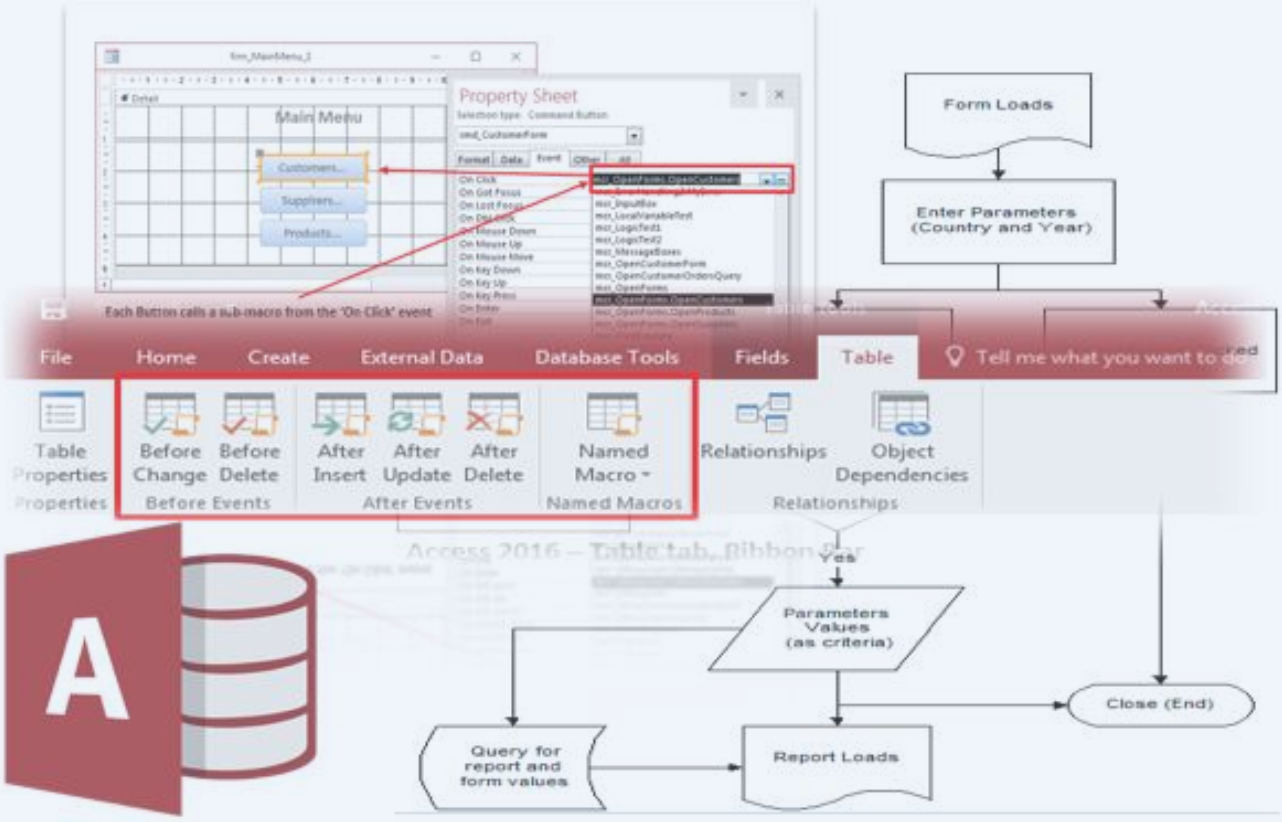
Microsoft Access 2016

Understanding and Using Access Macros



Welcome to Microsoft Access 2016

Thank you for your purchase and commitment to learn and master the tools and features of Microsoft Access 2016.

This comprehensive user guide covers the essential objects, techniques and concepts to get you started in the best possible way that I know and have taught to my many students over the years.

There are two ways you could use this reference guide. It can be used as a general reference guide jumping between sections in any suitable order to fill any gaps in your knowledge and establish an understanding as taught in my training courses or if you're a complete new beginner then simply start from the beginning and read the whole guide first and then go over each section again applying the examples shown to help manage and build a database application.

At first glance, learning about Microsoft Access may be deemed as a steep learning curve but by using the examples and understanding the theories covered in this guide, I aim to flatten that curve to the easiest and painless way possible.

At the end of day, it will require your investment of time and perseverance and having a open and positive mind set as at times it may seem all too much to take in. If this happens, take a break, go for a walk, do something else for a few hours and then revisit this at a later point

It will sink in, I promise!

All the examples used in this user guide can be tested with sample data which is available in the box below:

FREE TO DOWNLOAD

If you require some sample data to test the techniques and illustrations as shown in this document, you can go and download my sample Access database file consisting of six potentially related tables which these examples are all based on.

Sample Access Database File (ACCDB Format) - Zip File

(http://benbeitler.com/Order%20Processing%20Data%202016.zip)

DISCLAIMER OF WARRANTIES/DAMAGES

All software data files is provided "as-is," without any express or implied warranty.
In no event shall the author be held liable for any damages arising from the use of this software.
The user must assume the entire risk of using the software.

PLEASE DO NOT DOWNLOAD UNLESS YOU HAVE AGREED TO THIS DISCLAIMER.

I hope you find this user guide of value and welcome your comments on ways this user manual could be improved or if you wish to get in contact with me, please do so at ben@AccessDatabaseTutorial.com

Once again, many thanks for your commitment – *enjoy!*

CONTENTS

Understanding and Using Macros

Overview of Microsoft Access Macros

Here's a quick overview of what Access macros are all about and when you should use them in your database application.

When programming in Microsoft Access, you have essentially two ways of accomplishing this task. You can use VBA (*Visual Basic for Applications*) or Access macros.

> ***Note:*** *The term 'macro' may be a little confusing when talking about other Microsoft Office applications including Excel, Word, Outlook and PowerPoint as it refers to VBA (the code) which is one of the same. In Access however, they are two completely different processes and the term 'macro' here refers to a lower set of automating tasks (having its own language reference) leaving VBA (stored as Modules) to handle more complex programming functionality.*

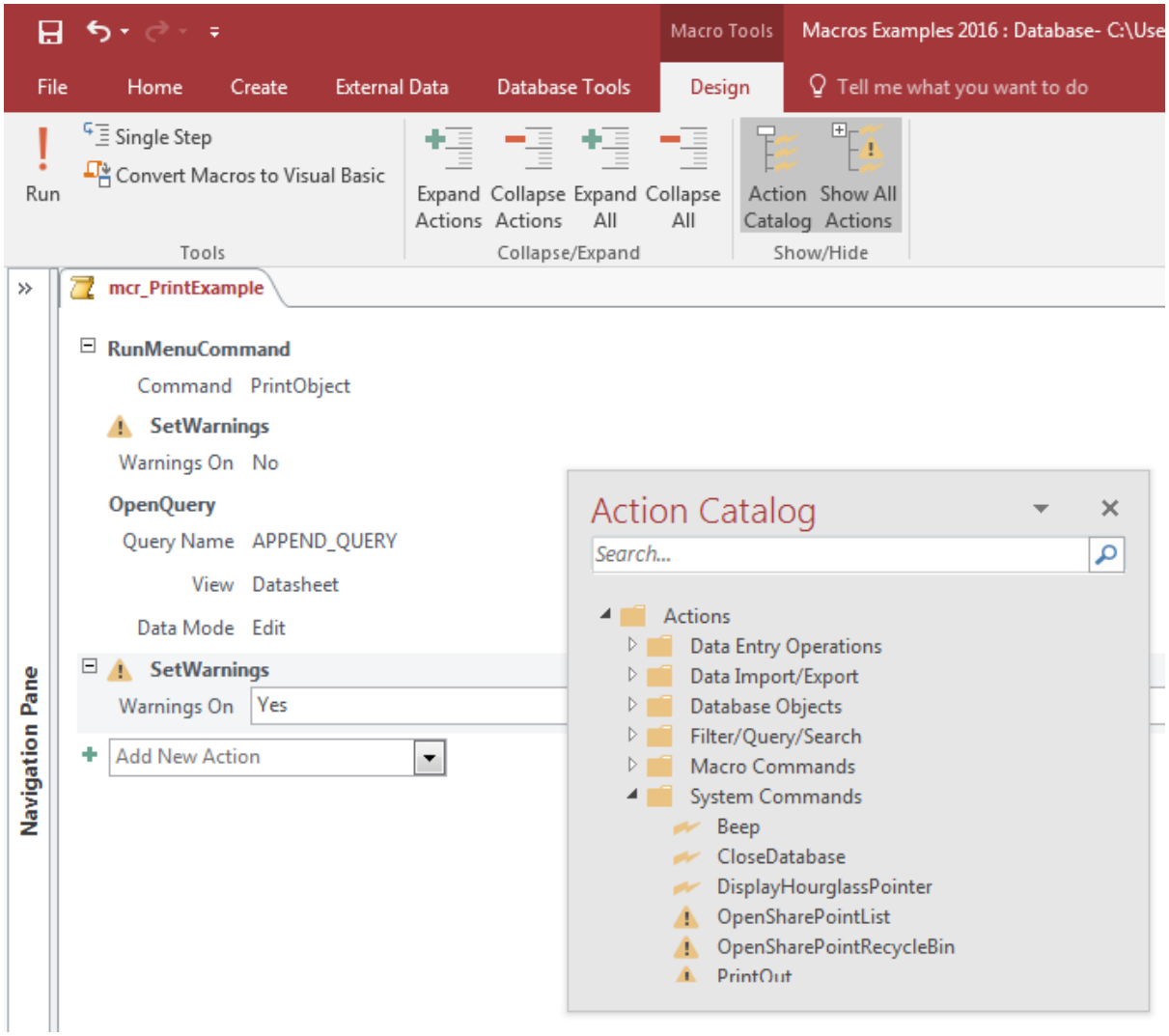
Tasks that macros typically are used for may include:

- Opening a form or a report.
- Executing a query.
- Setting a value to a control (*like a Textbox*).
- Importing data from external spread sheets.
- Exporting data to a PDF file.
- Quitting the application.

Every time you click on a button, open a form or close a report you can call a macro to perform that task (*a procedure*) and start to automate your application. A macro is the simple programming alternative to using the more powerful VBA programming feature of Microsoft Access.

The level of programming knowledge therefore required is '*zero*' and the only skill needed is having the time to learn and having a logical mind (*which by the way, we all have!*).

Continues/…



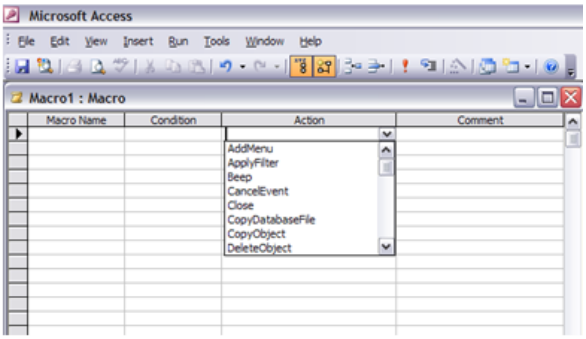
Originally, Access macros was the only way to program and automate your application (*which goes way, way back to the mid 1990's*) and has remained ever present since. During the different releases of Access over the years, Microsoft took a view and though it promoted macros as being the first choice to automating a database it soon leaned towards the more powerful approach of VBA. With the introduction of Access 2010, it included a newer set of macro commands with the addition of a '*Data Macros*' and now has swung the pendulum back to using macros as the first choice.

The one missing element to Access macros which some readers may have noticed is that unlike with Excel, Word or PowerPoint, you do not have a '*Macro Recorder*' feature (*which generated VBA for you*) and therefore means that you will start from a blank canvas and set your commands and parameters manually.

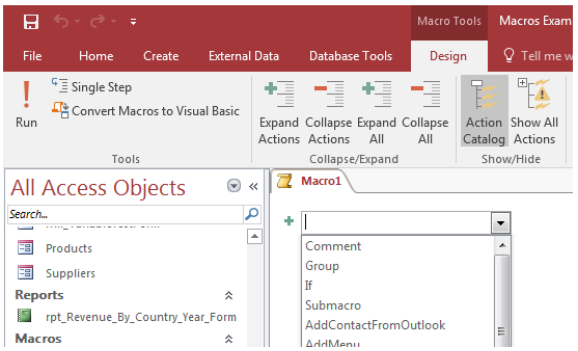
Today, macros still have an important role to play within your application especially for the latest release and can even be combined with VBA to bring a happy balance to controlling the simplest of tasks through to the more advanced routines that VBA is designed to do.

Finally, macro commands have changed (*and been increased*) from version to version and readers will need to be aware that I'm using Microsoft Access 2010 to show the latest features and the complete command list of which there are now 86 key commands. Earlier versions may have different keywords or no command available as illustrated in this guide.

If you are seasoned user migrating from an earlier version namely Access 2003, you will notice how the macro designer has changed for Access 2016 and the screen below shows an example of the new versus the older designer view.



Access 2003 (& earlier) – Older Macro Designer Tool



Access 2016 – New Macro Designer Tool

I will only be explaining the current Access 2016 designer interface in this guide.

The only reason for referring to previous versions is simply due to the possibility that some Access databases may be migrated to the current version and in some cases require re-editing and tidying up.

Of course, there may be an unusual requirement to demote your database application to an earlier version too!

Pros & Cons of Access Macros

In this section I'm going to list the advantages and disadvantages of using Access macros that is an exercise developers need to carry out to determine how they will programme their application going forward and avoid the dreaded result of having a '*pear shaped*' and inflexible database.

Benefits of using Access Macros

Here's the list:

1. ***Easier to write!*** You do not have to have a university degree in computer programming to understand and utilise macros. The command reference is intuitive and easy to apply. It just requires some investment of your time.

2. ***Disabled Mode*** – From version 2007 by default, any database opened that contained VBA code would not run as part of the security changes made to Microsoft Office applications and prevent unwanted macro virus threats that the VBA code could contain some malicious routines which some programmers seem to get a kick out of! Macros used within templates in Access are safe and run in normal mode.

3. ***Access Services*** – With the introduction of SharePoint server where you can now publish your Access database on the web in a secured environment, VBA code is not a web compatible procedural language and therefore will not run. Macros on the other hand are safe and will run via a web server.

4. ***Embedded Macros*** – from version 2007, you can now attach a macro inside a form or report as part of the host object and not have a dedicated separate macro object sitting in the navigation pane (*or database window for earlier versions*). This means when you copy a control like a command button which has an embedded macro the procedure copies across too, as it is part of the properties to the control.

5. ***Command Bars/Ribbon Bars*** – Macros have the ability to be attached to customised command buttons on a toolbar/menu bar (*pre Access 2007*) and ribbon bars (*from Access 2007*) which in turn can call VBA procedures should you need to.

6. ***Reserved Macro Names*** – There are two reserved macro names that automate the start-up and keyboard shortcut controls in your application without the need to code in VBA. A polished application will have reassigned keyboard shortcuts using the '*AutoKeys*' macro.

Continues/…

7. ***Variables are not reset*** – With VBA, public variables lose their values when a procedure ends or there is an error thrown. Macros can keep the values in place when an error occurs and have an advantage when handling errors in code.

Disadvantages of using Access Macros

1. ***Performance*** – VBA code is generally faster and more efficient that macros and is more noticeable for larger or longer procedures compared to a small piece of code where it is negligible. High end applications will use structured VBA more than macros.

2. ***Flexibility & Functionality*** – Macros allow you to do many things but VBA has the power to reach beyond and communicate with other applications and use features that macros simply cannot do, like play a sound, talk to web service or handle non Microsoft software.

3. ***Managing Code Procedures*** – Macros can be split into smaller units and now in Access 2016 you have a sub-macro tool; this can still be a restriction in handling control flows and other modular based calling procedures. Using external references like ADO is simply not possible and will be a deciding factor when planning your coded procedures.

4. ***Build Custom Functions*** – In VBA, you can build additional Access functions to sit alongside the standard functions and use them in queries, forms and reports like with any other function.

Spend the time analysing which set of programming tools you are going to apply. It may even be that a combination of the two will give you a fine balance and create a good working practice for end users. It could be a two tier structure whereby macros are used for basic processing that end users could manipulate and VBA is sealed in the background and critical workflows that no user is allowed to gain access to.

Types of Macros

With the introduction of Microsoft Access 2010 and follows through to this version, we now have three different types of macros to choose:

1. **Macro Objects**
2. **Embedded Macros**
3. **Data Macros**

Macro Objects (all versions)

These are physical objects which are stored globally in the Navigation pane and are stand-alone procedures that are typically called from one or more other objects including forms and reports.

This distinction and storage of such macros make them re-usable and is a good approach to exposing public based procedures that will typically call other objects like running a handful of queries in succession, exporting recordsets to other objects or applications and handling basic application commands (*i.e. quitting the application*).

This type of macro is available to all versions of Microsoft Access and is where your 'Auto' based macros are also stored (*AutoExec* and *AutoKeys*).

Embedded Macros (Access 2007 to 2016)

With the release of Microsoft Access 2007 through the current version, you now have the opportunity (*and is now the default*) to create a procedure attached to a specific control's event. For example, the click event of a *Command Button*, the load event for a form or an update event for a *Combo Box*.

There is no physical or dedicated macro object found in the Navigation pane as it becomes a member of the host object (*i.e. a form or report*).

The scope changes from being public to private as it can only be executed within the host when loaded and running.

The added advantage here is now when you copy a *Button* from one form to another and it has an event attached to it (*i.e. On Click*), the macro is also copied as it is part of the properties of the copied control.

This scales down (*and reduces memory*) of having to store too many macro objects and simple tasks like closing a form (*explicitly by name*) is an example why you bind an embedded macro instead.

Continues/…

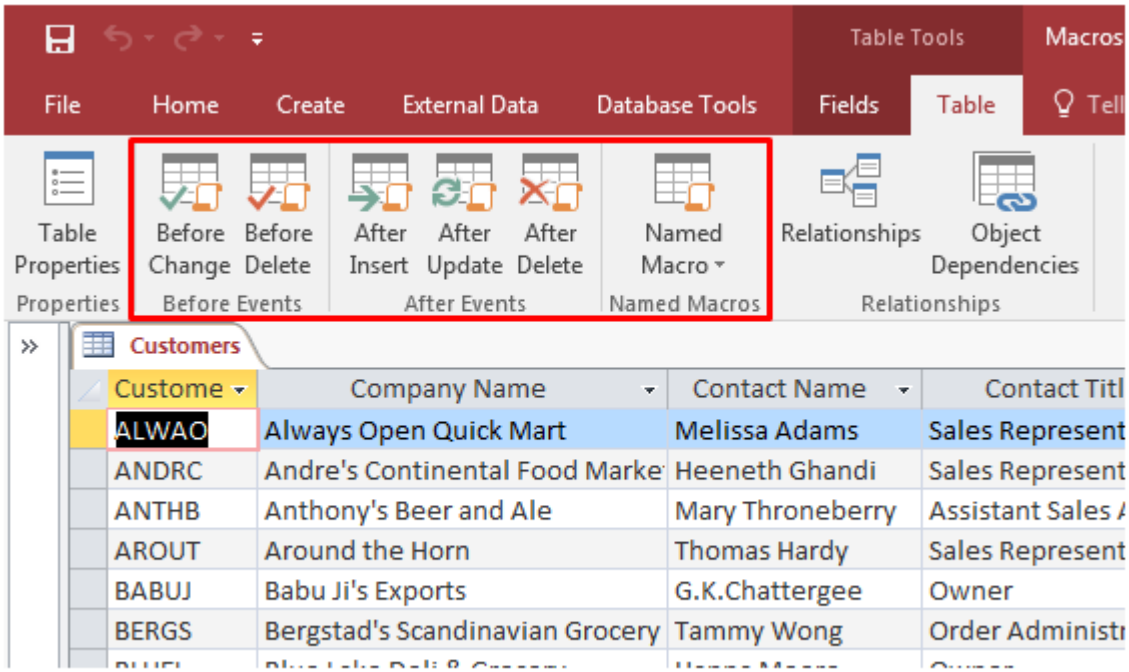
When you now create a *Button* via the wizard control, it will generate an embedded macro and not VBA code as with the case for versions pre 2007. The reason for this action is down to the 'Disable mode' feature in Access that VBA code cannot run unless it has first been trusted, whereas macros are safe to run in this mode and therefore enabled.

Data Macros (Access 2010 to 2016)

This is relatively a newer and more exclusive introduced from Access 2010 and is compared to the more powerful database application, SQL Server's Triggers.

This type of macro is attached to a table and executes when an event like **insert**, **update** and **delete** is carried out to a record change.

The added benefit to this type of macro is now if you decide to upscale your database to the web using SharePoint server (Access services), they will also run.



Access 2016 – Data Ribbon Bar Commands (Table tab)

The scope for this type of macro is deemed private as it only applies to the calling table.

Note: *You must use .ACCDB (Access database 2007 to 2016 format) and not the older .MDB (pre Access 2007) in order to utilise Data Macros.*

Introducing the Macro Designer Window

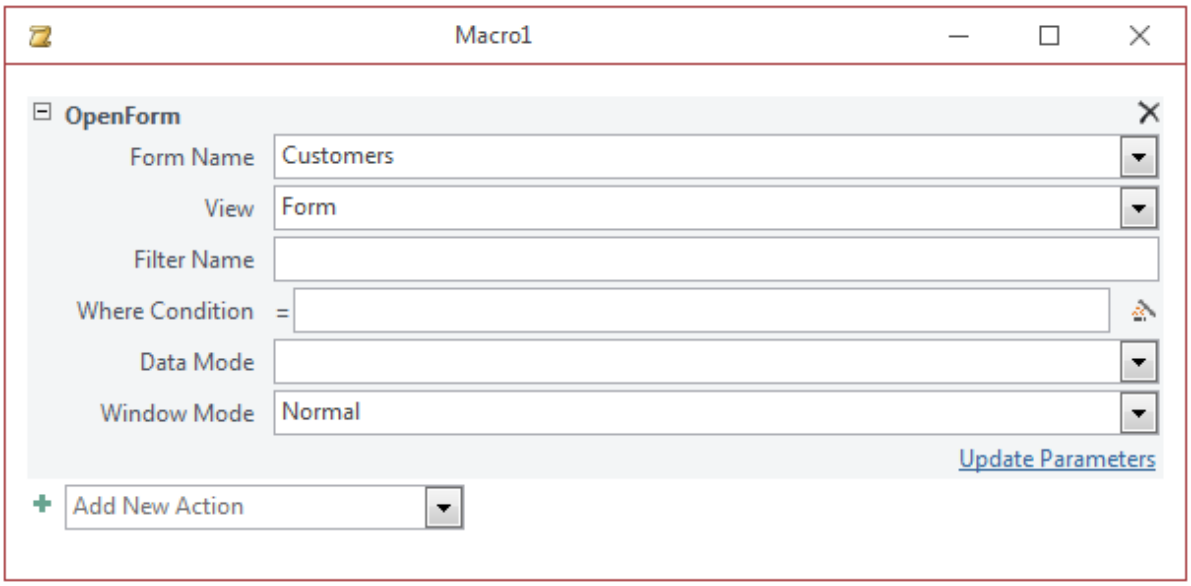
If I wanted to open a form (*called Customers*), you may have gathered by now that I could write some VBA code or create a macro instead.

VBA code would look something like:

```
Option Explicit
Option Compare Database

Private Sub CommandButton_Click()
    DoCmd.OpenForm "Customers", acNormal
End Sub
```

A macro would look something like:

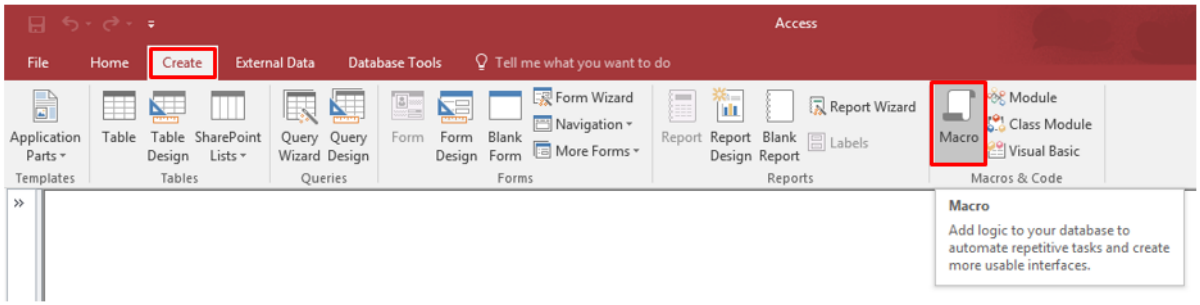


How often would you write a simple calling command to open form or report? Using a macro to hold multiple opening commands for forms (*known as sub-macros*) may be a better way to manage such procedures.

Notice in the macro illustration above that setting arguments (*i.e. View:* ***Form***) is cleaner than writing the VBA code (`, acNormal`) and is deemed more intuitive for the non-programmer!

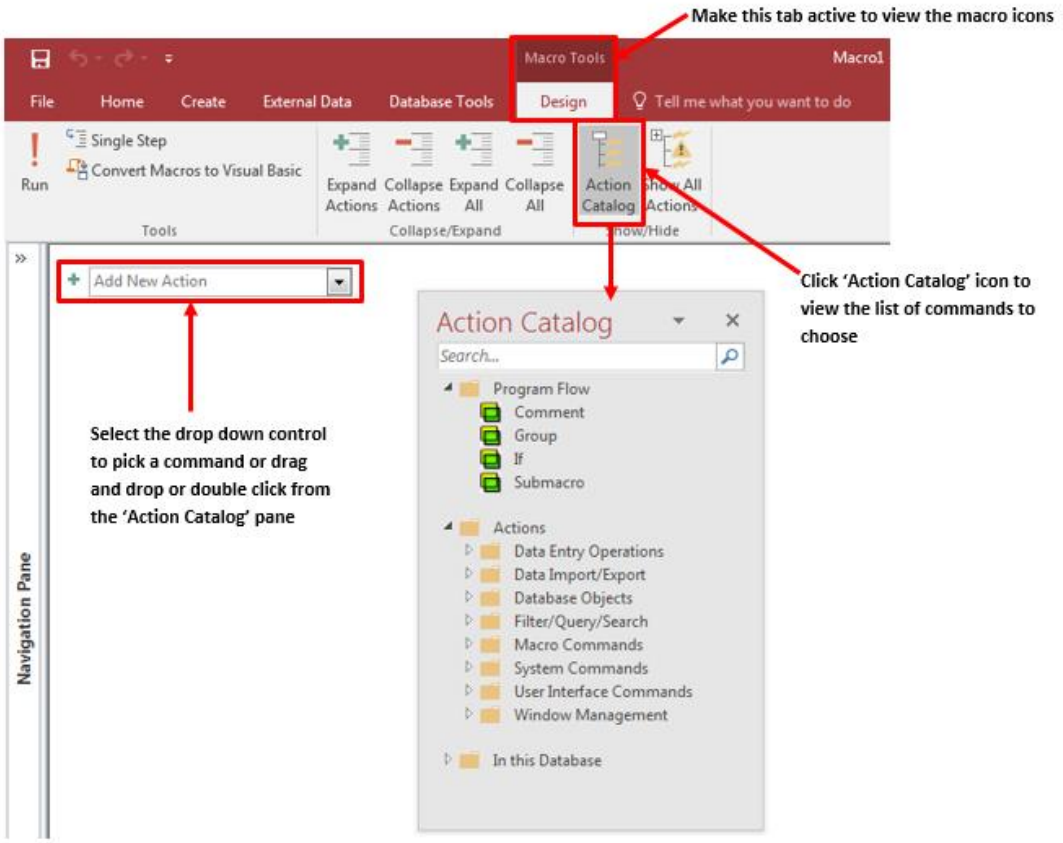
Access Macro Designer Window

Let's introduce you to the elements for this interface. First of all, start by going to the **Create** tab and under the section *Macros & Code*, choose the **Macro** icon.



Access 2016 –Ribbon Bar, Create tab, Macro icon

You are now taken into a new macro designer view and the following screen opens up:

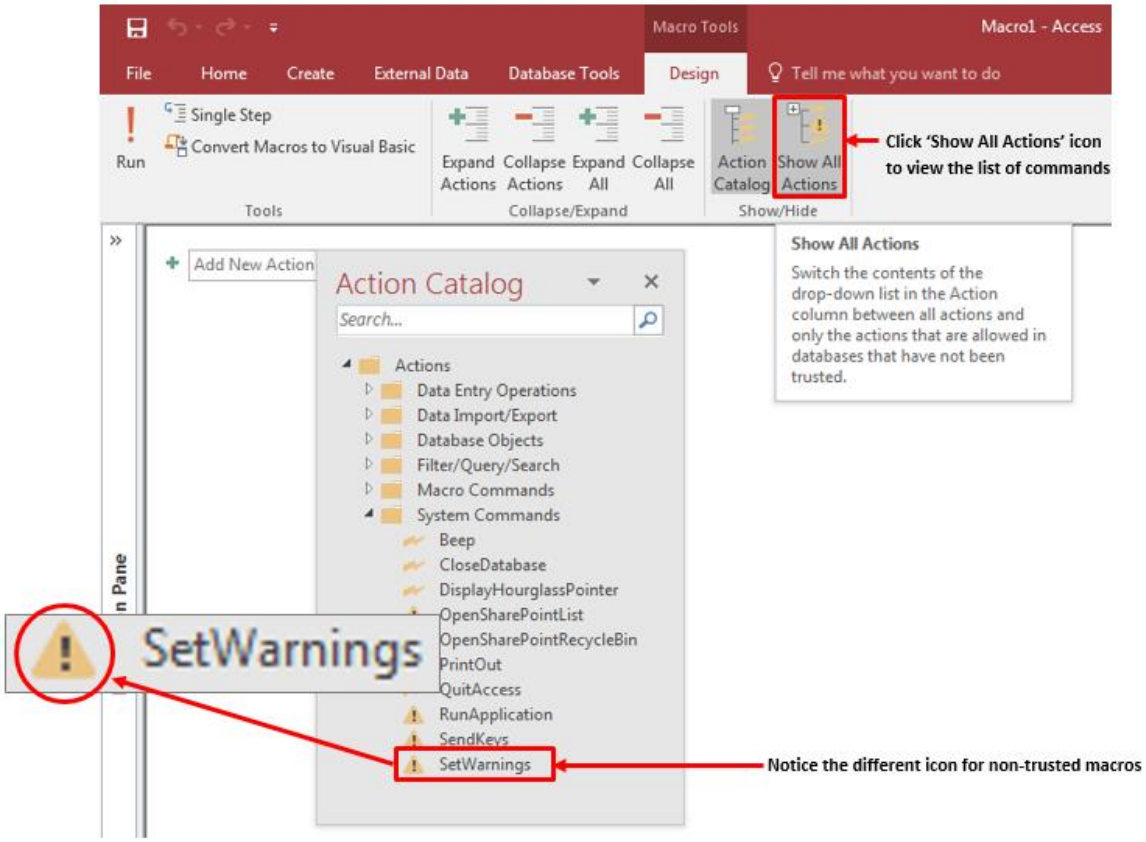


Access 2016 –New Macro Designer View

The '*Action Catalog*' pane view can be moved around the screen or docked to one side. It's not essential to have this listed and visible as you can choose from the '*Add New Action*' drop-down control instead.

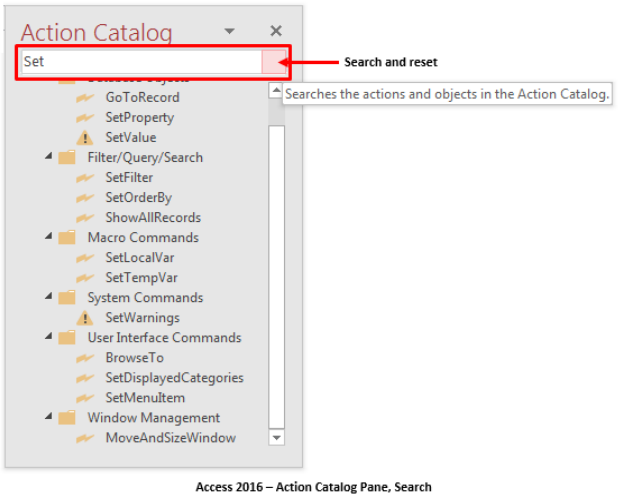
Continues/…

Notice there is another icon to the right of '*Action Catalog*' called *'***Show All Actions**' which appears to do nothing until you have changed and expanded one of the item folders in the '*Action Catalog*' pane.



Access 2016 – Action Catalog Pane

Some of the non-trusted macro commands are proprietary (*from earlier versions of Access*) and if you have been using macros for a while, you may want to view this element but be aware that macros like with VBA code will need to be trusted first to avoid having the extra step to enable your procedures when starting your database application.

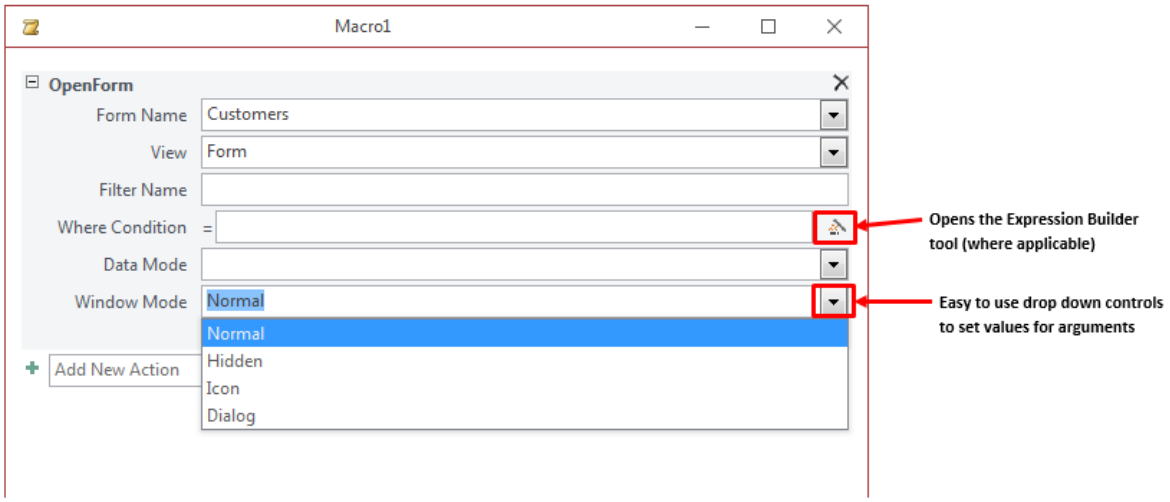


Access 2016 – Action Catalog Pane, Search

Also note that you can search for a keyword at the top of the '*Action Catalog*' pane to speed up locating new commands.

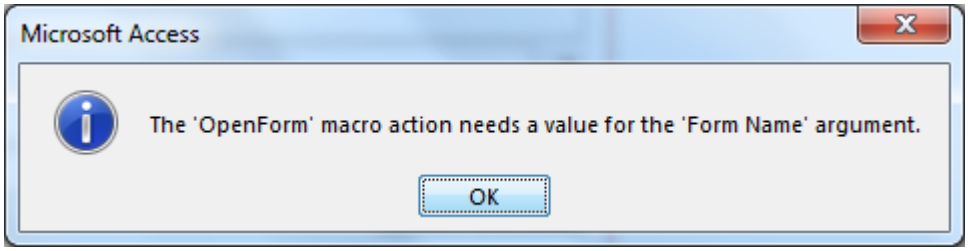
Continues/…

Choosing a command and setting arguments are intuitive and simple enough to do. Each command will have arguments that are either mandatory or optional (*some do not have any arguments i.e.* **Beep**) and can be assisted with additional tools including the *Expression Builder* that pops up to assist.



Access 2016 – Commands and Arguments

Mandatory arguments (*fields*) will be populated with a caption '*Required'* to prompt the user and they must complete a value before they can save the macro. A message prompt will remind users:



To run a macro, either close and save the macro first and attach it to a control's event or double-click the macro object from the Navigation pane (*if it's a global based macro*).

