

'DIY BI' Tips, Tricks & Techniques

Power Query Edition

By Excelguru Consulting Inc.



EXCELGURU

‘DIY BI’ Tips, Tricks & Techniques – Power Query Edition

© 2022 Excelguru Consulting Inc.

Excel, Power Query, Power Pivot, Power BI, and their logos are registered trademarks of Microsoft Corporation in the United States and/or other countries, and are property of their respective owners.

Power Query Academy Logos: © Skillwave Training

All rights reserved. No part of this eBook may be reproduced or transmitted in any form or by any means, electronic or mechanical, including photocopying, recording, or by any information or storage retrieval system without permission from the publisher. Every effort has been made to make this eBook as complete and accurate as possible, but no warranty or fitness is implied. The information is provided on an “as is” basis. The authors and the publisher shall have neither liability nor responsibility to any person or entity with respect to any loss or damages arising from the information contained in this eBook.

This eBook is acquired completely free-of-charge by simply signing up to our newsletter on www.excelguru.ca. If someone charged you for it, we suggest that you request a refund.

Author: Ken Puls
Layout & Design: Excelguru Consulting Inc.
Copyediting: Excelguru Consulting Inc.
Cover & Illustrations: Excelguru Consulting Inc.
Logos on pages 6, 13, 15, & 20: Skillwave Training

Originally published online by Excelguru Consulting Inc. on May 8, 2017

Updated version published on June 20, 2018

Updated version published on February 4, 2020

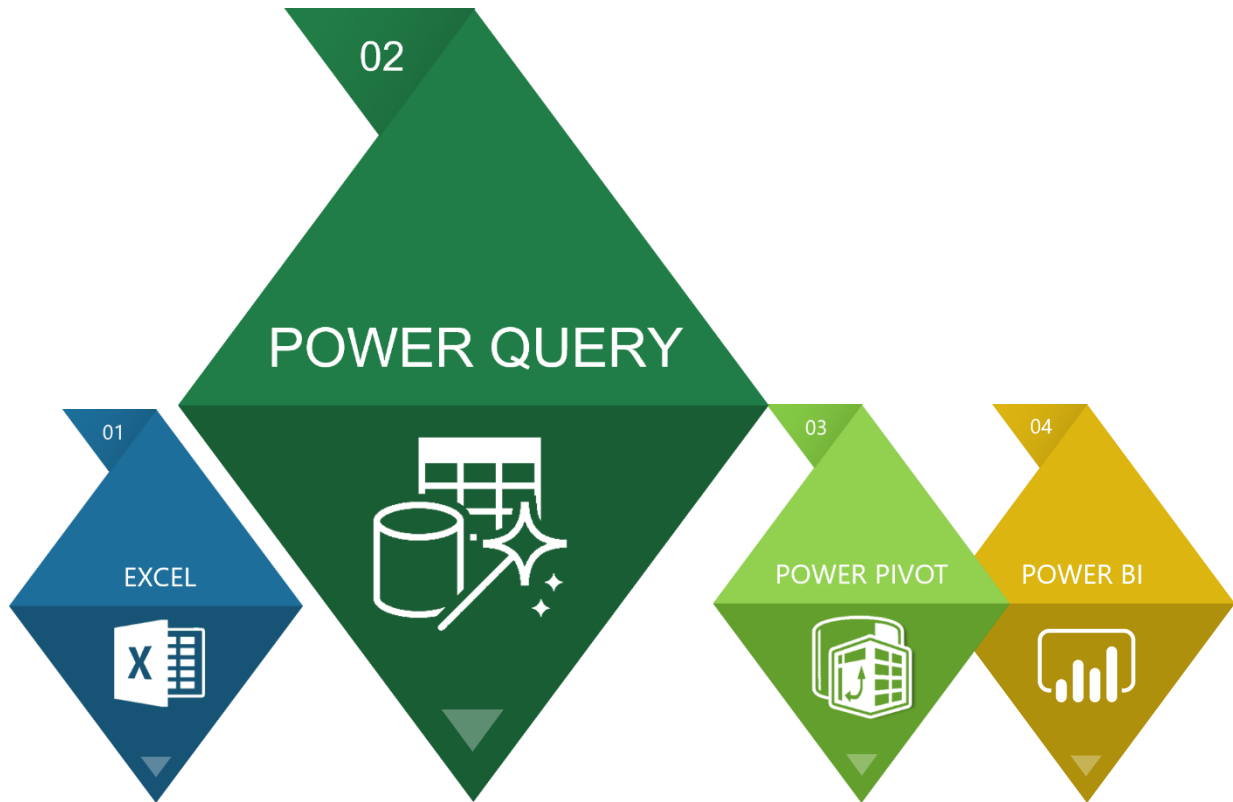
Updated version published on April 22, 2021

Updated version published on August 9, 2022



About this Book

This free eBook is the second in a series of four that will be released to our Excelguru email newsletter subscribers. Each book contains five of our favourite tips, tricks, and techniques, with one book each for Excel, Power Query, Power Pivot, and Power BI. The tips and tricks shared in these books have been developed over years of research and real-world experience.



Get the Entire Series

Don't miss out on the other three books in this series. They're all free, and you can get them as a reward for subscribing to the Excelguru newsletter. In addition to the free book, you'll also get our monthly(ish) email newsletter features the latest updates for Excel and Power BI, upcoming training sessions, new products, and other information.

If you are not already a subscriber, sign up today at <http://xlguru.ca/newsletter>.

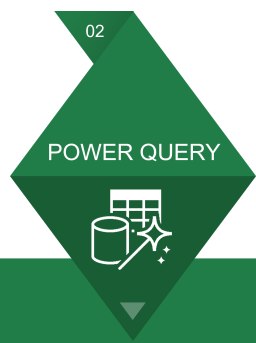


Table of Contents

| | |
|--|----|
| Tip 1: Generate Relative Path for Solution | 1 |
| Tip 2: Build Calendar Table from Two Inputs..... | 7 |
| Tip 3: Creating Custom Date Columns | 14 |
| Tip 4: Logging the Last Update Date/Time | 16 |
| Tip 5: One Setting to Cut (Power) Query Load Time..... | 18 |



Tip 1: Generate Relative Path for Solution

One of the big challenges when building solutions that use Power Query to source data is that all file paths get hard coded into the queries. This makes it difficult to send a solution to someone else if the file is stored in a different location, or if the path the recipient uses to connect to the data is different than that of the author.

Assume for instance that you build a new solution and store the files on your local PC. You then want to send the Excel workbook to your boss, but need to store it on the network. To make things worse, your boss has a different drive mapping to the folder than you do. This means that whoever opens the file will need to edit Power Query's M code in order to repoint that file path. This is obviously far from ideal.

Fortunately, in Excel we can get around this problem. To do so, we need to set up a table in the workbook, include a formula that retrieves the solution file path, and pull that into a query. We can then pass the results of that query into others, resulting in a dynamic file path instead of a hard coded one. And the best part is – since an Excel formula returns the file path for the solution – it will recalculate when the solution is moved. You'll never need to ask the recipient to update your M code again!

