User Excel, PowerQuery and Yahoo to manage your Portfolio

24TH OF JUNE, 2015 / PETER REID / NO COMMENTS

There are some new and powerful Business Intelligence options available to all of us, from scalable cloud platforms to the Excel on the desktop. So its a good time to demonstrate some of them and show where BI and Application Development meet.

Power Query and the language M have been quietly making their way into the Microsoft Business Intelligence stack and in particular Excel for some time. The set of "Power" components (Power Query, Power Pivot and Power BI) all sound a bit like the marketing guys won the naming battle again and don't really help explain the technology nor where they fit so I avoided it for a long time. That is until this latest version of Office where you can't really avoid it. So it came time to take Power Query and M by the scruff and learn it.

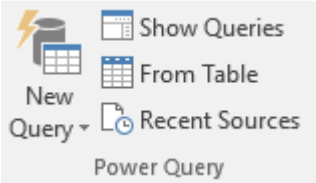
First let's put these new "Power" technologies in their relative places:

- M is a new language that takes on the ET from ETL (Extract Transform and Load). It is a scripting language which looks like an odd mix of VBScript and PowerShell and "R". It has constructs for importing and manipulating Tables, Lists and Records including the ability to do joining and grouping
- Power Query is a scripting environment which is an add-on to Excel adding GUI support for creating and editing M scripts. It hides away the M language from the punter but you'll quickly find the need to drop into the M script editor for all but the basic scripting chores.
- Power Pivot is a multidimensional analysis engine without the need to use the full MSSQL OLAP Analysis Services. PowerPivot is available as a plug in for Excel but executes over what is called the Data Model an in memory database capable of much larger datasets than Excel.
- Power BI Designer is a standalone BI development environment which includes Power Query and PowerPivot. It's for the power business user to develop reports from data without the need to drop into or distribute Excel.

So I set myself a goal to create a spreadsheet that auto updates with stock prices.

First get a version of Excel (2010 or above) which supports Power Query and download the Power Query installer this is already embedded in the later versions of Excel but may need an update. After

installing you should get a set of new buttons like this. The New Query button replaces the previous Get External Data features of Excel and is the entry point to load data into the Power Query environment.



Yahoo Finance

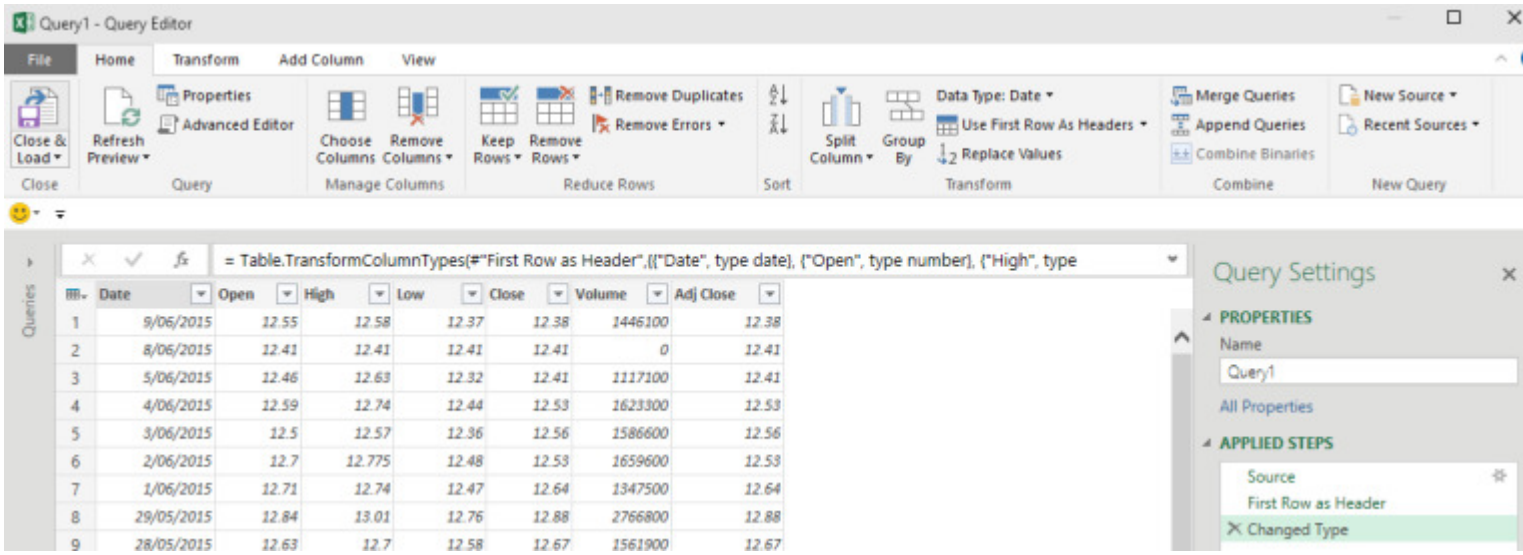
Yahoo Finance have kindly exposed a number of APIs for requesting stock price information (if used for non-commercial purposes), current and historical for just about every instrument you can think of. These APIs are simple web queries but predate the REST API conventions so don't follow nice RESTful best practice. The two services we'll use are:

