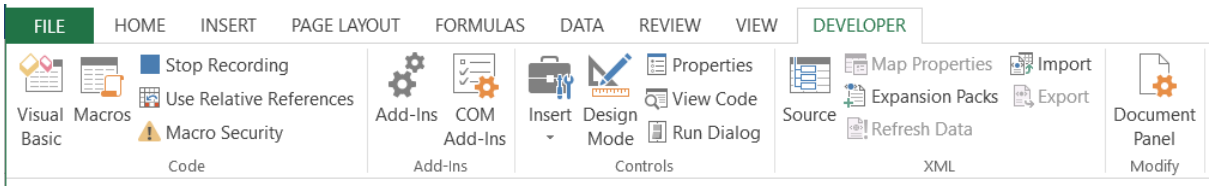
Now set up to Record a Macro, this is on the Developer Menu Ribbon which maybe hidden in the default setup of Excel.



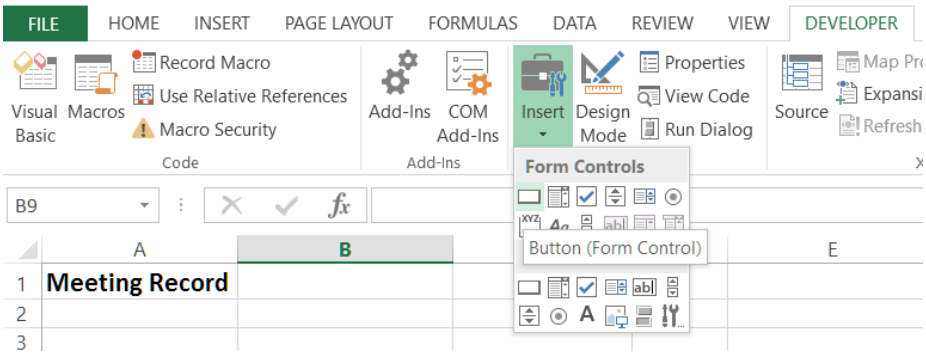
Press the Record Macro button. This will display a dialog box allowing the macro to be named



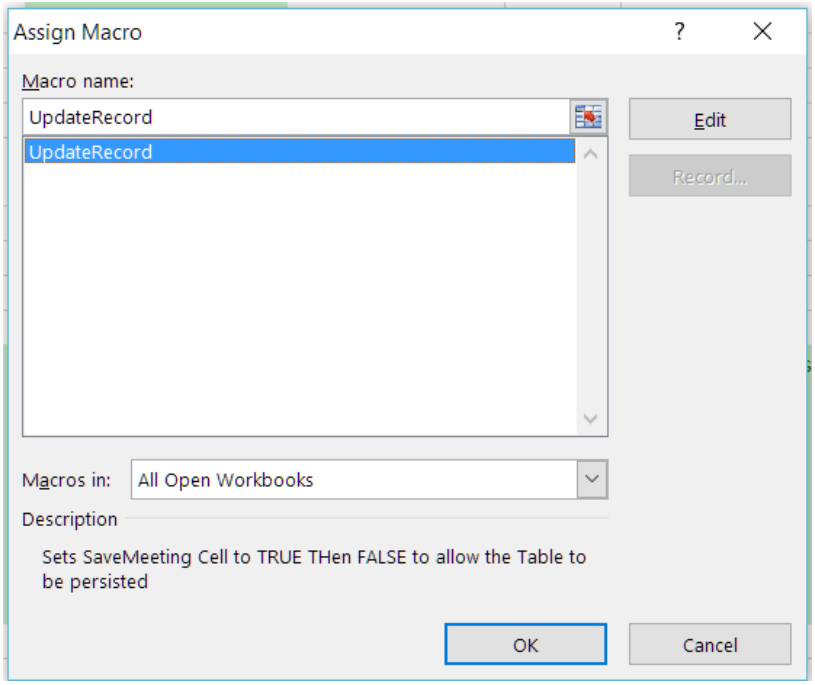
Now select the Guard Cell and enter TRUE followed by FALSE into the Cell. Then press the Stop Recording button



This will have created a macro that can be executed from a command button.