Solution Background

For the purpose of this example, assume that I have an Excel solution set up as follows:

- Path to the Master Workbook:

`C:\DIY BI\DIY BI - Power Query Edition.xlsx`

- Source Data Files:

`C:\DIY BI\Source Data\`

The “DIY BI – Power Query Edition.xlsx” workbook is the one that holds our Power Query solution, and the source data files could be anything. In this case, however, the folder contains a single txt file.

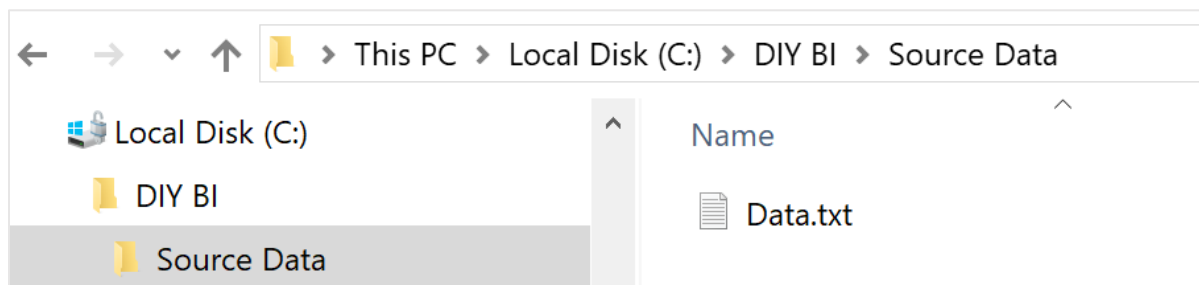


Figure 1 - Data.txt stored in the Source Data folder

Assume also that within this workbook there is already a query called “Actual” that pulls in data from Data.txt and lands it in a table. The problem, of course, is that the path is hard coded to `C:\DIY BI\Source Data\Data.txt`, which won't share very well.



Step 1: Setting up a Table for our File Paths

To create our dynamic solution, we begin by creating a table in our Excel workbook that looks like this:

| Parameter | Value |
|-------------|--------------|
| File Path | C:\DIY BI\ |
| Data Folder | Source Data\ |

Figure 2 - A parameter table for Power Query

The key things to recognize about this setup are:

1. This is a proper Excel table which is named “Parameters”. (If you’ve never created an Excel table before, you can do so by selecting any cell in the data, going to Home→Format as Table, then changing the table name on the Table Tools tab.)
2. The Value column for File Path uses the following formula to return the path to the current workbook. (Note that the workbook must be saved before it can return anything here.)
`=LEFT(CELL("filename",A1),FIND("[",CELL("filename",A1),1)-1)`
3. The Value column for Data Folder is just text, but ends with a trailing slash.

Step 2: Create Queries to Return the File Paths

To create the needed queries, we will:

- Select a cell in the table
- Pull the data into Power Query
 - Excel 2016+: Go to the Data tab → From Table/Range
 - Excel 2010/2013: Go to the Power Query tab → From Table



This will launch you into Power Query where the table will look as follows:

| | Parameter | Value |
|---|-------------|--------------|
| 1 | File Path | C:\DIY BI\ |
| 2 | Data Folder | Source Data\ |

Figure 3 - The Parameter table in the Power Query editor

We now need to create two queries; one for the File Path and one for the Data Folder. To do this quickly:

- Filter the Parameter column (using the arrow) to select only File Path.
- Right click the file path in the Value column → Drill Down.
- Rename the query on the right to call it “FilePath” with no spaces.

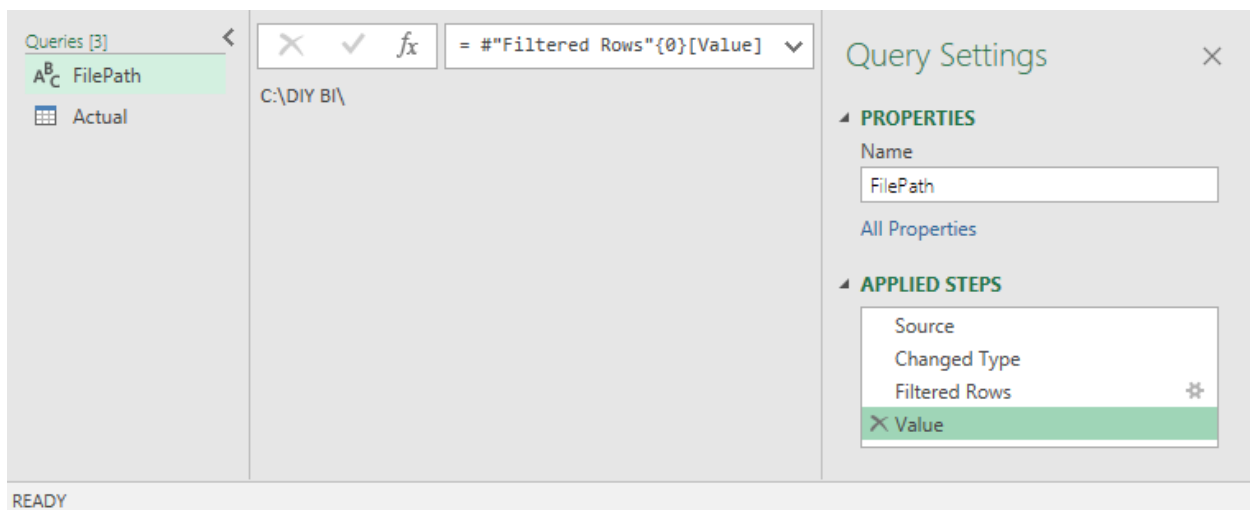


Figure 4 - The FilePath query

With this query done, we can now create the second query:

- If the Queries pane on the left is collapsed, expand it by clicking the > character.
- Right click the FilePath query → Duplicate.
- Rename the query on the right to “DataFolder”.
- In the Applied Steps area, click the gear icon next to Filtered Rows.
- Change the filter to user “Source Data” instead of “File Path”.
- Select the Value step.



The results should now look like this:

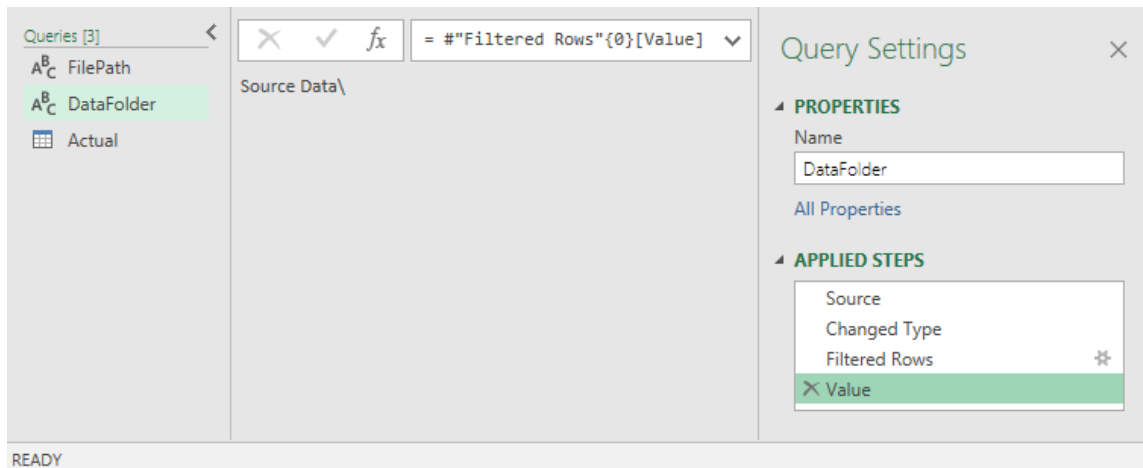


Figure 5 - The DataFolder query

We are now at the point where we can finalize these queries. We don't really need them to load to worksheets or the data model, however, we so need to create Connection Only queries. To do this:

- Go to Home → Close & Load → Close & Load To...
- Choose to Only Create Connection.