Historical: http://ichart.finance.yahoo.com/table.csv?s=CPU.AX&a=00&b=4&c=2014&d=07&e=01&f=2015&g=d&ignore=.csv
This will return historical prices between two dates.
Quote: http://download.finance.yahoo.com/d/quotes.csv?s=CPU.AX&f=nsl1op.
This will return the latest quote for the stock CPU.AX with a number of fields identified by the f parameter.

Both APIs are very well documented (counter intuitively) at Google Code
Pulling this data into Power Pivot is really easy from which it's even easier to post process and report.

Historical Query

So let's import one of these services. The New Query button has a huge number of different data sources. For this purpose, we'll use simply "Web" and import the historical data service by pasting in the above URL. This will query the service, display the data and launch the Power Query editor.



Here's our first glimpse of the new data staging area. There's a whole lot of buttons here for transforming and massaging the returned data and everything that is done with the buttons in the GUI become M statements in an M query which can be a great way to learn the language. And there is a data grid (not Excel this one is part of Power Query) to show the query results.

Try hitting the button "Advanced Editor".

```
let

    Source = Csv.Document(Web.Contents("http://ichart.finance.yahoo.com/table.csv?s=CPU.AX&a=00&b=4&c=2014&d=07&e=01&f=2015&g=d&ignore=.csv"),[Delimiter=",",Encoding=1252]),

    #"First Row as Header" = Table.PromoteHeaders(Source),

    #"Changed Type" = Table.TransformColumnTypes(#"First Row as Header",{{"Date", type date}, {"Open", type number}, {"High", type number}, {"Low", type number}, {"Close", type number}, {"Volume", Int64.Type}, {"Adj Close", type number}})

in

    #"Changed Type"
```

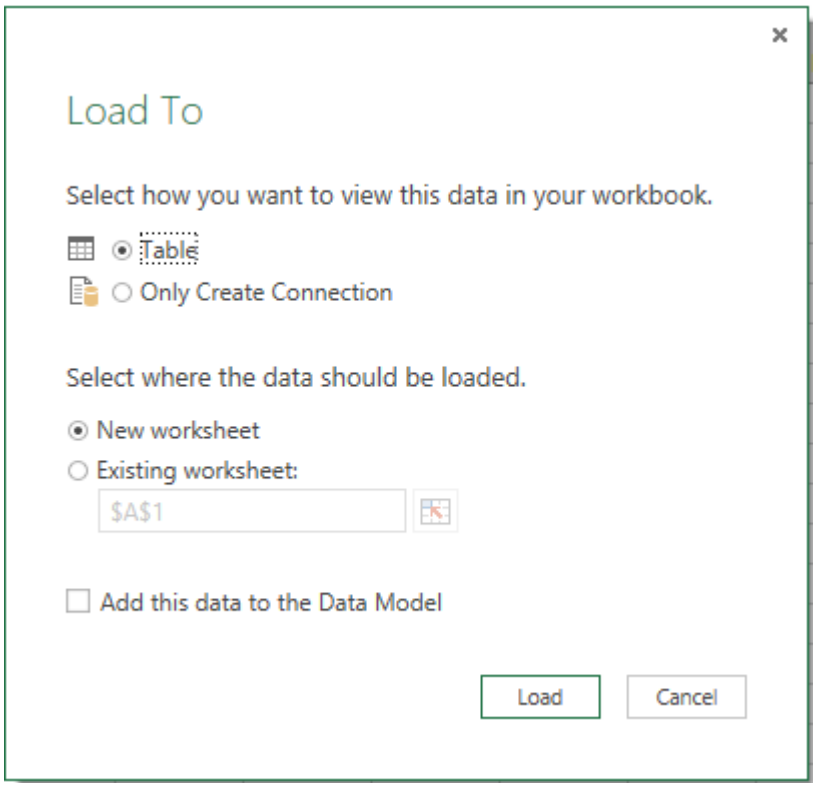
Ah, now we get our first glimpse of M. The Web import wizard created a set of M code that queried the URL, determined the type of data returned (csv), determined the first row is headers and had a guess at the column types of the data and returned that data as a Table.