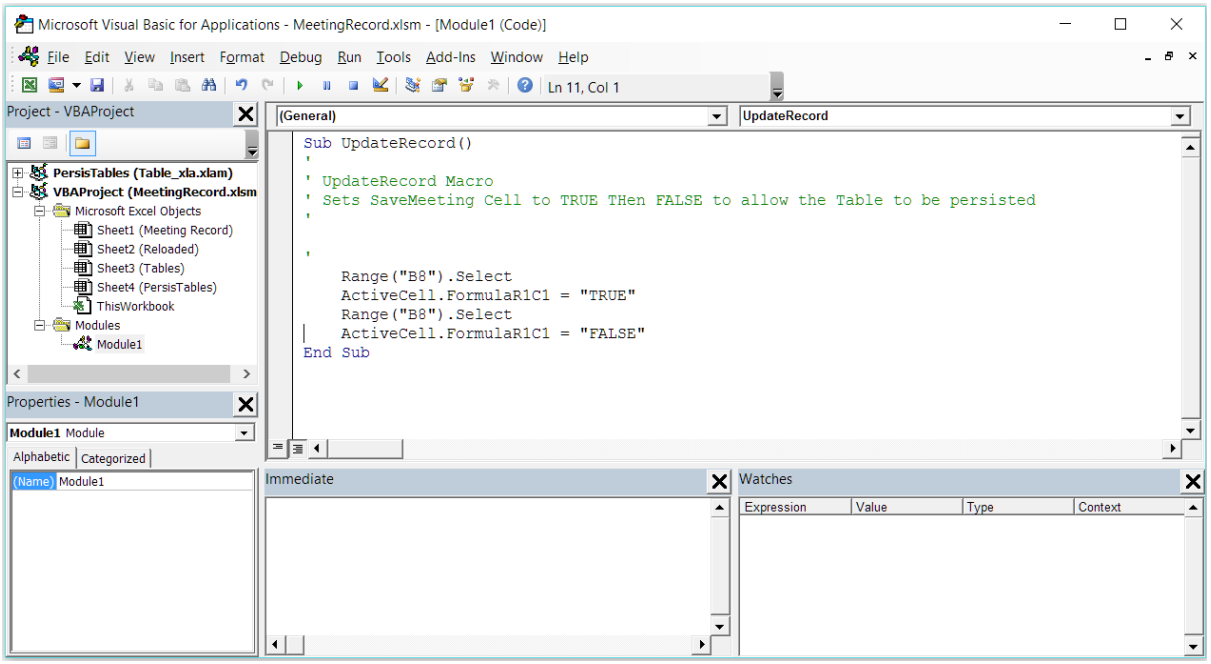


The command button can be created using the Insert Form Controls in the Developer Menu. Clicking on the Button (Form Control) displays the Assign Macro dialog.



Selecting the Macro just created will associate the macro with the button being clicked. Once the OK button is clicked the button will appear on the spreadsheet. Right clicking on it and selecting 'Edit Text' allow the button caption to be set.

Opening the Visual Basic editor will show the macro in Module1 of Modules which should look like this



The button should now trigger the Guard Cell to allow the PersisTable to be saved and force it to be reloaded.