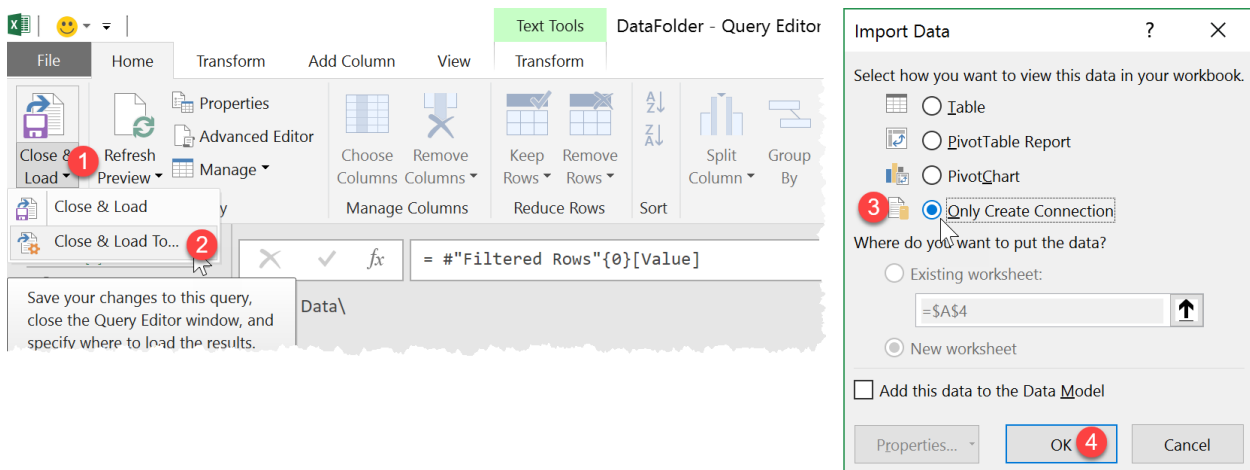


Figure 6 - Saving the queries as Connection Only

NOTE:

When you create multiple queries in the Power Query editor, the load behaviour you choose will be applied to all of them. You can't pick and choose. If in doubt, choose to load as Connection Only. It's faster, leaves less litter in the form of new worksheets/data model tables, and you can always change the behaviour later.



Step 3: Modify the Existing Query

The next thing we need to do is edit the existing query to take advantage of the queries that we've just built.

Locate the query you wish to modify:

- Excel 2016+: Go to the Data tab → Queries & Connections
- Excel 2010/2013: Go to the Power Query tab → Show Pane

The Queries pane will open on the right side of the Excel window. Locate the query you wish to modify (in this case the Actual query), right click it and choose Edit. You'll then be taken into the Power Query editor again.

Select the Source step for the query:

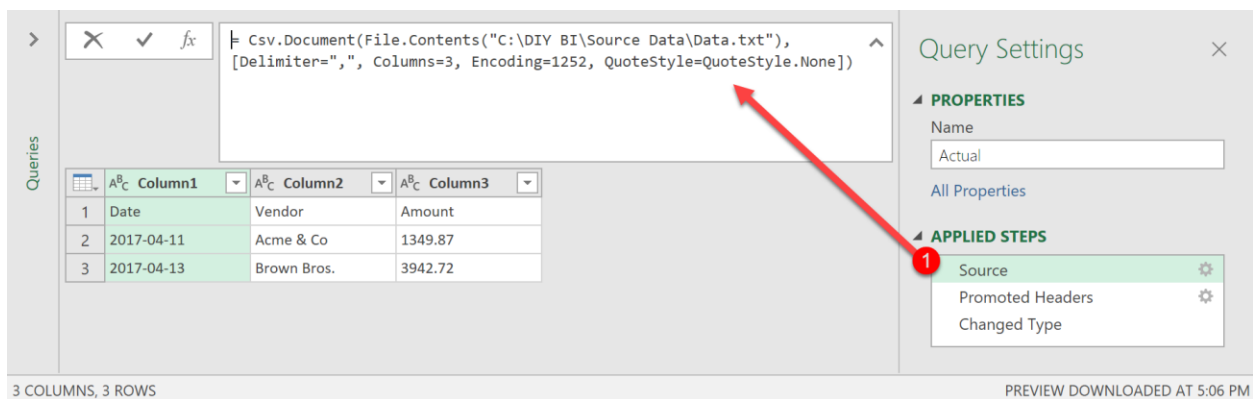


Figure 7 - Selecting the Source step for the original query

NOTE:

If you don't see the formula bar, you can display it by going to the View tab → Formula Bar.

The trick here is to replace the existing file path with the queries we created earlier. So in this case:

```
= Csv.Document(File.Contents("C:\DIY BI\Source  
Data\Data.txt"),[Delimiter=",", Columns=3, Encoding=1252,  
QuoteStyle=QuoteStyle.None])
```

Becomes:

```
= Csv.Document(File.Contents(FilePath&DataFolder), [Delimiter=",",  
Columns=3, Encoding=1252, QuoteStyle=QuoteStyle.None])
```

Once you've made the change, go to the Home tab → Close & Load to save the change.

02

POWER QUERY



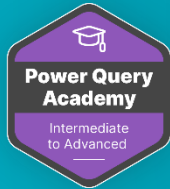
The End Effect

At this point, the query will no longer fail if the solution is moved to a new folder. As soon as the Excel workbook is opened, the CELL function will recalculate to show the current file path. And when the queries are refreshed, they will rely on that path to refresh the data.

The implications of this are huge:

1. You can zip an entire folder, send it to someone, and when they refresh it, it will just work.
2. If your co-worker uses a different drive mapping to get to your solution than you do, it's no big deal. It will still refresh just fine.

Please keep in mind that in order for this solution to function, the Power Query solution needs to be opened in Excel so that the file can recalculate the file path (i.e. the file path won't recalculate if you retrieve the parameters table via a Power Query solution in another workbook since it doesn't kick off Excel's calculation chain).



Extend Your Learning

Would you like to see this technique in action? It's covered in the module on Parameters & Custom Functions of our [Power Query Academy!](#)
Get started for FREE with our [Power Query Fundamentals](#) course.



Tip 2: Build Calendar Table from Two Inputs

If you are building solutions that use Power Pivot, you know that a calendar table is critical to creating proper time intelligence solutions. But where do you get one? There are a variety of places, but each has some drawbacks.