In the macro design interface, you can now use the **F5** function to run the open macro (*design view only*).

There are other features like **Groups** which allows blocks of macros to be grouped together and then expanded and collapsed to keep logical workflows tidy but are all still executed as with all macro actions.

Note: *For seasoned Access database developers who have worked with earlier versions of the macro utility may have noticed that features like Macro Name, Condition & Comment have vanished as they are not found within the macro command actions itself.*

Creating Macros

It would be impossible to give examples of all the types of processes that macros would be ideal for and later in this guide I will expand on more examples and scenarios to help exercise the logical mind and give some ideas (*and hopefully inspiration*) to apply some of the techniques explained.

For now, this section will show some simple examples highlighting the steps required to familiarise you more with the design interface.

You can use macros for a variety of tasks in Microsoft Access and the first is to introduce you to the following macros principles to give you the confidence and an overview and are as follows:

1. Opening objects
2. Printing

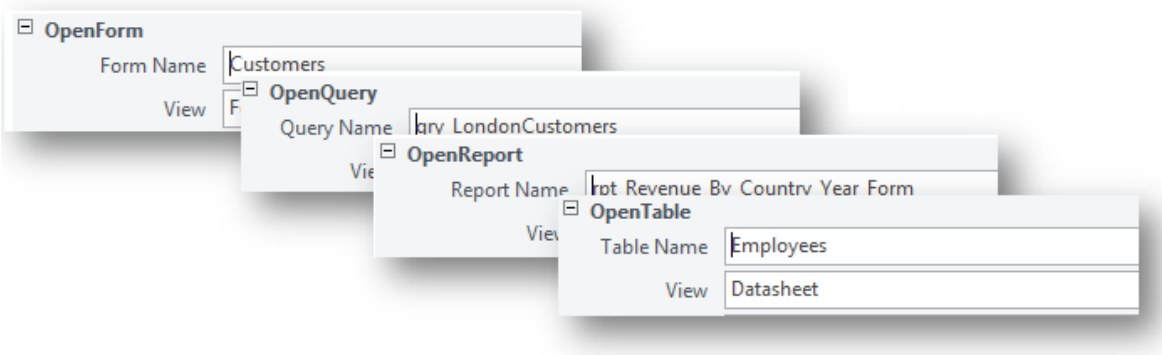
Opening objects

In Access 2016, there are four 'safe' '*Open'* objects to choose from which are already enabled (*from 'Action Catalog'*):

1. **OpenForm**
2. **OpenQuery**
3. **OpenReport**
4. **OpenTable**

They all perform the same action and open the object type by a chosen name. There are arguments that can be added to manage the options but the mandatory argument is to choose at least a name.

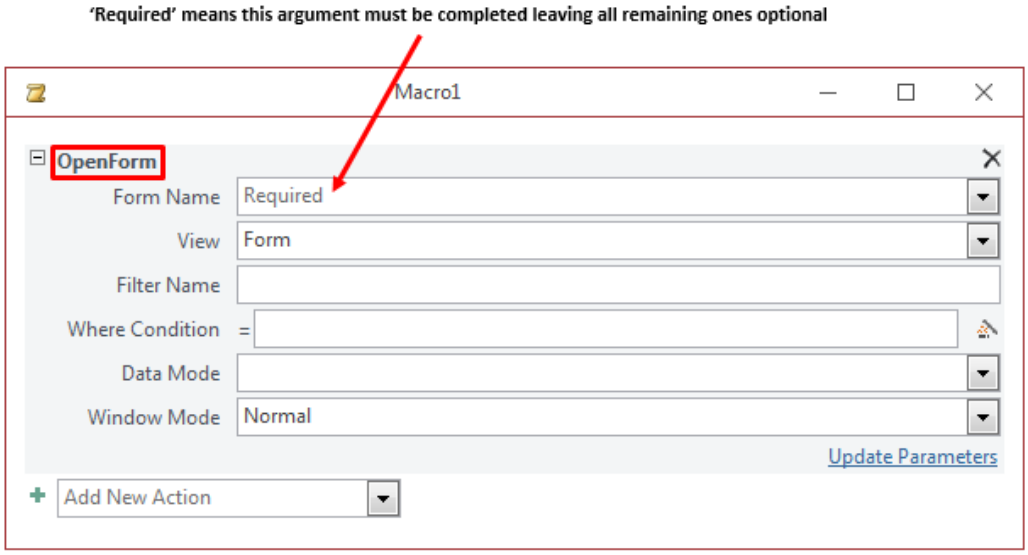
> ***Note:*** *There are three other 'Open' based commands deemed from the unsafe list and include OpenSharePointList, OpenSharePointRecycleBin and OpenVisualBasicModule.*



Continues/…

1. Start a new macro object (*to store in the Navigation pane*) making sure you see a new canvas and the **Action Catalog** pane (*usually loaded to the right of the screen*).

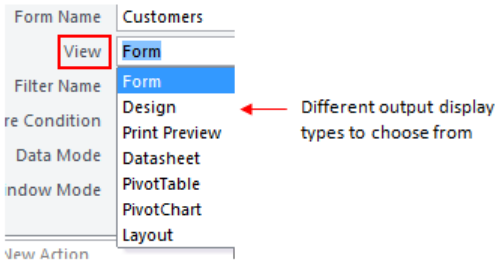
2. From the '*Add New Action*' choose the action **OpenForm** command to reveal the following screen:



Access 2016 – New Macro Design, 'Open Form' Action selected

3. Choose the form name required (*my example is* '*Customers*').

4. Optionally set arguments required which includes how to view the form:



5. Click the save button and give it a name (call it '***mcr_OpenCustomerForm***') and close macro.

In the Navigation pane, double-click your new macro! The form loads.

> ***Note:*** *You do not have to close the macro to run it. There is an icon (red exclamation) that will execute the macro or use the F5 function key but the macro must first be saved.*

Change the other '*View*' arguments and see the impact, remembering to save changes each time.

Continues/…

There are other arguments to the **OpenForm** action which are as follows:

Filter Name – To restrict recordsets like a query and can be either a query object or a saved filter.

Where Condition – The more powerful option to store SQL statements and can be used with **Filter Name** argument as the *WHERE* clause (which is why you do not use the word *WHERE* in this SQL field).

Data Mode – Like with a bound form, you can set its editing mode to either 'Add', 'Edit' or 'Read Only'.

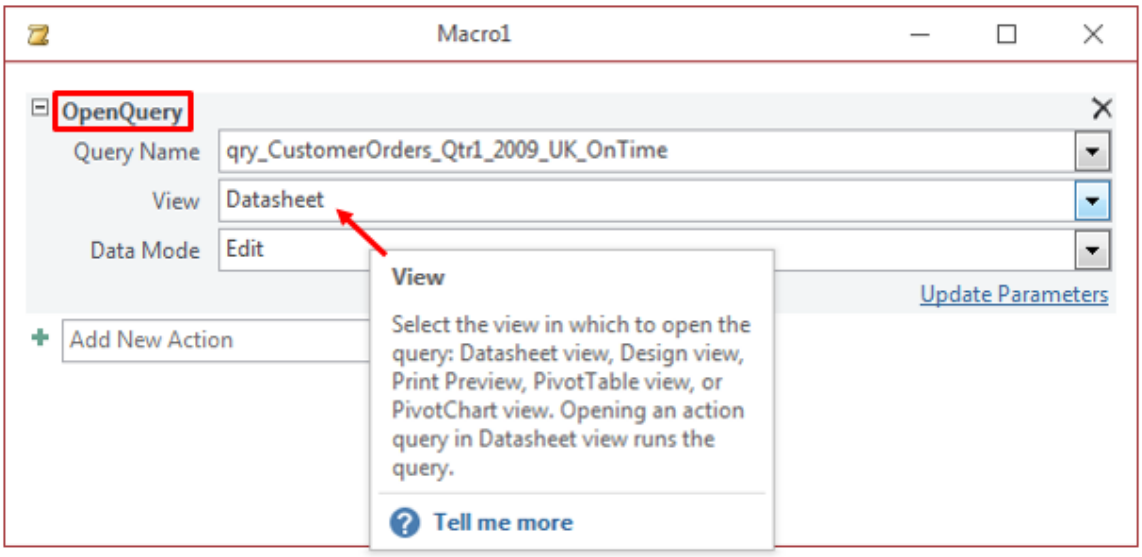
Window Mode – Decide if you want to view the form or hide it but keep it open or even make it a '*Dialog*'.

Note: *To see more on the above action and the arguments, see* http://msdn.microsoft.com/en-us/library/ff823095.aspx.

Each of the ***Open…*** action commands will vary but essentially they provide same basic functionality to when that object is being called.

The **OpenReport** action is identical to the example given above.

An example for an **OpenQuery** action (*which the* **OpenTable** *is identical to this*) is as follows:



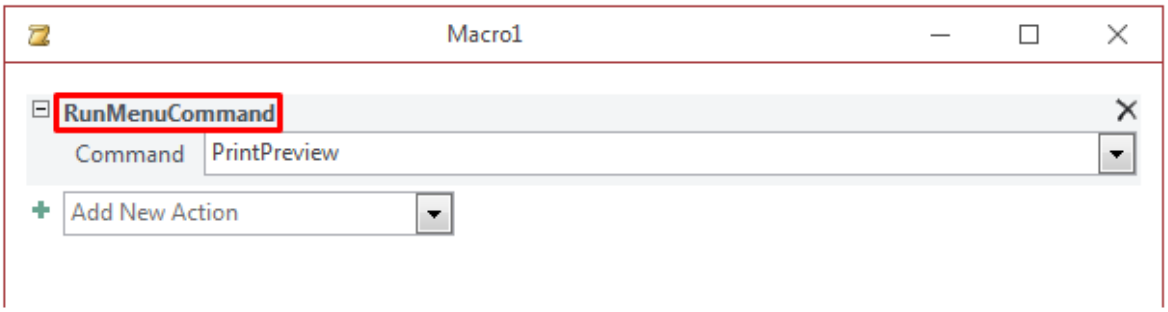
Access 2016 – New Macro Design, 'Open Query' Action selected

This query will open in normal datasheet view (*default*) and allow records to be edited. Other view options include *Print Preview*, *PivotTable*, *PivotChart* and even loading the *Design view* too.

Printing

With the introduction of Access 2010 you now have the **RunMenuCommand** which simulates a menu command available to the correct loaded object at the time and run with the command when called.

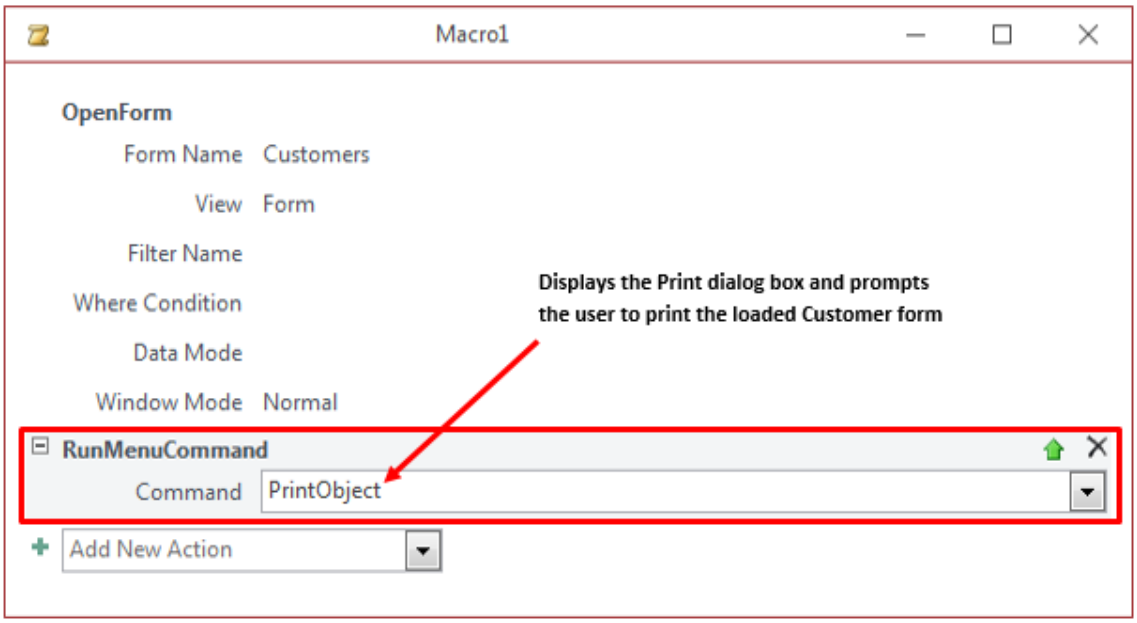
In this case, to print an object which is already loaded (*the current view*), you can use the **PrintPreview** action which is listed as *PrintPreview* but converts the display to the following:



Access 2016 –Macro Design, 'Print Preview' Action selected

There is no real added value to use the one command as shown as with this example it would run the macro print preview (*being the window open*).

Instead, it is added as an additional command to previous macro commands and the **PrintObject** action can be used to print the selected object loaded as shown below:



Access 2016 –Macro Design, 'Print Object' Action selected

Continues/…

The Print dialog will be shown and requires the user to send document to the printer in the normal manner.

Notice there are no arguments for this action and to delete a command, simply click the '**X**' icon located to the right side of the action command.

Additionally, there is a green arrow in this case pointing up to allow you to re-arrange the order of macro commands to be executed, as macros execute one line at a time in the order they appear.

Macro Actions & Arguments

In Access 2016 there are 87 macro action commands including all the unsafe commands too. In most cases, each action will have mandatory and optional arguments (*parameter values*) to increase the permutation of each command action making this a flexible programming tool.

The following table shows some of the more popular command actions and differences between version 2016 and 2007 (*which also applies to the earlier versions too*):

Action Name 2016	Action Name 2007	Brief Description	New?	Web?
ApplyFilter	ApplyFilter	Applies a filter to the currently open object.	No	No
BrowseTo	N/A	Navigates to the specified form or report.	Yes	Yes
CloseDatabase	CloseDatabase	Closes the database	No	No
CloseWindow	Close	Closes the specified object window.	No	Yes
DeleteRecord	RunCommand DeleteRecord	Deletes the current record.	No	Yes
DisplayHourglassPointer	Hourglass	Sets or clears the hourglass cursor.	No	No
EMailDatabaseObject	SendObject	Sends an object in the database as an attachment in an e-mail.	No	No
ExportWithFormatting	OutpuTo	Exports an object in the database in the specified format.	No	No
GoToControl	GoToControl	Moves focus to the specified control.	No	Yes
GoToRecord	GoToRecord	Moves to the specified record as an offset of the current record.	No	Yes
ImportExportData	TransferDatabase	Imports or exports data or objects from another Access database.	No	No
ImportExportSpreadsheet	TransferSpreadsheet	Imports or exports data or objects from Excel.	No	No
ImportExportText	TransferText	Imports or exports data from text files.	No	No
MessageBox	MsgBox	Displays a message to users.	No	Yes
OnError	N/A	Defines behaviour for error handling.	No	Yes
OpenForm	OpenForm	Opens the specified form.	No	Yes
OpenReport	OpenReport	Opens the specified report.	No	Yes
OpenQuery	OpenQuery	Opens the specified query.	No	Yes
PrintPreview	RunCommand PrintPreview	Opens Print Preview for the current object.	No	No
QuitAccess	Quit	Exits Microsoft Access.	No	No
RemoveAllTempVars	RemoveAllTempVars	Clears the `TempVars` collection.	No	Yes
RemoveTempVar	RemoveTempVar	Removes the specified `TempVar` from the `TempVars` collection.	No	Yes
Requery	Requery	Re-executes the query for the current object.	No	Yes
RunCode	RunCode	Runs the specified VBA user-defined function.	No	No
RunDataMacro	N/A	Runs the specified data macro.	Yes	Yes
RunMacro	RunMacro	Runs the specified macro.	No	Yes
RunMenuCommand	RunCommand	Runs the specified command.	No	Yes
RunSavedImportExport	RunSavedImportExport	Runs the specified import or export specification.	No	No
SaveRecord	RunCommand SaveRecord	Saves the current record.	No	Yes

Continues/…

Action Name 2016	Action Name 2007	Brief Description	New?	Web?
SearchForRecord	SearchForRecord	Searches for the specified record	No	No

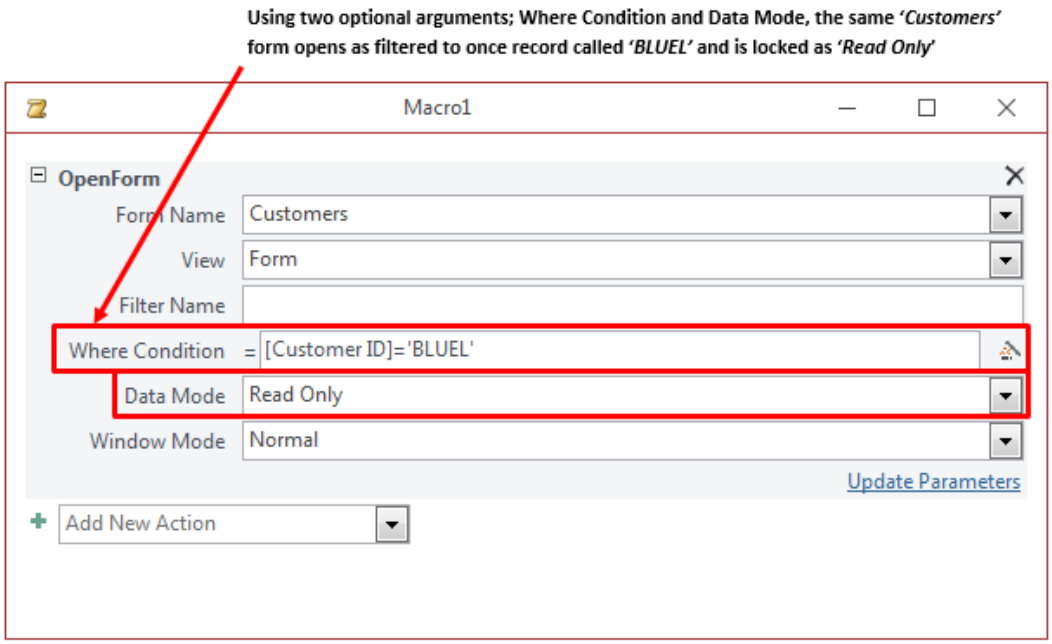
		using a SQL WHERE clause.		
SetFilter	N/A	Applies a filter to the specified control or current object.	Yes	Yes
SetLocalVar	N/A	Creates or sets a local variable.	Yes	Yes
SetOrderBy	N/A	Applies a sort to the specified control or current object.	Yes	Yes
SetProperty	SetProperty	Sets a property value for the specified control.	No	Yes
SetTempVar	SetTempVar	Creates or sets a `TempVar`.	No	Yes
StopAllMacros	StopAllMacros	Stops all currently active macros.	No	Yes
StopMacro	StopMacro	Stops the current macro from running.	No	Yes

Some of the above macros actions listed that are available for the web (*Access services*) will have some of the arguments available where applicable, as the options listed in arguments are specific to Access only. For example, the **`OpenForm`** command **`View`** and **`FilterName`** arguments would not be applicable to the web.

Note: *Not all Actions will be available to earlier version of Microsoft Access and users should refer to the online help for clarification and confirm if there is an alternative keyword command.*

Let's take a close look at how some of the arguments can be set and I'm going to continue to work on the **`OpenForm`** action which demonstrates various techniques when setting values.

Take a look at the following screen for this action (*and arguments*):



Access 2016 –Macro Design, Open Form example

Continues/…

The **Where Condition** is a useful argument as it can reduce the overall number of separate query objects and various saved filter options making use (& re-use) of the same Customers

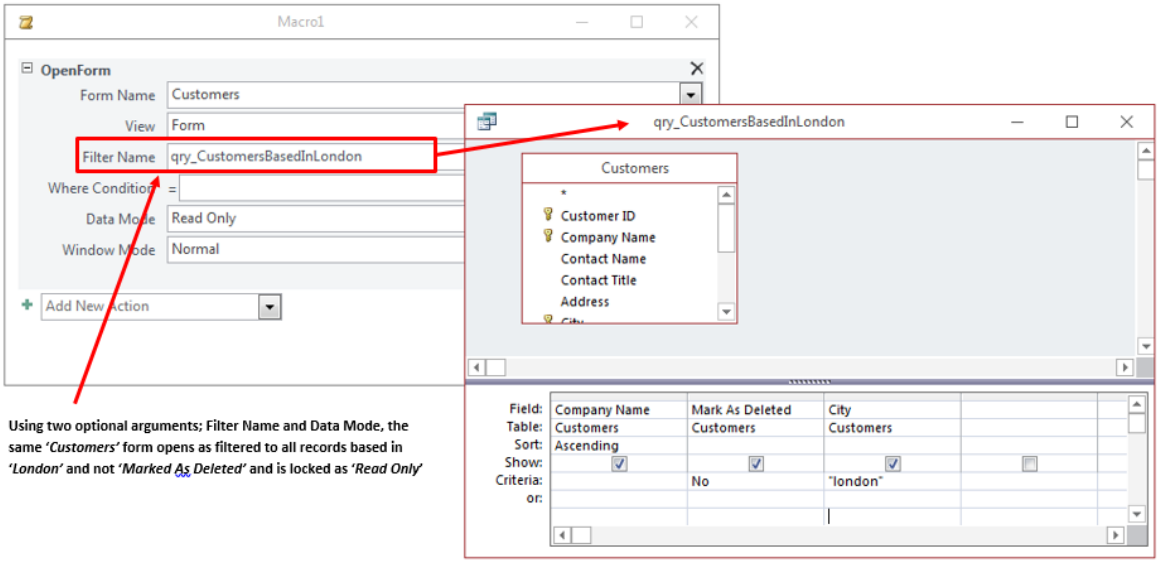
form. Notice it refers to a control; in this case **[Customer ID]** which is a text value looking for the unique value '*BLUEL'.*

Additionally, I also requested this form to load and lock all the data so it cannot be edited using the **Data Mode** argument.

Arguments can be chosen from a drop-down box if available, via the build button if available, or simply typed into the argument box.

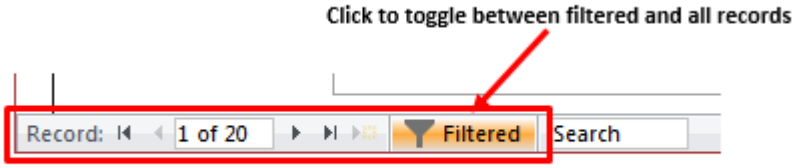
Using the **Filter Name** argument is also available to apply a filter to the form which is the same as applying a filter to the table (*datasheet view*). It can be a saved query which must contain the fields in the form/table for the filter to apply correctly.

See the following example screen below:



Access 2016 –Macro Design, Open Form example

A dedicated query called '*qry_CustomersBasedInLondon'* contains all the fields for criteria (*as shown above*). By typing the same name query into the **Filter Name** argument, the customers form opens up with a filter applied to it showing 20 records.



Access 2016 – 'Customers' f rm open

You can still toggle between all records and the currently applied filter but in my example this is still locked (*read only*).

Some arguments are more complex than I've given above but with help files (*and other online resources*), you will quickly pick up the benefits of using arguments.

Note: *When setting text criteria, you can use either conventions of the double-quotes or single-quotes. Also, criteria and values in arguments are not case sensitive.*

Sub-Macros (Named Macros)

A nice tidy feature introduced with Access 2010 is the ability to divide macros into smaller and more manageable units keeping the number of macro objects to a minimum.

These were previously known as *Macro Names* (*Microsoft Access 2007 or earlier*) and were called by the hierarchy of the main macro name followed by the sub-name (**`MacroName.SubMacroName`**)

There are useful for creating related functionality all stored in a single macro object. Some examples might include:

- **Opening and outputting forms and reports**
- **Running queries grouped to similar processes**
- **Exporting and importing data between other applications**
- **Keeping general functions (*VBA driven*) together**

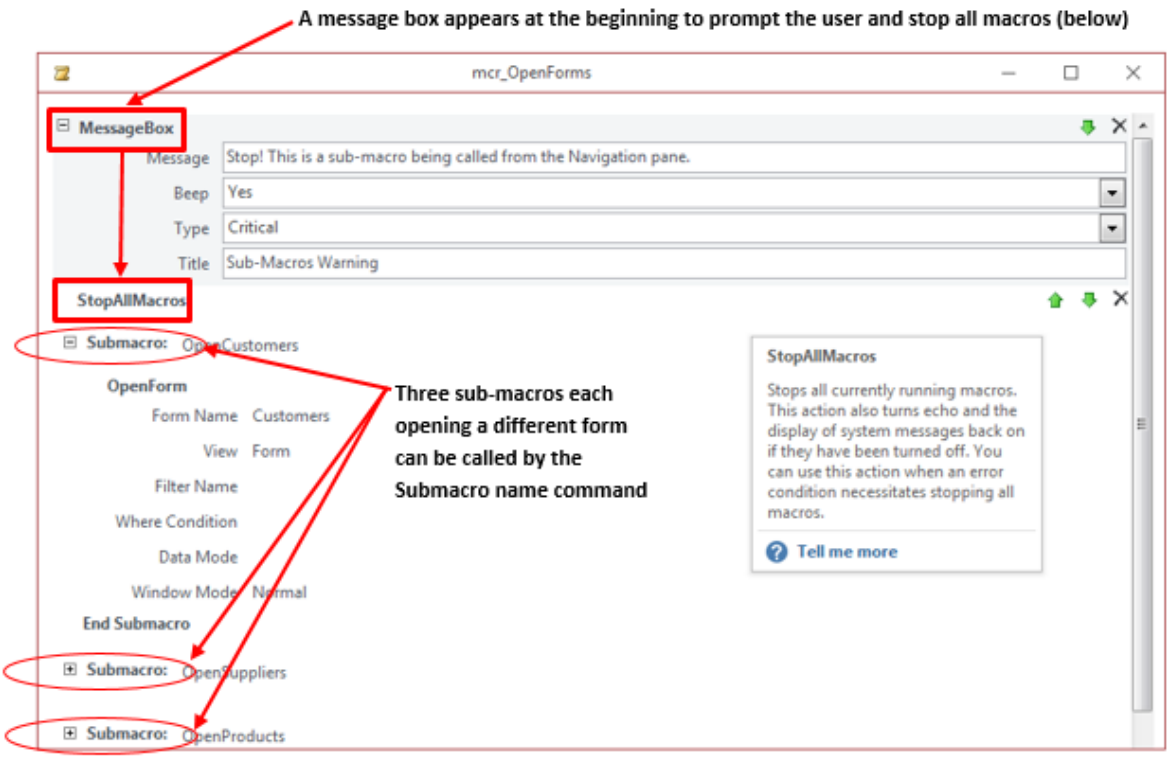
Before giving you an example of a sub-macro use, there some remarks to be noted about this type of macro. First of all, you cannot call a sub-macro from an embedded macro though there is a work-around when creating this via an embedded macro (*using the* **`OnError`** *macro*).

Any macro objects that are called directly from the Navigation pane that contains sub-macros will only run the first sub-macro listed. Therefore, these types of macros are going to be called from different places within your application and not directly from the Navigation pane.

To help prevent a macro being called directly via the Navigation pane for this type of macro (*sub-macros*), we could add a **`MessageBox`** command at the beginning (not as a sub-macro) and stop the macro continuing with the **`StopAllMacros`** command.

Continues/…

See illustration below:

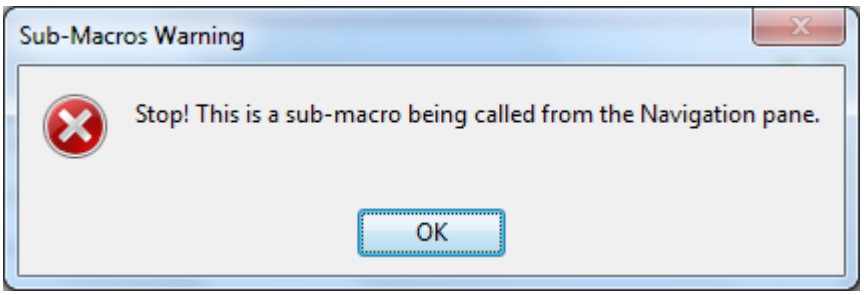


Access 2016 – Macro Design, Sub-Macros example

In the above illustration, there are three sub-macros; *OpenCustomers*, *OpenSuppliers* and *OpenProducts* which each contain the same **`OpenForm`** command.

The first sub-macro is expanded to show the **`OpenForm`** command and its arguments, with the other two collapsed but having the same configuration other than the name of the calling defined forms.

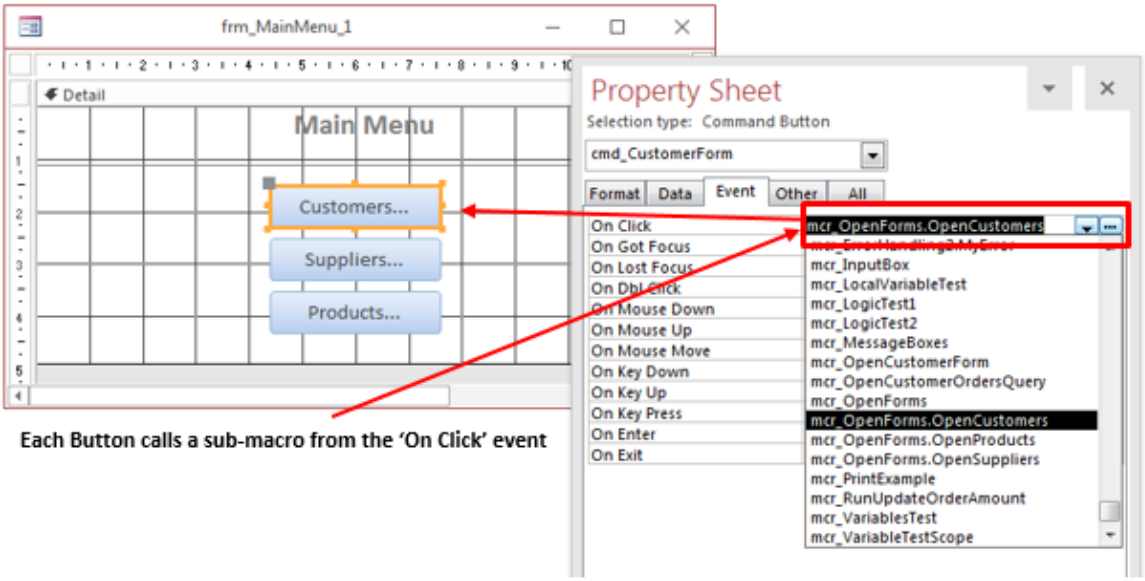
If I then call this macro which was named *mcr_OpenForms* from the Navigation pane, it will display the following message box:



This adds a little more protection (*an additional layer*) to prevent any further macros being executed within this object should users gain access to the Navigation pane and attempt to execute this macro.

The next step is to call a sub-macro and typically you might have a dedicated form (*customised menu screen*) with **Buttons** calling a sub-macro.

In the example below I have three **Buttons** (*one for each calling sub-macro*) to display a form:



Access 2016 – Form Design, calling Sub-Macros example

Notice the hierarchy for a sub-macro call. It starts with the main macro name followed by the sub-macro name; **`mcr_OpenForms.OpenCustomers`**.

Control Flows and Interaction

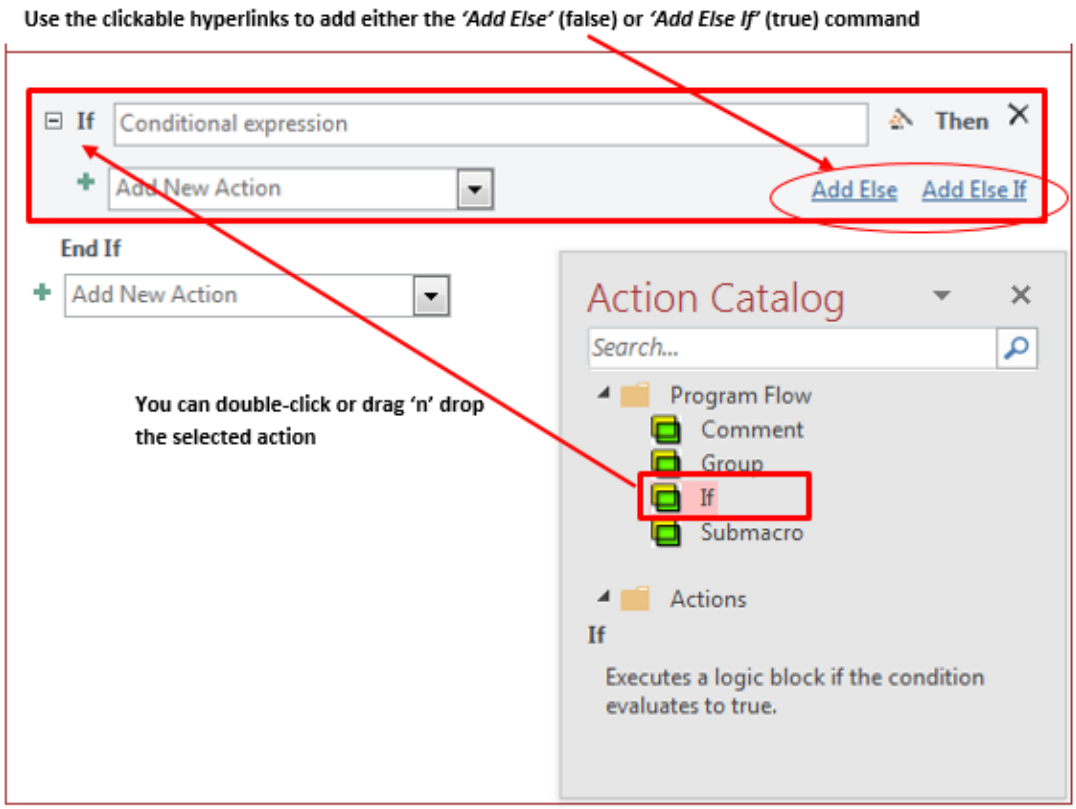
A macro simply executes one action (*command*) at a time in the order it has been listed in the macro designer tool. To allow a macro to *think for itself*, you need to add some logic or control the flow of execution.

Another term used to handle control flows is a ***macro construct*** and has been made easier with Access 2010. Previous versions of Microsoft Access used the '*Condition*' column combined with ellipses (…) to logically flow '*true*' expressions (*see an example later in this section*).

A Logical condition

Setting a simple logical test requires adding a single **If** action block (*for a true response*).