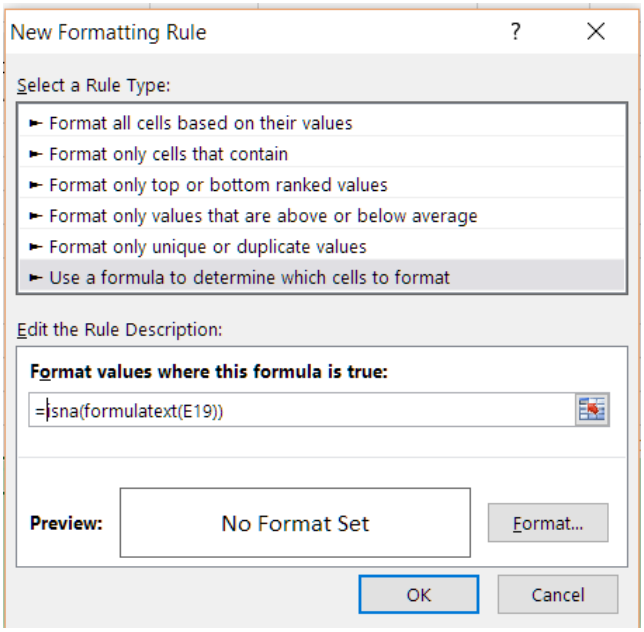
Conditional Formatting of Formula Overwrite

To highlight a cell when a formula contained in the cell has been overwritten by the user, the following steps need to be taken.

Highlight the cell containing the formula.

Select Conditional Formatting Option from Home menu.

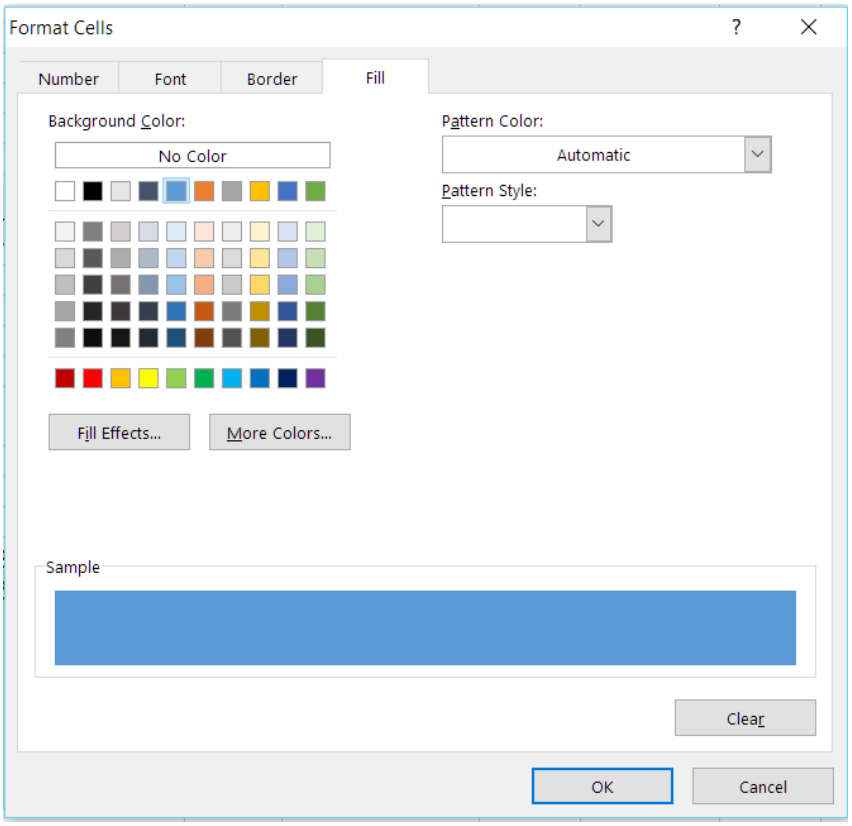
This displays the conditional formatting dialog



Select “Use a formula to determine which cells to format” option and enter the formula

=isna(formulatext(<cell>))

into the formula box. Next select the “Format..” button



Select the Fill tab and choose a suitable colour to highlight modified cells.