For my money, I use a combination of Excel formulas and Power Query to build a self-updating calendar that always covers the date range I need.

Step 1: Record the Start and End dates in Excel

Like in our previous tip, we start by entering the dates into an Excel Table. And since we already have one in this file, why not use it?

| | A | B | C | D |
|---|---|--------------|---|---|
| 1 | Parameter table to dynamically drive queries | | | |
| 2 | | | | |
| 3 | Parameter | Value | | |
| 4 | File Path | C:\DIY BI\ | | |
| 5 | Data Folder | Source Data\ | | |
| 6 | Calendar Start | 1/1/2017 | | |
| 7 | Calendar End | 12/31/2017 | | |
| 8 | | | | |

Figure 8 - Adding calendar start and end dates to our Parameter table

The biggest question really, is what dates do you use? Personally, as I work with a lot of budgets, I like to ensure that my calendars start on the first day of the year, and end on the last day of either the current or next year.

In the case of the values shown in Figure 8, I chose to hard code the start date to Jan 1, 2017, and use the following formula to return Dec 31, 2017 (the end of the 11th month from where I started):

`=EOMONTH(B7,11)`

The tables on the following page show some formulas that you may find useful when building dynamic calendar formulae.



Useful formulas for start date:

| To Generate | Use this formula in Excel |
|------------------------|-------------------------------------|
| Today's Date | =TODAY () |
| End of Current Month | =EOMONTH (TODAY () , 0) |
| First of Current Month | =EOMONTH (TODAY () , -1) +1 |
| Jan 1 of Current Year | =DATE (YEAR (TODAY ()) , 1 , 1) |
| Jan 1 of Last Year | =DATE (YEAR (TODAY ()) -1 , 1 , 1) |

Useful formulas for end date:

| To Generate | Use this formula in Excel |
|--|---------------------------------------|
| End of Current Month | =EOMONTH (TODAY () , 0) |
| End of 12 th Month From Now | =EOMONTH (TODAY () , 12) |
| Dec 31 of Current Year | =DATE (YEAR (TODAY ()) , 12 , 31) |
| Dec 31 of Next Year | =DATE (YEAR (TODAY ()) +1 , 12 , 31) |

Step 2: Create Queries for the Start and End Dates

With the start and end data for our calendar now contained in the parameter table, we now want to create two simple queries that we can call with the start and end dates for our calendar. Those will be called “Calendar_Start” and “Calendar_End”.

To do this, we will follow steps similar to what we did to load the file path into Power Query in the previous tip.

Creating the Calendar_Start Query

As many of the steps are quite similar to what was show in the previous tip, I've kept the required steps to a bullet point summary. Those are:

- Select a cell in the Parameters table.
- Create a new query From Table/Range.
- Filter the Parameter column to include only “Calendar Start”.
- Change the Value column to a Date.
- Right click the date in the Value column → Drill Down.
- Rename the query in the Applied Steps window to be “Calendar_Start”.

Your query should now look similar to what is shown in Figure 9 on the next page.



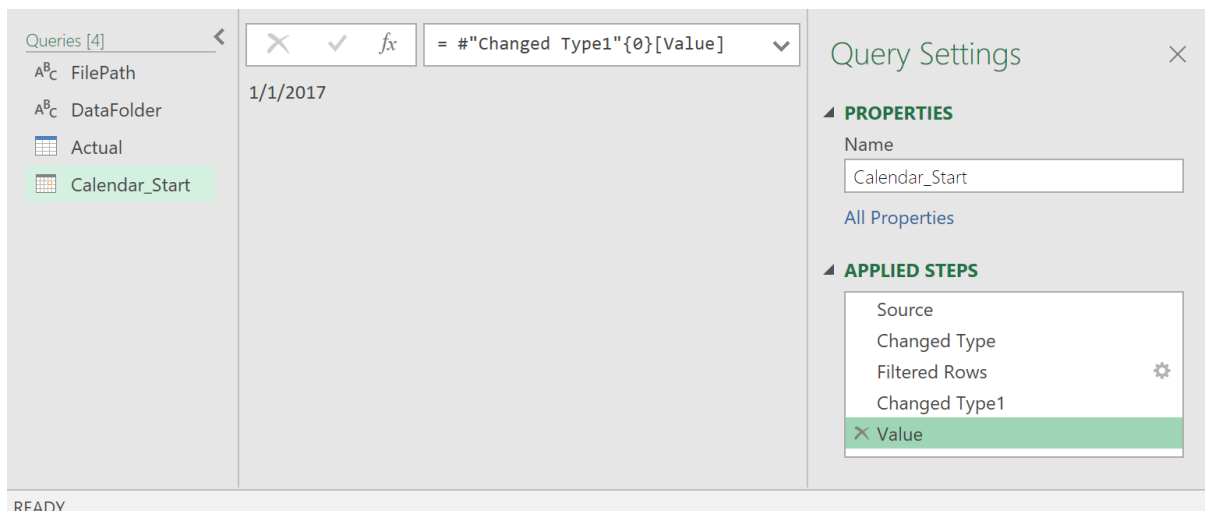


Figure 9 - The Calendar_Start query

With this built, we can now create the Calendar_End query.

Creating the Calendar_End Query

The easiest way to create the Calendar_End query is again to simply duplicate the Calendar_Start query, and modify the key parts that need to be different. To do this:

1. Right click the Calendar_End query in the Queries pane on the left → Duplicate.
3. Change the query name to Calendar_End.
4. Click the gear next to the Filtered Rows step and change the filter to “Calendar End”.
5. Select the Value step in the Applied Steps window to make sure it worked correctly.

The resulting query should look as follows:

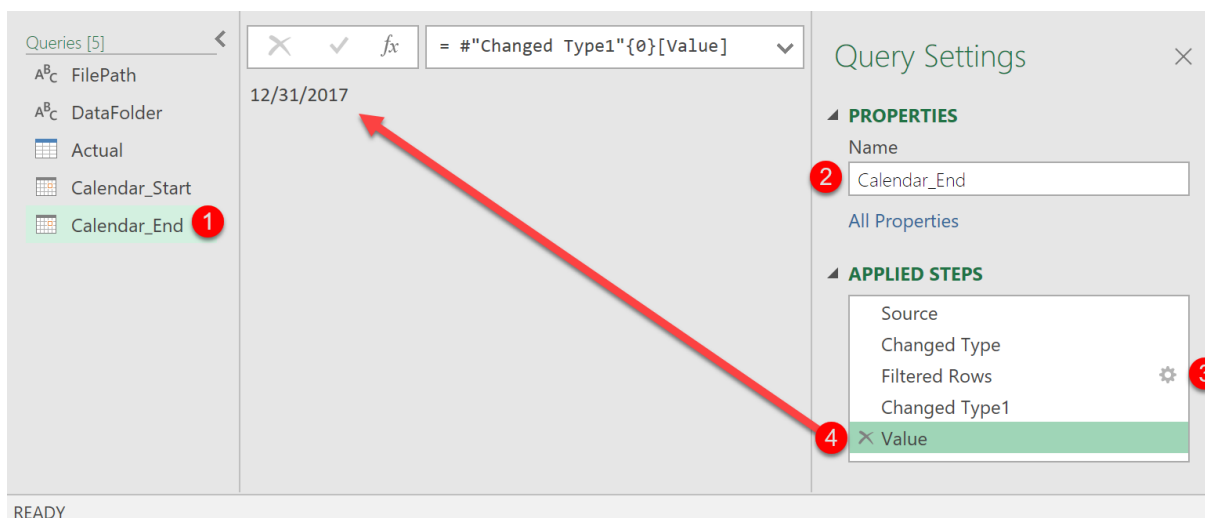


Figure 10 - Creating the Calendar_End query



Step 3: Create the Dynamic Calendar Query

We've now got our start end date values set up to read from the spreadsheet whenever the query is refreshed. And because those values are coming from Excel formulas, they are dynamically updated whenever the workbook is recalculated. Now we get to the really cool part; creating the fully dynamic calendar.