In a new macro, add a program flow command, **If**:



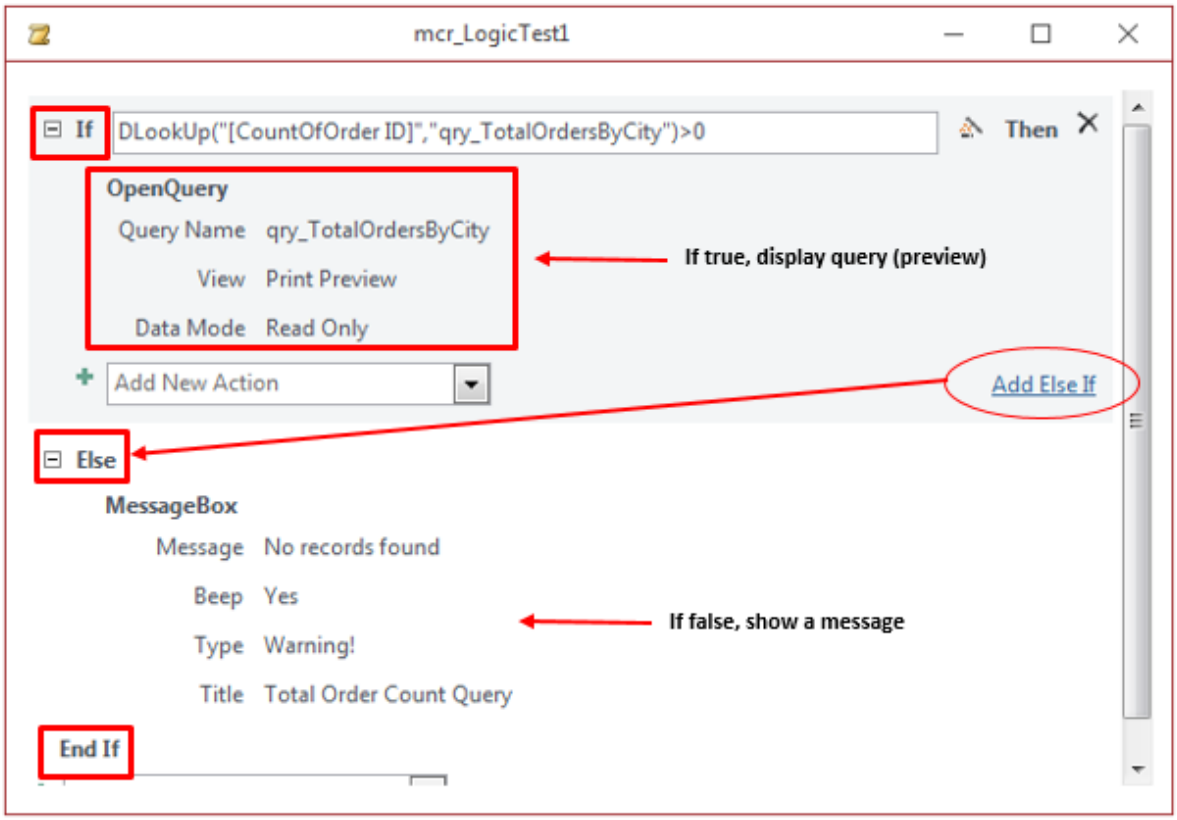
Access 2016 – Macro Designer, If Action

Next, you add one or more actions that will be executed if true.

Notice the '**End If**' block terminating the control flow but not the macro itself unless you call the **`StopMacro`** action command.

Continues/…

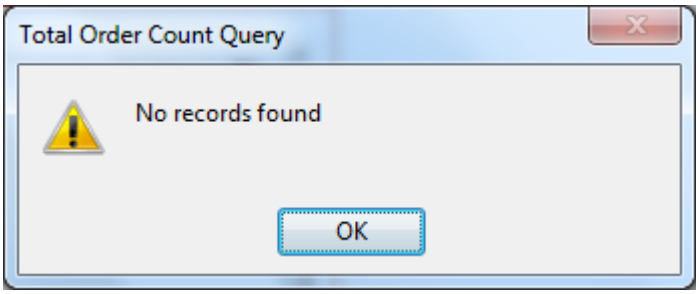
The following example displays a query in print preview mode providing there are records (*summary value of orders by city*) or displays a **MessageBox** if there are no records.



Access 2016 – Macro Designer, If, Else If Example

In the condition of the If test, I'm using a **DLookUp** function to return a value if one of the fields (*CountOfOrderID*) is greater than zero. If true, then open the query in print preview mode.

If false (i.e. no records found) then display the following message box:



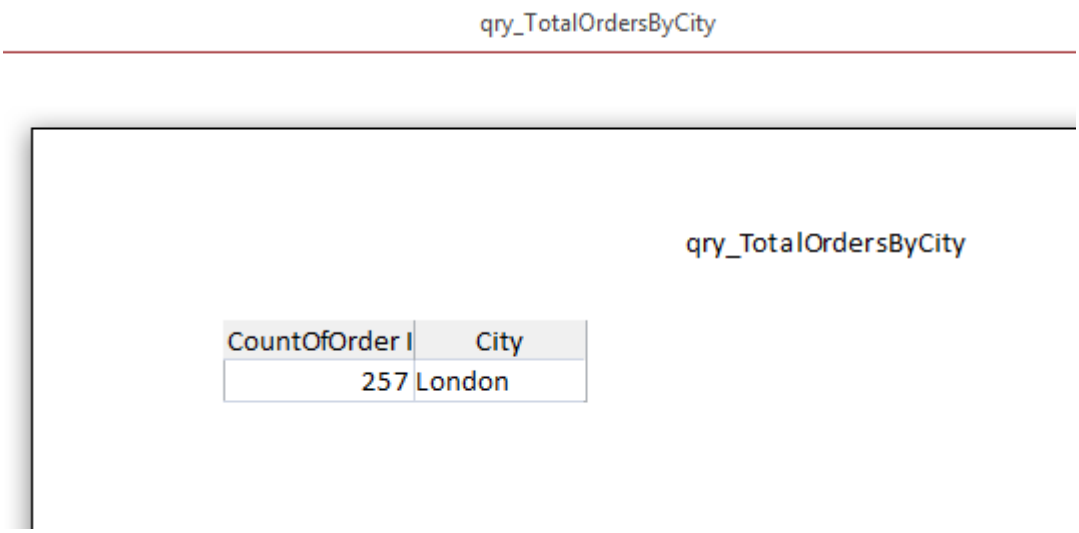
The query displays a record total number of orders by criteria for the city field (for the UK).

Continues/…

SQL (for my query example):

```
SELECT Count(Orders.[Order ID]) AS [CountOfOrder ID], Customers.City
FROM Orders INNER JOIN Customers ON Orders.[Customer ID] = Customers.[Customer ID]
WHERE (((Customers.Country)="uk"))
GROUP BY Customers.City
HAVING (((Customers.City)="london"));
```

By changing the city criteria from '***London'*** to an unknown city '***Exeter'*** will logically test the macro example.



Multiple conditions

You can use more than one **If** test in one of two ways:

1. *Nested* – **If** inside an **If**.
2. *ElseIf* – **If** block with two or more conditions using **Else If**.

It's the same action command as previously shown for a logical condition and an example of an **Else If** is shown below:

Continues/…

This scenario is going to take an empty table (*acting as the template*) which has the structure for claiming expenses for each working day. The macro will copy a new table from this template and name with the day of the week (i.e. *tbl_Monday*).

First, create a new table as follows:

Field Name	Data Type	Field Size & Properties
ExpID	AutoNumber	Long Integer, Primary Key
ExpenseDate	Date/Time	-
Description	Short Text	Size 255
Amount	Currency	-
Authorised	Yes/No	-
PaidDate	Date/Time	-

Save the table as **tbl_ExpensesTemplate** and close.

The above table will not have data (*records*) as it will act as the template for new tables that the macro will copy (**`CopyObject`** *action*).

In fact, you can create any table, fields and data types for this example as your template.

I thought you might want to have some inspiration here!

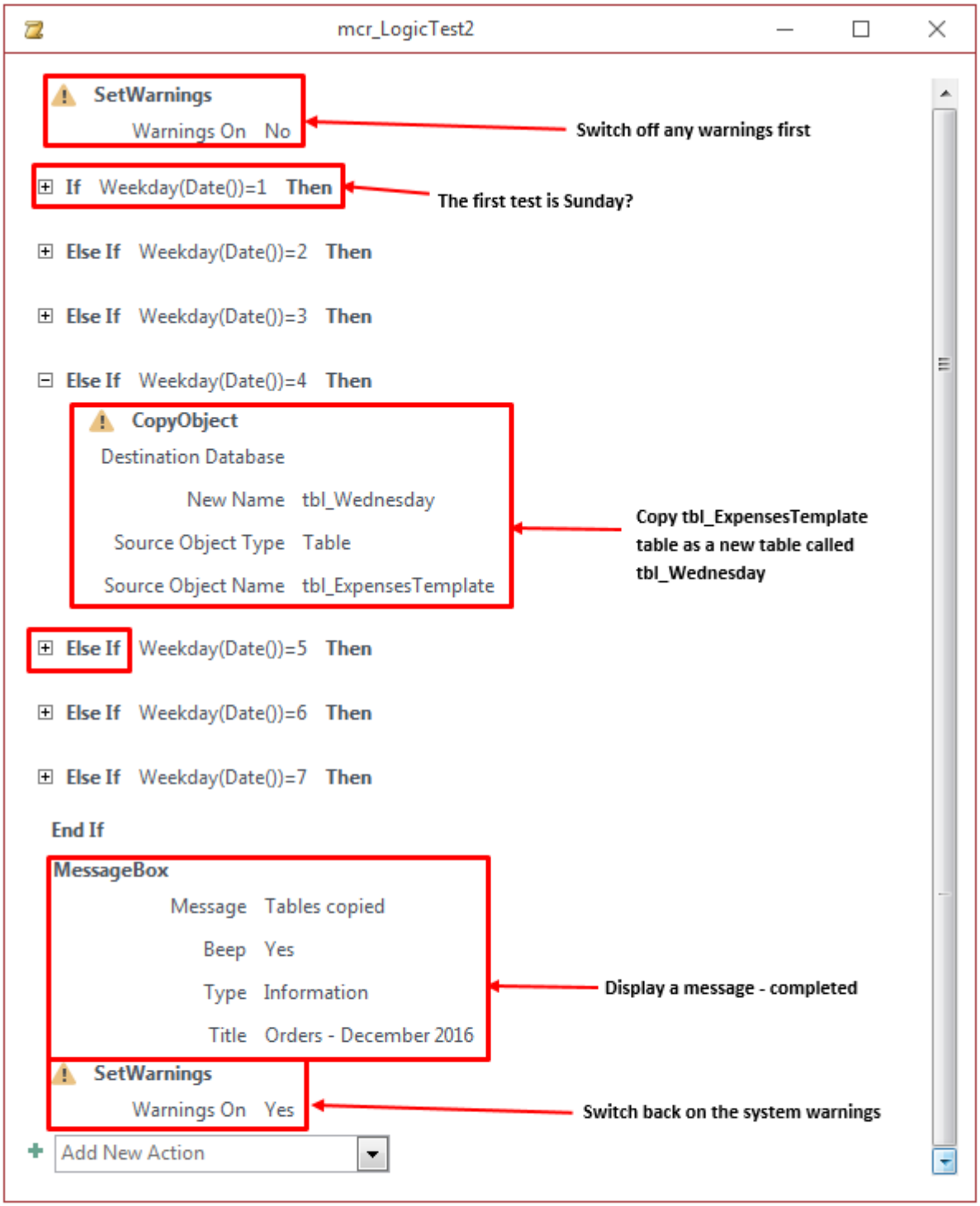
Next step is to focus on creating a new macro where we are going to logically test which day of the week it is and create a new table.

The following macro actions are required:

- **`SetWarnings`** – which switches on and off a warning when over writing a table (*or any object*) that may already exist.
- **`If`**…**`Else If`**…**`End IF`** – to handle the logical control workflows (*for which day it is*).
- **`CopyObject`** – defines the new object name (*in this case a table*) from the original table.
- **`MessageBox`** – to prompt the user that the macro has completed its procedure.

Continues/…

Take a look at the screen below:



Access 2016 – Macro Designer, If, Else If Example

In the above example, I'm using seven logical tests (*one for each day of the week*) using the Weekday function (*an Access function which returns a value between 1 and 7 starting with Sunday*).

Continues/…

Therefore, the following condition using two Access functions (nested) returns 4 which means Wednesday:

```
Weekday(Date())=4
```

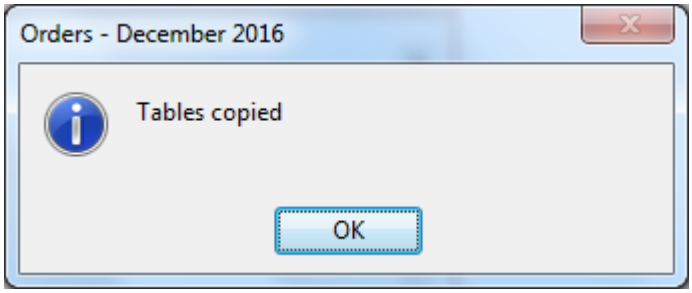
It logically calls the **CopyObject** action for the current day and copies a new table as defined in the ***New Name*** argument for this action.

Also notice that I use the **SetWarnings** action at the beginning and re-used it again at the very end of the macro to temporarily disable the warning you see when deleting or overwriting an existing object (*in this case a table*).

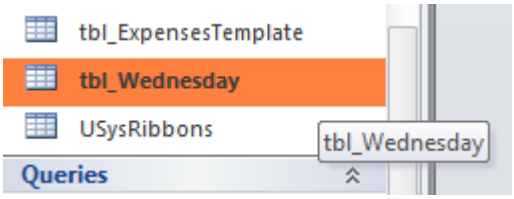
To add as many **if** tests as you like, simply click the blue hyperlink '**Add Else If**' and choose at least one action after the logical test.

Save the macro and run to test it out.

If today was Wednesday, then a message box appears:



The following table has been created in the Navigation pane:



More practical uses for this type of procedure may be found as an embedded macro (*covered later in this guide*) which logically tests controls in a form.

Message box

As you have seen several times in this guide the **`Message Box`** action is a way to communicate to the user (interact). Previous versions of Microsoft Access use **`MsgBox`** instead and have the same argument settings.

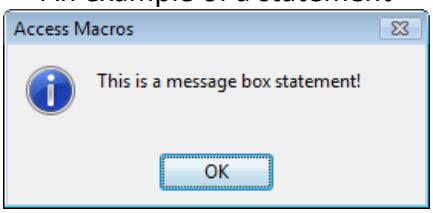
There are two types of use for this action:

1. Message box statement
2. Message box (MsgBox) function

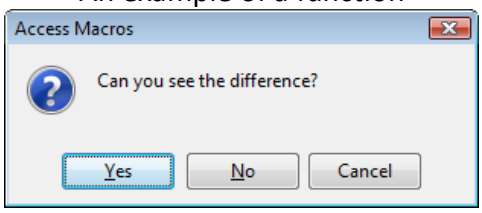
What's the difference? A statement is a *one-way* communication message which tells you (*the user*) when you want to communicate a message and that's **the end of this action. Whereas the function is a *two-way* communication message which waits for users** to respond to the prompt, returning a value and is used to logically test the value returned.

A function is the same as statement other than it answers you back - what an attitude!

An example of a statement



An example of a function



The Message Box function sits inside the condition of the **`If`** action and uses the older **MsgBox** (*Access*) function as follows:

```
MsgBox("Can you see the difference?",35,"Access Macros")
```

The syntax (structure) for the MsgBox function is as follows:

```
MsgBox(Prompt [, Buttons] [, Title])
```

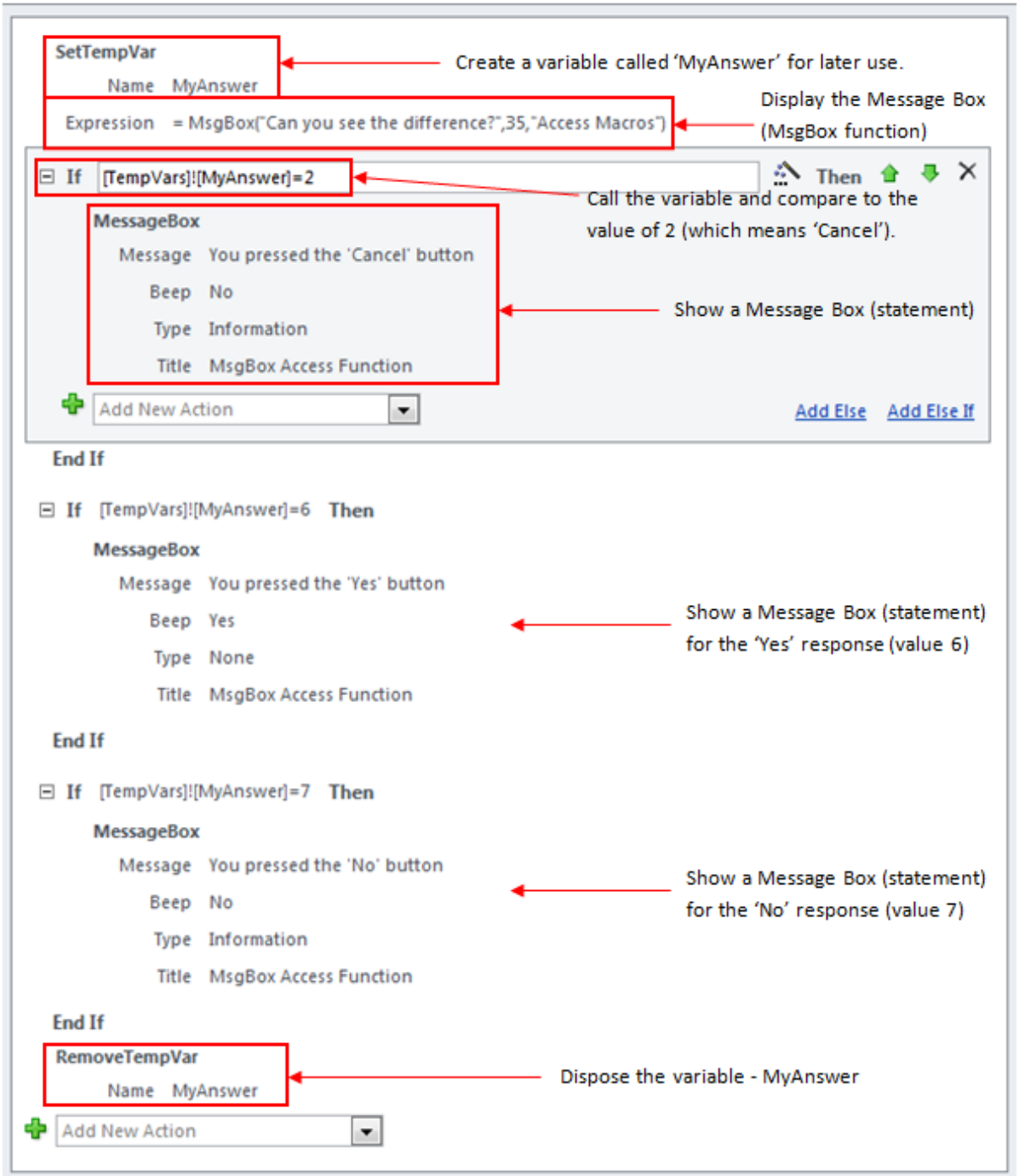
The first argument is the **`Prompt`** which is simply text in quotations of the message to display.

The second argument (*optional*) is a value which represents a permutation of what **`buttons`** and icon to display. In the case, 35 means show the 'Yes', 'No', 'Cancel' buttons and display the 'Question Mark' icon too.

The third argument (*optional*) is the **`Title`** that appears in the title bar of the message box and is enclosed with double-quotation marks.

> ***Note:*** *For further help on the MsgBox function, check out this link* http://office.microsoft.com/en-us/access-help/msgbox-function-HA001228886.aspx

Here's a simple use for the message box function and a message box statement to illustrate the difference in how you apply them.



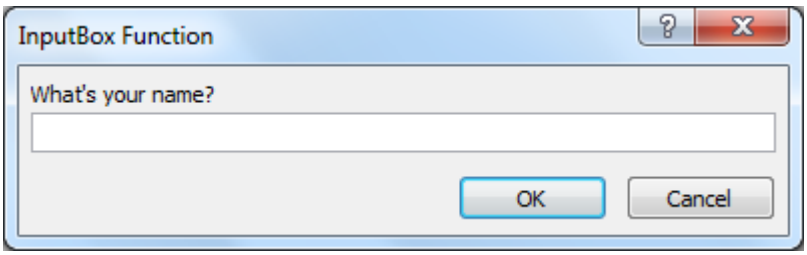
Access 2016 – Macro Designer, Message Box examples

In the above example, I have used a variable (*covered later in this guide*) to set the response of a button clicked from the **`MsgBox`** function which is then tested in separate If actions which in turn are using the **`Message Box`** (*statements*). Finally, I dispose of the variable as it has no other use.

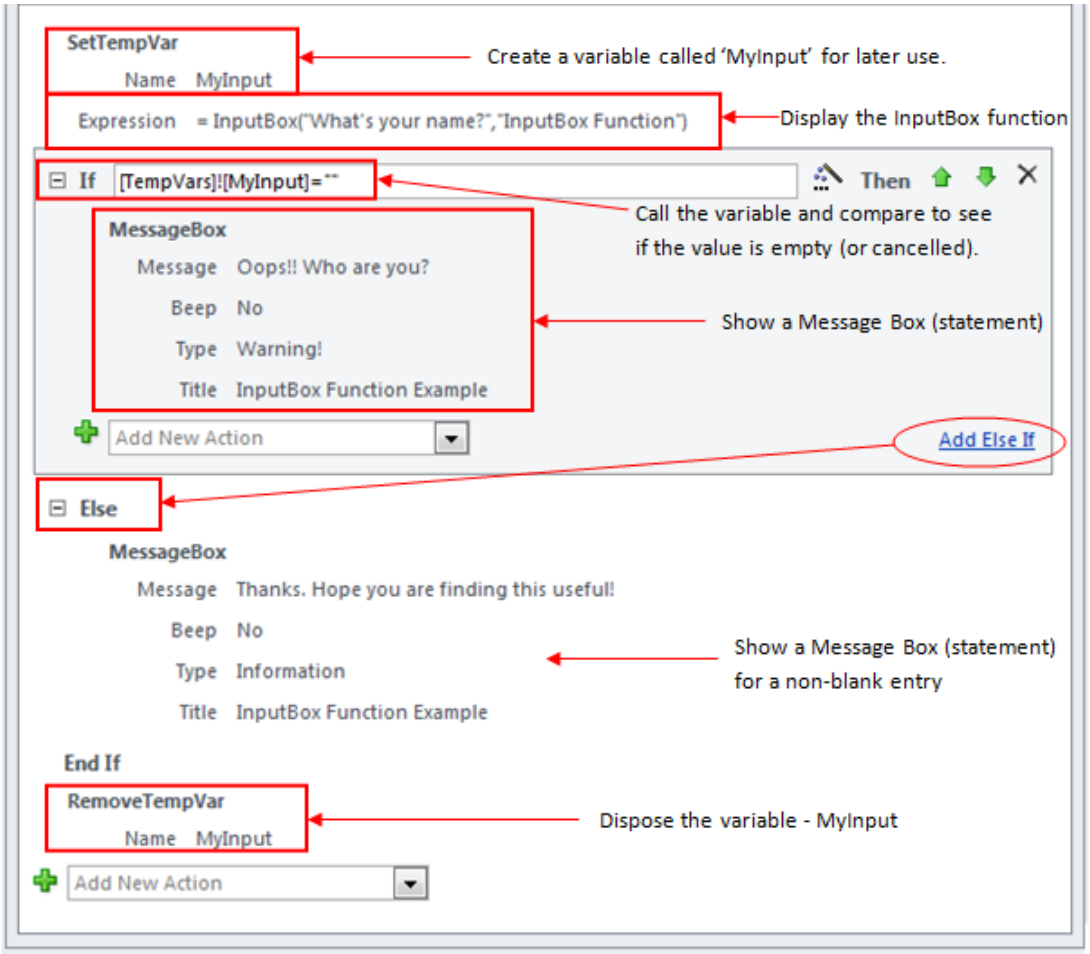
Input box

An **`InputBox`** comes in the form of a function only and returns the contents of the single textbox as entered by the user and can therefore be typically used in the **`If`** action (*or any action that can hold a value*).

An example of an **InputBox** function



Create a macro that calls the above **`InputBox`** and add a logical If action to handle the returning value which in this example tests for an empty value (*or not*).

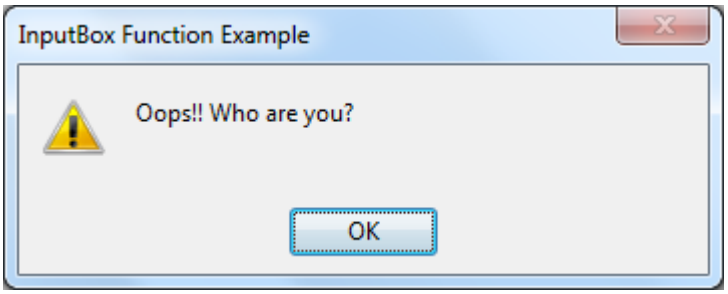


Access 2016 – Macro Designer, Inputbox example

Continues/...

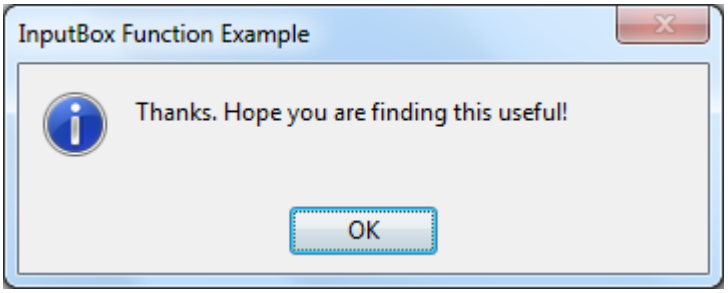
In the above example, the **`InputBox`** function is set to a variable called *MyInput* and then it is tested in an If…Else action.

If the value of the variable is empty (“”) then display a **`Message Box`** (*statement*):



The variable will return an empty value if you clicked ‘OK’ with no content or clicked the ‘Cancel’ button.

If the value of the variable is not empty, then display the following **Message Box** (*statement*):



The variable is then disposed of and the macro ends.

Note: *Using the* ***InputBox*** *function is a way to interact with a macro and ultimately the database but since one of Microsoft Access’s strengths is Forms, it would be better to build your own input box and have the full flexibility of managing one or more controls with embedded macros attached.*

Introducing Variables

A variable is a location that holds a value in the computer's memory which is then recalled as often as required during a macro call. It avoids the need to archive data anywhere else (*which improves performance too*).

It's the equivalent of saying to you to remember the following *telephone number* being "+44 (0)208 897 4567" where the *telephone number* is the name of the variable (*location*) and the value is the number itself. At some point you will want to recall the *telephone number* and use it somewhere.

With the introduction of Access 2007, variables were added to Access macros and this gave you one more reason why you could dispense with VBA code.

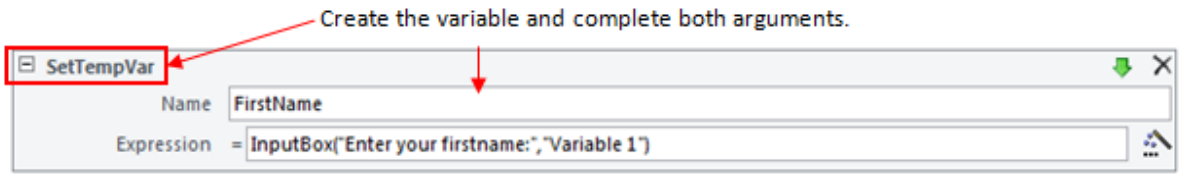
This version continues to use the **`TempVars`** collection and two newer variable actions have been added with the release of Access 2010 which are called **`LocalVars`** and **`ReturnVars`**.

Let's a take a closer look at these variables:

TempVars

The **`TempVars`** collection was added and allow programmers to create as many different variables required which are known as a **`TempVar`**. Notice the two keywords; **`TempVars`** (*plural*) v **`TempVar`** (*singular*) which is a member (*or element*) of the plural keyword (collection).

To use any variable, you must first use the action **`SetTempVar`**.



Access 2016 – Macro Designer, SetTempVar Action

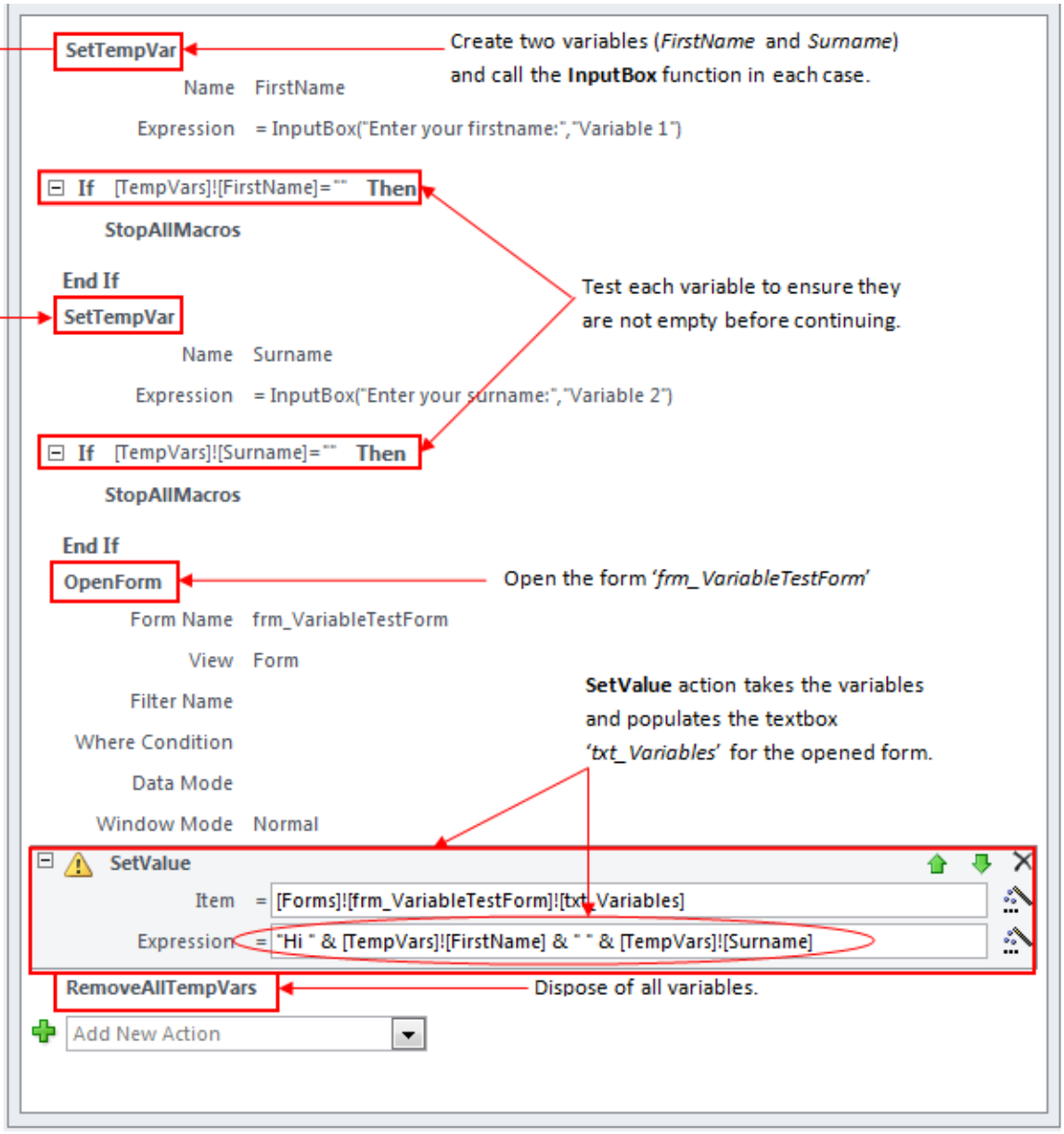
Each variable requires a separate **`SetTempVar`** which is added to the collection of **`TempVars`**.

To remove a variable or all variables in a macro, use either **`RemoveTempVar`** or **`RemoveAllTempVars`**.

If you do not remove a variable, then any variable defined remains in memory until the database closes and not when the macro closes. This is why it's important to dispose of variables and their values when you no longer need them.

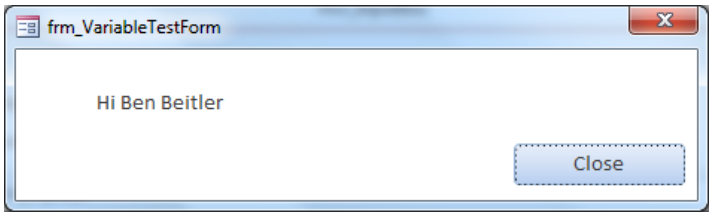
Continues/...

In this example, I've created two variables, prompting the user each time with an **`InputBox`** function which is then tested with an **`If`** action (*for empty values*) and loads a form populating a textbox control from the values supplied into the two variables:



Access 2016 – Macro Designer, Variable Example

The form loaded looks like:

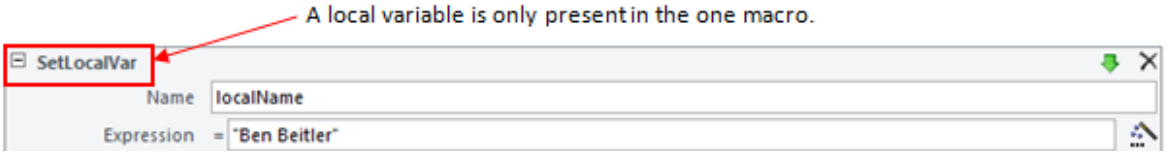


If I failed to include the action **`RemoveAllTempVars`** at the end of the macro, other macros could still utilise the two variables which can sometimes be a little dangerous.

LocalVars

A different type of variable called **`LocalVar`** which is a member of the **`LocalVars`** collection and behaves exactly the same way as **`TempVars`**.

This is a local variable which only exists in the macro it was defined in and therefore can help manage how and where macro variables are utilised.



Access 2016 – Macro Designer, Local Variable Example

There is no reason and therefore no action to remove a local variable since it's disposed of when the macro ends.

ReturnVars

There are many times in an application when you want to read a value from a table and use it somewhere else. This is true for applications running in Access itself and in the browser. For example, you might receive an input from a text box on a form, and need to retrieve the ID number that matches the input.

Another place where you might want to return a value is from a data macro that performs some calculation across a set of data.

To return a value, you can use the **`SetReturnVar`** action, which adds to the **`ReturnVars`** collection. The **`SetReturnVar`** action is used in a data macro to return a value from a table to a macro object or embedded macro.

The overall workflow goes something like this.

1. The macro object calls the **`RunDataMacro`** action to execute the specific data macro.
2. The data macro in question performs some calculation or retrieves a value. It then calls **`SetReturnVar`** to the value to return.
3. The macro object then uses the **`ReturnVars`** collection to retrieve the value.

There are a couple of things to be aware of with the **`ReturnVars`** collection. First, the values in the collection are only available in macros that run in the user interface — that is, macro objects or embedded macros. These values cannot be used in expressions on forms or reports.

Continues/…

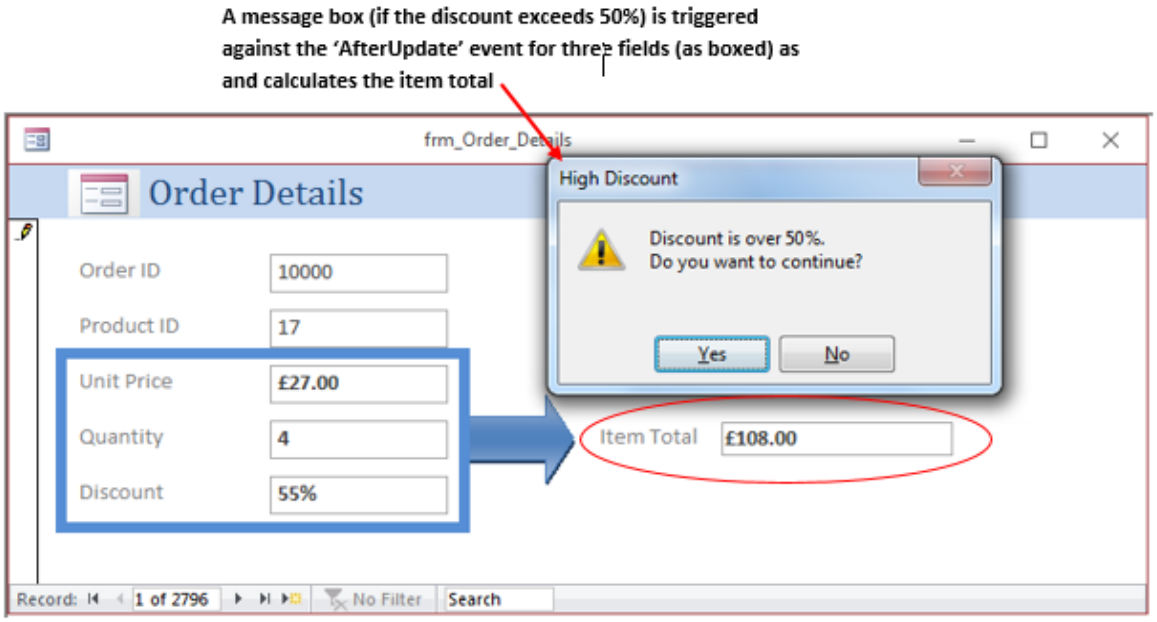
Second, the return value can only be used in the macro that called it. In other words, you cannot call **`RunDataMacro`** from one macro and use the **`ReturnVar`** in another macro. Even with these limitations, **`ReturnVars`** are still a valuable tool for developing using macros and data macros (*covered later in this guide*).