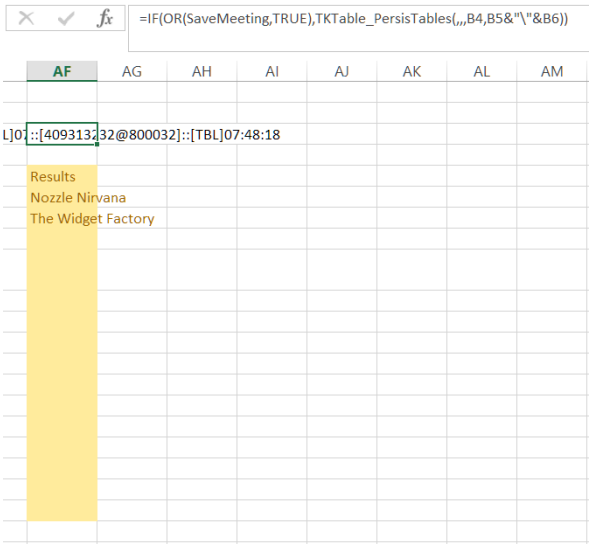
This conditional formatting works because the cells have a formula in to copy retrieved data into the cell. Overtyping the formula with a value will cause the test to return true and therefore the cell will be highlighted in the selected colour.

As long as the formulae in the adjacent cells are the same it is possible to just copy and paste the cell contents across the adjacent cells to get the same effect for all of them.

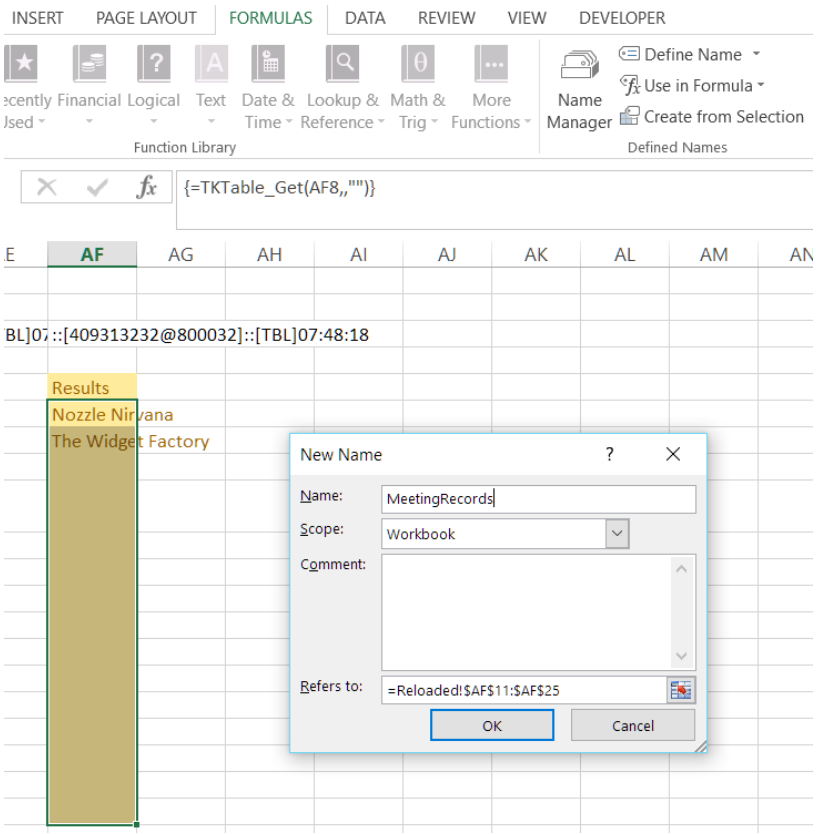
Creating a List from a Table

It can be useful to create a pick list from the contents of a Table. Here is a quick description of the process. The Meeting Reports worksheet demonstrates the end results.