You'll notice that I haven't closed and loaded any of the queries yet. If you have, it's not the end of the world, but you'll need to create a new blank query, or edit one of your existing queries in order to continue. Personally, when I build these tables, I just create my next blank query in the Power Query editor as follows:

- Right click the whitespace in the Queries pane.
- Choose New Query → Other Sources → Blank Query.

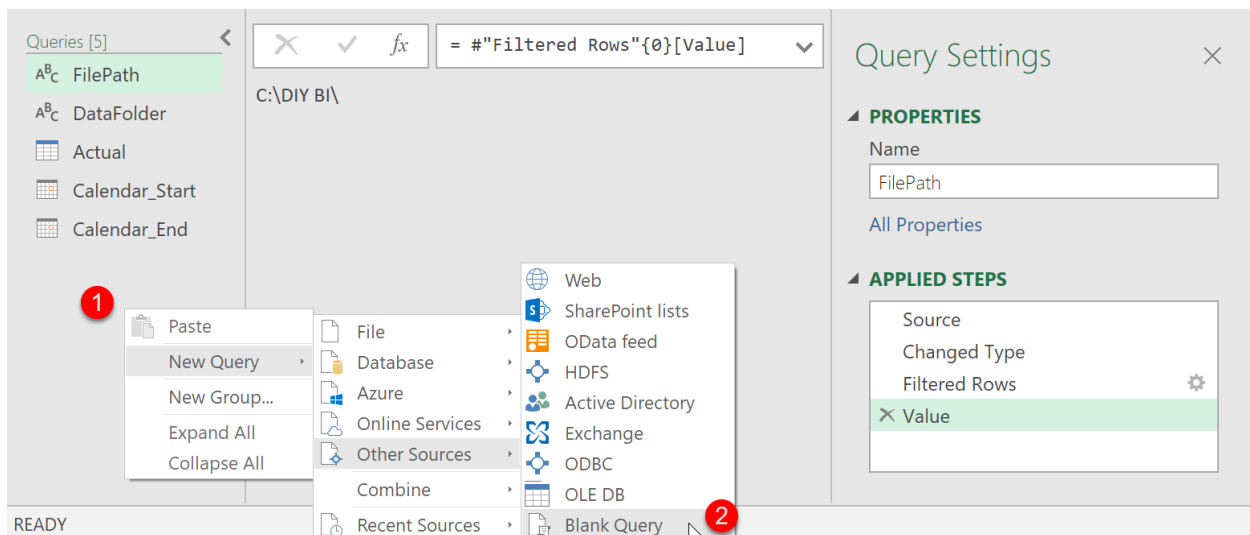


Figure 11 - Creating a new blank query inside Power Query

This will result in a new blank query where we can make some magic happen:

- Start by renaming the new query to “Calendar”.
- In the formula bar, type the following:

```
= {Number.From(Calendar_Start) .. Number.From(Calendar_End) }
```

It is a little bit ugly, and those curly braces and parentheses all need to be correct, but once you're done, you should see a List, as shown in Figure 12 on the next page.



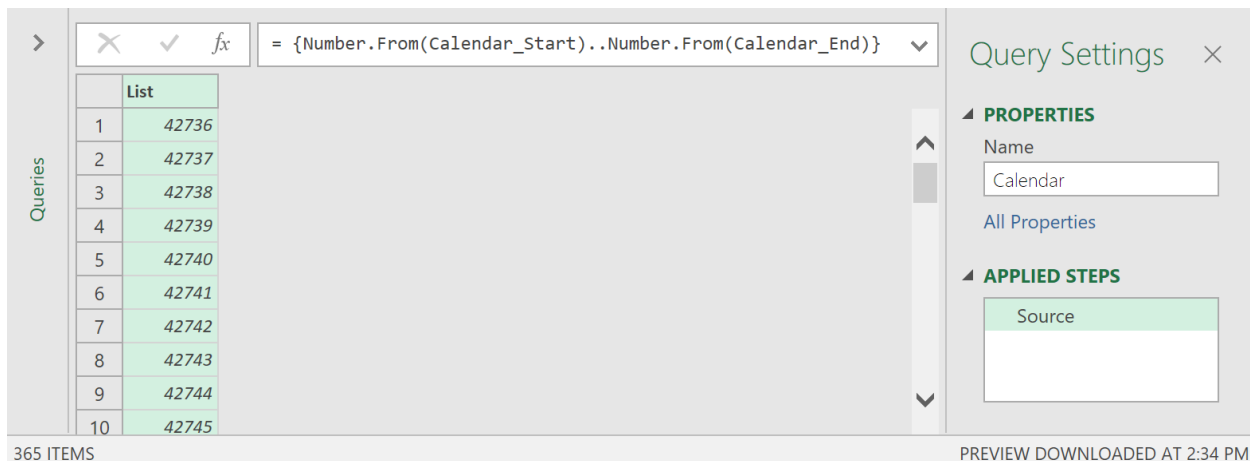


Figure 12 - A list of... something?

It doesn't look like much, but this is a list of date serial numbers that runs from our start date through our end date.

NOTE:

The reason I used an underscore when naming my Calendar_Start and Calendar_End queries is that it makes it easier to type here. Had I used spaces instead of underscores, I would have needed to write = {Number.From("#Calendar Start")..Number.From("#Calendar End")} which is longer and just plain ugly to write.