Embedded Macros

This type of macro applies from version 2007 through to the current version (2016) and is a way to store macros within the host object (*as embedded*) which can be a form or a report.

The benefits of allowing the host object to directly store macros protects and hides macros from being executed outside the host object. The added use therefore, makes this a private macro and better for interrogating controls in forms and reports.

In this simple example, I'm going to set three embedded macros to the '*After Update*' event of each field which triggers the same macro. Take a look at the following screen:

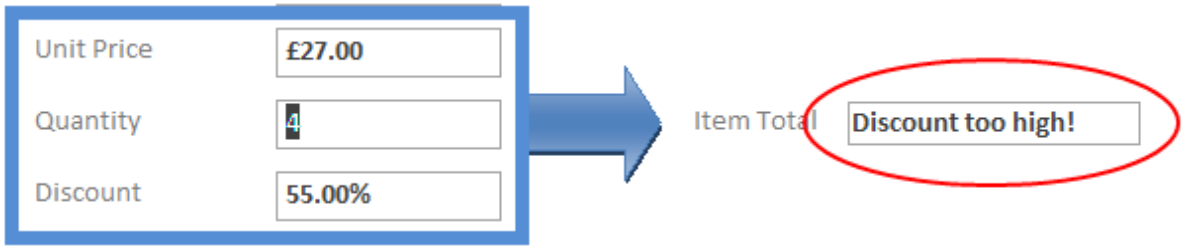


Access 2016 – Embedded Macro Example

A message box appears if any value is edited within the three fields (as boxed in blue) and the *Discount* field exceeds 50%.

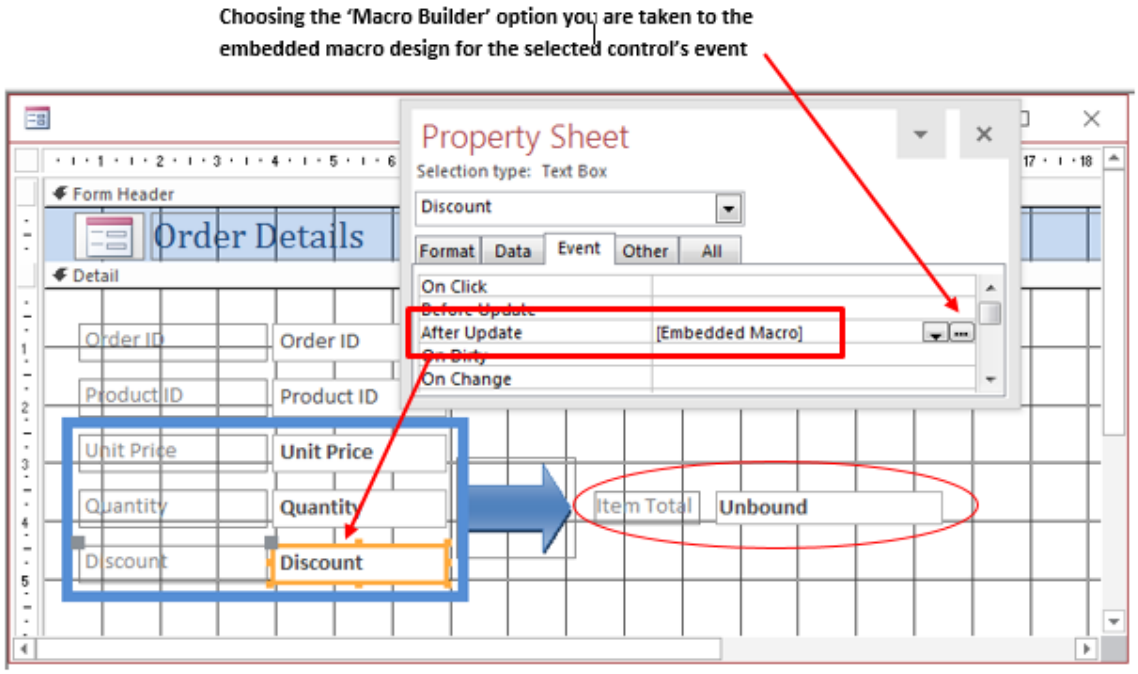
Continues/…

The user then confirms the prompt and if they choose yes, it calculates the new total into the *Item Total* field. If no, then the same field populates with the following narrative ***Discount too high!***



Needless to say, if the discount is equal to or less than 50% then the *Item Total* field just recalculates using the formula: **([Unit Price]*[Quantity])*(1-[Discount])**.

Take a look at the design view for the same form before we look at the macro code:



Access 2016 – Embedded Macro Example, Form Design view

Each control (*text boxes boxed above*) has their own dedicated events (***Event*** *tab from the Property Sheet*) and for this example to fully work, we will need to repeat the following coded embedded macro to the *Quantity* and *Discount* fields respectively to their own events.

> ***Note:*** *You will need to build a form (based on the* ***Order_Details*** *table) using the quick create form icon (or any other of your choice). Make sure you have the key fields in place. Add an unbound Textbox and name it txt_Total. Save changes.*

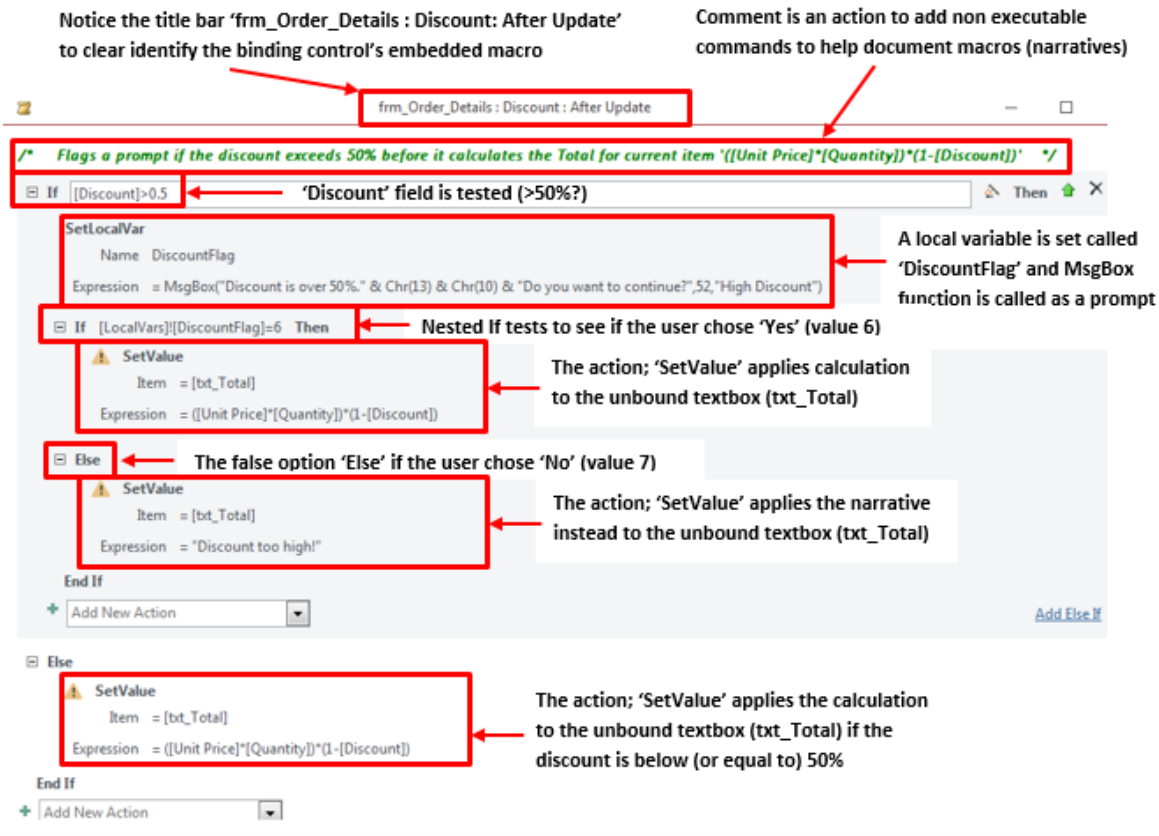
Continues/…

For the macro itself, we are going to:

1. Add a **`Comment`** (*something new for you here!*)
2. Set a local variable (**`SetLocalVar`**)
3. Have a main **`If`**…**`Else`** statement
4. Add a nested **`If`**…**`Else`** statement to the true condition of the main **`If`** test
5. Set value to the *Item Total* field (**`SetValue`**).

Note: *You will need to click the '**Show All Actions**' icon to reveal the unsafe macro actions (command called 'SetValue')*

Let's take a look at the macro designer code for the **[Discount]**, *After Update* event:



Access 2016 – Embedded Macro Example, After Update (Discount)

The green text at the top of the screen is a way to add comments which will not execute in the order they appear. It's good practice to add comments especially for the more complex and longer based procedures to help document workflows. Comments can be added at any point and you just add the **`Comment`** action which is listed from the *Program Flow* category in the **Action Catalog** window pane.

The rest of the code has been explained already in this guide and you need to revisit the relevant sections to explain workflows (*as illustrated*).

The final point here is to the message box itself. Notice how the prompt appears over two lines due to two appended function calls in the argument prompt; **… & Chr(13) & Chr(10) & ….** (*Return + New Line*).

Note: *One could argue the fact that the above example may have been written using calculation (expressions) and no macros. But remember that expressions simply return one answer to the control whereas macros can run several actions with the returning answer.*

Embedded macros are typically used for events like the *Click* event for a Command Button, the *Load* event for a form, the *After Update* event to a Combo Box control and the *On Lost Focus* event for a control.

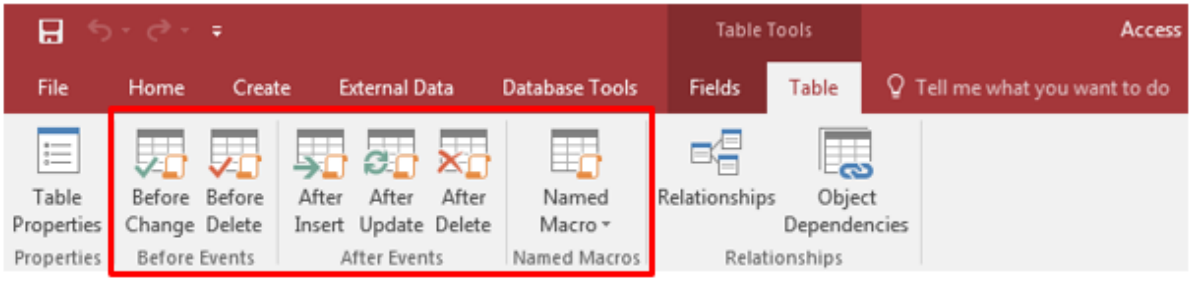
Events are probably the best way to trigger a macro as the system listens for the event to occur and no user intervention is required, making your application seamless and fully automated.

Data Macros

This is a relatively newer feature (introduced from 2010) and allows events in a table to run a macro which can also be called from macro objects too.

They appear in two categories with five event options to choose:

1. **Before Events**
    a. **Before Change**
    b. **Before Delete**
2. **After Events**
    a. **After Insert**
    b. **After Update**
    c. **After Delete**



Access 2016 – Table tab, Ribbon Bar

Before Events

If you have worked with the ***Before*** events in a form, then you will be familiar with how these events work in a table. The ***Before Change*** event executes just before the record is updated and the ***Before Delete*** would execute just before the record is deleted.

These events can also be cancelled as they are triggered and are used to typically validate data, if for some reason you haven't used primary and secondary keys (*indexes*).

Continues/...

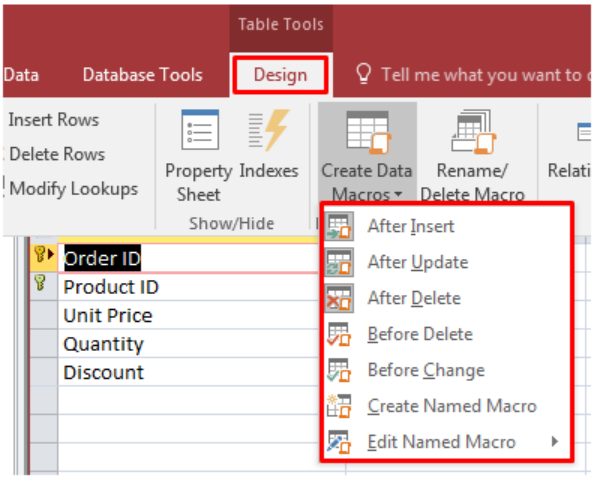
Here's an example using the *Order_Details* table:

Order_Details

Order ID	Product ID	Unit Price	Quantity	Discount	Click to Add
11068	28	£45.60	8	15.00%	
11068	43	£46.00	36	15.00%	
11068	77	£13.00	28	15.00%	
11069	39	£18.00	20	0.00%	
11070	1	£18.00	40	15.00%	
11070	2	£19.00	20	15.00%	
11070	16	£17.45	30	15.00%	
11070	31	£12.50	20	0.00%	
11071	7	£30.00	15	5.00%	
11071	13	£6.00	10	5.00%	
11072	2	£19.00	8	0.00%	
11072	41	£9.65	40	0.00%	
11072	50	£16.25	22	0.00%	
11072	64	£33.25	130	0.00%	
11073	11	£21.00	10	0.00%	
11073	24	£4.50	20	0.00%	
11074	16	£17.45	14	5.00%	
11075	2	£19.00	10	15.00%	
11075	46	£12.00	30	15.00%	
11075	76	£18.00	2	15.00%	
11076	6	£25.00	20	25.00%	
11076	14	£23.25	20	25.00%	
11076	19	£9.20	10	25.00%	
11077	2	£19.00	24	20.00%	
*			1	0.00%	

Access 2016 – 'Order_Details' Table tab – 5 fields

Open the *Order_Details* table in either design view or open view (*they both can set a data macro*).

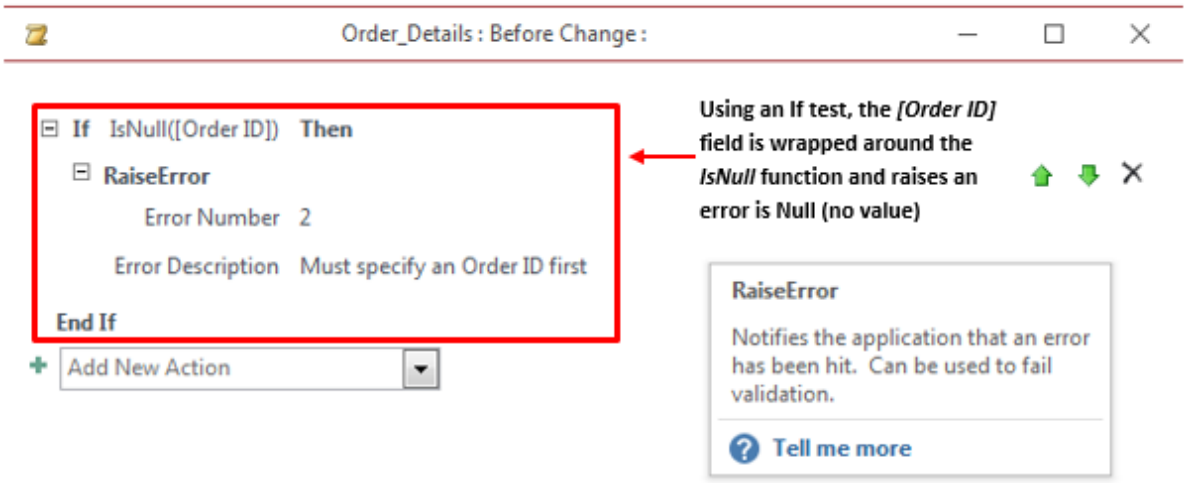


Access 2016 – 'Order_Details' Design tab, Data Macros

I'm going to open (*run*) the table.

Continues/…

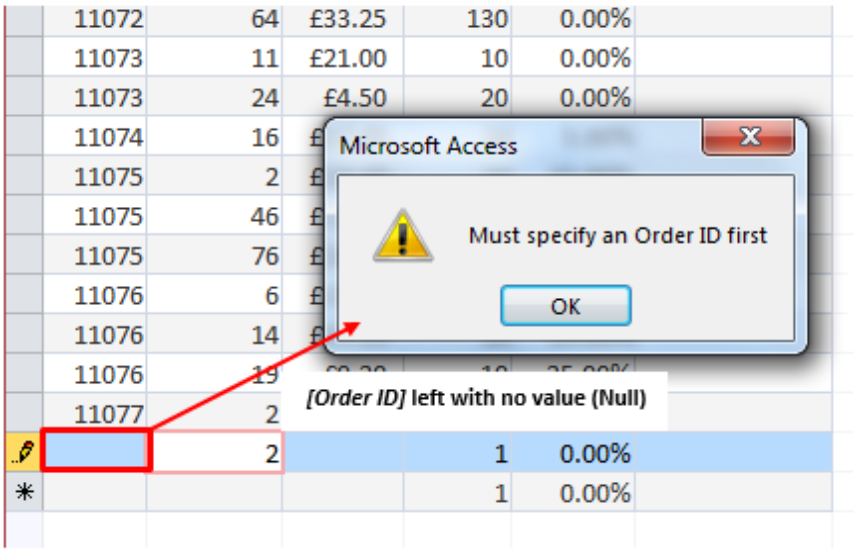
Select the **Table** tab from the Ribbon Bar and click the ***Before Change*** event icon. The following macro designer window opens and I have set the following actions:



Access 2016 – Macro Designer for the 'Order_Details' table (Before Change event)

In this example, I'm trapping for the *[Order ID]* field which really should be (*and is*) indexed but we will assume it's not and therefore trap it with your own validation instead by using the **`If`** action and a condition involving the **IsNull** function.

If the condition is true (*no value has been entered*) when attempting to save the change, the before change event is triggered and the **`RaiseError`** action is fired.



Access 2016 – Order ID – Raises the error

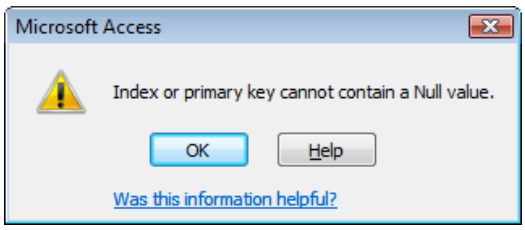
Continues/…

The two arguments to the **`RaiseError`** action must be completed but the ***Error Number*** has no meaning and can be any value (*I chose the number 2!*).

You will need to save your changes to keep the macro or you will be prompted anyway when closing the window view.

***Note**: If you are working via the design view mode, you will need to save the table too.*

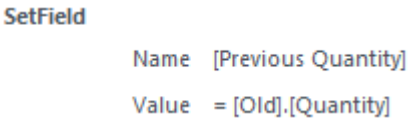
Since my table has an index set to both the *[Order ID]* and *[Product ID]* fields then because indexed fields are mandatory and cannot be left without a value, you get the standard Access message when attempting to save the record.



Using this type of event allows users to automate and flag data values for records in a table like for instance; a *Status* update field for an order which starts life as a new record. Then it changes from an *Updated* state to a *Dispatched* state and other any other status including perhaps *Cancellation*.

Here's a quick illustration using the **[Old]** keyword *property* command to the *[Quantity]* field.

First, create a new field to the *Order_Details* table called *[Previous Quantity]* and make it Number data type (as Integer). Take a look at this before changing the macro:



Save changes and test it out!

The **[Old].[Quantity]** field is populated by first remembering the *[Quantity]* value and then populating the new *[Previous Quantity]* field using the **`SetField`** action.

The **[Old]** keyword is very easy and useful!

***Note:** When creating a data macro, you may have noticed some of the icons from the Ribbon Bar are enabled and include 'Run', 'Single Step', 'Convert Macros' and 'Show All Actions' as they are features for macro objects & embedded (excluding 'Convert Macros') ones only.*

Also note that the action catalog list is filtered for the two different event categories and not all actions are available as listed with macro & embedded objects.

After Events

There are three *After* events for data macros: *After Insert*, which fires after a record has been created/added; *After Update*, which fires when a record has been updated/saved; and *After Delete*, which fires after a record has been deleted.

***Note**: The 'After Update' event does not fire when a new record is added (only for existing records when saved).*

If multiple records are affected at once, the *After* events will fire for each one. For example, let's say that you copy ten records and paste them into a table that has the *After Insert* event defined. The *After Insert* event will fire once for each insert, or ten times. Similarly, if you were to delete ten records, the *After Delete* event fires for each record.

There are different macro constructs over and above the ones we have seen so far which include:

- **Comment**
- **If, Else, Else If**
- **Group (simply for logical grouping)**
- **Call other named macros passing parameters**
- **For Each Record (iterate over a recordset or SQL query)**
- **Lookup Record (returns a single record)**
- **Create Record**
- **Edit Record**

The number of actions varies too and are listed below:

- **Set Field**
- **Cancel Record Change (breaks out of Create and Edit Record changes)**
- **Delete Record**
- **Send Email**
- **Run Data Macro**
- **Set Local Variable (scope to the current instance of a macro)**
- **On Error**
- **Raise Error**
- **Clear Macro Error**
- **Stop Macro**
- **Stop All Macros**
- **Exit For Each Record**

***Note:** Data macros will not run with linked tables but in the back-end database file instead (which must be ACCDB). Also, you cannot call VBA procedures from a data macro but you can with a named macro and one with parameters.*

*I will demonstrate an example of one of the After events once I've covered the next topic called **Named Data Macros**…*

Named Data Macros

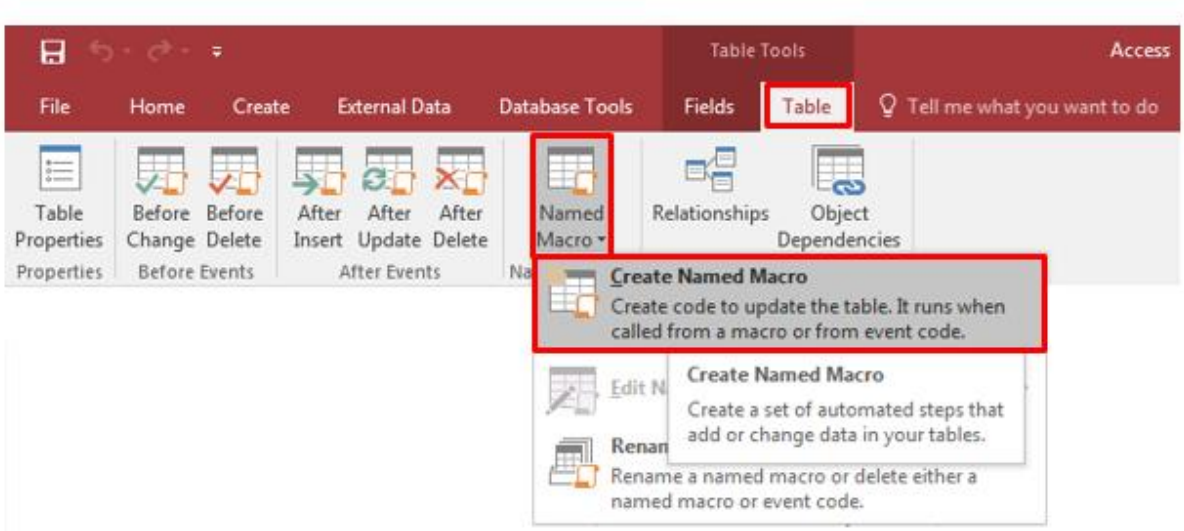
In this section I'm going to cover another important aspect which expands the way these events are typically used and they are **Named Data Macros**.

A Named data macro is a stand-alone macro that is associated to a table but not for a specific event (*remember, there are five events available*).

The benefits of using this type of macro is more for re-use and planning ahead where you can create 'generic' data macros and attach them to one or more tables (*their events*) or create a macro object and set parameters to pass into the event being triggered.

To run a named data macro from another macro, you use the **`RunDataMacro`** action. This action provides a box for each parameter that you create, so that you can provide the necessary values.

To create a named macro, you must first open a table where you wish to store your macro and choose from the following icons:



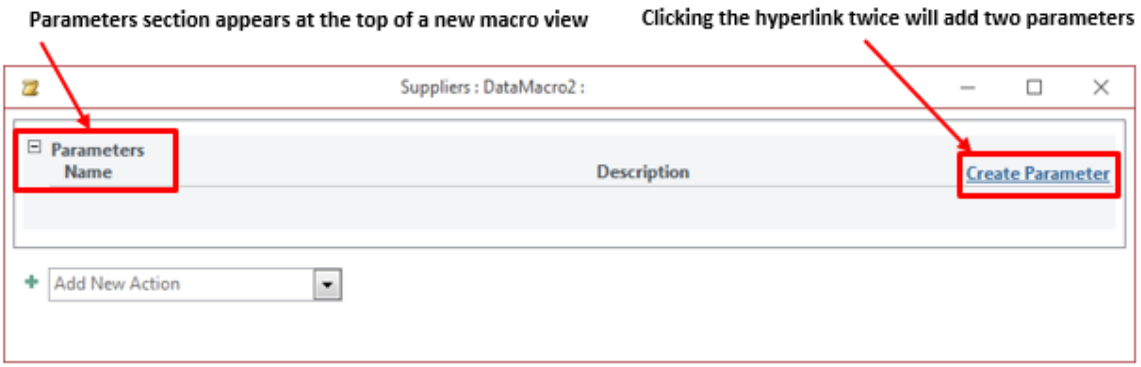
Access 2016 – Named Data Macro Icons

Make sure the table is relevant to the macro procedure as it will make it easier to refer to fields of the table local to it rather than having to use the full referencing of other tables and fields.

Once you have at least one saved named macro, the **Edit Named Macro** option is enabled, displaying a sub menu of one or more items to edit.

Continues/…

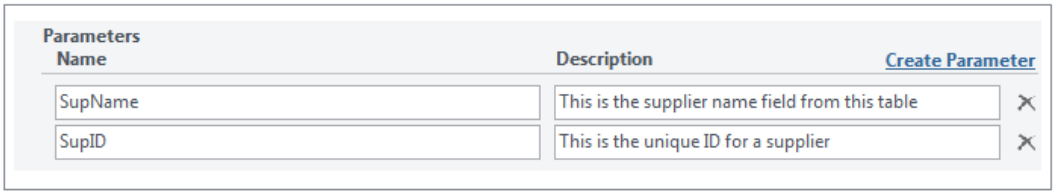
When you create this type of macro, the following design view is similar to a normal macro canvas with the addition of choosing one or more **Parameters**.



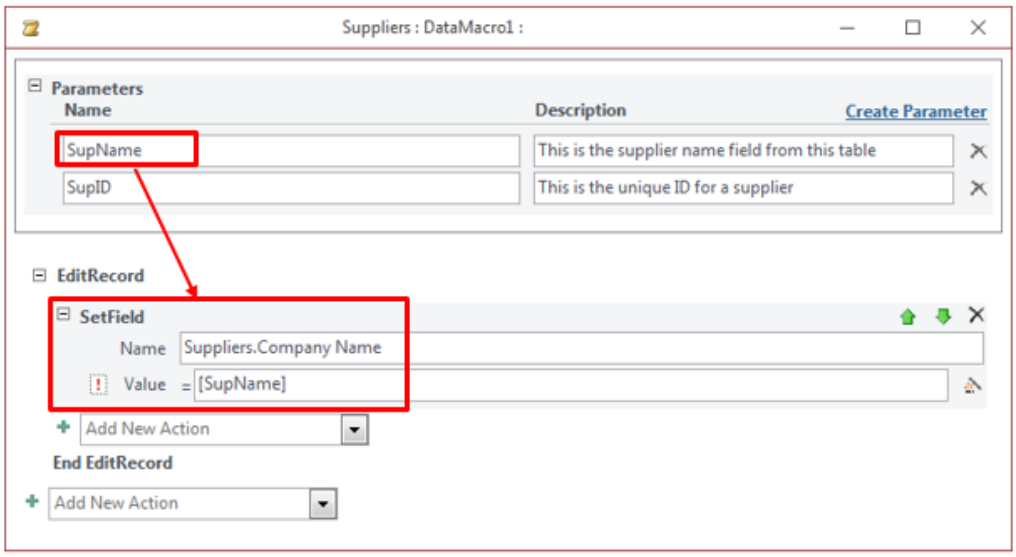
Access 2016 – Named Data Macro, Parameters Section (at the top)

Parameters are of course optional and are the equivalent to variables holding a value that can be passed to other macros (*and their events*).

To create a parameter, just set a name and add a description:



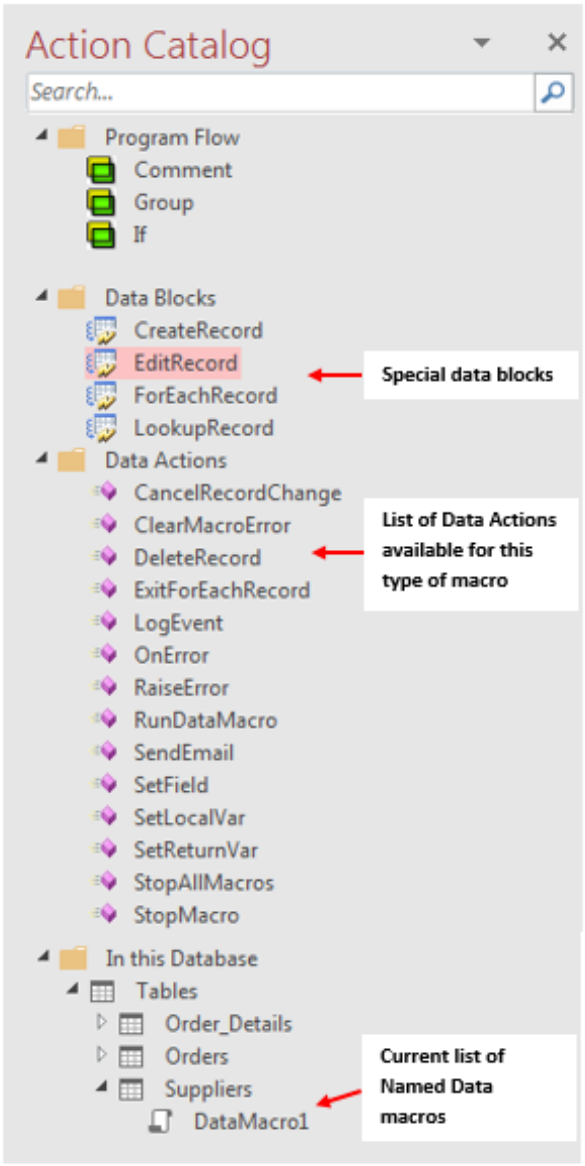
To set a value for a defined parameter, in the macro itself you choose an action which in this case could be **`SetField`** which is a member of the **`EditRecord`** data block:



Access 2016 – Named Data Macro Design, SetField Action

Continues/...

With this type of data macro, you have a different list of Actions available which can be found in the **Action Catalog** pane:



Access 2016 –Action Catalog (Named Macros)

When you save the macro you are prompted to name it and it then becomes a member of the table associated.

Note: *Named data macros cannot be debugged (stepped into), converted, executed from the design view and therefore must be first saved which is attached with the table and executed via an event of any table. There are no extra hidden actions available, only what's listed and therefore the 'Show All Actions' icon is also dimmed in grey.*

To call and use a named macro, you use the **`RunDataMacro`** action which can be called via any type of macro

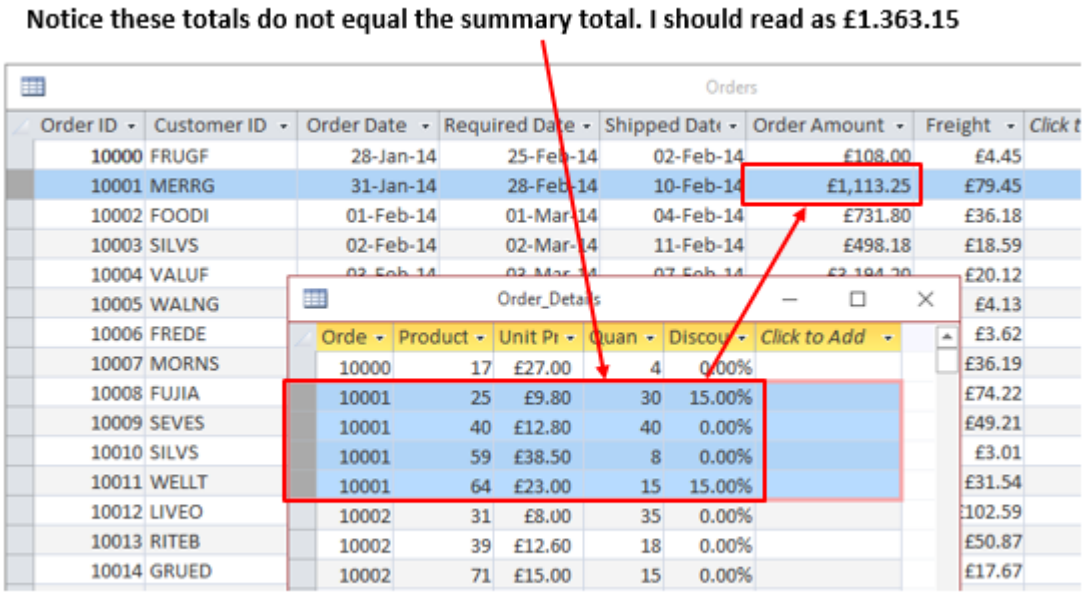
Example of a Named Data Macro

In this example, I want to tie up some loose ends to this type of macro procedure and bind together a named data macro being triggered by one of the *After* events in a table.

From various books on this subject, there are very few examples around and I want to give a real world example to help you see the power and benefit of using these macros.

The scenario:

In the *Orders* table, I have an *Order Amount* field which is the total order value based on one or more products sold specifying a *Quantity*, *Unit Price* and any applied *Discount* from the *Order_Details* table.



Access 2016 – Order and Order_Details Table Relationship

Quick math check - the four items with [Order ID] 10001 in the 'Order_Details' table when taking discount into consideration should total £1,363.15 but the single [Order ID] 10001 in the 'Orders' table's [Order Amount] field is incorrect.

If I add or amend an item in the *Order_Details* table, it will not update the *Order Amount* field (*from the Orders table*) unless I trigger a macro via an event.