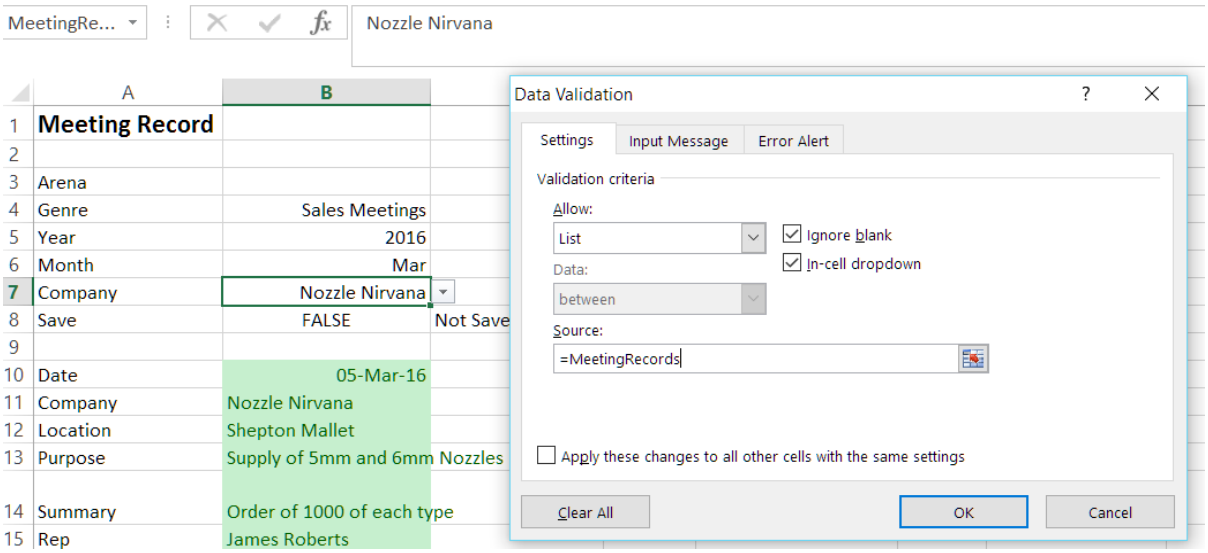
In this example the TKTable_PersisTables() function has created a Table listing the PersisTables in the Sales Meetings Genre under the Categories 2016\May. The contents has been extracted into the Yellow Area of the screen using the TKTable_Get() function and using a blank fill string.



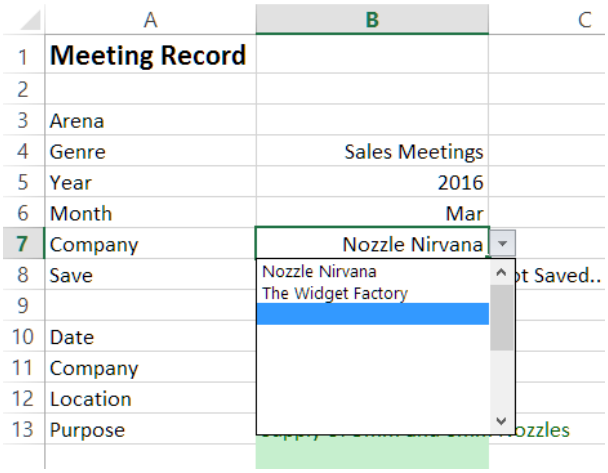
The first step is to create a named Range for the results excluding the Results heading. We could just highlight the range holding the list of calendar names and defining a name for the range using the Define Name command on the Formulas menu tab (Excel 2013)