Breaking down the code:

- A Query starts with (a very BASIC looking) “let”
- A set of case sensitive statements (or Steps) that are separated by commas
- Variables look like #”variablename” are created by assignment and adopt the returned type (much like Powershell a var in C#)
- M is optimised for speed and therefore uses short circuit evaluation; that is a statement line wont execute if it isn’t required for the end value.
- A Query end in an “in” statement to represent the data returned from the query.

Load

Now choose Close and Load To…. This is the point where Power Query tips into Excel or other tool you may be using to display data. The choice here is either to load the query into an Excel Table the PowerPivot Data Model or simply save the code into a stored query “Connection” for later use.



Now that’s given us a Choose Table and you should end up with something like this

Date	Open	High	Low	Close	Volume	Adj Close
6/11/2015	12.29	12.375	12.23	12.35	1405200	12.35
6/10/2015	12.37	12.46	12.27	12.4	830900	12.4
6/9/2015	12.55	12.58	12.37	12.38	1446100	12.38
6/8/2015	12.41	12.41	12.41	12.41	0	12.41
6/5/2015	12.46	12.63	12.32	12.41	1117100	12.41
6/4/2015	12.59	12.74	12.44	12.53	1623300	12.53
6/3/2015	12.5	12.57	12.36	12.56	1586600	12.56

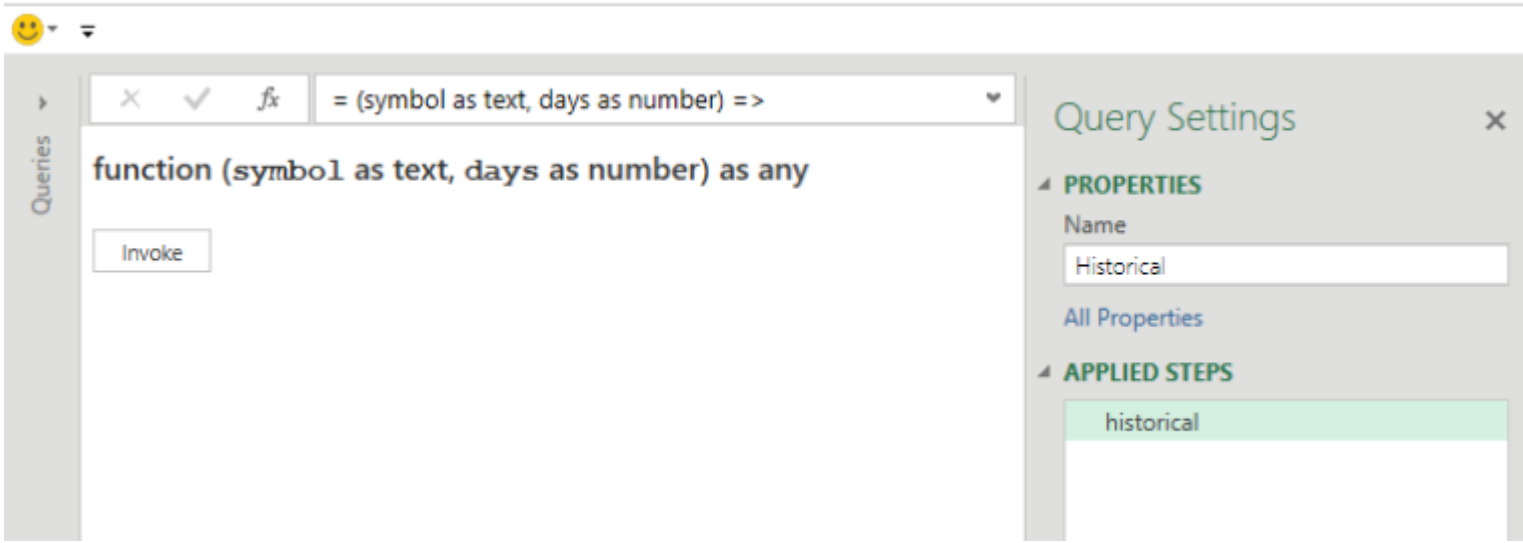
Function

Now wouldn't it be good if we could parameterise that query so it could be used for other stocks and dates. We can upgrade that original query to be a function. Open the Advanced Editor again on the query and overwrite with this.