Continues/…

I'm going to create three macros:

1. A Named Data Macro in the '*Orders*' table to run and update all the orders *[Order Amount]* values.

2. A 'general' macro object to call the Named Data Macro from the '*Orders*' table.

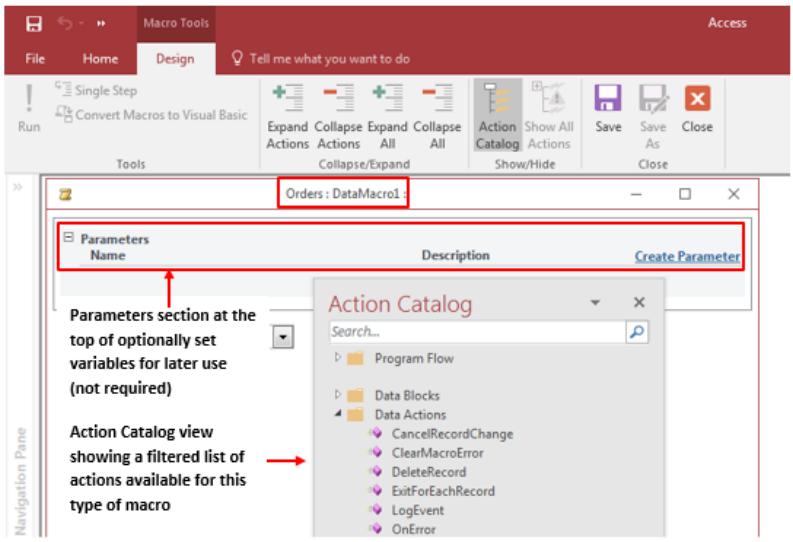
3. A Named Data Macro in the '*Order_Details*' table to be triggered from the *After* events within this table collecting the [Order ID] value from the '*Orders*' table and update the *[Order Amount]* field (in the '*Orders*' *table*).

Named Data Macro in the Orders Table

This macro will do the following:

1. Loop through all the records in this table.
2. Create a variable called *TotalOrderValue* setting its value to zero for the iteration.
3. Locate the '*Order_Details*' table where there is an *[Order ID]* match between the two tables.
4. Loop through all the record of the '*Order_Details*' for the same *[Order ID]* match (*as there is potentially more than one item for an order*).
5. Set its variable as calculation between the *[Quantity]*, *[Unit Price]* and *[Discount]* fields for each matching record in the *'Order_Details'* table accumulating the previous found record.
6. Edit the *[Order Amount]* field's value in the '*Orders*' table for the matched *[Order ID]* value between the two tables.
7. Move to the next order record.

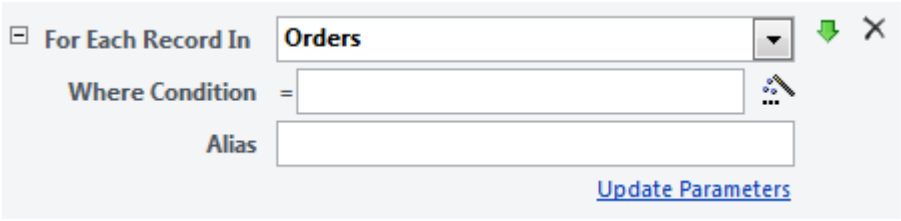
Open the '*Orders*' table and locate the **Table** tab (*from* **Table Tools**) and choose from the **Named Macro** icon, **Create Named Macro** and you will be taken to the following screen:



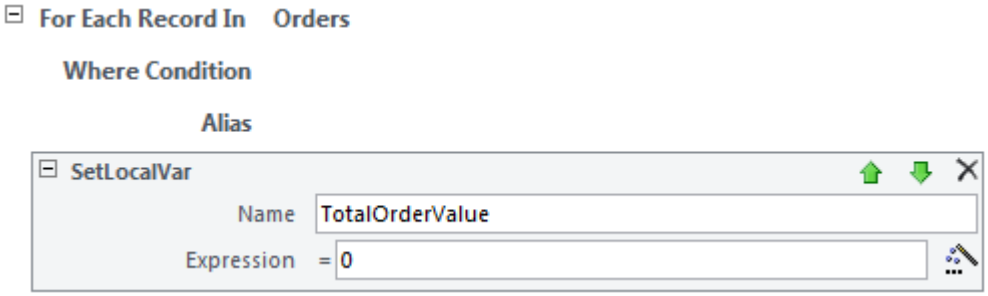
Access 2016 – Orders, Data Macro Design View

Continues/...

Add the action **`ForEachRecord`** and choose the 'Orders' table from the drop-down list provided.

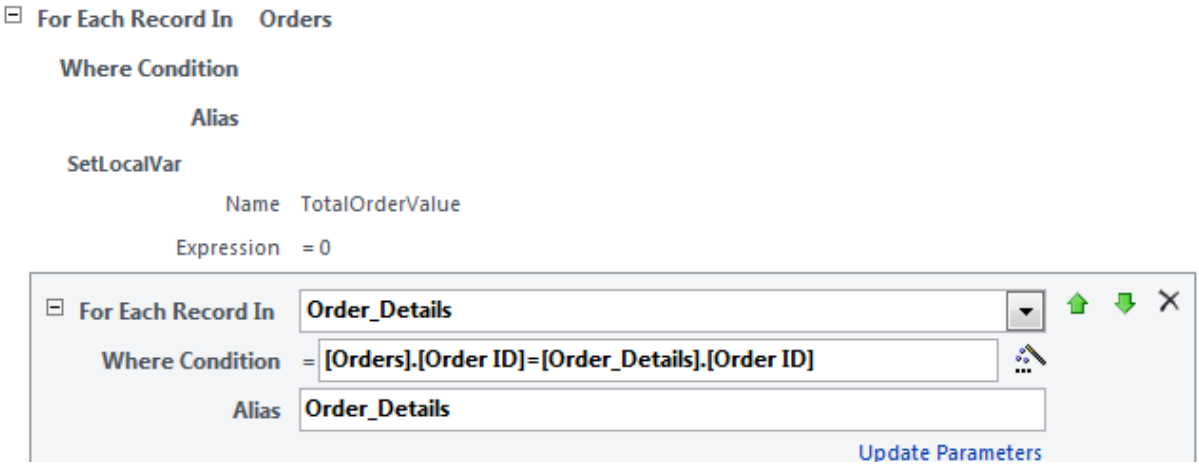


Now add another action below (but inside the **`ForEachRecord`**) to set a local variable. The action is called **`SetLocalVar`**.



Enter a name for your new variable (*I've called it* **TotalOrderValue**) and set its expression to zero as this needs to be reset for each repeating order before calculating the accumulative value later on in the '*Order_Details'* table.

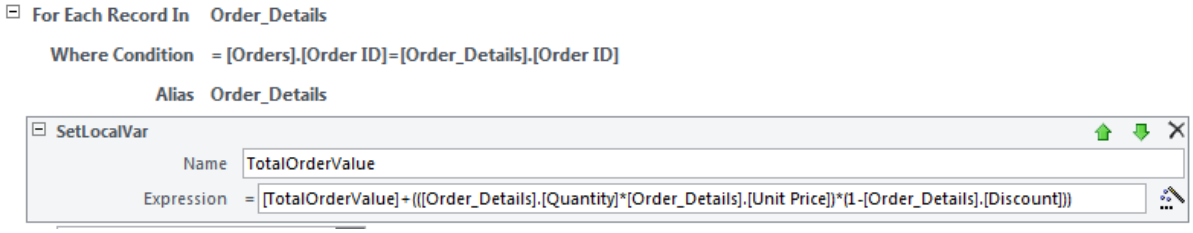
Next add another **`ForEachRecord`** action (*still within the first* **`ForEachRecord`** *section*) which will loop through records from the '*Order_Details'* table matching the common *[Order ID]* field between the two tables.



The Where Condition filters for the unique *[Order ID]* (*from* '*Order's*) to one or more matching *[Order ID]* values in the '*Order_Details'* table.

Continues/…

Within this loop section, we need to set our local variable (*same name as before*) but this time calculating between *[Quantity]*, *[Unit Price]*, *[Discount]* fields and any previous matched items for the same order (*variable name*).

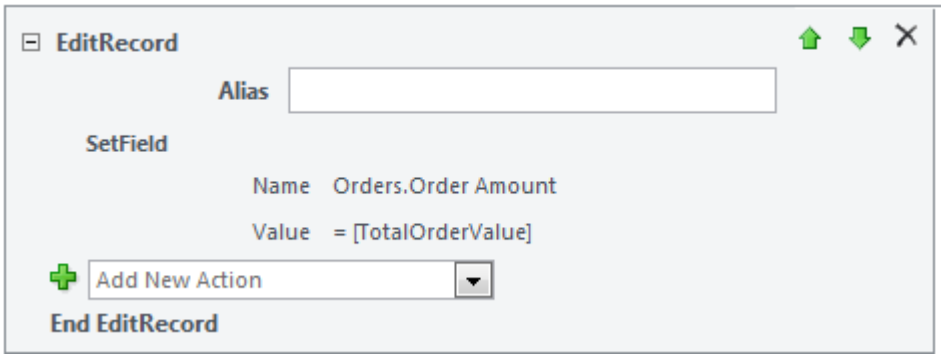


The expression to calculate a line item total is as follows:

(([Order_Details].[Quantity]*[Order_Details].[Unit Price])*(1-[Order_Details].[Discount]))

Make sure all references to tables and fields are wrapped in square brackets and not to be confused with normal parenthesis. Use the build tool to help build the expression.

Now we need to add another action below the variable called **`EditRecord`** which is a block in its own right and we call the **`SetField`** action inside it.



The **`SetField`** action has two arguments; **Name** to locate which table and field you want to set and the **Value** which could be a value, expression or in this case a variable (*TotalOrderValue*).

Save this as **UpdateOrderAmount** macro and close.

You can add as many different named macros as required, edit, delete and rename it too.

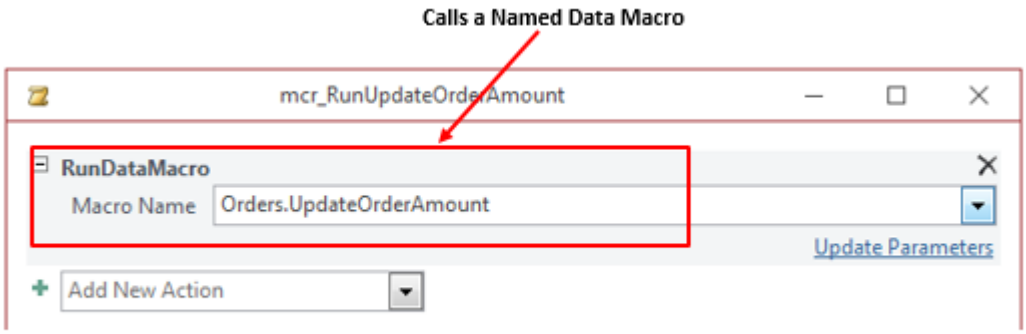
You will not be able to run the macro here however as it requires either an event to trigger it or for this example, create a macro object to call this named data macro which is covered in the next part of this example.

Continues/…

General Macro Object updating the values in the Orders table

This macro will be added as a general macro object to the navigation pane and can be executed globally within your Access application.

It's a simple macro with one action call – the **`RunDataMacro`** action:



Access 2016 –Macro Design View, RunDataMacro Action example

Add the action **`RunDataMacro`** and choose from the drop-down list the desired data macro (*Orders.UpdateOrderAmount*).

Notice the hierarchy of the table name with a dot separator followed by the data macro name.

Save the macro as **mcr_RunUpdateOrderAmount** and close.

Run the macro and nothing appears to have happened but be assured it did call the data macro in question, updating all orders *[Order Amount]* value from the '*Order_Details'* table.

Go and check the order amount value for an order against its details and see if they balance.

Now edit an item for an order in the '*Order_Details'* table and re-run the macro to see the update changes applied to the '*Orders'* table.

Of course, you could add a **`MessageBox`** action at the end of this macro to make it more user-friendly and inform the user the macro had completed – *I'll leave that for you to apply!*

This macro can be attached to any other object, perhaps as some update routine for general maintenance but in the real world it would make sense to update as changes are made to the '*Order_Details'* table itself for the matching order only.

In the next section, the last part of this example explains how to do that.

Continues/…

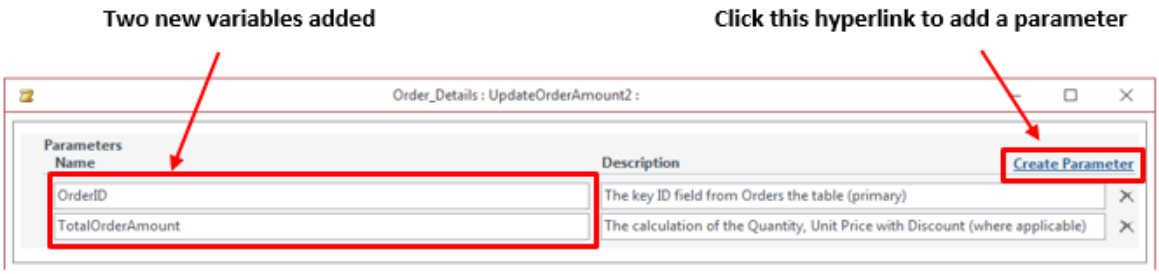
Named Data Macro and *After* events in the Order Details Table

The advantage of running an update order amount macro at the time of changes to the '*Order_Details'* table is that it only updates the related *[Order ID]* and does not loop through all the records of the '*Orders'* table (*which can be a performance overhead challenge especially when handling many thousands of records*) and it is also a more natural way to update data.

You would need to trigger this macro to all the *After* events; **Insert**, **Update** and **Delete** but I'm going to show you the main one when a record changes (*using the After Update* event).

Open the '*Order_Details'* table and create a new Named Data Macro.

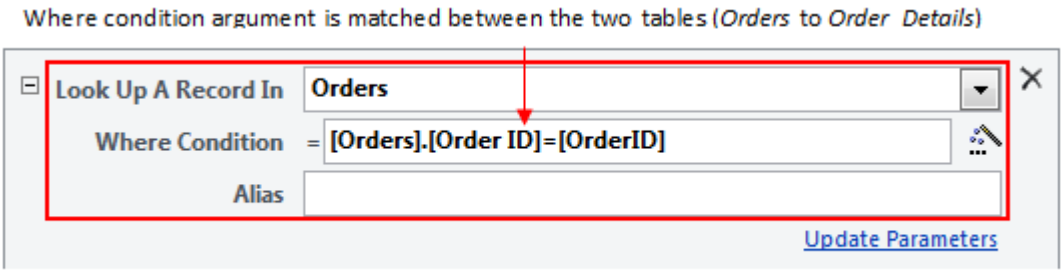
This time, we are going to utilise the **Parameters** section that appears at the top of the new macro design view. Therefore, click the blue hyperlink; **Create Parameters** twice (since we will need two variables):



Access 2016 –Macro Design View, Order-Details table

I have added two unique names with optional descriptions as shown above. The ***OrderID*** parameter will collect *[Order ID*] value to be passed into this macro when being called from an event and the ***TotalOrderAmount*** will keep count of the total order value for one or more items.

Next, I'm going to add the **`LookUpRecord`** action which is one of the data blocks available to tell the system to find a record (*a matching [Order ID]*).

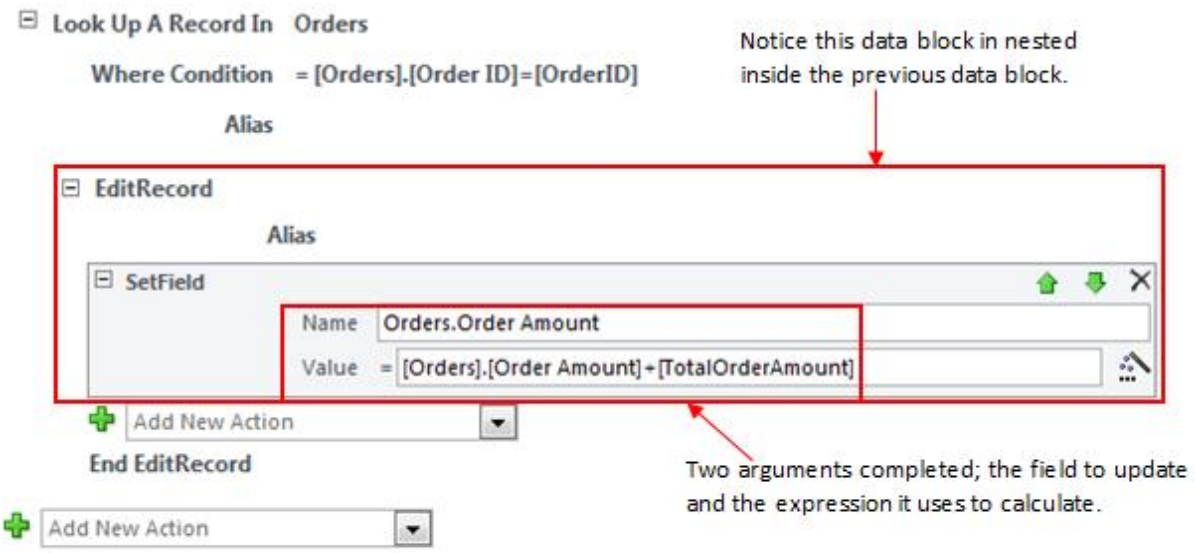


Access 2016 –Macro Design View, LookUpRecord Data Block

Continues/…

Choose the table required from the drop-down list (*which is in this case, Orders*) and set the **Where Condition** argument using the expression; **[Orders].[Order ID]=[OrderID]**.

Now we need to add another data block action **`EditRecord`** and add a final action into this section to set the field using the **`SetField`** action of the *[Order_Amount]* (*from the 'Orders' table*) and then update the running totals of all matching items from within this table ('*Order_Details*').



Access 2016 –Macro Design View, EditRecord Data Block & SetField action

The ***Name*** argument is the destination field (*with table alias*) to update and will assist you when typing a value or reference. The ***Value*** argument is a calculation taking the existing *[Order_Amount]* and adding to the new value (*variable*).

Remember, this is a data macro that will handle and pass values (*via its parameters*) when being called by one or more of the five events to this table.

Save the macro as ***UpdateOrderAmount2*** and close.

In this last part, I'm now going to show you the *After* event call which is going to trigger the above macro.

Continues/…

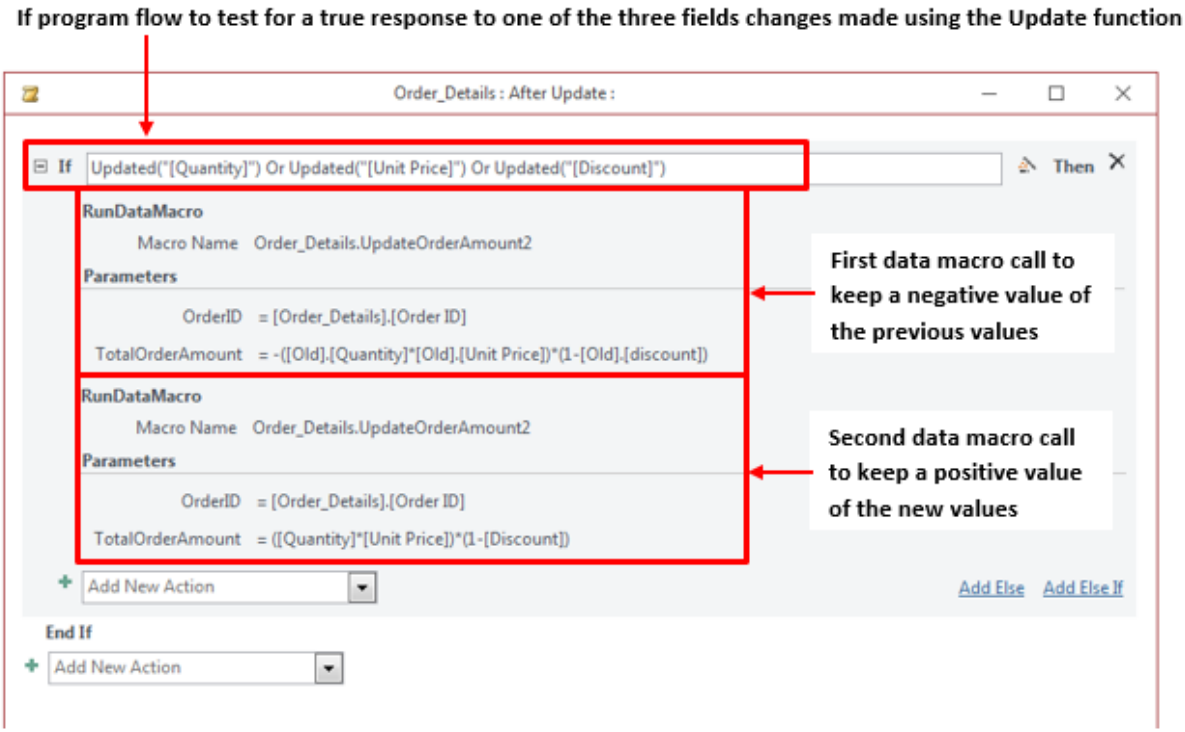
The workflow in this event is as follows:

1. Using an **`If`** program flow action, test for three fields that may have been changed using an Access function called **Update**.

2. If true, then use the **`RunDataMacro`** to call the new data macro and supply values (*as expressions*) to the two parameters remembering the previous values for the record using the **[Old]** property (*as a negative value*).

3. Once again, call the same macro but this time calculate the value (*total of order item*) as a new positive value. The balance between the two calls is updated to record and saved.

Take a look at this screen shot below:



Access 2016 –Macro Design View, After Update Event for Order-Details table

The **Update** function returns either a **`true`** or **`false`** value which is perfect for our If program flow action above. Any changes to the *Quantity*, *Unit Price* or *Discount* fields will trigger a true return value.

Continues/…

If true, the first **RunDataMacro** which has two parameters, is set using the following expression:

OrderID parameter = **[Order_Details].[Order ID]**

TotalOrderAmount parameter = **-([Old].[Quantity]*[Old].[Unit Price])*(1-[Old].[discount])**

Notice the **[Old]** property which takes the previous value recorded against each field and the negative sign returns a negative value ready for the second data macro call.

The same data macro is called again using the **RunDataMacro** action but this time takes the current values from the three fields using the following expression:

TotalOrderAmount parameter = **([Quantity]*[Unit Price])*(1-[Discount])**

Between the two ***TotalOrderAmount*** parameter values, a balance is returned.

Close and save macro – *test it out!*

Note: *You will need to add macro code to both the After Delete and After Insert events too and this is available from the download file at the end of this guide.*

Data Macro Errors – USysApplicationLog Table

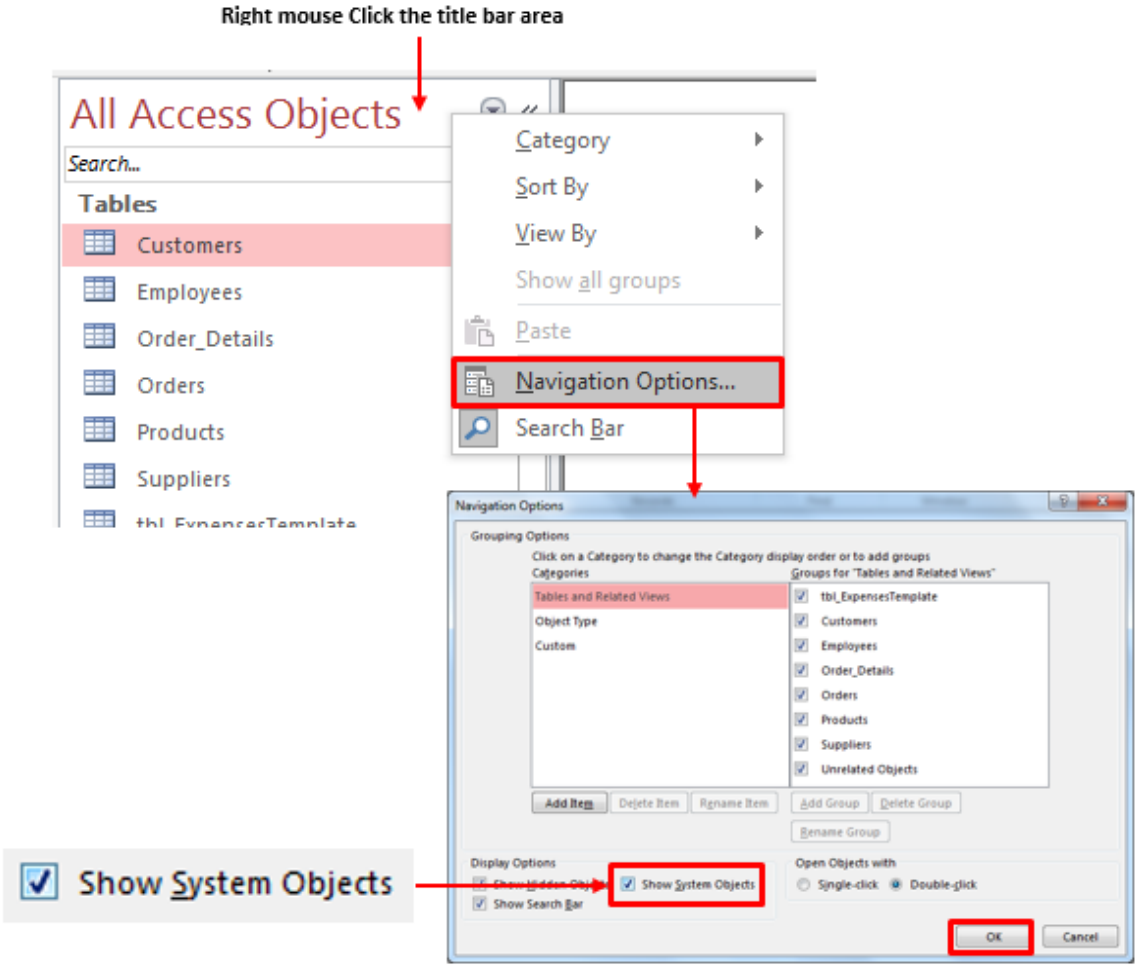
Because you cannot ***step*** and ***debug*** data macros as you can with general and embedded macros, a system table is generated automatically for you to view and interrogate.

This table will not appear until there are errors to report (*running a data macro which did nothing but has an error*).

There are two ways to locate this table which is called **USysApplicationLog**:

1. Unhide the navigation pane options to **Show System Objects**.
2. Go to the *BackStage* (**File** *tab*) and locate the Info section for **View Application Log Table** icon.

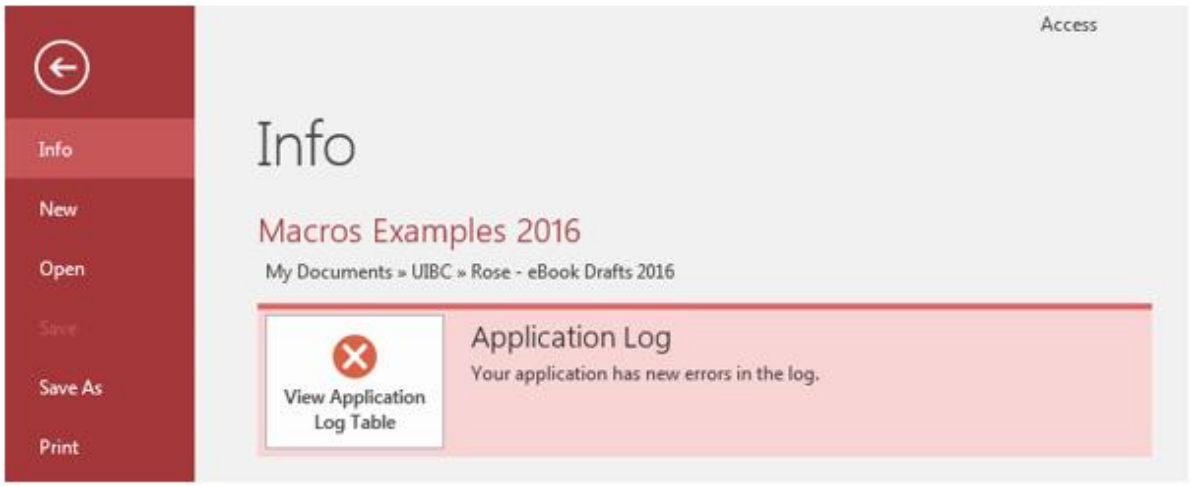
To show the system tables, right mouse click on the banner (*top grey bar*) and choose the **Navigation options…** command:



Access 2016 – Unhide System Objects

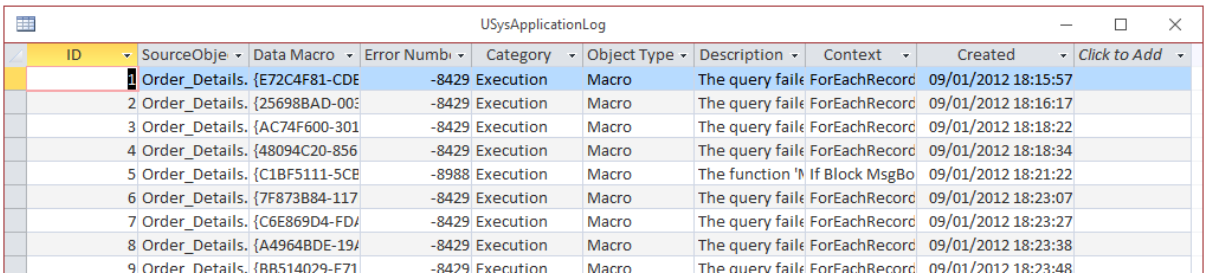
Continues/…

Alternatively, go to the *BackStage* using the **File** tab, ***Info*** section and click the **View Application Log Table** icon.



Access 2016 – Backstage via the File tab, Info Section

Open the **USysApplicationLog** table and view the data which contains basic information to help give you some clue to what happened, including a date stamp which can be sorted.



ID	SourceObje	Data Macro	Error Numb	Category	Object Type	Description	Context	Created	Click to Add
1	Order_Details.	{E72C4F81-CDE	-8429	Execution	Macro	The query faile	ForEachRecord	09/01/2012 18:15:57	
2	Order_Details.	{25698BAD-003	-8429	Execution	Macro	The query faile	ForEachRecord	09/01/2012 18:16:17	
3	Order_Details.	{AC74F600-301	-8429	Execution	Macro	The query faile	ForEachRecord	09/01/2012 18:18:22	
4	Order_Details.	{48094C20-856	-8429	Execution	Macro	The query faile	ForEachRecord	09/01/2012 18:18:34	
5	Order_Details.	{C1BF5111-5CE	-8988	Execution	Macro	The function 'N	If Block MsgBo	09/01/2012 18:21:22	
6	Order_Details.	{7F873B84-117	-8429	Execution	Macro	The query faile	ForEachRecord	09/01/2012 18:23:07	
7	Order_Details.	{C6E869D4-FDA	-8429	Execution	Macro	The query faile	ForEachRecord	09/01/2012 18:23:27	
8	Order_Details.	{A4964BDE-19A	-8429	Execution	Macro	The query faile	ForEachRecord	09/01/2012 18:23:38	
9	Order_Details.	{BB514029-F71	-8429	Execution	Macro	The query faile	ForEachRecord	09/01/2012 18:23:48	

Do not bother to edit or format the table, it will serve as no additional benefit to you –it's just an information& reference table.

Macro Example (Using a Form, Query & Report)

In this section, I'm going to show an example of how Access objects typically work together. In this scenario a form will load prompting the user to choose a value (*or two*) as parameters for a report which is triggered by the '**Run'** button and is supported by an underlying query passing criteria to the report.

This report is a list of customers showing a summary order amount for one or all countries for a chosen year:

Revenue for UK

Year: 2014

CompanyName	Total Revenue
Around the Horn	£1,211.50
Babu Ji's Exports	£61.20
Blum's Goods	£3,117.50
B's Beverages	£803.00
Dunn's Holdings	£3,329.60
Eastern Connection	£6,345.80
Empire Trading	£5,477.70
Fraser Distributors	£7,321.34
Hanover Poultry	£406.00
Island Trading	£1,056.50
Kingsgate Goods	£1,121.40
North/South	£261.00
Piccadilly Foods	£2,282.55
Queen's Express	£3,990.00
Richmond Sugar	£13,981.22
Seven Seas Imports	£4,053.33
Special Delights	£89.60
Sugar and Spice	£6,934.65
Trade More	£547.80
Wellington Trading	£589.05
Wolverton House	£162.50
Total Revenue	£63,143.24
Number ofTrans	21

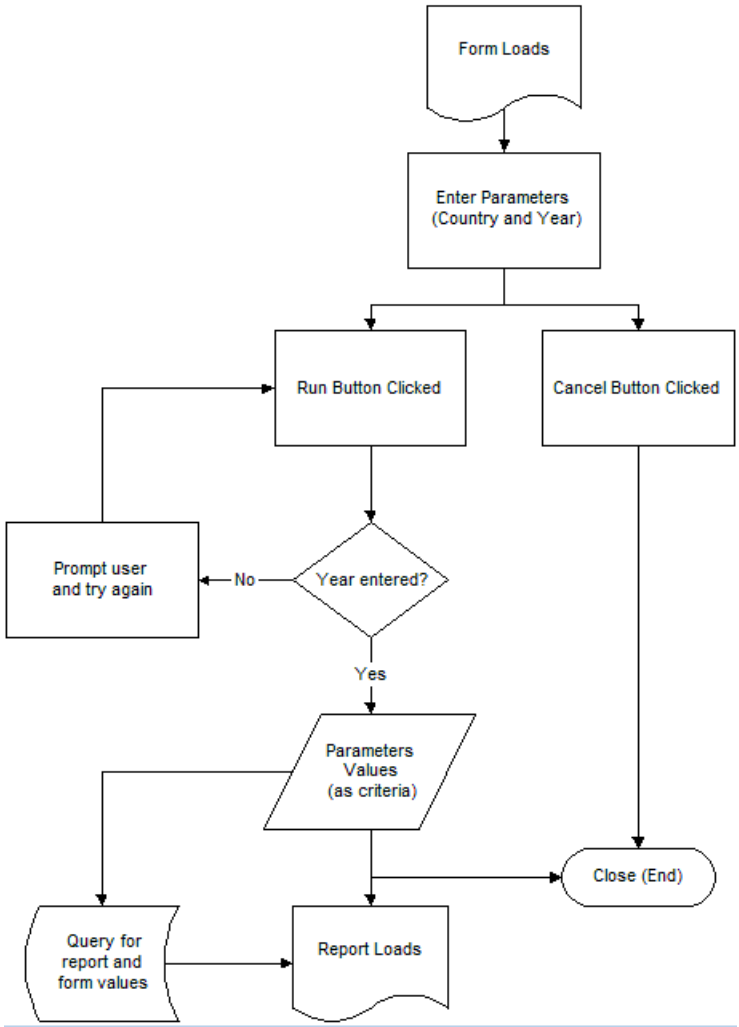
The workflow starts with a form prompting the user to enter either a defined country or leave blank for all countries and to choose from a pre-populated list of years:

Continues/…

The '**Cancel'** button simply closes and ends the procedure. The '**Run'** button will first validate that at least the year was selected and run the associated report (which has an underlying query and passes the parameters values from this form) before it closes too.