We now want to do something with our date serial numbers, but you'll find that most of the Power Query transformations are greyed out. The reason for this is that we currently have a list, not a table, so let's fix that:

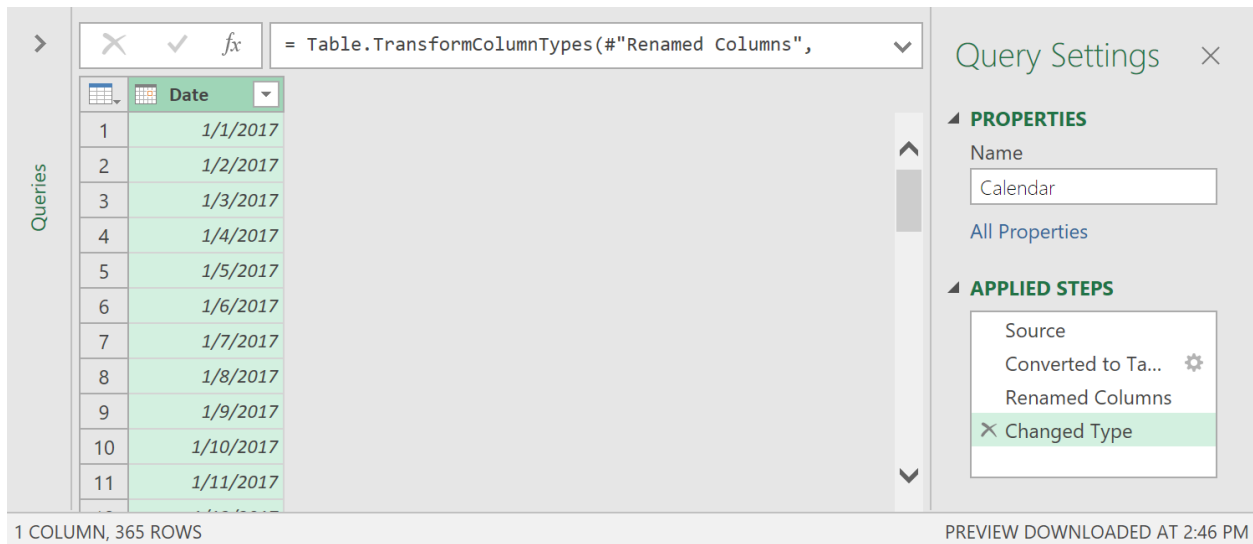
- Go to the List Tools → Transform tab → to Table.
- Accept the default options by clicking OK.

At first, the only real noticeable change is that the first column is now called Column1 instead of List, but on digging deeper you'll find that you once again have a rich array of transformations available to you again. Let's use those:

- Right click Column1's header → Rename → Date.
- Right click Column1's header → Change Type → Date.



And voila! A calendar table is born:



| | Date |
|----|-----------|
| 1 | 1/1/2017 |
| 2 | 1/2/2017 |
| 3 | 1/3/2017 |
| 4 | 1/4/2017 |
| 5 | 1/5/2017 |
| 6 | 1/6/2017 |
| 7 | 1/7/2017 |
| 8 | 1/8/2017 |
| 9 | 1/9/2017 |
| 10 | 1/10/2017 |
| 11 | 1/11/2017 |

1 COLUMN, 365 ROWS

PREVIEW DOWNLOADED AT 2:46 PM

Figure 13 - A basic calendar table

You can now add as many columns as you like from a large collection of one-click transforms available to you. To do this:

1. Select the Date column.
6. Go to Add Column → Date.
7. Choose the format of the date you'd like to add.
8. Repeat steps 1-3 for any other formats that you'd like on your calendar.

Step 4: Loading the Queries

Now that we have our dynamic calendar created, we need to load it to our model. Remember, however, that we created three queries here, and we really don't need to land the Calendar_Start or Calendar_End, as they are only really used to drive the calendar.

As all queries will use the same load behavior, it's important to load the queries as Connection Only, then change the Calendar later to load to the correct destination. So let's do that:

- Go to the Home tab and click the bottom half of the Close & Load button.
- Choose Close & Load To... → Only Create Connection.

This should complete quickly, and all queries will be marked as "Connection Only" in the Excel Queries pane. Now let's load the Calendar somewhere where it can be used:

- In the Excel queries pane, locate the Calendar query and right click it.
- Choose Load To...
- Change the load behavior to load to a table or the data model.



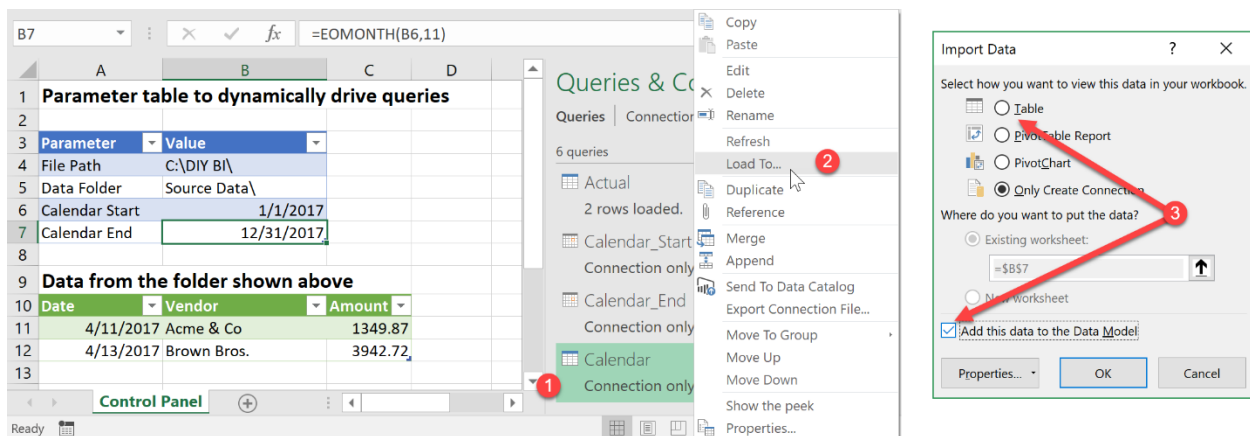


Figure 14 - Changing the load behaviour of an existing query

And you now have a completely dynamic calendar table!

Adapting this Trick to Power BI Desktop

We can create calendar tables in exactly the same fashion for Power BI, the only difference is where we source our Calendar_Start and Calendar_End values. In the case of Power BI, since we don't have a grid with dynamic formulas, we can target these off either Parameters (if we want to hard code the values), or via a query against the data set.

Generally, when I'm doing this, I will set up my start and end dates via the following method:

- Locate my transaction table, right click it and choose Reference.
- Rename the query to Calendar_Start.
- Right click the date column → Remove Other Columns.
- Filter the data column → Date Column → Is Earliest.
- Right click the date column → Remove Duplicates.
- Go to the Transform tab → Dates → Year → Start Of Year.
- Right click the date → Drill Down.

I'd then do a similar thing to drill into the calendar end date and use the rest of the steps from this tip to generate my calendar. The great thing is that it will then dynamically generate a calendar that always spans the entirety of the dates I need.



Extend Your Learning

Would you like to see this technique in action? It's covered in the module on Date and Time Techniques of our [Power Query Academy!](#)
Get started for FREE with our [Power Query Fundamentals](#) course.



Tip 3: Creating Custom Date Columns

Extending your Calendar table is actually very easy in Power Query. As mentioned in the tip above, when you're in the query, you simply:

1. Select the Date column.
2. Go to Add Column → Date.
3. Choose the date format you want.
4. Repeat steps 1-3 for any other columns you want to add.

But what about when you want to add a date format that just isn't in the offered transformations? For example, how do you create a date like the following:

- May 9, 2017
- 2017-05-09 (Tue)
- 9 May 17

None of these exist in the offered transformations, so we need to build them manually. To do this, we need to:

- Go to Add Column → Add Custom Column.
- Use a custom formula to generate the date format that looks similar to this:

```
=Date.ToText(Date, "<format>")
```

The trick here is all about how to build a date from character codes. It's quite similar to custom number formatting in Excel, but with a couple of differences.