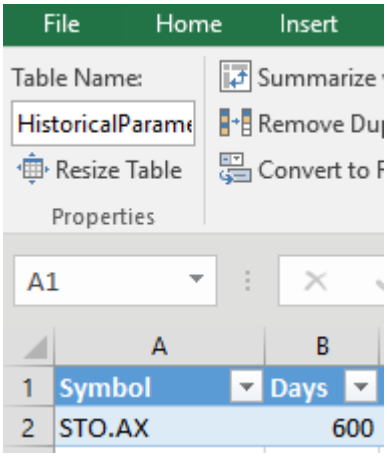
```
let

     historical= (symbol as text, days as number) =>

let

    #"toDate" = DateTime.LocalNow(),

    #"fromDate" = Date.AddDays(#"toDate", 0-days),

    #"fromDateString" = "&a=" & Text.From(Date.Month(fromDate)-1) & "&b=" &
Text.From(Date.Day(fromDate)) & "&c=" & Text.From(Date.Year(fromDate)),

    #"toDateString" = "&d=" & Text.From(Date.Month(toDate)-1) & "&e=" &
Text.From(Date.Day(toDate)) & "&f=" & Text.From(Date.Year(toDate)),

    Source =
Csv.Document(Web.Contents("http://ichart.finance.yahoo.com/table.csv?s=" & symbol &
#"fromDateString" & #"toDateString" &
"&ignore=.csv"),[Delimiter=",",Encoding=1252]),

    #"SourceHeaders" = Table.PromoteHeaders(Source),

    #"Typed" = Table.TransformColumnTypes(#"SourceHeaders",{{"Date", type date},
{"Open", type number}, {"High", type number}, {"Low", type number}, {"Close", type
number}, {"Volume", Int64.Type}, {"Adj Close", type number}})

in

    #"Typed"

in
```

```
historical
```

First thing to note is the code structure now has a query in a query. The outer query defines a function which injects some variables into the inner query. We've parameterised the stock symbol, and the number of days of history. We also get to use some of the useful M library functions as defined in here Power Query Formula Library Specification to manipulate the dates.
Now rename the new "function" as Historical and choose Home=>Close and Load.



At this point it's easy to test the function using the Invoke command which will prompt for the parameters. To execute it from Excel we'll provide the data from Excel cells. In Excel use Insert=>Table to insert a table with two columns named Symbol and Days and name the table "HistoricalParameters". Note we have just created an Excel Table which is different to a Power Query Table, although you can use a Power Query Table to load data into and read data from an Excel Table (confused yet?).



Now it’s time for some more M. New Query=>Other Sources=>Blank Query and paste in this code.

```
let

    Source = Excel.CurrentWorkbook(){[Name="HistoricalParameters"]}[Content],

    #"Symbol"=Source{0}[Symbol],

    #"Days"=Source{0}[Days],

    #"Historical" = fHistorical(#"Symbol",#"Days")

in

    #"Historical"
```