An embedded macro will be used

Here's the flowchart…



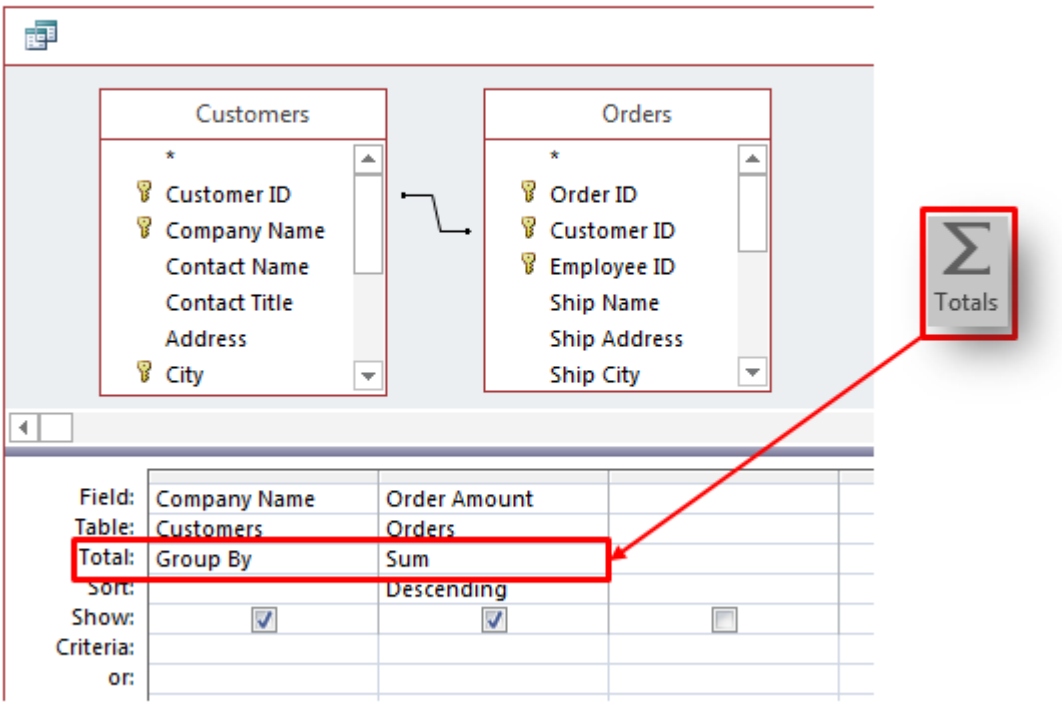
The following points are the recommended approach to the steps in building this solution:

1. Build a query and specify the fields, sorting and calculations but do not add any criteria yet.
2. Create a report based on the query and format layout and other options.
3. Create a form (*as a dialog box*) and set controls, values and formats in place. Make sure you take the time to name controls to help both the macro and the query later on.
4. Create two embedded macros (*Cancel and Run buttons; on click event*)
5. Modify query by adding criteria from the form controls (*year and country values*).

The Query

Create the following query making sure you test that query before saving it as '***qry_Revenue_By_Country_Year_Form***':

Make sure the two tables have a relationship and have applied the Sum function to the *[Order Amount]* field and Group By to the *[Company Name]* field



Access 2016 – New Query (Customer and Orders) – Groups and Total

We will return to this query later to add criteria. For now, save changes and keep open.

***Note**: Here's the SQL if you prefer this approach:*

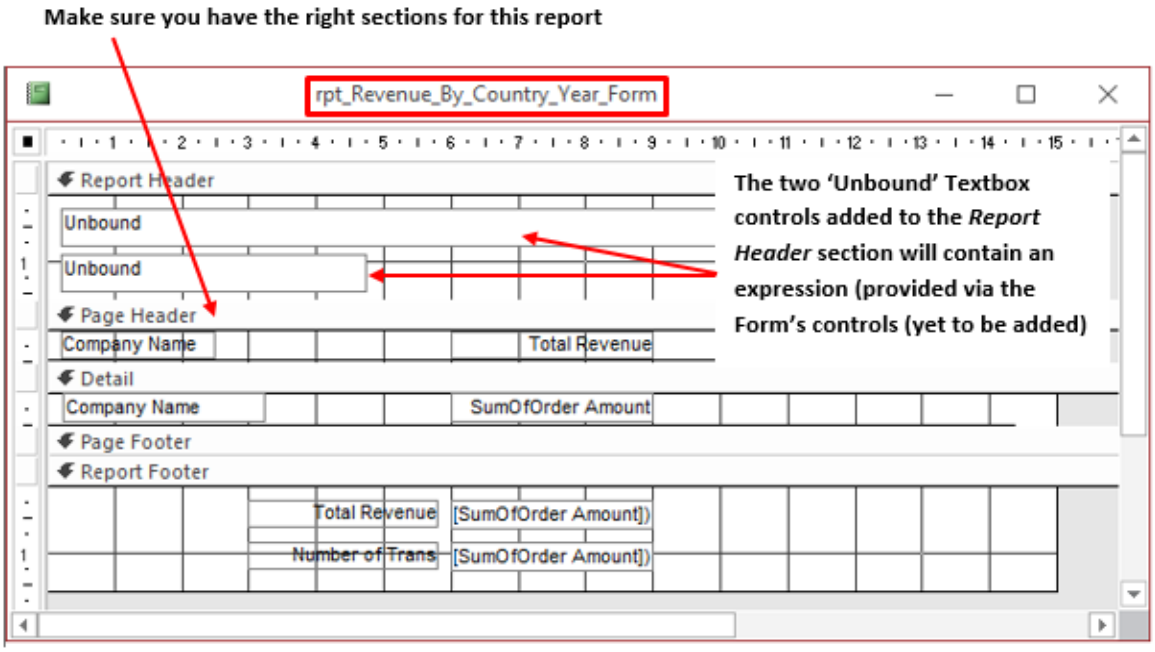
```
SELECT Customers.[Company Name], Sum(Orders.[Order Amount]) AS [SumOfOrder Amount]
FROM Customers INNER JOIN Orders ON Customers.[Customer ID] = Orders.[Customer ID]
GROUP BY Customers.[Company Name] ORDER BY Sum(Orders.[Order Amount]) DESC;
```

The Report

There are several ways to create an Access report and I'm going to start with a standard report template and make some alterations.

Make sure you base this report on the pre-defined query '***qry_Revenue_By_Country_Year_Form***' by either choosing this data source or by setting the Record Source property (*Property Sheet for the whole report*).

Continues/…



Access 2016 – New Report (based on query: *qry_Revenue_By_Country_Year_Form*)

It's not essential to have the exact type of report as long as you have the key fields; choose whether you wish to have summary totals, detail per company name etc. It's an example of connecting the '*dots*' here.

Make sure you have two *TextBox* controls (*unbound for now*) where they will be modified later to collect values from our yet to be created form.

You can test (*preview*) the report (*as it has no criteria*) to make sure it looks something like:

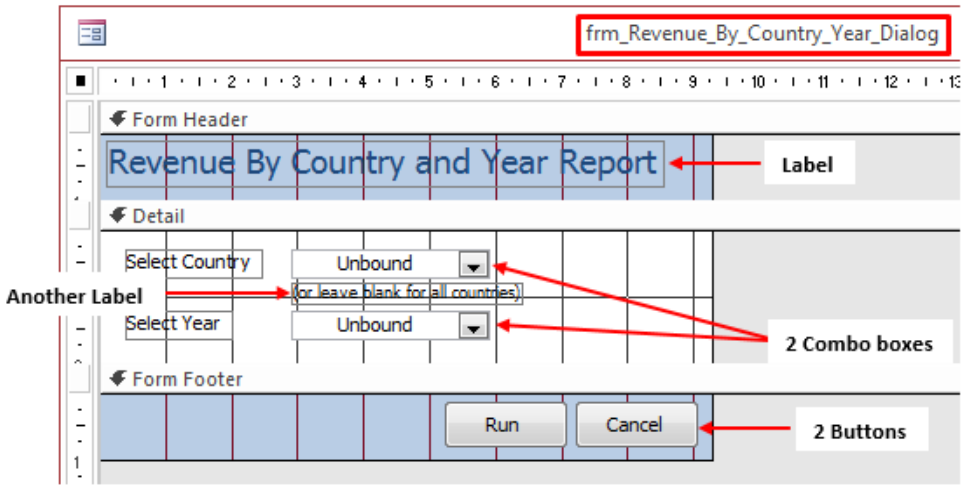
Company Name	Total Revenue
Around the Horn	£11,603.95
Babu Ji's Exports	£12,699.77
Blum's Goods	£4,715.16
B's Beverages	£3,401.00
Commoner's Exchang	£513.75
Consolidated Holdings	£931.50
Dunn's Holdings	£1,347.10
Special Delights	£2,423.35
Sugar and Spice	£11,672.60
Trade More	£5,364.66
Wellington Trading	£3,842.56
Wolverton House	£3,072.95
Total Revenue	£244,738.17
Number of Trans	24

Save changes and keep it open for now.

Continues/…

The Form

Next, create a form (*I'm starting with a blank canvas*) and add the following controls:



Access 2016 – New Form (Unbound – no data source but set as a dialog box)

It will help to name controls where you will either code as a reference or refer to other objects (as *in this report*) and I'm going to use the following:

- **Combo Box 1**: which has the caption '*Select Country*', name: **cbo_Country**
- **Combo Box 2**: which has the caption '*Select Year*', name: **cbo_Year**
- **Button 1:** which has the caption '*Run*', name: **cmd_Run**
- **Button 2:** which has the caption '*Cancel*', name: **cmd_Cancel**

All other controls can be optionally named but will not be used in reference anywhere else.

Next, you need to add values to each of the Combo Boxes to list all countries and years from the *'Customers'* and *'Orders'* tables respectively. To help make your lists more dynamic so that when additional countries and orders (by year) are added to the database and accommodate the new values, you add the following SQL statements to each of the control's **Row Source** property:

cbo_Country

```
SELECT DISTINCT Customers.Country FROM Customers ORDER BY
Customers.Country;
```

cbo_Year

```
SELECT DISTINCT Year([Order date]) AS [Year] FROM Orders;
```

Make sure other properties are correctly set for each of the above Combo Boxes and save changes (*still keeping it open*).

Embedded Macros

Next, we are going to create two embedded macros to the existing form, one for each **Button**.

The '*Cancel*' button will simply close the form and the macro ends.

The '*Run*' button will first validate that a year has been selected before opening our report (*which triggers the query*) and then closes the form.

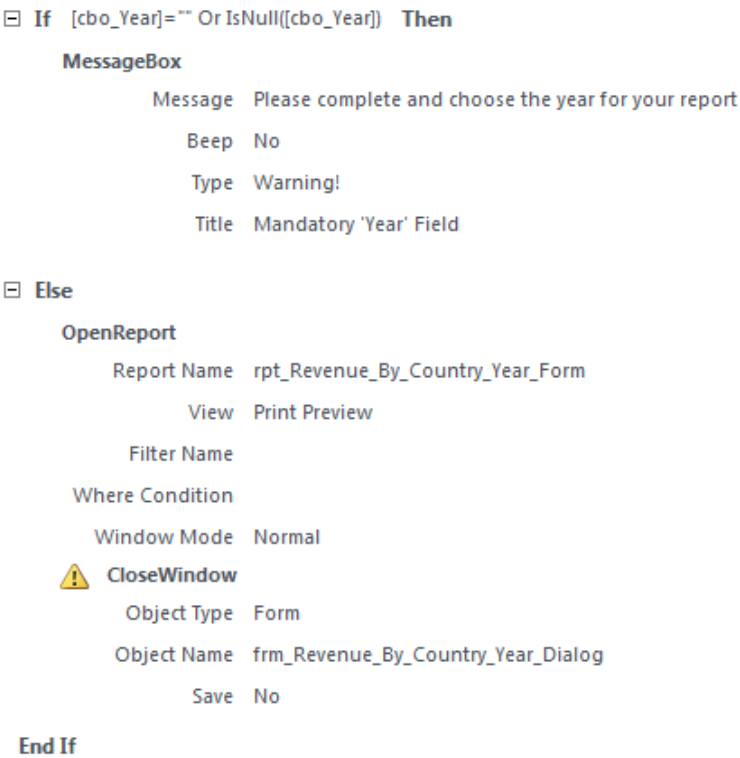
Click on **cmd_Close** and go to its *'On Click'* event and choose the **Macro Builder** option.

Add the following macro:

Make sure you explicitly set the object type to close a form and the correct name of the form, setting the save option to no (*no prompt*).

Close and save changes.

Next, click on **cmd_Run** and go to its *On Click* event and choose the **Macro Builder** option.

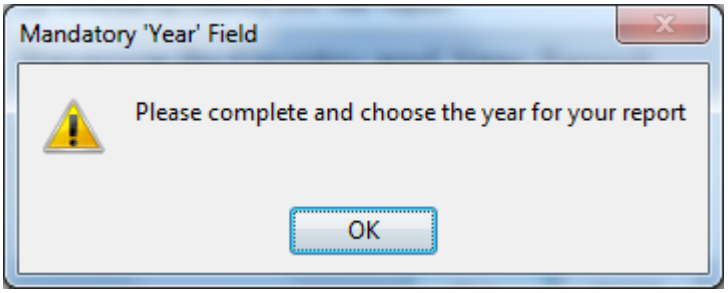


Continues/...

The first action is a validation to see if the user had selected a year from **cbo_Year** Combo Box control from the form. I have added two different expressions just to catch an empty value or a null value using the '*Or*' operator.

```
[cbo_Year]="" Or IsNull([cbo_Year])
```

If either is true then display the following message box:

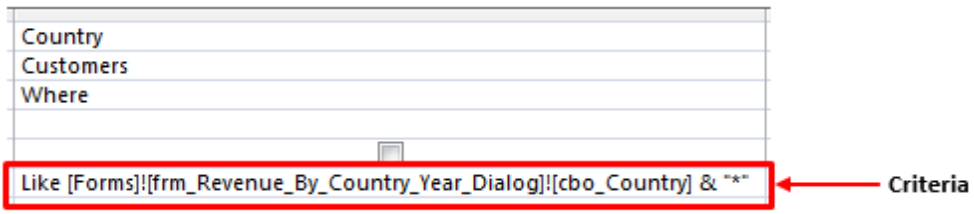


If false, then proceed and run the report. Once the report opens in print preview, the form is closed and the macro ends.

Don't test this part for now - we still have one more step! For now back in the design view of the form (*macros are closed*), save changes and keep the form open.

The final Steps

In the query, add a new field (*column*) and click into the criteria where we are going to set the country value that will be passed from the form's *cbo_Country* control.



Access 2016 – Query criteria (Country)

You can call the build expression tool in this area (CTRL + F2) and pick from the list of objects; the **cbo_Country** control or type as above.

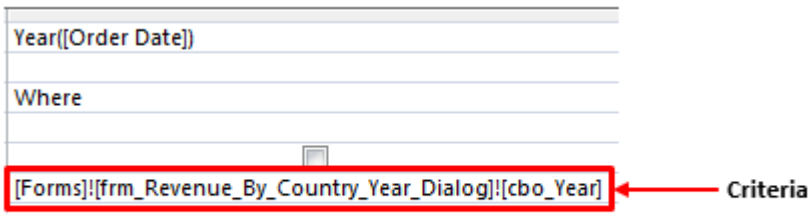
```
Like [Forms]![frm_Revenue_By_Country_Year_Dialog]![cbo_Country] & "*"
```

The reference to the form's value (*Country Combo Box*) show the hierarchy; ***[Forms]![Form Name]![Control Name].*** The ending allows a wild card to be appended (*) which means the criteria can be left empty for all countries (*with the operator* ***Like***).

Make sure you have set the **`Where`** clause from the ***Total*** row of this query.

Continues/…

The other criteria will be for the Year field:



Access 2016 – Query criteria (Year)

Make sure you have set the **`Where`** clause from the ***Total*** row of this query and that the field has the **Year** function wrapped around the *[Order Date]* (*as shown above*).

Close and save changes.

> ***Note**: Here's the final SQL if you prefer this approach:*
>
> ```
> SELECT Customers.[Company Name], Sum(Orders.[Order Amount]) AS [SumOfOrder
> Amount]FROM Customers INNER JOIN Orders ON Customers.[Customer ID] =
> Orders.[Customer ID]WHERE (((Customers.Country) Like
> Forms!frm_Revenue_By_Country_Year_Dialog!cbo_Country & "*") And (Year([Order
> Date])=Forms!frm_Revenue_By_Country_Year_Dialog!cbo_Year)) GROUP BY
> Customers.[Company Name] ORDER BY Sum(Orders.[Order Amount]) DESC;
> ```

Back in the design view for the report (*header*), we are going to add two references to the two *Unbound* controls.

The first control will return the following expression:

```
=IIf(IsNull([Forms]![frm_Revenue_By_Country_Year_Dialog]!
[cbo_Country]),"All Countries","Revenue for " &
[Forms]![frm_Revenue_By_Country_Year_Dialog]![cbo_Country])
```

The second control will return the following expression:

```
="Year: " & [Forms]![frm_Revenue_By_Country_Year_Dialog]![cbo_Year]
```

> ***Note**: I'll let you research the above two expressions and to understand how it works!*

Close and save changes to all three objects.

Test it out!

Auto Macros

Judging by the term *'Auto'*, there are two pre-defined named macro objects that apply to all versions that execute when the application (*database file*) starts.

There are:

1. **AutoExec**
2. **AutoKeys**

You must use the above names only to bind and create the event in question when the applications starts up.

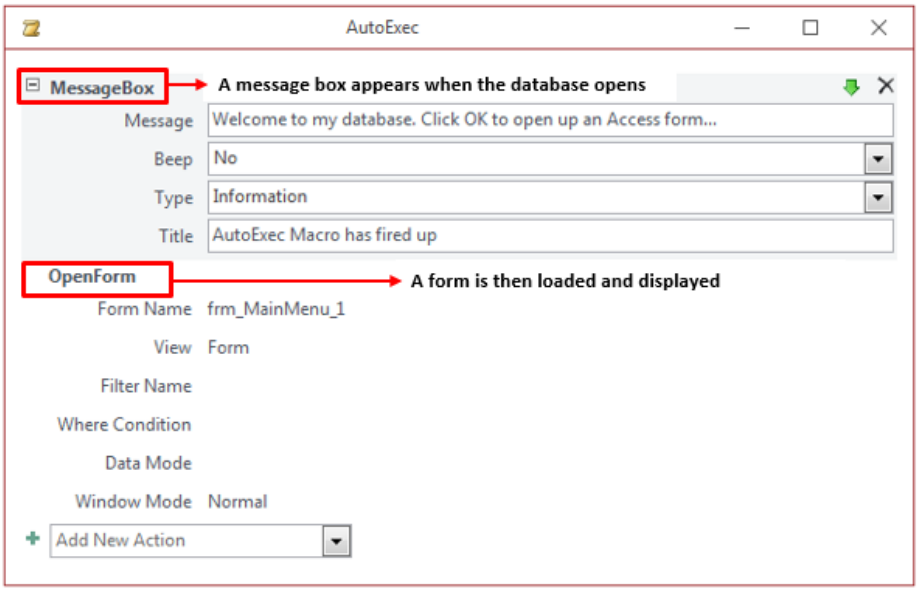
AutoExec

This macro will automatically execute when you open the database file (*accdb*) and runs event before the *StartUp* switch in **Access Options**.

Historically, this macro has been around way before the *StartUp* switch was added to later and current versions but it still can be used as an alternative way to bind procedures. Today, they both can be used together as the order they execute is *AutoExec* then *StartUp*.

You may want to use the *AutoExec* macro to run pre-processing code which may include initialising a connection to external tables (*other source files*), importing data, setting initials values and running VBA code for more complex procedures. The *StartUp* switch could be used to open the starting form ready for use.

Here's how to create the macro.



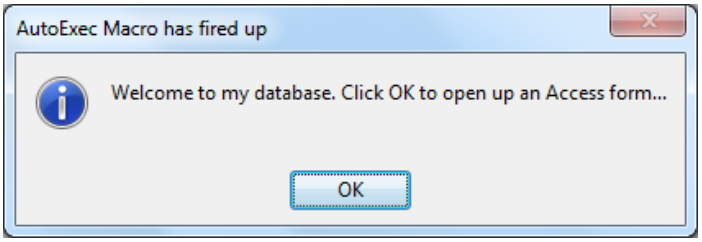
Access 2016 – AutoExec macro

Continues/...

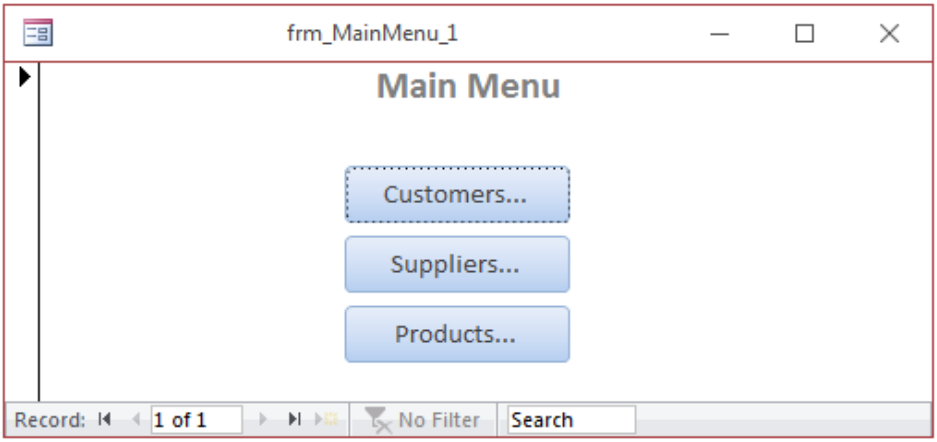
Create a new macro object and set your actions and their arguments. Save the macro as **AutoExec** (*do not be creative and name it anything you like - this is a reserved keyword*).

Close the database file and re-open (*or just run the **Compact & Repair** tool*).

The *AutoExec* macro fires up when the database file loads with the **`MessageBox`** action being called first:



Click the OK button and the form (*frm_MainMenu_1*) opens with the **`OpenForm`** action:



You could of course bind this form to the *StartUp* switch instead but this example is simplified and both could be used together.

A personal note:
I have used the *AutoExec* macro object with the *StartUp* switch combined splitting processes down into logical (*and sometimes business workflows*) units. For instance, the AutoExec macro would execute pre-processing routines which include establishing links with external files, importing data sets, initialising global variables and running more complex VBA code.

I would then bind a start-up Access form using the *StartUp* switch to load and welcome users to the database (*menu screen*).

Even though the starting form has a load event which can trigger an embedded macro, the *AutoExec* macro ran first and only once at the time the database started and not for each instance the Access form is re-opened.

AutoKeys

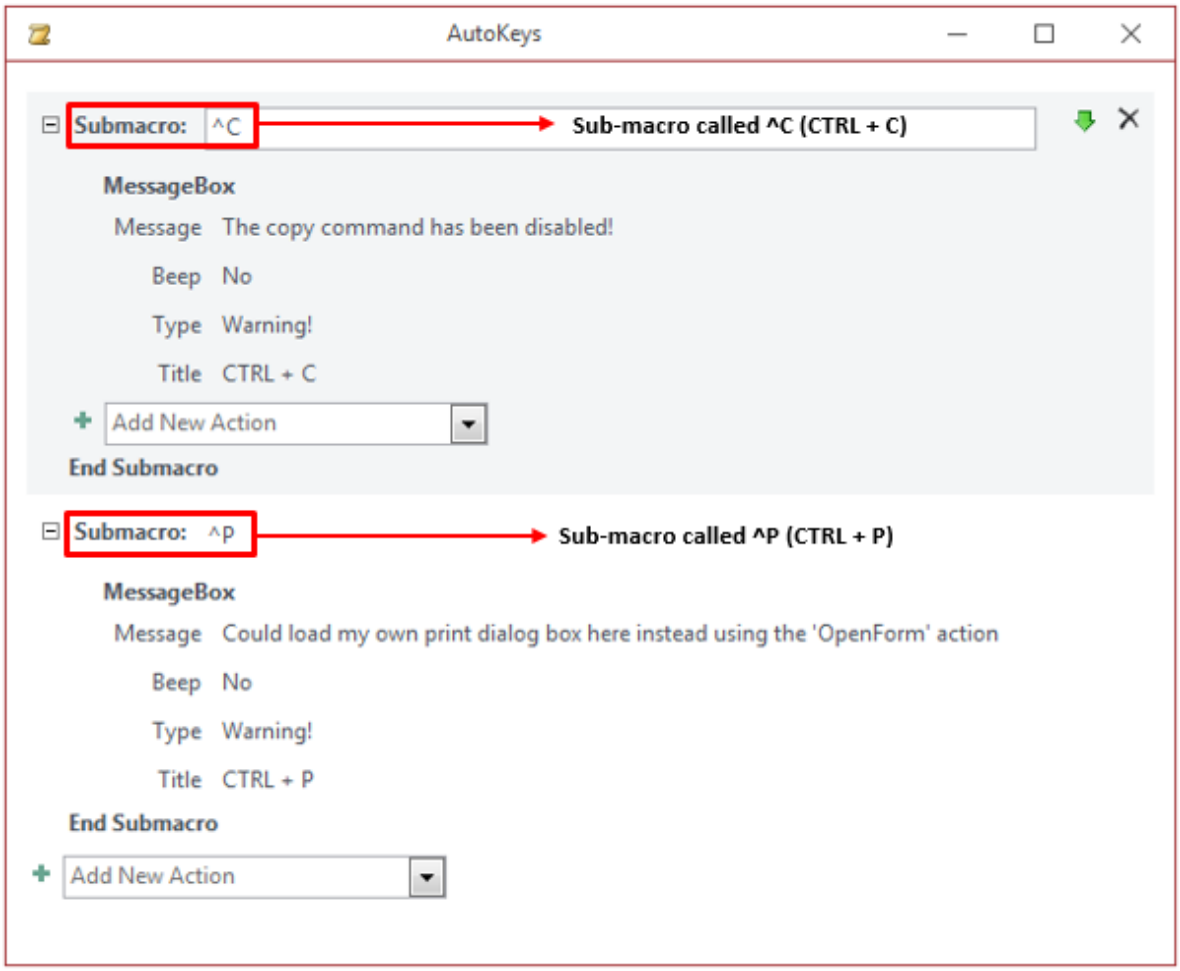
This macro is also triggered when the application (*database file*) is opened. It's used to control keyboard shortcuts within Microsoft Access that users take control of.

For example, the print command could be disabled via the keyboard shortcut (CTRL + P) and you can place your own customised print dialog (*Access form*) instead.

Here's how to create the macro.

Create a new macro and save it as **AutoKeys**.

Using the **`Submacro`** action, set the following example



Access 2016 – AutoKeys macro

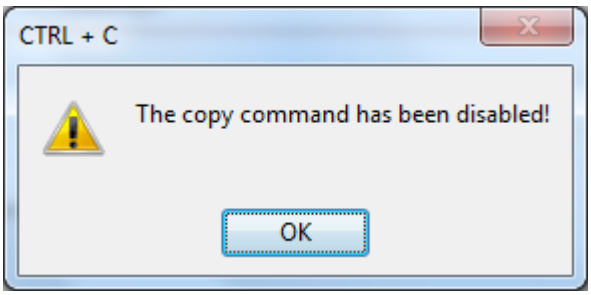
Using a sub-macro for each keyboard shortcut is important otherwise only one keystroke could be stored for this macro.

Close the database file and re-open (*I just carry out a* ***Compact & Repair***).

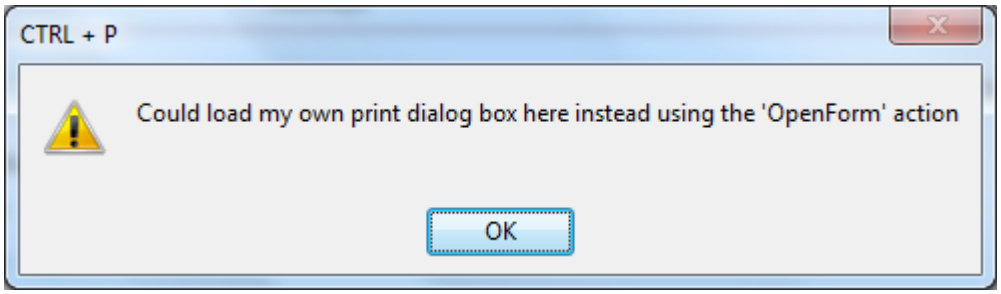
Continues/…

Reserved characters combined with the keys create a link to the shortcuts. For example, when I call either the following keyboard shortcuts, the following happens:

CTRL + C (*which is the copy command in Access and of course most Window applications*):



CTRL + P (*which is the print command in Access and of course most Window applications*):



To get a feel for how you can control keyboard shortcuts and accelerators, here's a list of some examples:

- {F1} calls the **F1** function key
- ^S calls the **Ctrl + S**
- +P calls the **Shift + P**
- +^O calls the **Ctrl + Shift + O**
- +{F4} calls the **Shift + F2**
- +^{F4} calls the **Ctrl + Shift + F4**

The accelerators:

^ = CTRL

+ = SHIFT

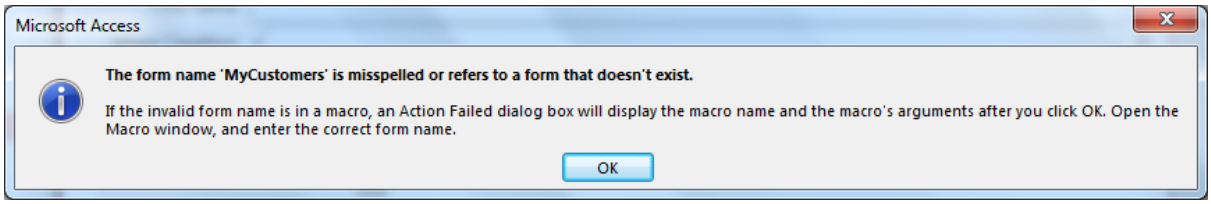
The **ALT** is not available for AutoKeys names and you will need VBA to handle this accelerator instead.

Debugging Macros

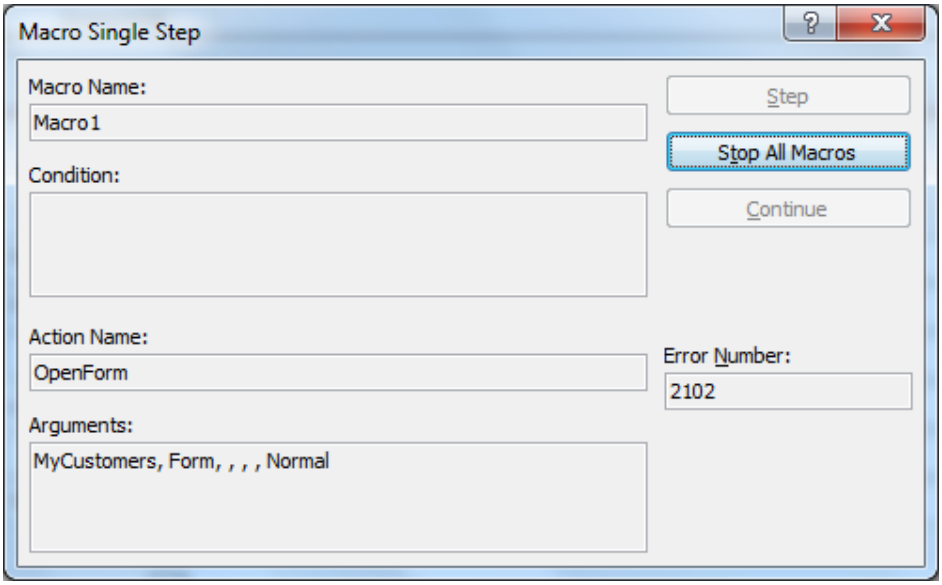
You may have noticed when an error occurs in Microsoft Access (*macros*) that screens will appear to help inform and give a clue to what the issue could be.

However, this makes your application ugly! Un-user-friendly and interrupts the flow of your application.

In some cases, Microsoft Access displays a warning to tell the user as much as possible (*if it can*) about the macro call that's failed. For example, the following screen appeared due to the calling form of '*MyCustomers*' not being found in the database:



This is a standard Access prompt and would be acceptable practice if loaded by the user from the Navigation pane (*Database Window*). However, if this was part of a macro, the following screen appears:



This is not what users should see and is the knock-on effect of a macro error. The only option is to click the **Stop All Macros** button.

This error tells the operator user the name of the macro, any conditions where applicable, the action that caused offence along with its argument values. There is an error number (*in this case,* **2102**) which could help developers. Other than that, there are little benefits for the end-user.

Continues/…

To debug any errors, you will need to step through the actions one line at a time and view and investigate the argument values and determine how to resolve them.

There are a couple of ways to do this:

1. Use and switch on the **Single Step** feature for a macro.
2. Use the **`SingleStep`** action in your macro.

> ***Note**: Depending on which version of Microsoft Access you use you may only have one option! Pre Access 2007 does not have the action* **SingleStep** *available to add into a macro and therefore you can only utilise the first option (**Single Step** icon).*

Single Step Icon

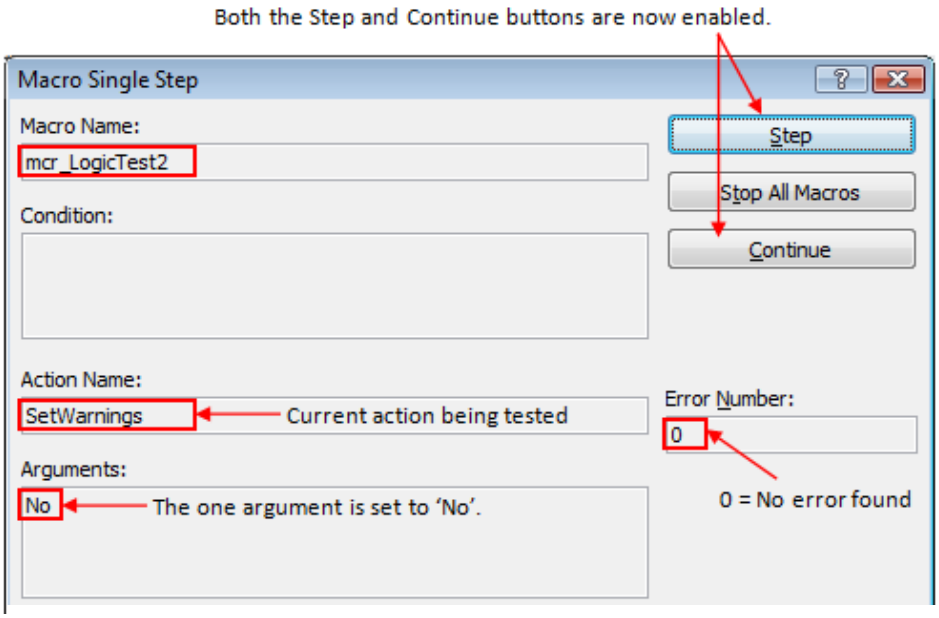
All modern versions of Access will have the **Single Step** icon tool available, which must first be switched on before running a macro.

I'm going to use the macro mcr_LogicTest2 (created earlier).

Open the design view of the macro to test and step into.

Locate the **Single Step** icon from the Ribbon Bar.

Either close or run the macro and the following screen (similar) is shown:

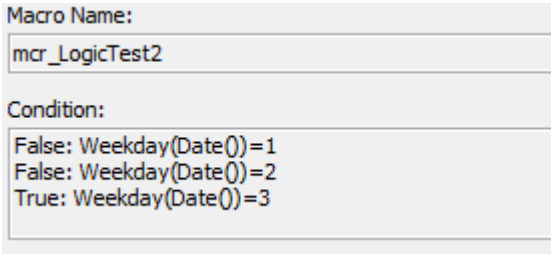


Access 2010 – Macro Debug Error screen

Click the **Step** button to take you into the next action (*line*) which in this case calls the first of several logical tests using the **`If`** action.

Continues/…

I have clicked this two more times where the third one is evaluated as **True**:



If I continue to click the **Step** button it will take me through to the end of the macro or if I had clicked the **Continue** button it will run at normal speed and complete the macro too – there were no errors in this example.

At any time, you can stop the macro by clicking the **Stop All Macros** button.

Also be aware that when you click the **Continue** button it automatically switches off the **Single Step** feature and you will need to re-enable it again.