Here are the formulas for the 3 dates shown earlier:

| Date | Formula |
|------------------|--|
| May 9, 2017 | =Date.ToText([Date], "MMM d, yyyy") |
| 2017-05-09 (Tue) | =Date.ToText([Date], "yyyy-MM-dd (ddd)") |
| 9 May 17 | =Date.ToText([Date], "d MMMM yy") |

And here is the output of those formulas:

| | Date | A ^B C MMM d, yyyy | A ^B C yyyy-MM-dd (ddd) | A ^B C d MMMM yy |
|---|----------|------------------------------|-----------------------------------|----------------------------|
| 1 | 1/1/2017 | Jan 1, 2017 | 2017-01-01 (Sun) | 1 January 17 |
| 2 | 1/2/2017 | Jan 2, 2017 | 2017-01-02 (Mon) | 2 January 17 |
| 3 | 1/3/2017 | Jan 3, 2017 | 2017-01-03 (Tue) | 3 January 17 |
| 4 | 1/4/2017 | Jan 4, 2017 | 2017-01-04 (Wed) | 4 January 17 |
| 5 | 1/5/2017 | Jan 5, 2017 | 2017-01-05 (Thu) | 5 January 17 |
| 6 | 1/6/2017 | Jan 6, 2017 | 2017-01-06 (Fri) | 6 January 17 |

Figure 15 - Some custom date columns



So what do the characters mean? Here is a list of the reserved characters for date and time formatting:

| <i>Dates</i> | <i>Character</i> | <i>Times</i> | <i>Character</i> |
|--------------|------------------|--------------|------------------|
| Year | y | Hour | h |
| Month | M | Minute | m |
| Day | d | Second | s |

All other characters are treated as text, except for quotes, which can be used to return a text pattern (just like in Excel.)

The trick is all about combining the correct characters together to build your date. And the key here is to include sufficient characters to get the correct date format, as one “y” does something different than two “y” characters.

The following table shows what would happen for a date of July 9, 2017 (a Sunday) depending on how many of each character you combined:

| <i>For</i> | <i>Character</i> | <i>1</i> | <i>2</i> | <i>3</i> | <i>4</i> |
|------------|------------------|----------|----------|----------|----------|
| Year | y | 17 | 17 | 2017 | 2017 |
| Month | M | 7 | 07 | Jul | July |
| Day | d | 9 | 09 | Sun | Sunday |

This means:

- In order to get “Jul” you would use 3 “M” characters, or “MMM”.
- In order to get “July”, you would use 4 “M” characters.

If we want an ISO date (2017-07-09), we would use:

```
=Date.ToText ([Date] , "yyyy-MM-dd")
```

Do be aware of the following:

- These characters are case sensitive (lower case “m” is minutes, not months!).
- These patterns return textual results, not true dates (which may be important to some scenarios).



Extend Your Learning

Would you like to see this technique in action? It’s covered in the module on Date and Time Techniques of our [Power Query Academy!](#)

Get started for FREE with our [Power Query Fundamentals](#) course.



Tip 4: Logging the Last Update Date/Time

One nice thing to add to your dashboards is when they were last updated. In fact, if you liked the tip in the Excel version of this series about showing a message when Pivot Tables sourced from Excel worksheets are out of date, you'll like this as well. While we can't tell without running the query if the data has changed, we can log the last time it was updated, and kick off a message if the last refresh exceeds a certain threshold.

The overall method to do this is as follows:

- Create a query that logs the last refresh date and time.
- Land it in a table.
- Calculate the difference between the current date/time and the date/time of the last refresh.
- Display a message if the difference is greater than the threshold you are comfortable with.

Step 1: Recording the Last Update Date/Time

For this trick, we will again start from a blank query:

- In Excel create a new Query → From Other Sources → Blank Query.
- Rename the query Last Updated.
- In the formula bar type the following formula:

```
=DateTime.LocalNow()
```

This will return something similar to the following:

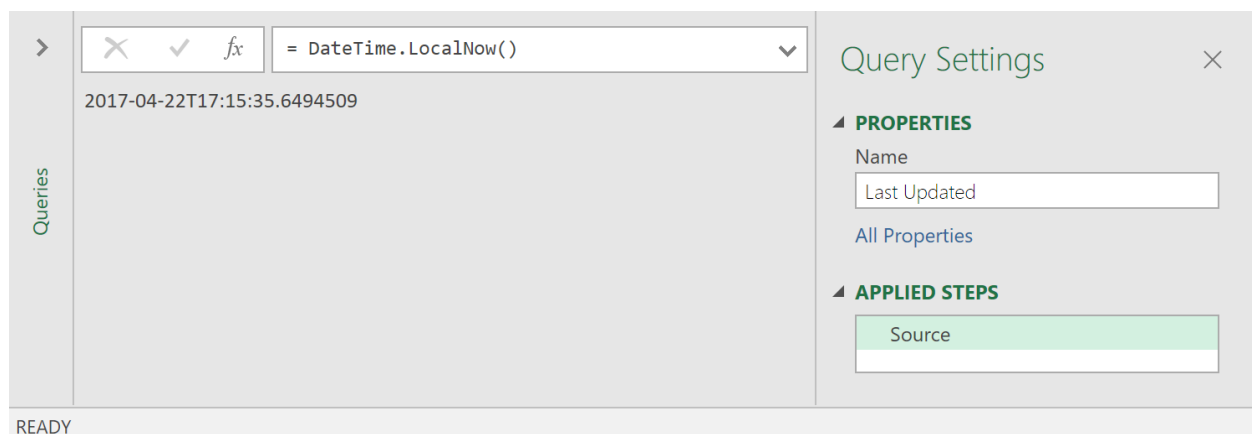


Figure 16 - Creating the Last Updated query

Step 2: Land the Query in a Table

You can now choose to load the query to a worksheet by choosing Home → Close & Load (or Close & Load To if you have modified Power Query's default behaviour.) Even though this query isn't actually returning a table, it will still land it to an Excel table called "Last_Updated" for you.



Step 3: Calculate the Time Difference

Now we need to figure out how often your data should be updated. Daily? Within the last 30 days? Within the last 3 hours, 1 hour, or 15 minutes? This will drive the formulas we need.

Just to keep things simple, we're going to assume the following:

- D2 will hold our current date/time.
- E2 will have the formula to calculate the difference.

In addition, since Power Query loads out to a proper Excel table, it really doesn't matter where it is stored. We can read the last updated value from our table by targeting the table names with the following formula:

`=Last_Updated[Last Updated]`

Here are the formulas required to generate the required differences:

| <i>Difference Required</i> | <i>"Current" Formula (D2)</i> | <i>Difference</i> |
|----------------------------|-------------------------------|--|
| Days | <code>=TODAY ()</code> | <code>=D2-ROUNDDOWN (Last_Updated[Last Updated], 0)</code> |
| Hours | <code>=NOW ()</code> | <code>= (D2-Last_Updated[Last Updated]) *24</code> |
| Minutes | <code>=NOW ()</code> | <code>= (D2-Last_Updated[Last Updated]) *24*60</code> |

Figure 17 below, shows these formulas at work:

| | A | B | C | D | E | F |
|---|----------------|---|---------|---------------|------------|---------|
| 1 | Last Updated ▾ | | | Current | Difference | |
| 2 | 4/21/17 17:15 | | Days | 4/22/2017 | 1.00 | Days |
| 3 | | | Hours | 4/22/17 17:38 | 24.38 | Hours |
| 4 | | | Minutes | 4/22/17 17:38 | 1,462.63 | Minutes |

Figure 17 - Calculating time elapsed since last Power Query refresh

Step 4: Add this into Your Error Checking

As this topic was covered in the Excel book of this series¹, we won't cover that here. Armed with the formulas above, however, you should be able to easily add this into your error checking logic.