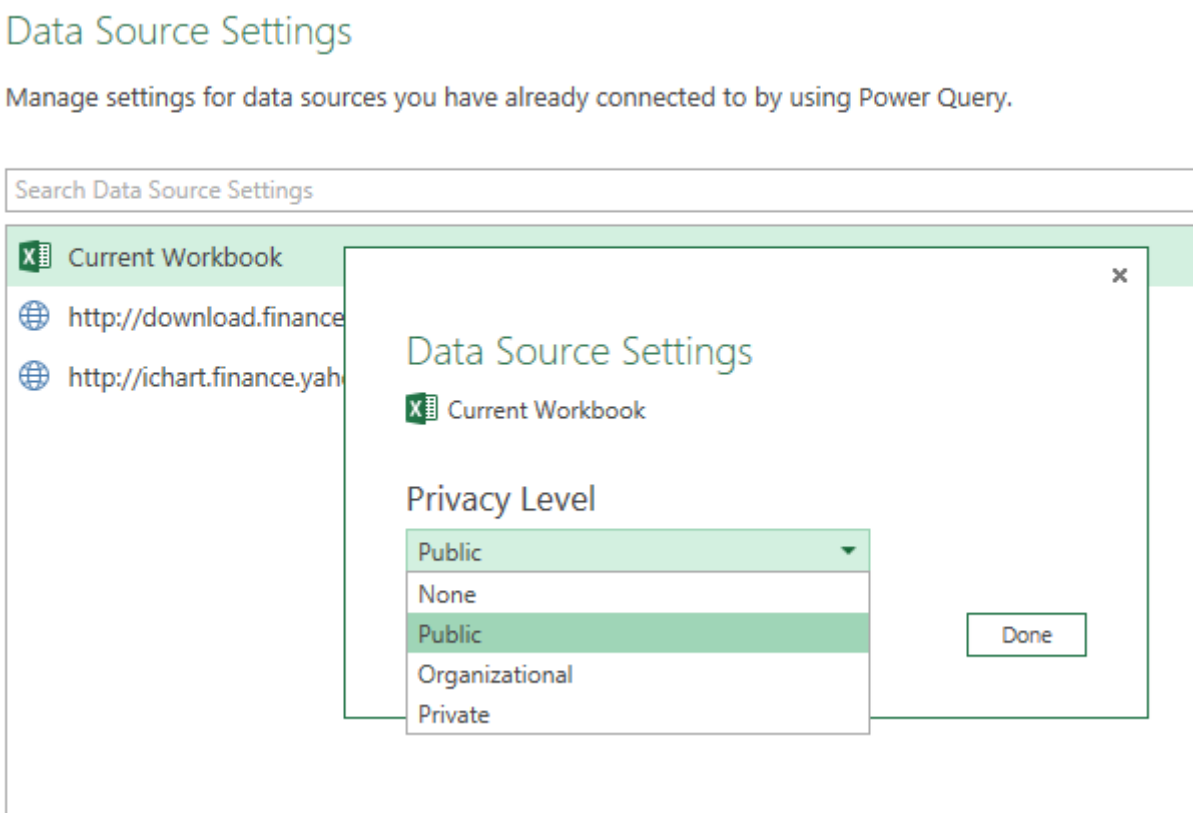
Take the HistoricalParameters Table we created in Excel, pull out the data from the Symbol and Days columns. And use that to call our fHistorical function. Close and Load that query into an Excel table called Historical. I put both the Historical and the HistoricalParameters tables on the same Worksheet so you can update the Symbol and the Days and see the result. It looks like this. To get the query to run with new data either hit the Data Refresh button in Excel or Right Click and Refresh on the Historical table.

	A	B	C	D	E	F	G
1	Symbol	Days					
2	STO.AX	600					
3							
4	Date	Open	High	Low	Close	Volume	Adj Close
5	19/06/2015 0:00	8.35	8.44	8.245	8.36	7186300	8.36
6	18/06/2015 0:00	8.19	8.325	8.125	8.31	9067600	8.31
7	17/06/2015 0:00	8.05	8.17	7.95	8.16	3832600	8.16
8	16/06/2015 0:00	8.1	8.11	7.96	7.99	3035800	7.99
9	15/06/2015 0:00	8.22	8.235	8.09	8.12	3717800	8.12
10	12/06/2015 0:00	8.27	8.35	8.17	8.34	4852900	8.34
11	11/06/2015 0:00	8.17	8.28	8.16	8.27	4950400	8.27
12	10/06/2015 0:00	8.14	8.17	8.02	8.07	4064200	8.07
13	9/06/2015 0:00	7.82	8.14	7.8	8	3655300	8
14	8/06/2015 0:00	7.91	7.91	7.91	7.91	0	7.91
15	5/06/2015 0:00	7.96	7.97	7.79	7.91	3447900	7.91
16	4/06/2015 0:00	8	8.09	7.83	7.96	4549900	7.96
17	3/06/2015 0:00	8.1	8.19	7.98	8	4285000	8
18	2/06/2015 0:00	8.33	8.33	8.08	8.08	4379800	8.08
19	1/06/2015 0:00	8.35	8.38	8.17	8.32	3654900	8.32
20	29/05/2015 0:00	8.23	8.31	8.18	8.25	7970700	8.25
21	28/05/2015 0:00	8.22	8.23	8.11	8.17	3456000	8.17
22	27/05/2015 0:00	8.2	8.31	8.09	8.23	4174900	8.23

Note once you have data in an Excel Table like this, it's easy to add additional calculated columns into the table auto update when the query does.