***Note**: When you apply the **Single Step** option, it applies to all macros and there is no distinction/option to set per macro.*

SingleStep Action

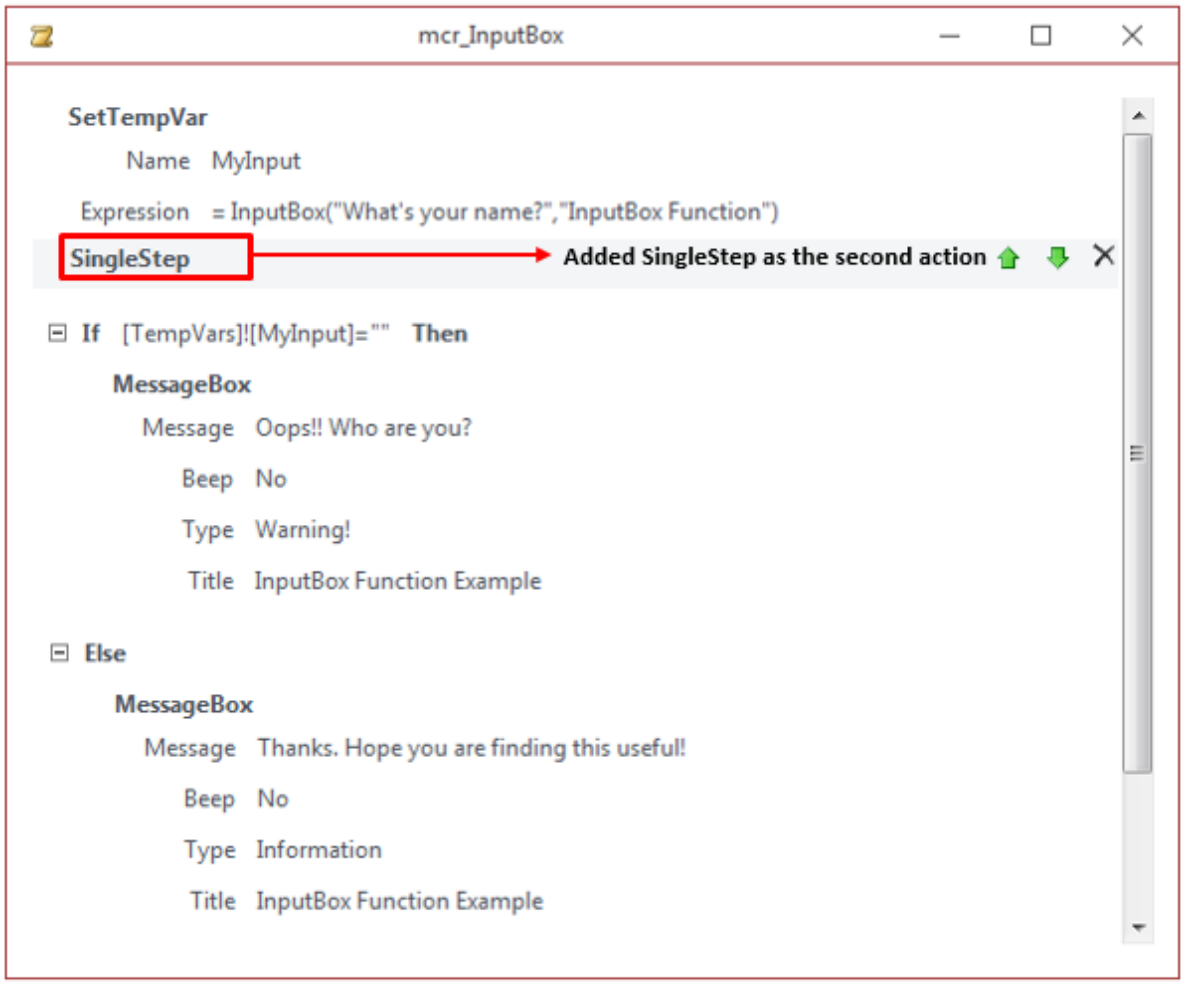
This is an action that can be added into a macro which has the following benefits:

1. It can be applied to a single macro and not for all macros (*as mentioned with the* **Single Step** *icon*).
2. For longer and larger macros, you don't have to step into a macro from the beginning (*again using the* **Single Step** *icon*) you just simply mark where you want to pause the macro.

The **`SingleStep`** action has no arguments; it is simply added where you need it and the macro is paused showing the same debugging screen as before.

Continues/…

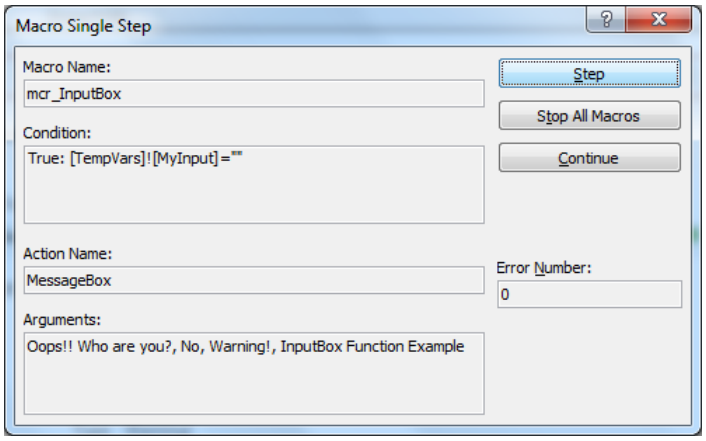
I'm going to open the macro ***mcr_InputBox*** (*created earlier*) and add this action as follows:



Access 2016 –mcr_InputBox macro – SingleStep Action

Save the macro and run at normal speed.

You have now paused the macro (*still running*) just after the Input box was called and you either enter something or leave it blank allowing you step through the code.



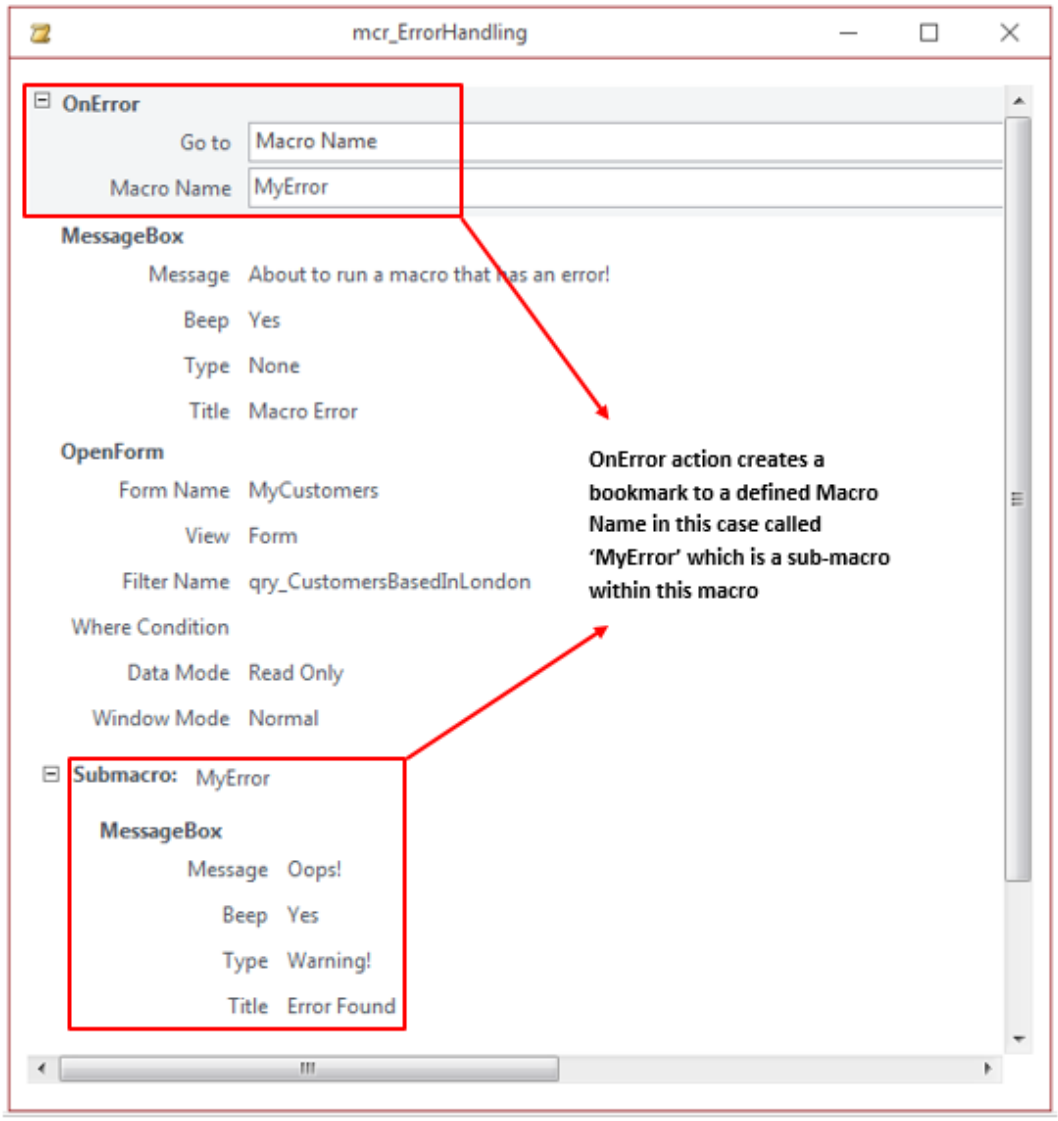
Handling Errors

Seeing an error screen is too late to catch workflows and to avoid this screen altogether, you need to trap for this in the first place and display a user-friendlier way to handle any unforeseen errors.

Now, macros are not ideal when it comes to error handling though with the later versions (*Access 2007 to current version*) you now have better functionality to handle some of the errors gracefully within your macro routines.

The **`OnError`** action is added to your macro and *listens* for an error to occur.

Here's an example of how to use this type of action where I will first create the **`OnError`** action followed by a **`MessageBox`** and the **`OpenForm`** actions.



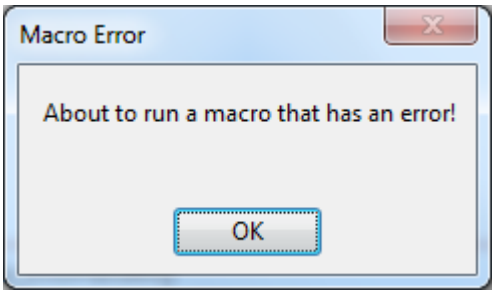
Access 2016 –mcr_ErrorHandling macro example

Continues/...

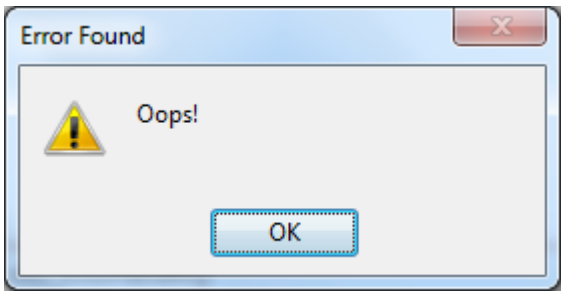
Close and save the macro.

When you run this example the first thing it does is to create an error handler (*or listener*) for any errors that are triggered during the macro routine.

A message box appears to inform the user that this macro is now running and it's a deliberate error macro example for the purposes of illustrating and highlighting this point:



Click the 'OK' button to continue and the next response is the attempt to open a form called '*MyCustomers*' which doesn't exist and triggers the sub-macro called **MyError** (*bookmarked from* **`OnError`**).

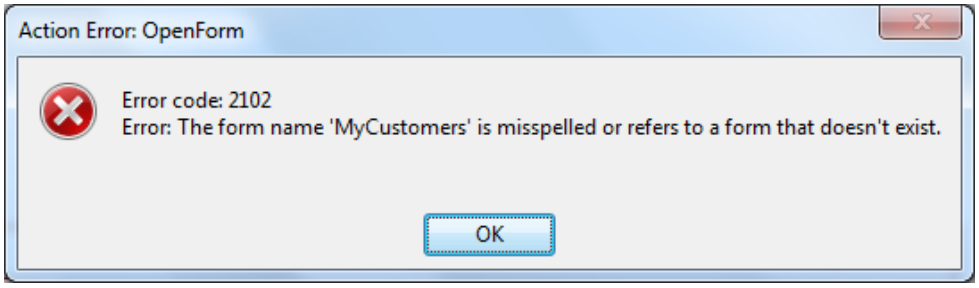


The macro ends.

If there wasn't an error and the form did exist then no error message is displayed and the form opens instead.

You can also create dynamic messages using the **`MsgBox`** or **`MessageBox`** actions.

Take a look at an example below of a dynamic message box:

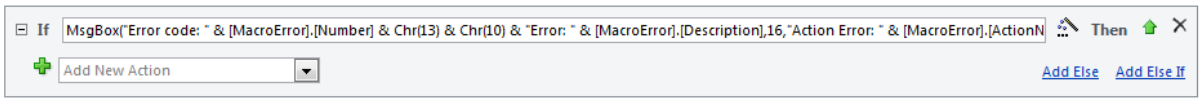


Continues/…

Notice the title says which action caused the error (*OpenForm*). See the error code it generated (**2102**) and the description of the error narrative below.

The only way to produce this type of dynamic error (*other than use VBA*) is to use the **`MsgBox`** function (*covered earlier*) or set an expression for the **`MessageBox`** action. I've opted to use the **`If`** action to respond to this call but not actually test the logic further (*I could have used any action or variable instead*).

See the example below:



I removed the standard **`MessageBox`** inside the **`SubMacro`** action and replaced it with the above illustration.

In addition, I used the **[MacroError]** construct with the following three properties:

1. **[Number]** – returns the error code.
2. **[Description]** – returns the narrative of the error generated.
3. **[ActionName]** – returns the action that triggered the error.

When using a constructor, it outputs the value and therefore to call the error code you would use the syntax **[MacroError].[Number]**.

Here's the complete **`MsgBox`** call including adding a new line (**Chr(10) & Chr(13)**):

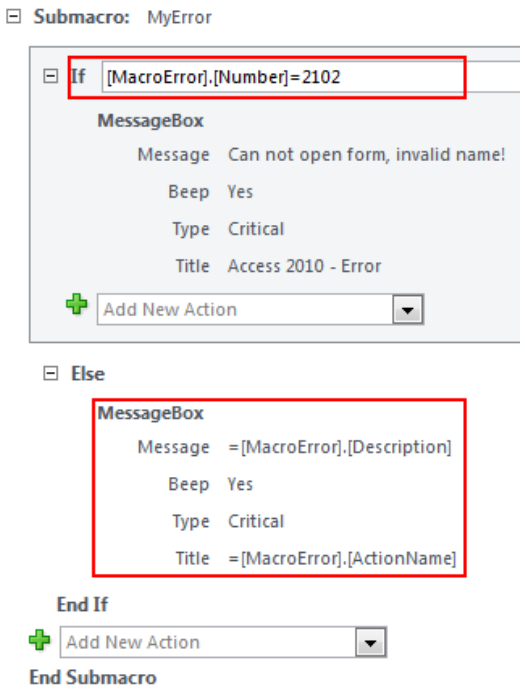
```
MsgBox("Error code: " & [MacroError].[Number] & Chr(13) & Chr(10)
& "Error: " & [MacroError].[Description],16,"Action Error: "
& [MacroError].[ActionName])
```

(All on one line)

Continues/…

Of course you can just use an **`If`** action to test for a type of macro error (*by its code number*) and have different customisable messages instead.

Here's an example using the **`MessageBox`** action with a defined fixed message for an unknown form or a generic message for all other errors using the **=[MacroName].[Description]** property for the message itself.



Hopefully, you should now have the confidence to automate your database application using macros Spend the time learning this topic well and it will pay dividends in the long term.

DOWNLOAD EXAMPLE FILES FOR THIS USER GUIDE

If you would like a copy of some of the macro examples and other objects for this user guide, please go to:

Microsoft Access Macro Example File - Zip File
(http://benbeitler.com/Macros%20Examples%202016.zip)

DISCLAIMER OF WARRANTIES/DAMAGES
All software data files is provided "as-is," without any express or implied warranty.
In no event shall the author be held liable for any damages arising from the use of this software.
The user must assume the entire risk of using the software.

PLEASE DO NOT DOWNLOAD UNLESS YOU HAVE AGREED TO THIS DISCLAIMER.

Administration Tools

Introduction

In this section, we will look at some of more popular Access features and tools that will assist you in managing and configuring your database and include:

1. **Backup and Restore** - a simple feature to maintain archives and reinstate data and their objects.

2. **The Compact & Repair** tool - will help you to optimise your database for best performance and resolve any inconsistencies.

3. **Protecting & Encrypting** your data - looking at how to add a basic password- and encrypt your database to prevent others from opening it either with Access or with another tool.

4. **Database Documenter** tool- that will enable you to use the tool to produce printed documentation of your database design.

5. **The Analyze Performance** tool - database tool will give you advice and tips on improving your database.

6. **The Analyze Table** tool - you use the *Table Analyzer Wizard* to restructure a table into a more consistent format for use in a relational database.

7. **Object Dependencies** tool - Inside Access is a map of how all your tables, queries, forms, and reports are dependent on each other. The Object Dependencies feature allows you to examine this map and quickly locate dependent objects.

All the above are not essential to your database application but it's recommended you get to know some or all of the above as they can keep your database clean, tight and allow the database to perform well.

If you intend to hand over your hard earned efforts, having back-end technical documentation and a recommended user guide should include some of the above too.

Backup and Restore

Backing up your Access 2016 database is an important step to preserve the integrity and availability of your data The process of backing up an Access database is simple step-by-step procedure (and may seem obvious).

Protecting the integrity of information stored in a database is one of the most important responsibilities of a database administrator. Making regular database backups will pay dividends as you know without taking this quick and simple action and back up, *you will certainly need to recall your data!*

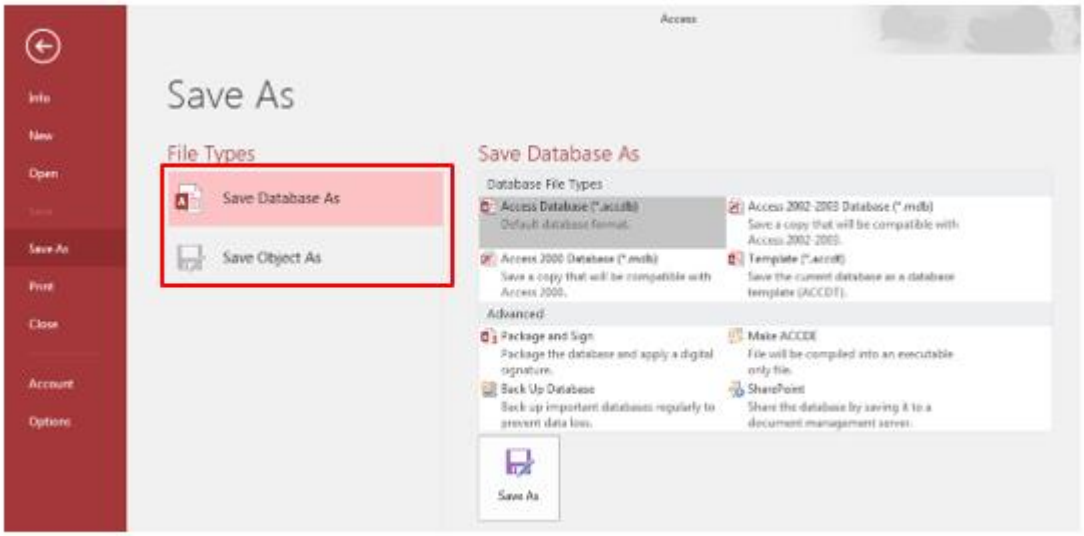
It is important to remember that Microsoft Access backups take place on a database by database basis. You will need to repeat these steps for every database that you use.

Backing up one database will not back up other databases that you may have stored on the same system.

Backup

1. Make sure your database is open and that all the objects are all closed showing just the Navigation pane.

2. Click the **File** tab and choose **Save As** (as shown):

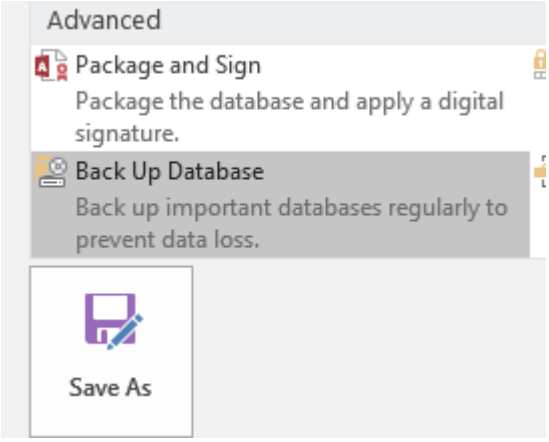


Access 2016 – File tab, Backstage, Save As

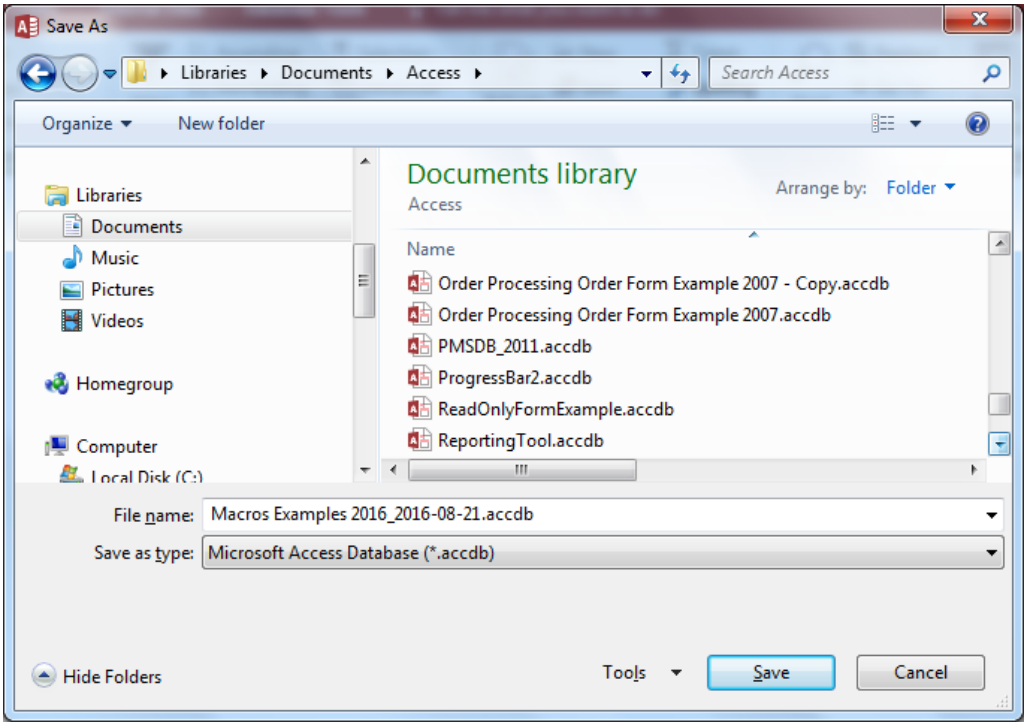
3. Decide which of the two actions to call. **Save Database As** will save all the objects in the current database and **Save Object As** will only be available if you had left one of the objects opened being the current object to save as

Continues/…

4. In The 'Advanced' section, choose the **Back Up Database** option and click the **Save As** icon.



5. You will see the standard save as dialog box for the current file format (.accdb).



6. Navigate to your desired folder location and click **Save**.

You may have noticed the save as type is the '*accdb*' file extension and not '*mdb*' *(or any other file format)*. You will need to manually choose the **Save Database As** type (*upper half section*) should you wish to demote to an earlier version and all objects must be closed before you can successfully downgrade your database.

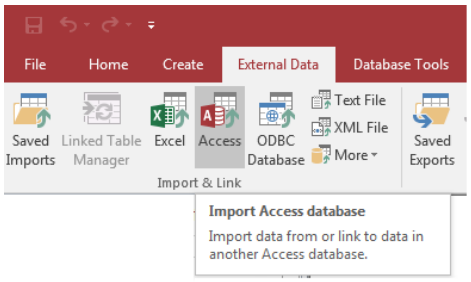
Restore

Ready for the complicated steps here? *There's no official tool to restore your Access database!*

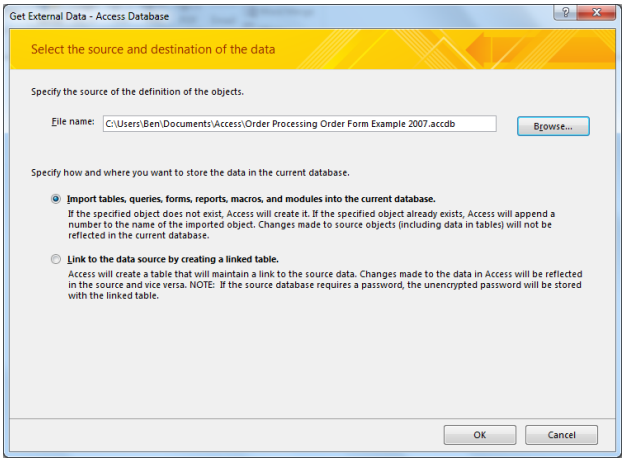
Since you have backed up your database as a new separate '*accdb*' file, it's just another file and that can be copied over the old file in the same location.

It's treated like any other file you want to manually restore. Just make sure the old database file is first closed so it releases the '*laccdb*' locking file.

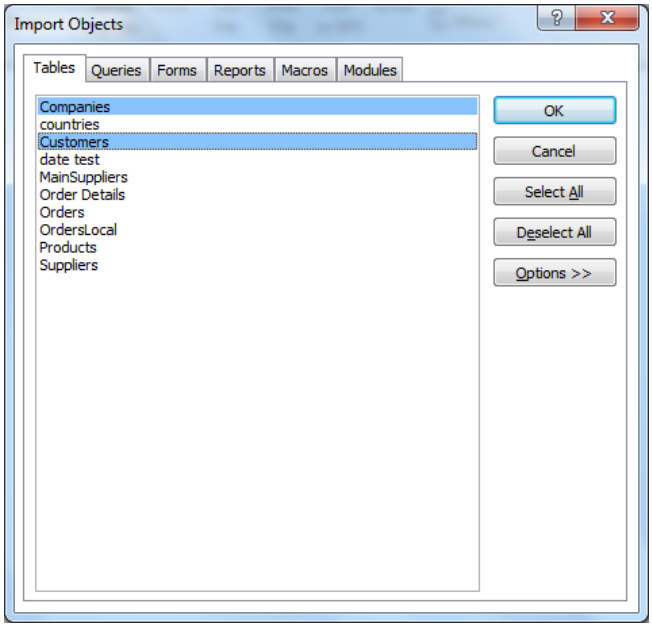
If you want to restore an object from your backed up database file, you import the one or more objects using the **Access** icon from the '*Import& Link*' section via the **External Data** tab.



You will see:



Follow the simple steps and choose your object(s) across one or more tabs.



The Compact & Repair Tool

As changes are made to your data over a period of time, such as when data is deleted or updated, the data will eventually no longer be ordered via the physical storage (hard-drive) in the most efficient manner. Over time, as objects are created and removed, the database will also grow in size; artificially inflating your database size.

Compacting the database reduces the size of the files (*squeezing the air out*) and makes the database operate faster by reorganising the physical data.

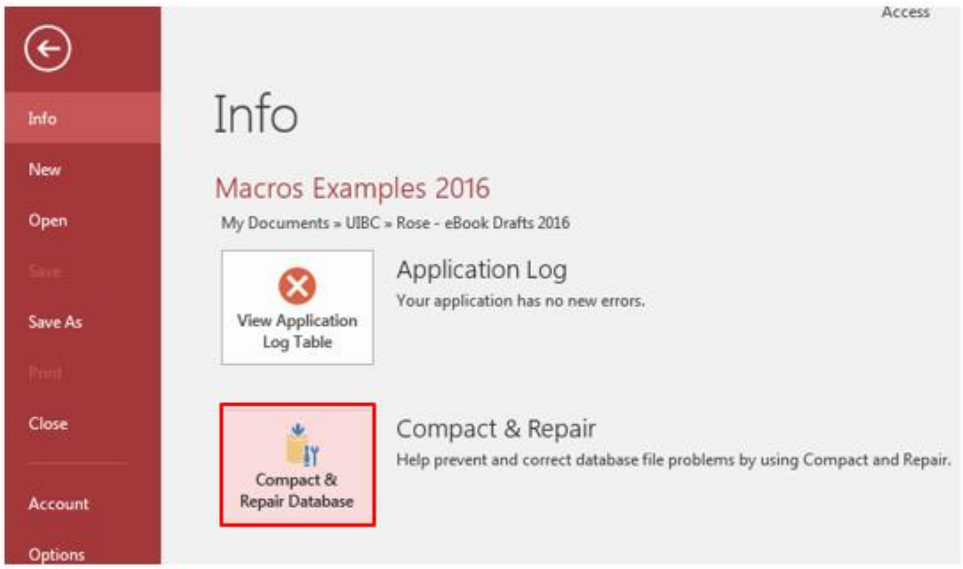
If you are sharing an Access database (*not covered in this guide*) AND Before you can compact or repair your shared database, you must ensure that no one is using it and that ideally it's open in '*Exclusive'* mode (*covered in Protecting your data later in this guide*).

The repair operation corrects for any problems in the consistency of the data or indexes in the database. A single process is used to both compact and repair a database and releases any locking files if they were not released during the normal close action of your database.

Compact & Repair

1. Make sure your database is opened and all objects are saved and closed first. Ideally, you need to be the only person with the database opened (*if being shared*).

2. On the File tab and into the Backstage under the Info command, you will the

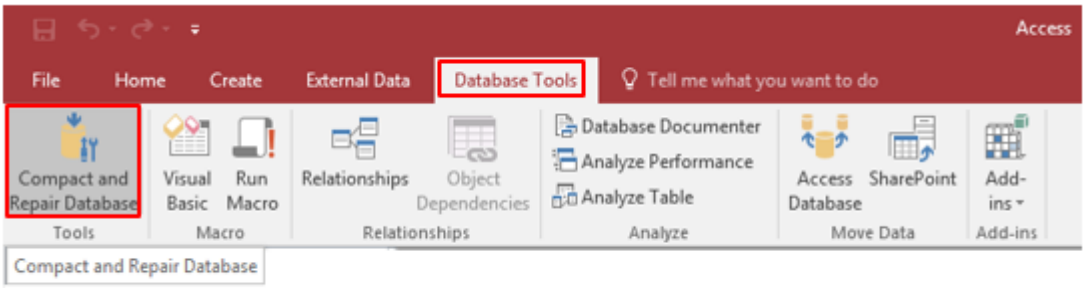


Access 2016 – File tab, Backstage, Compact & Repair Database

3. Click the icon shown above and job done!

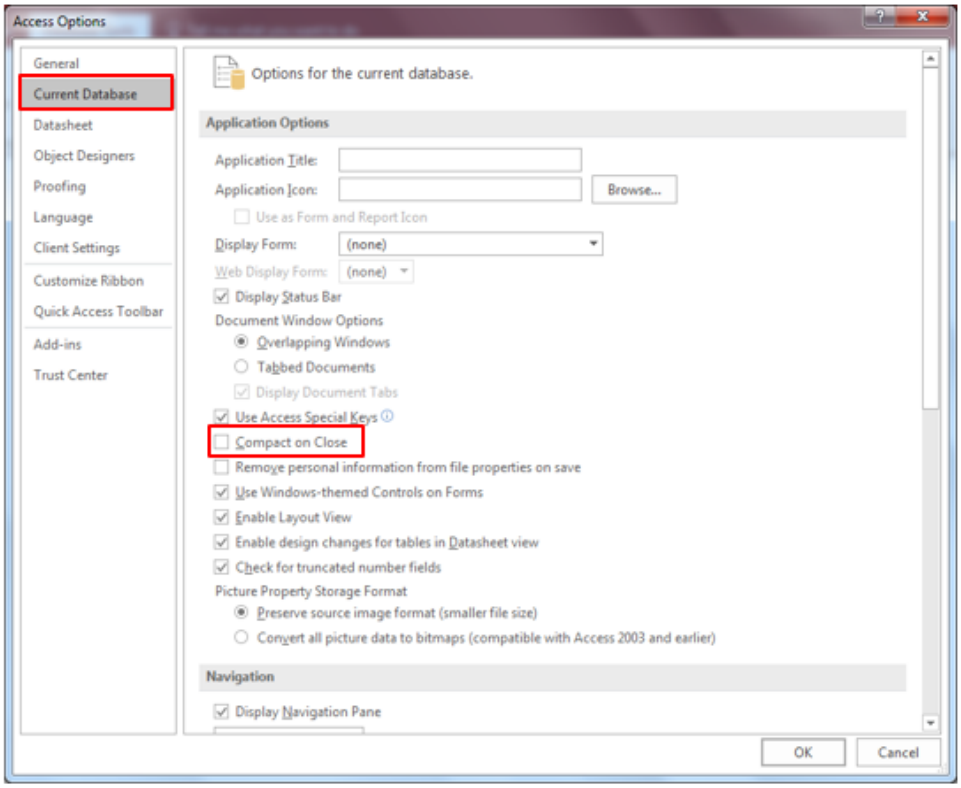
Continues/…

You can also call the same command via the **Database Tools** tab inside Access.



Access 2016 – Database Tools tab, Compact & Repair Database

If you want Access to automatically compact your database when close your file, you can set the option via Access **Options** in the **Backstage**.



Access 2016 – Access Options, Compact on Close option

Just remember two points about this. Firstly, this should only be used if this is not being shared and secondly, this will increase the general overhead to your application and has been known to take a while to close for larger databases.

Finally, you can even create a desktop shortcut with an added switch command at the end of *Target* properties file location and filename path using **/Compact** to run this as the database open.

Protecting and Encrypting your data

A database can be protected with a single and simple password. This prevents a low level to unauthorised users from successfully opening the database.

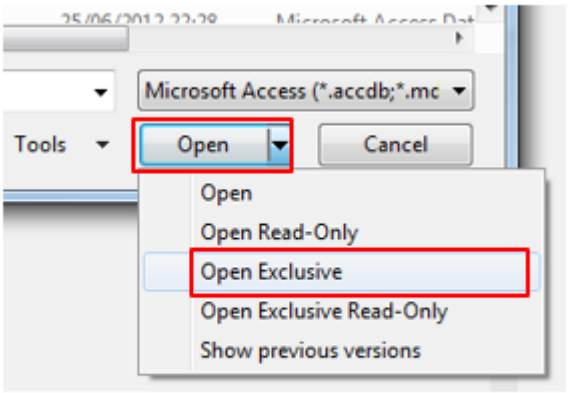
As part of the password protection process, the database is encrypted to prevent other software tools from being used to examine the data.

To add or remove a password, you must access the database by using a special sequence of steps that will open the database with the **Open Exclusive** option.

Protect a database

1. With the database closed, go to the File tab and Click the **Open** command. Do not use the recently used bookmarks to re-open a database, you actually need to see the Open dialog box.