¹ See the section on "Show Message if Your Pivot Data is Stale" (pages 2-4) of *DIY BI – Excel Edition* by Excelguru Consulting Inc.



Tip 5: One Setting to Cut (Power) Query Load Time

One of the things that we know about Power Query is that its refresh times can be lengthy. While a few seconds is tolerable, longer refreshes, particularly if you perform them frequently, can be frustrating.

Lengthy refreshes can certainly be in part due to the way you've coded things. But there something that is affecting almost every user with regards to refreshes.

The Effect of Data Privacy Settings

As it turns out, one of the big culprits for increasing the refresh time is the Privacy model employed by Power Query. But before I tell you how big the differences can be, you should know a bit about it and what it is intended to do.

What is Data Privacy?

Power Query includes a Privacy model for data which is intended to stop you from “leaking” data when combining data from different sources. What does this mean? The worry is that you could collect confidential data in a workbook (like social insurance numbers) and try to combine them with data from a server or a web source. As your data may be sent to the server to generate the query, you could accidentally send out private data to a public source, not even realizing you're doing it.

To protect against this potential issue, Microsoft built the Privacy model to allow you to classify your data as public, private or organizational. This prevents you from accidentally sending private or organizational data to a public source and potentially leaking something important.

Some people get around this by just choosing “Organizational” for all their data sets, which may even be correct. But the problem is that the Privacy model still checks all the records anyway, and it causes a significant performance hit along the way.

Where are the Data Privacy Settings found?

The Privacy settings are accessed through the Power Query options, which you can access via the following methods:

- Excel 2016+: Go to the Data tab → Get Data → Query Options
- Excel 2010/2013: Go to the Power Query tab → Options

These options are broken down into two areas; Global and Current Workbook:

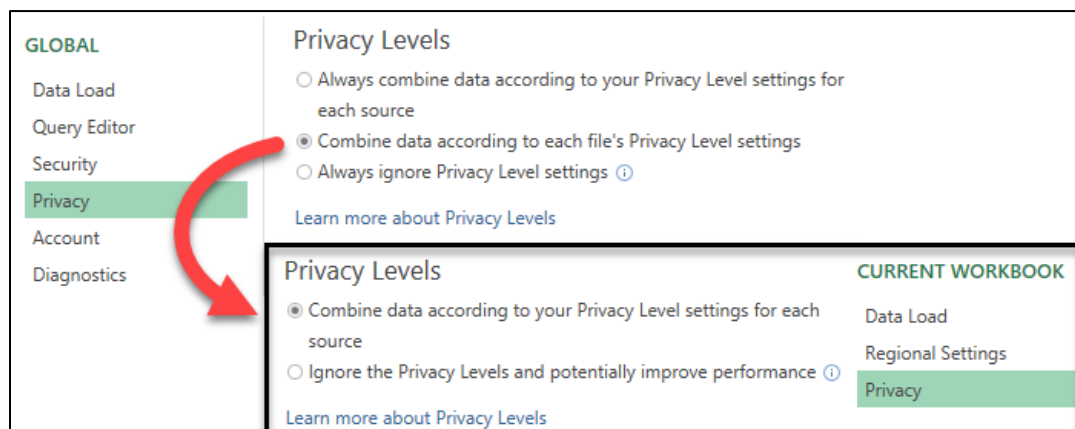


Figure 18 - Privacy level options

Be aware that:

- Current Workbook privacy settings can only be applied if the Global setting is using “Combine data according to each file’s Privacy Level settings”.
- “Always ignore” will turn off privacy and significantly increase your performance.
- The Global “Always combine” is the safest, but slowest setting.

So how much of a difference does privacy make to performance? Good question...

Performance Testing Data Load Options

I’ve tested several models to come to this conclusion, but will focus on some stats from one which is big enough to drive home the point. This model:

- Uses 46 interlinked Power Queries to source its data.
- Data comes from within the Excel workbook only.
- The queries create 8 data model tables.
- Total rows of data in the model are less than 16,000.
- The Query Node Caching fix ([described here](#)) had not been implemented by Microsoft.

The Power Query structure is quite complicated, but it’s not a huge amount of data overall. Based on the model described above, I have tested the load times of some individual tables as well as testing the refresh using Excel “Data → Refresh All” method. I ran each test several times, logging the results, and the results shown in Table 1, below, show the average of those run times in seconds.

Also, in addition to testing the effects of the Privacy model, I also included the setting for Fast Data Load, which is located on the main page of the Query Options dialog.

| Query | Privacy=On Fast Data Load=Off | Privacy=On Fast Data Load=On | Privacy=Ignore Fast Data Load=Off | Privacy=Ignore Fast Data Load=On |
|-------------|----------------------------------|---------------------------------|--------------------------------------|-------------------------------------|
| Categories | ~9 seconds | ~9 seconds | ~6 seconds | ~6 seconds |
| Forecast | ~135 seconds | ~134 seconds | ~45 seconds | ~45 seconds |
| Refresh All | ~180 seconds | ~180 seconds | ~57 seconds | ~57 seconds |

Table 1 - Power Query refresh test results (in seconds)

There are a couple of surprising findings in these results:

1. The Privacy model is dreadfully impactful. Even if your data is all organizational and privacy isn’t a concern, the default settings can significantly extend your refresh times.
2. The Fast Data Load settings seems to be completely ineffective or has been superseded by more efficient refreshes.

For optimal query refresh times – if you are not concerned about Data Privacy – I would suggest setting the Privacy Levels to “Ignore Privacy” and also checking the Fast Data Load option. (Even though the latter doesn’t seem to do anything, it can’t hurt.)

CAUTION!

While the Workbook Level privacy settings do stick between your Excel sessions, they don’t seem to stick with a workbook when you send it to someone else. You can get around this by having your audience set their privacy levels at a global level, but of course the settings will then apply to any workbook they open or create.

02

POWER QUERY

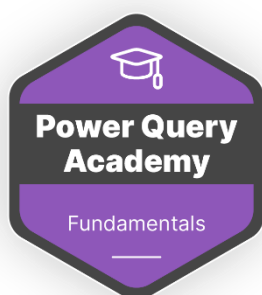




Extend Your Learning

If you are interested in learning more about Power Query, check out our on-demand video course:

[Power Query Academy](#)



Get started for FREE

Sign up for our FREE [Power Query Fundamentals course](#) and start becoming a power user of Excel and Power BI today!

02

POWER QUERY





'DIY BI' Tips, Tricks & Techniques

©2022 Excelguru Consulting Inc.

Don't miss out on the other books in this series!

Register for our newsletter today at

<http://xlguru.ca/newsletter>.