Formula Firewall

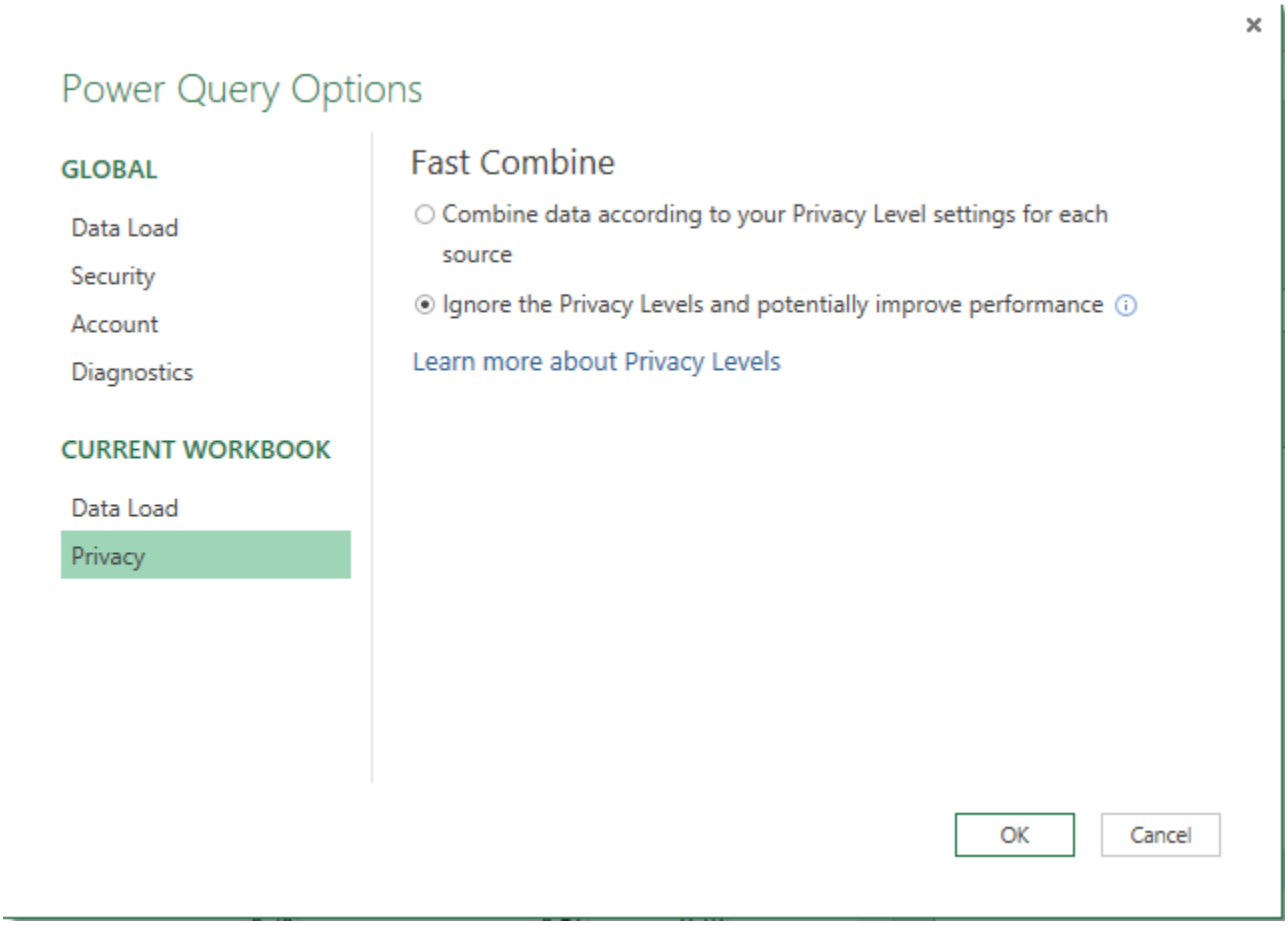
One of the areas that can be confusing is the concept of Privacy Levels. Power Query is built with sharing of queries in mind. Users can share queries with others but a Data Source must be declared with its privacy level to ensure that data from secure sources aren't combined with insecure sources. This is done via the very fancy sounding Formula Firewall.



You can control the privacy level of every data source or but when developing a query the restriction imposed by the firewall can be confusing and can lead to the dreaded error message.