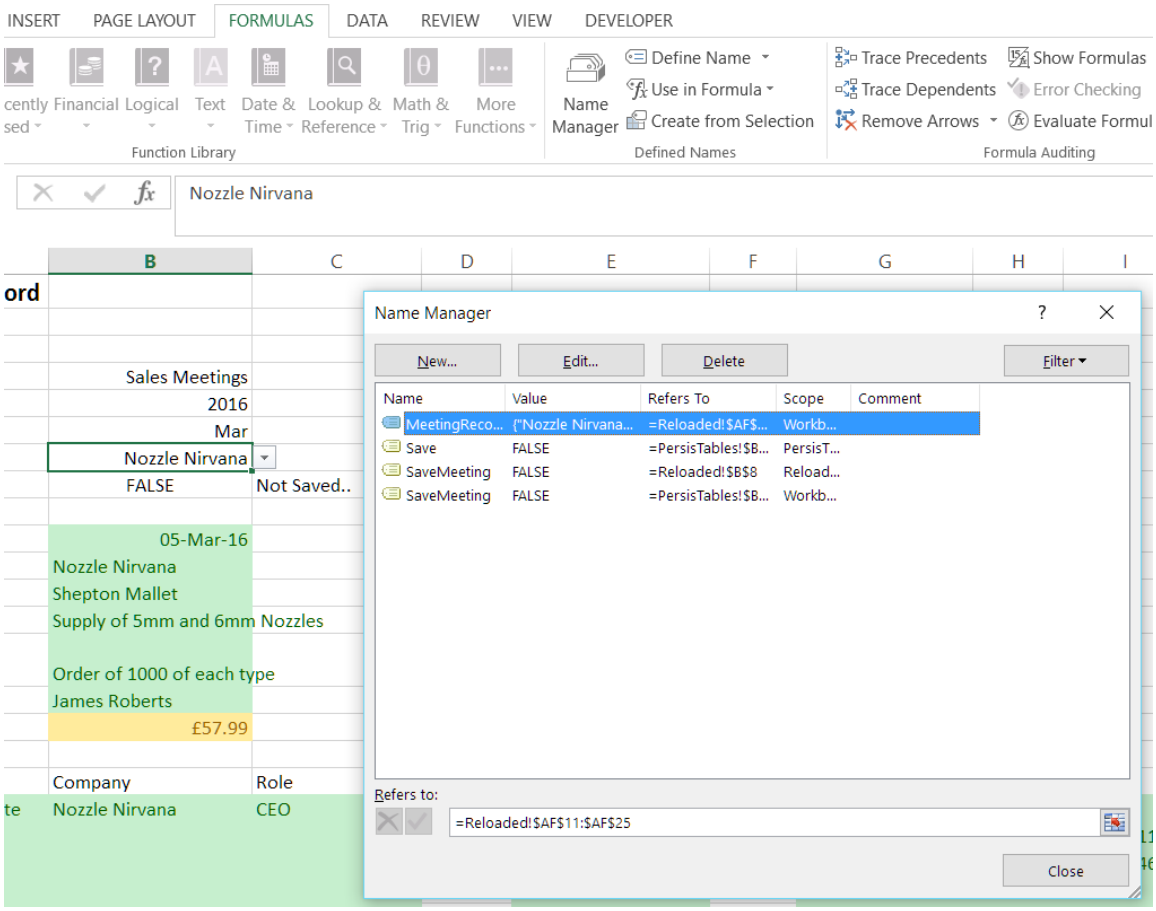
We can then assign the list of names to a drop down list on the cell we want to specify the Meeting Reports name to load. This is done using the Data Validation option on the Data Menu tab (Excel 2013). Selecting List as the Allow type and assigning the range name ‘MeetingRecords to the Source will set up the selection we want.