2. In the Open dialog box, choose from the '**Open'** command the drop-down option and select **Open Exclusive**.

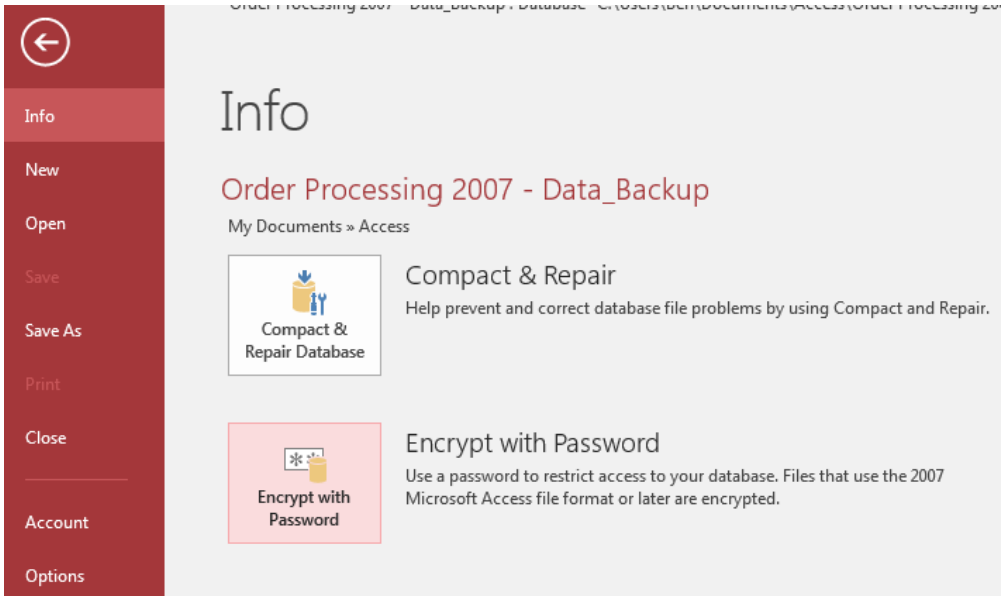


Access 2016 – part of the Open Dialog box, Open drop-down option

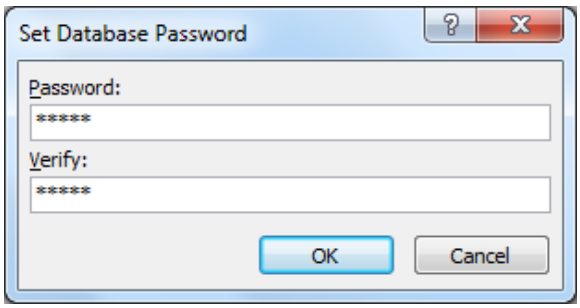
3. You will be taken to your database in the normal. Close any auto loading objects (*if you had set any*).

Continues/…

4. Return to the **File** tab and the **Backstage** and choose the **Encrypt with Password** icon from the '*Info'* section.



5. Enter your matching password and click **OK**.



6. You will see a prompt about record locking which you can ignore by clicking **OK**.

When you next re-open the database (in the normal way), you will be first prompted for this password.

To remove the password, you will need to first re-open the database in exclusive mode and then from the '*Info'* section of the **Backstage**, you will see instead the icon **Decrypt Database** where you can complete the steps.

This a very basic protecting tool and ideally you may want a more sophisticated version that Access doesn't provide.

Instead, take a look at my utility offer: http://accessdatabasetutorial.com/offer/

Database Documenter Tool

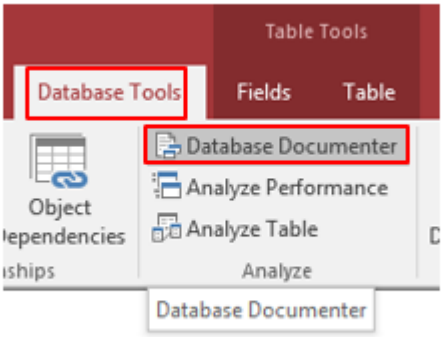
The Database Documenter tool provides you with a basic level of documentation for your Access database.

This will save you time having to manually compile technical details of your database for others to follow and support especially if you are handing over responsibilities.

Run The Documenter

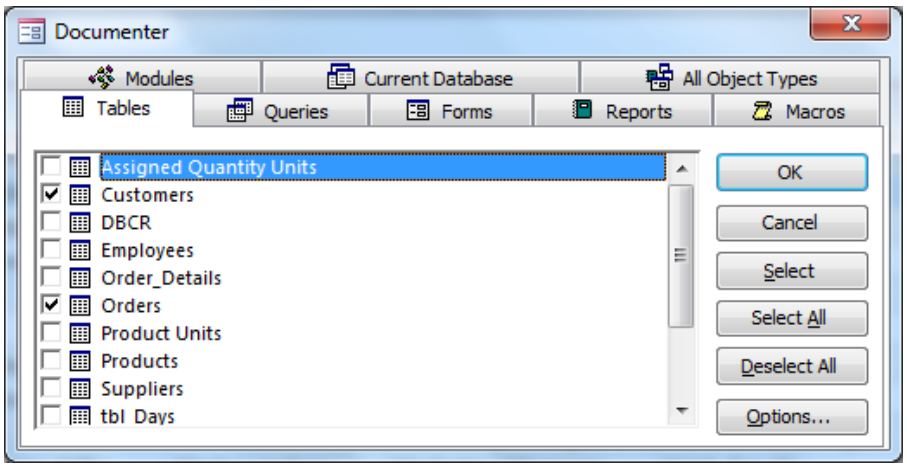
Ideally, ensure tyou close off all objects first though it's not necessary as you are prompted at the end before as the report is generated which is part of this routine.

1. Under the **Database Tools** tab and in the '*Analyze*' section, choose the **Database Documenter** icon.



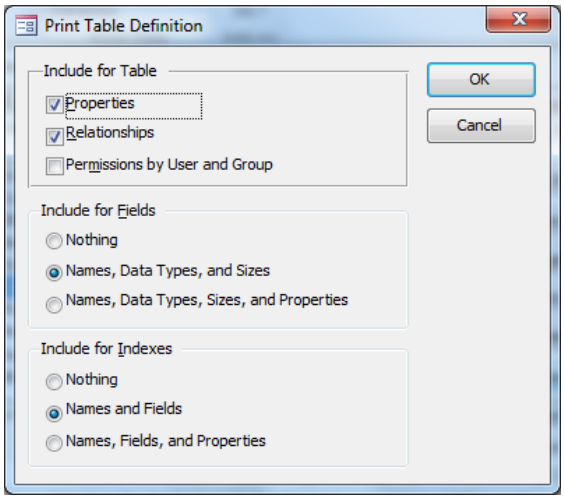
Access 2016 – Database Tools tab, Database Documenter icon

2. Choose one or more objects across the several tabs



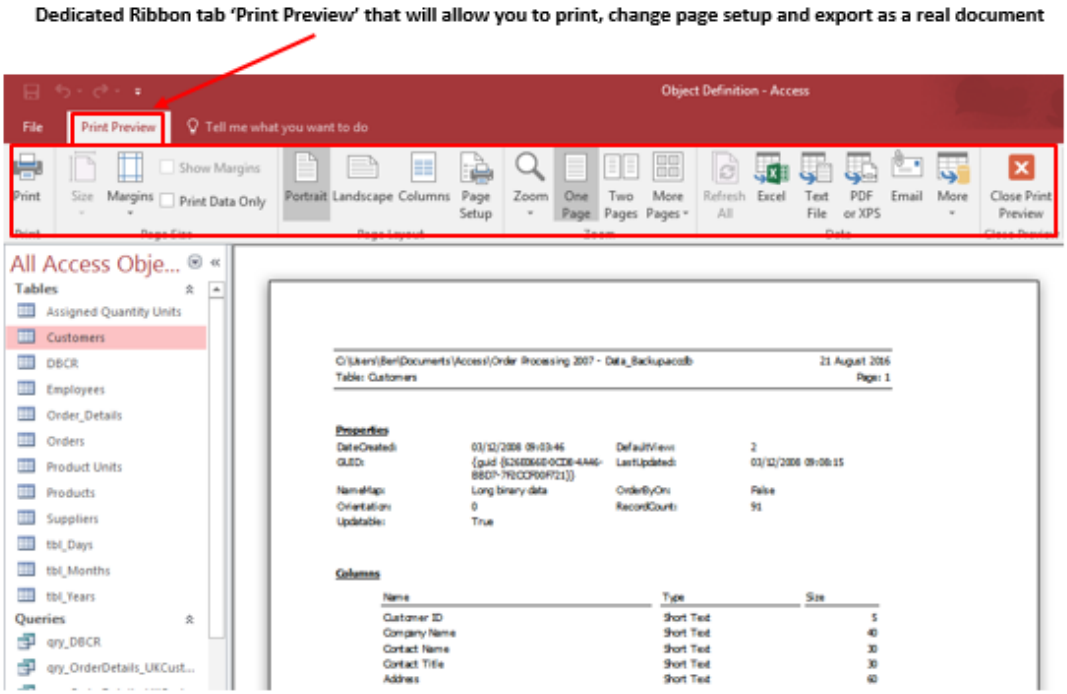
Continues/...

3. Optionally, you can click on the **Options…** button which will load the following screen:



The more you choose, the larger (number of pages) the report for each object selected. If you have a basic database with no relationships and little indexing, then switch off '*Relationships'* and demote Indexes to '*Nothing'*.

4. Click **OK** (and **OK)** again to generate a report preview in Access.



Access 2016 – Database Documenter report example

Continues/…

You can optionally export this report to Microsoft Word and save it as a separate document but be aware the larger the database, the more objects selected with the additional options, the larger the number of pages your report will be. Expect to see over 100 pages for a medium sized database!

This is a static report and can be generated as many times as required and is normally carried out of the end of the design phase cycle.

***Note**: There's an option that included 'Permissions by User and Group' via the Options dialog box which referred to an earlier version of Access (2003) and is not supported to this current version.*

The Analyze Performance Tool

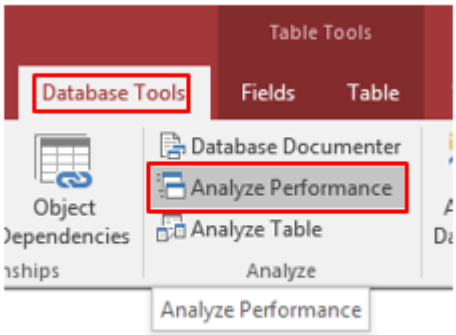
The Analyze Performance tool looks at the structure of your database and provides recommendations for any revisions that you can make to the design of the database

It's a checker tool that can provide some useful information. However, if you've been following my guide (*previous 400 pages*), *you shouldn't need it!*

Analyze Performance

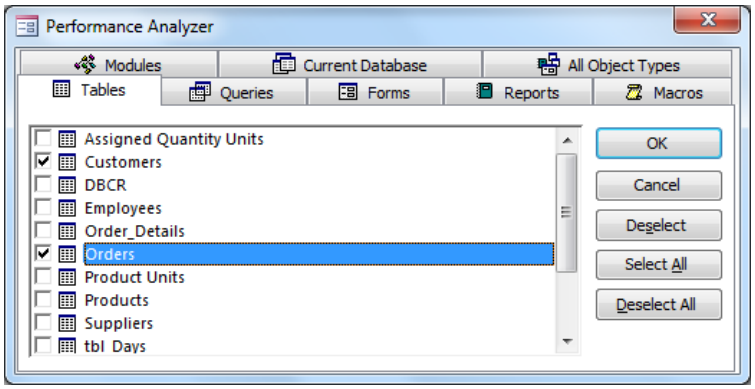
1. Ensure you close off all objects before proceeding as it will skip any opened objects if you had selected the said object as part of the analysis routine.

2. Under the **Database Tools** tab and in the '*Analyze*' section, choose the **Analyze Performance** icon.



Access 2016 – Database Tools tab, Analyze Performance icon

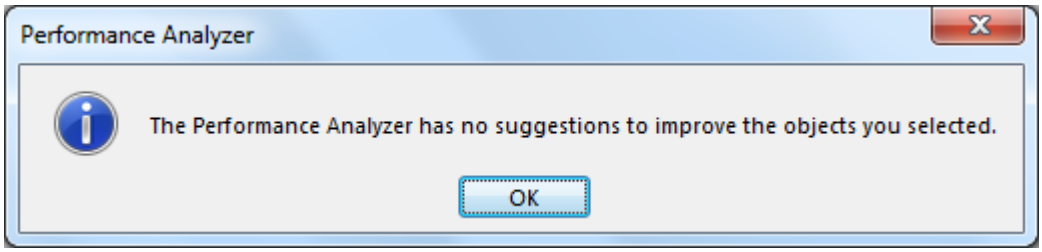
3. The following dialog box appears where you pick one or more objects across tabs.



Continues/…

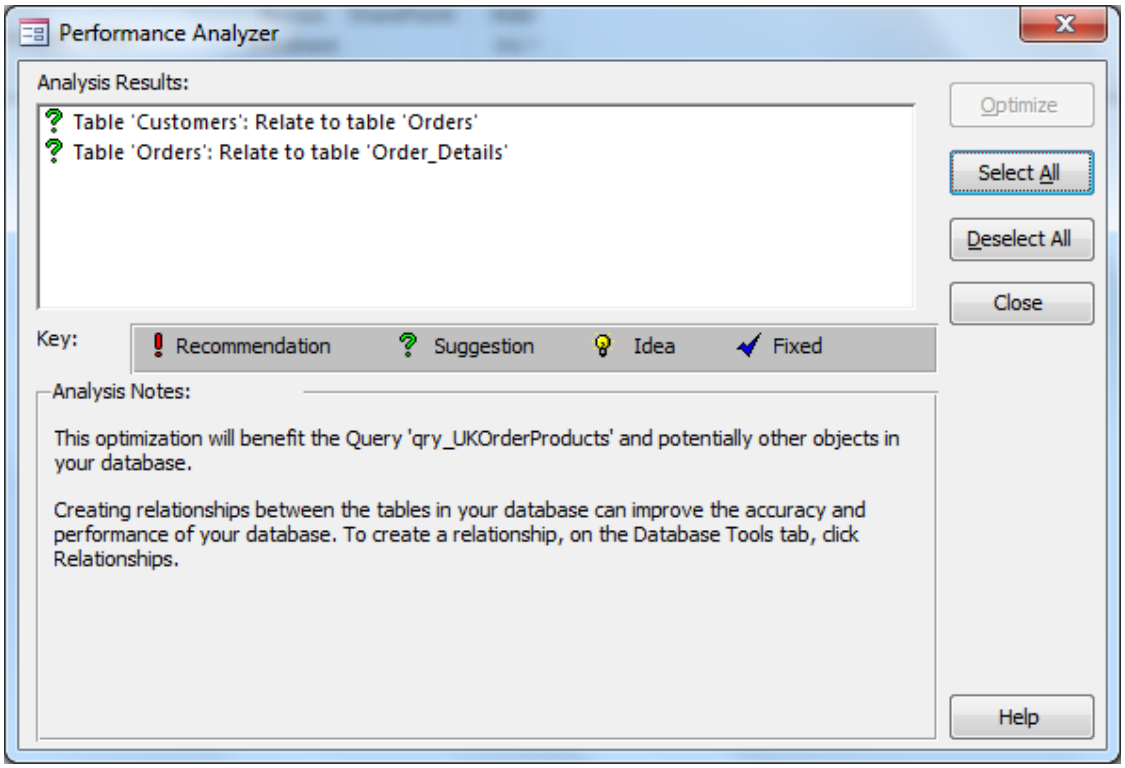
4. Click **OK** and see the results.

Now, if you have a perfect database structure, the following screen will appear:



However, it's not unusual for either the database developer to have overlooked a good design structure or perhaps Access being a little fragile and sensitive suggesting what could be better improved.

If for example you neglected to enforce good normalisation techniques and set tables in a solid relationship (*see topic on Understanding Access Database Relationships (RDBMS)*), you may experience the following screen:



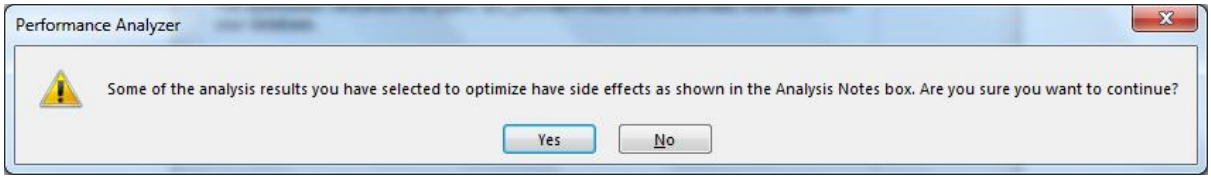
In the above example, there are two items in the results pane for your attention with additional comments in the lower half for reference.

Notice the icon types to indicate the sensitivity and importance level for each item with the '**?**' meaning just a suggestion and could be left alone (as passive).

Continues/…

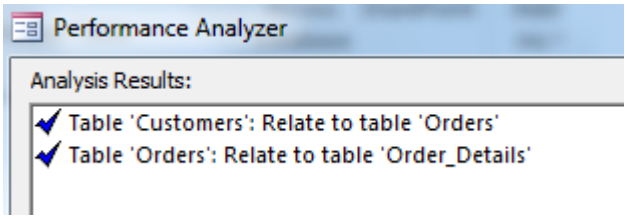
Bu selecting at least one item (*can choose more the one – use the **Select All** button*), the **Optimize** button is enabled and can be clicked.

Depending on what the item is and the possible outcome, you may experience additional prompts like as an example:



***Note**: At this point, you may want to consider taking a backup of your database just in case it has adverse effects that cannot be reversed.*

If you proceed, the system will attempt to fix the item and you are returned to same main screen with a blue tick (f*ixed*) icon for each selected item.



The Analyze Table Tool

The Analyze Table tool can restructure tables to provide for a more flexible and relational structure.

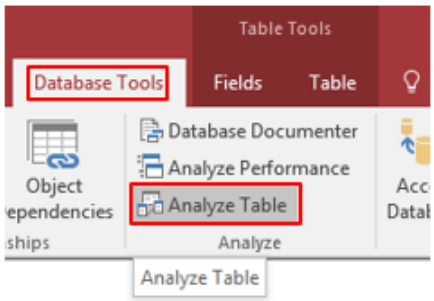
You may feel the previous analyze tool should be sufficient but tables are a very important type of object and the root of all good Access databases and this tool focuses just on all the selected tables checking for further properties like correct indexing and duplicate values which could be further normalised and so on.

There are several different outcomes where you can choose to manually take control or let the wizard attempt to build changes for you. Either way, it is highly recommended that you backup your database first.

Analyze Table

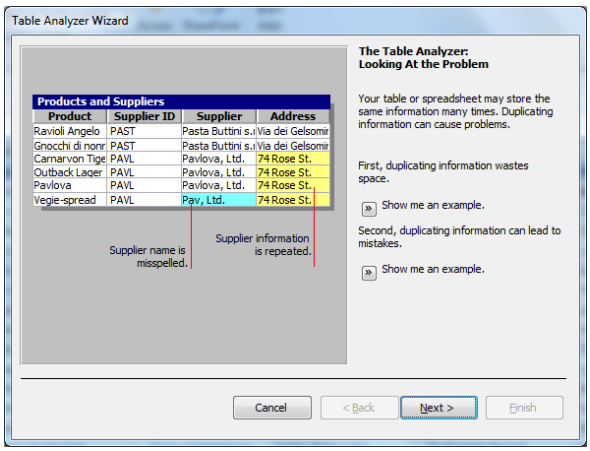
1. Ensure you close off all objects before proceeding.

2. Under the **Database Tools** tab and in the *'Analyze'* section, choose the **Analyze Table** icon.



Access 2016 – Database Tools tab, Analyze Table icon

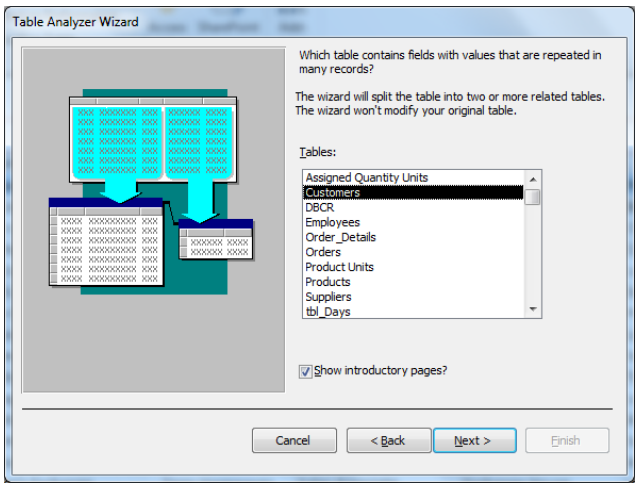
3. The following screen appears:



Continues/...

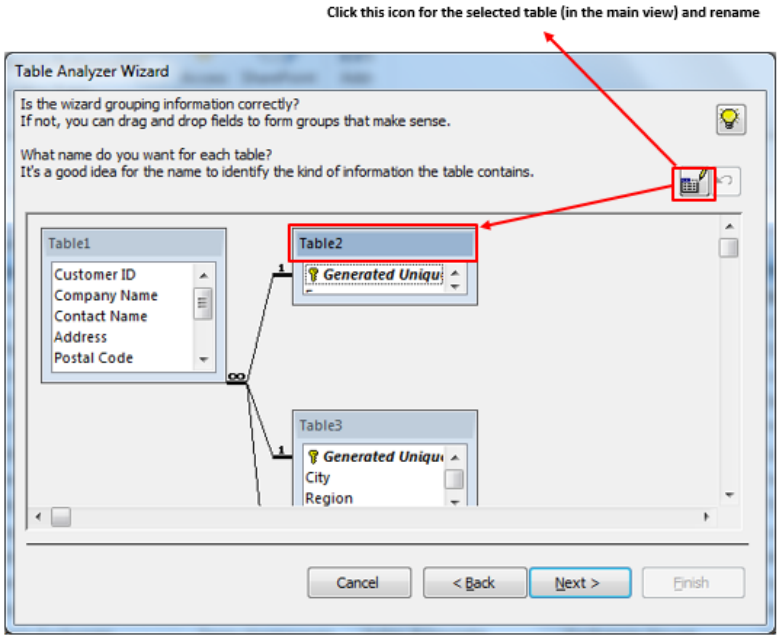
4. Step through the wizard screens and use the prompts and example button to understand more of the options available.

5. You will be asked to select only one table at a time to performance this test.



6. The next screen then invites you to either let the wizard do the next task or for you to decide. I would recommend letting the system take the lead here (*but feel free to engage!*)

7. The system will suggest a split table action for ID based fields and hat you should apply sensible names to each one:



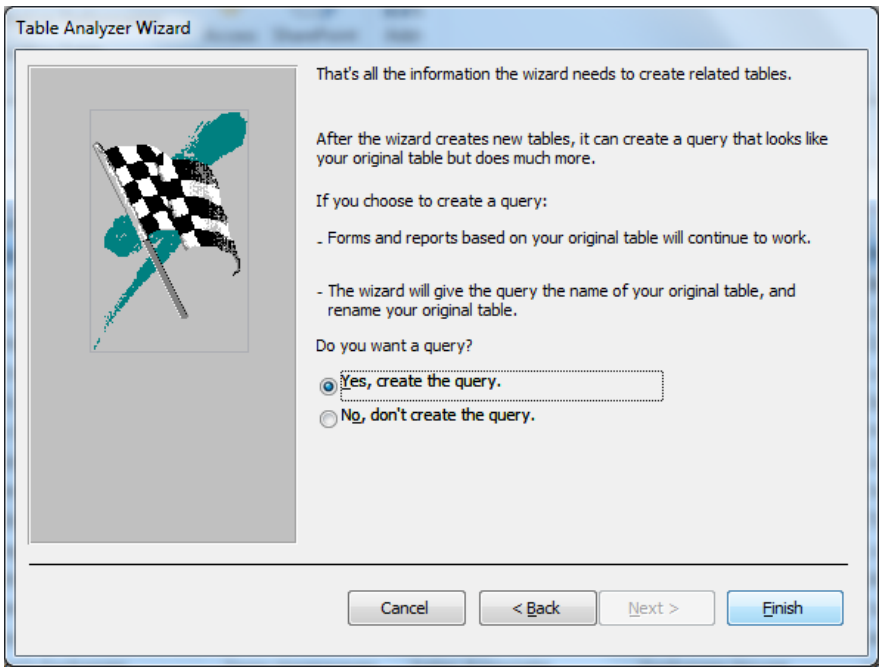
Access 2016 – Table Analyze Wizard tool

Continues/…

8. It will continue to enter into the data values making suggestions about similar looking values and whether you want to edit and change them. It's a time consuming task but may help to evaluate any inconsistencies etc.

   You need to spend time here and step though one or more prompts. If you do nothing, it will still prompt you and to continue or move onto the next step.

9. Repeat for each step until the last screen is shown



10. Notice the question to actually create new tables with their splits and also to generate a new query as per the original table so you can easily restore this query into a new real table and start again (*using a Make-Table query!*).

Object Dependencies Tool

The database has an internal map that tells Access where an object like a table is used in queries, in relationships to other tables, and on forms and reports. This feature enables the product to automatically modify the design of dependent objects when you make changes in an object.

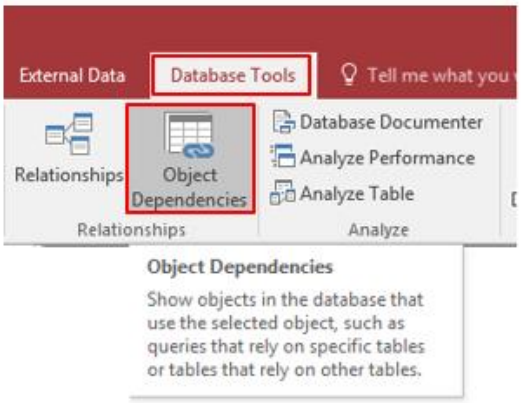
For example, say you have a table with a field called [Contact Name], and you decide to remove the space and rename it [ContactName] – *Just good conventions!*

Now imagine that you have queries, forms, and reports that all refer to [Contact Name]. However, because Access has a map of the dependencies, it can automatically correct references to [Contact Name]. Note that there are some limitations on these changes, relating to using the object name in more complex expressions and calculations.

Object Dependencies

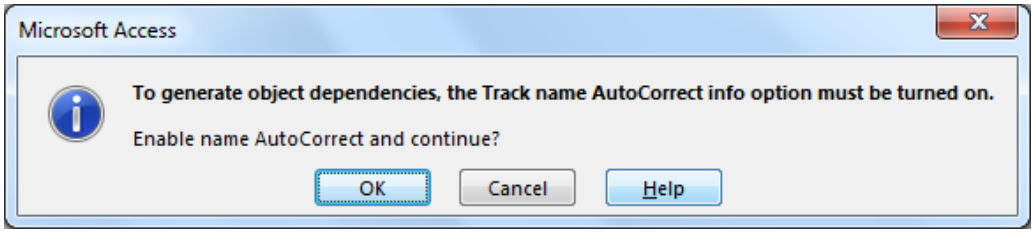
1. Select one object from the Navigation pane like the '*Customers*' table.

2. Under the **Database Tools** tab and in the '*Relationships*' section, choose the **Object Dependencies** icon.



Access 2016 – Database Tools tab, Object Dependencies icon

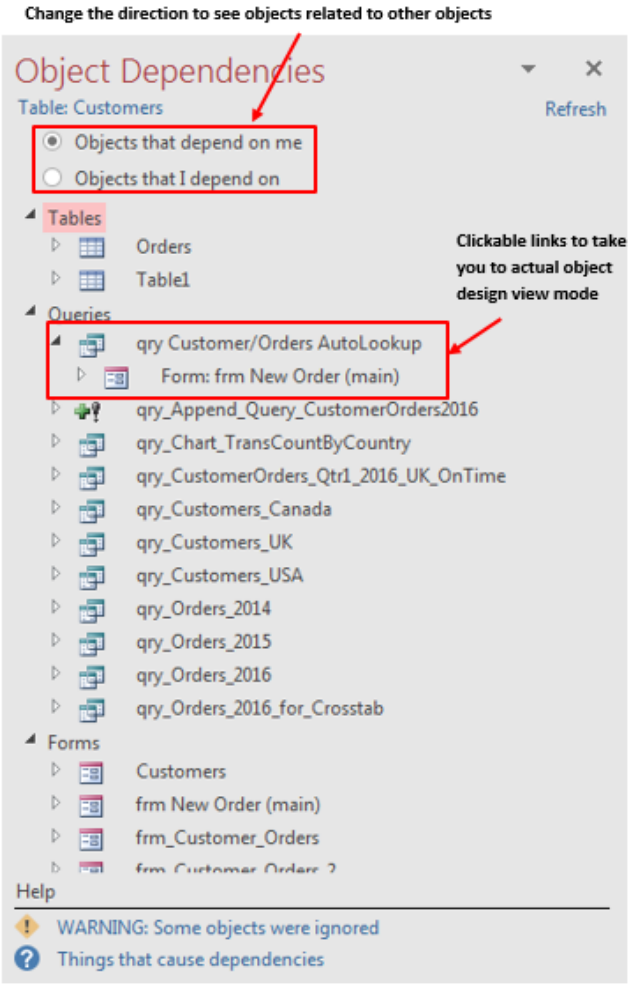
3. If you haven't set something called '**Name AutoCorrect**' from the **Options** screen, your will see the following prompt to switch this on (which will be required):



Continues/...

***Note**: The 'Name AutoCorrect' is not normally enabled for an Access database as it can drain the general performance of the application. If you switched this feature on to fulfil this task, then consider switching it back off when finished. (go to the File tab, Options, Current Database and scroll down the page to 'Name AutoCorrect Options')*

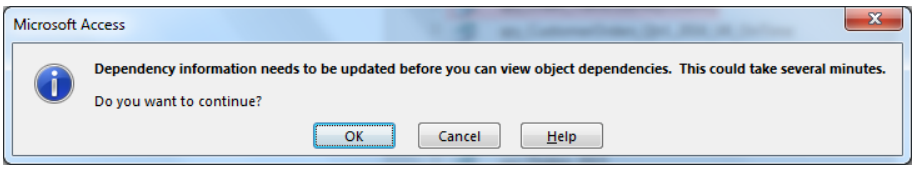
4. The following pane is displayed:



Access 2016 –Object Dependencies pane example

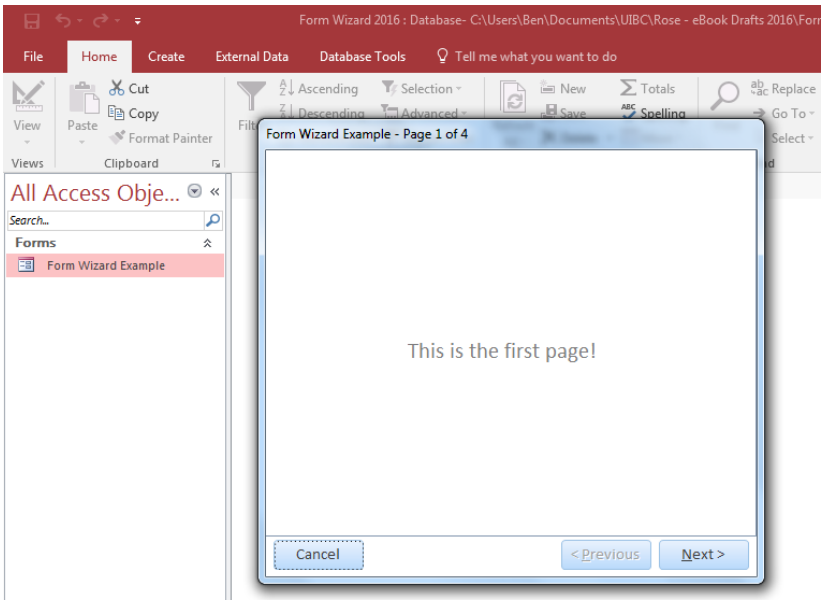
It's not a full and comprehensive list and can sometimes be frustrating to use but it will give you an overview where objects relate and if you change a field name in table, it will update all reference to the other objects.

You will see a prompt when changing to another new link which in turn has its own dependencies.

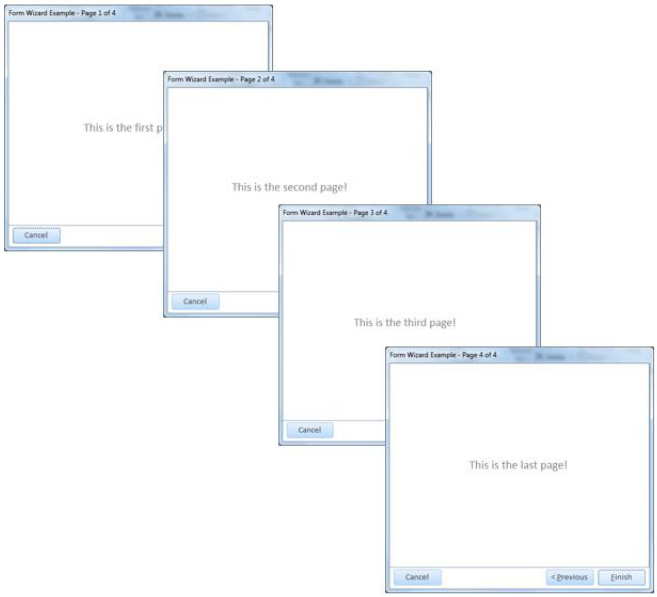


Custom Form Wizard Tool (Bonus)

Here's a bonus custom form wizard with some VBA code for you to use as a template for building your own wizard tools that can be used to help complete form filing processes, attach macros to it and generally provide a user-friendly interface for your database application.



This example form has four pages and can be modified to add or remove any number of pages:



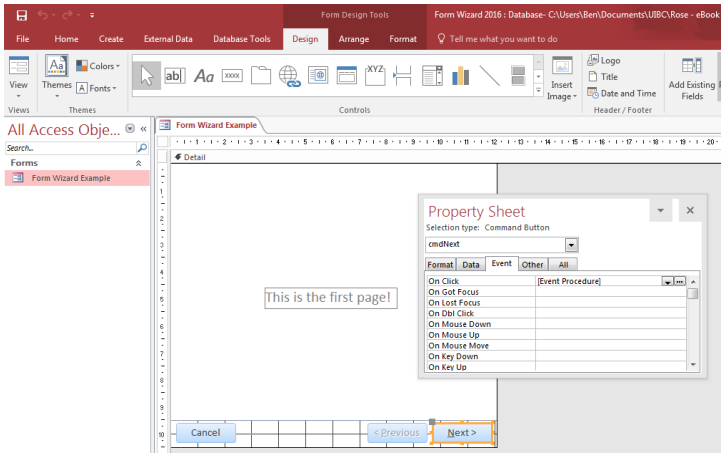
Access 2016 – Custom Form Wizard – 4-page example

Continues/...

You can collect this database from [Custom Form Wizard 2016 tool] and load it.

DISCLAIMER OF WARRANTIES/DAMAGES
All software data files is provided ”as-is,” without any express or implied warranty.
In no event shall the author be held liable for any damages arising from the use of this software.
The user must assume the entire risk of using the software.
PLEASE DO NOT DOWNLOAD UNLESS YOU HAVE AGREED TO THIS DISCLAIMER.

It contains just the one form called ‘*Form Wizard Example*’ and can be run to test the four pages or go to its design view mode and pick apart the components including visiting the competed VBA code (which is very basic).



Here’s the complete VBA code (*should be easy to amend*):

```
Private Sub cmdNext_Click()

    If Me.tab.Value = Me.tab.Pages.Count - 1 Then
        DoCmd.Close
        Exit Sub
    End If

    Me.tab.Value = Me.tab.Value + 1

    If Me.tab.Value = Me.tab.Pages.Count - 1 Then
        Me.cmdNext.Caption = "&Finish"
    Else
        Me.cmdPrevious.Enabled = True
    End If

    Me.Caption = "Form Wizard Example - Page " & Me.tab.Value + 1 & " of " & Me.tab.Pages.Count

End Sub

Private Sub cmdPrevious_Click()

    Me.tab.Value = Me.tab.Value - 1

    If Me.tab.Value = 0 Then
        Me.cmdNext.SetFocus
        Me.cmdPrevious.Enabled = False
    Else
        Me.cmdNext.Caption = "&Next >"
    End If

    Me.Caption = "Form Wizard Example - Page " & Me.tab.Value + 1 & " of " & Me.tab.Pages.Count

End Sub

Private Sub Form_Load()
    Me.Caption = "Form Wizard Example - Page " & Me.tab.Value + 1 & " of " & Me.tab.Pages.Count

End Sub
```

There you have it.

I hope you found this guide very useful?

We'd love to hear your comments about this guide, so if you would rather email us rather than blogging, send your comments to ben@AccessDatabaseTutorial.com – and don't forget that my website has lots more articles and free videos to help you plan, build and implement an Access database.

Ben Beitler – **"Your Access Database Expert"**

Please visit our website:

AccessDatabaseTutorial.com

Thank You