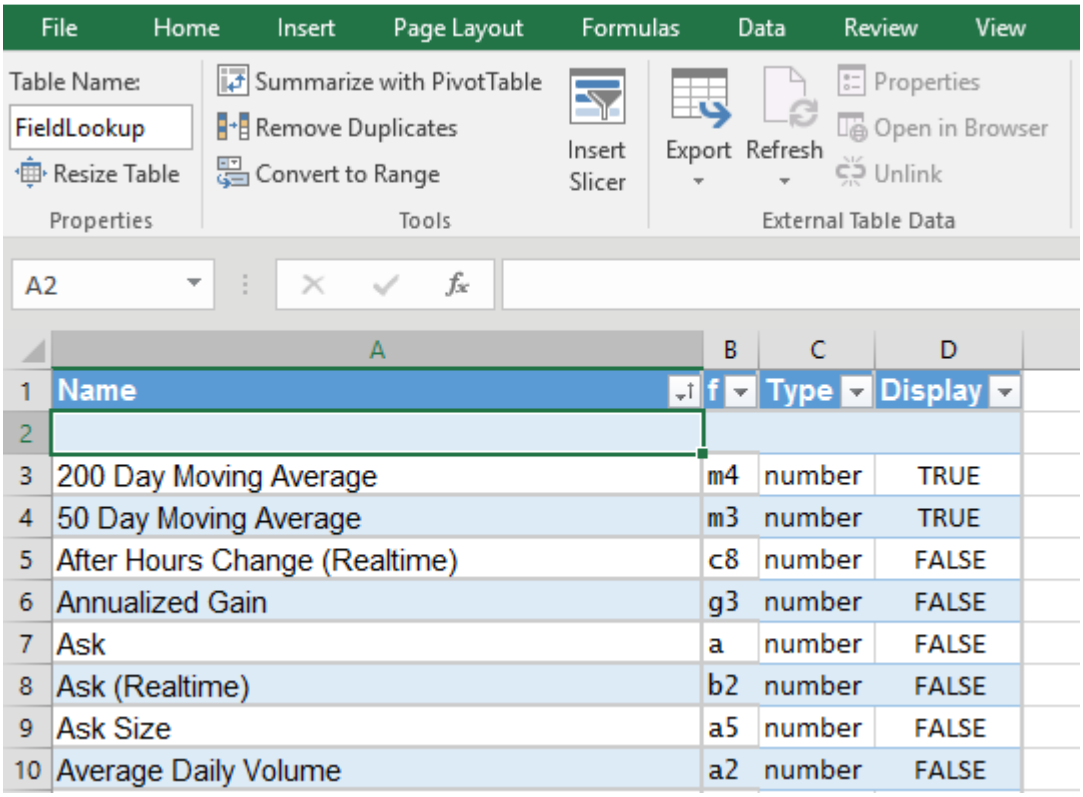
```
"Formula.Firewall: Query 'QueryName' (step 'StepName') references other queries or steps
and so may not directly access a data source. Please rebuild this data combination."
```

So for now I'm going to turn it off. In the Power Query editor window go to File=>Options and Settings=>Options and set the Ignore Privacy Levels option.



Portfolio Query

OK now onto something more complex. The Quote Yahoo URL takes two parameters, s and f. “s” is a comma delimited list of stock symbols and f is a list of fields to return. There are actually 90 fields which can be returned for a stock by concatenating the list of alphanumeric codes for the fields. What would be good is to create a query that can request a number of stocks for a user defined set of columns. The set of codes is well described in this page so let’s take that data and add it into an Excel worksheet and make it into an Excel Table by pasting the data into an Excel Worksheet, highlight the cells and select Insert=>Table which will convert it to an Excel Table and rename it “FieldLookup”. Now rename a couple of columns and add a “Type” and “Display” column. So you end up with something like this.



	A	B	C	D
1	Name	f	Type	Display
2				
3	200 Day Moving Average	m4	number	TRUE
4	50 Day Moving Average	m3	number	TRUE
5	After Hours Change (Realtime)	c8	number	FALSE
6	Annualized Gain	g3	number	FALSE
7	Ask	a	number	FALSE
8	Ask (Realtime)	b2	number	FALSE
9	Ask Size	a5	number	FALSE
10	Average Daily Volume	a2	number	FALSE

The “Display” column is used to determine which fields are to be queried and displayed. The “Type” column is used to encourage Excel to use the right column type when displaying (everything will still work using the default Excel column type, but Numeric and Date columns wont sort nor filter properly).