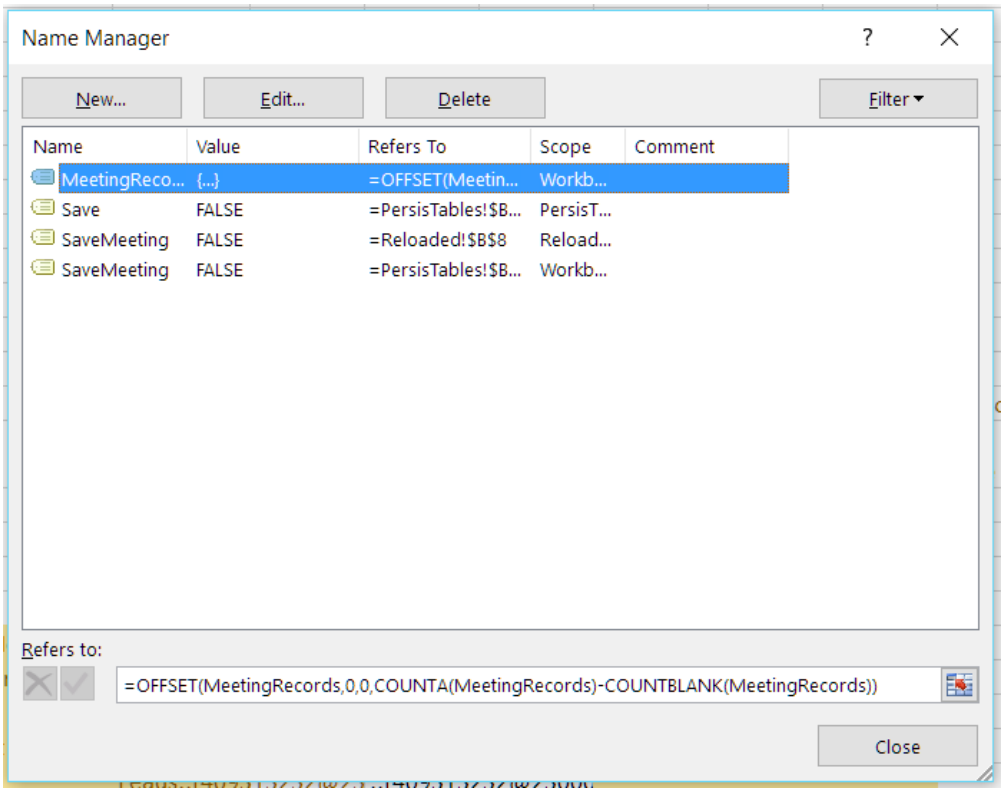
We can now see the list of Meeting Reports available to load



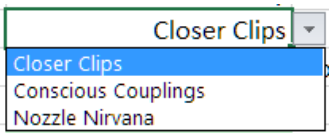
Note that we have a lot of blank lines in the list, this is the empty cells in the range. We can improve the behaviour by creating a dynamic range. This will adjust to the length of the list of items we need to display. We need to go back to the 'Name Manager' on the Formula Menu tab.



We select the MeetingRecord and press Edit. This allows us to redefine the range. If we use the Offset formula we can allocate a dynamic range including just the PersisTable Names.

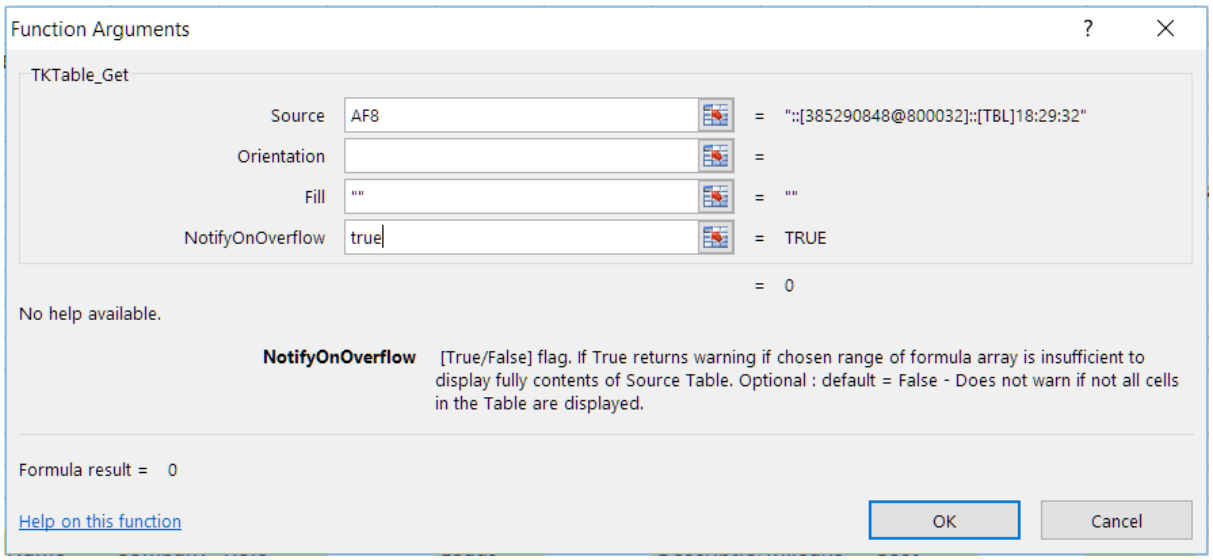


Here we have defined a formula that creates a range matching the PersisTable names listed. When we now look at the drop down list we can only see these items. The Offset command allows a range to be dynamically defined. By selecting the first item in the list and counting the number of rows populated in the original range we can adjust the size of the range to match the items returned from the TKTable_Get() function.



This is a useful technique for allowing a user to navigate through the PersisTable store without using the Persistence Browser.

Note that the data extracted from the Table must fit into the range supplied. To prevent data disappearing the TKTable_Get() function can have the NotifyOnOverflow flag set to true.



This will highlight when the area assigned for the list of PersisTables available is no longer large enough.

Copyright Acknowledgements

The PersisTables toolkit makes use of a modified version of the XLW version 2.0 package. The following copyright notice applies to that component

Glossary

Table	A Table is the basic unit of information captured by the PersisTable toolkit which can represent a range of data on an Excel spreadsheet. It may be one or two dimensional. It is held in memory and exists for the duration of an Excel session. It can be referenced by PersisTable functions, displayed or saved to an external storage. The data in a Table can reference other Tables to create hierarchical data structures.
PersisTable	A PersisTable is the representation of a Table that has been saved outside of an Excel worksheet and can be reloaded into Excel independently of the workbook it was originally created in.
Handle	Provides the means to access information within a Table, it is a combination of a user defined name and a unique suffix and can be used by the PersisTable Toolkit functions to interact with Table data
Key	Is a tag which can be used to access a slice of data from a two dimensional Table. It typically makes up the first row or first column of data that is passed in when creating a Table. The location of the Keys to a Table are defined by the Orientation of the Table, by default Keys are assumed to be in the first row of data supplied whn creating a Table.
Item	Is the offset within the slice of a Table defined by a Key. A Key and Item identify a single element of data within a Table. Item 1 is the first element beneath the Key while Item 0 is the Key.
XPath	Is a method of describing how to access data embedded within an hierarchy of Tables.
Attributes	These are additional pieces of information that can be associated with the primary Table data to explain or manage that data. For example it holds information on where the Keys were located on the input data when a Table was created so it can be displayed with the Keys in the same orientation when extracted.
Arena	The name of a physical storage location where PersisTables can be saved to Genres. The Arena is associated with a device driver that allows data to be saved and retrieved from that storage location. The current version of the PersisTables Toolkit allows data to be saved to the Windows file system only.
Genre	A Genre is a logical division of PersisTables where the information within a Genre share a similar or identical business function.
Category	A sub-division of PersisTables data stored within a Genre by some relevant criteria to simplify searching for previously created PersisTables
Moniker	The moniker is a short form name that provides information on the Arena, Genre, Categories where a PersisTable should be saved to or retrieved from. It also allows control over the way the PersisTable data is encoded and validated
Macro	An Excel worksheet function or a user defined function available for execution on an Excel worksheet
Array Macro	This is an Excel technique that allows the execution of a single macro to return information into a range of cells.
Template	An Excel template is a special type of workbook that can be loaded into another to provide a specific format worksheet. It is typically used in the Toolkit to support the TableToWorksheet macro which displays a Table on its own worksheet. By creating Genre specific templates and registering them, this macro will load a Table of that Genre onto the template worksheet allowing the user to view the contents in an appropriately formatted manner.