Now in a new Excel sheet and add table called Portfolio with a single column Symbol and add some stock code symbols. It is this table which will be used to fill with data for all the required stocks and using all of the required display fields.

Now create a new Query

```
let

    PortfolioSource = Excel.CurrentWorkbook(){[Name="Portfolio"]}[Content],

    #"sParam" = Text.Combine(Table.ToList(Table.SelectColumns(PortfolioSource,
"Symbol")),","),



    #"Fields" = Excel.CurrentWorkbook(){[Name="FieldLookup"]}[Content],

    #"DisplayFields" = Table.SelectRows(#"Fields",each [Display]=true),
```

```
    #"fParam" = Text.Combine(Table.ToList(Table.SelectColumns(#"DisplayFields",
"f")),""),



    #"DisplayColumns" = Table.ToList(Table.SelectColumns(#"DisplayFields", "Name")),

    #"TypeNames" = Table.SelectRows(Table.SelectColumns(#"DisplayFields",
{"Name","Type"}), each [Type]="number"),

    #"ColumnTypes" = Table.AddColumn( #"TypeNames", "AsType", each type number),

    #"ColumnTypesList" =
Table.ToRows( Table.SelectColumns(#"ColumnTypes",{"Name","AsType"})),



    #"YahooSource"=
Csv.Document(Web.Contents("http://download.finance.yahoo.com/d/quotes.csv?s=" &
#"sParam" & "&f=" & #"fParam"), #"DisplayColumns"),

    #"TypedYahooSource" =
Table.TransformColumnTypes(#"YahooSource",  #"ColumnTypesList")

in

   #"TypedYahooSource"
```

Here's what is happening

- PortfolioSource: Get the Portfolio table and take just the Symbol column to create a single column Table
- #"sParam": Now combine all the rows of that table with commas separating. This will be the "s" parameter of the yahoo Quotes request
- #"Fields": Now get the table content from the FieldLookup table
- #"DisplayFields": Select just the rows that have a Display column value of true. Here's we see the first use an inline function executed for every row using the "each" keyword (a bit like a C# Linq query).
- #"fParam": Combine all the rows with no separator to create the "f" parameter.
- #"DisplayColumns": Choose the Name column from the table to be used later
- #"TypeNames": Choose the Name and Type column where the type is a number

- #"ColumnTypes": Add a new column *of type number.* This is the first time we see the type system of M there are all the simple types and some additional ones for Record, List and Table etc.
- #"ColumnTypesList": Create a list of lists where each list item is a column name and the number type
- #"YahooSource": Now make the query to yahoo finance with our preprepared "s" and "f" parameters
- #"TypedYahooSource": Run through the returned table and change the column types to number for all required number columns

Now Close and Load the Query. This will generate a query like this.

http://download.finance.yahoo.com/d/quotes.csv?s=BKN.AX,MRM.AX,CAB.AX,ARI.AX,SEK.AX,NEC.AX&f=m4m3c1m5m7p2er1j4e7ql1nt8r5rr2p6r6r7p1p5s6s7st7yvkj
And a Portfolio table like this

Symbol	Name	200 Day Moving Average	50 Day Moving Average	Change	Ch
BKN.AX	BRADKEN FPO	2.83	2.24	-0.1	
MRM.AX	MMAOFFSHOR FPO	0.827	0.598	-0.02	
CAB.AX	CABCHARGE FPO	4.67	4.63	-0.01	
ARI.AX	ARRIUM FPO	0.19	0.17	0	
SEK.AX	SEEK FPO	16.94	16.18	-0.45	
NEC.AX	NINE ENT FPO	1.97	1.98	-0.03	

Every column that has a "true" in the Display column of FieldLookup table will get a column returned with data. To add and remove stocks from the portfolio, just add, remove or edit them right there in the Symbol column of the Portfolio table. Hit Refresh and the data is updated!

Heres the final spreadsheet YahooFinanceStocks
As I was feeling around PowerQuery and M I'm left with a nagging feeling. Do we need another language? And another development environment, without debugging, intellisense, or syntax highlighting? Definitely not!

M is reasonably powerful but seems to have snuck into the Microsoft stack before what I call the "open era". That is Microsoft's new model of embracing open source rather than competing against it. Which brings me to R. R can do all of that and more and there is much more community backing. R is now a language appearing across interesting new Microsoft products like Azure Machine Learning and even inside SQL Server.