

Programming in M and Developing Custom Connectors



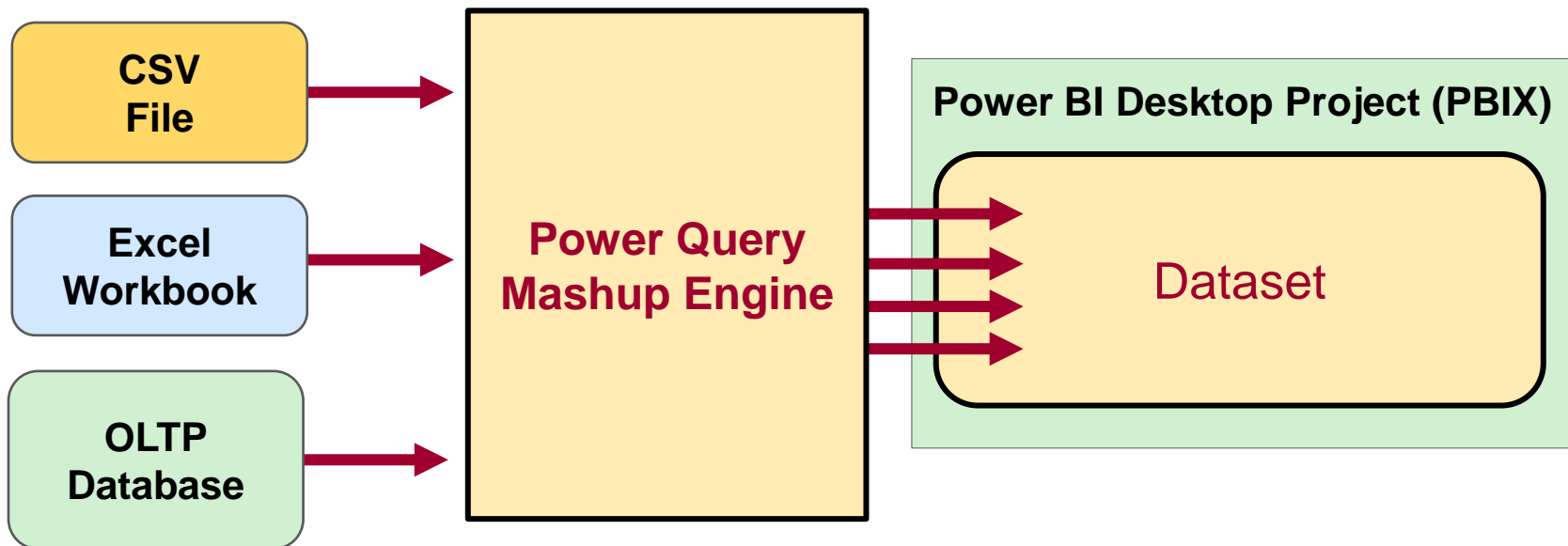
Agenda

- Power Query Mashup Engine
- M Programming Fundamentals
- M Function Library
- Query Functions
- Query Parameters
- Custom Data Connectors



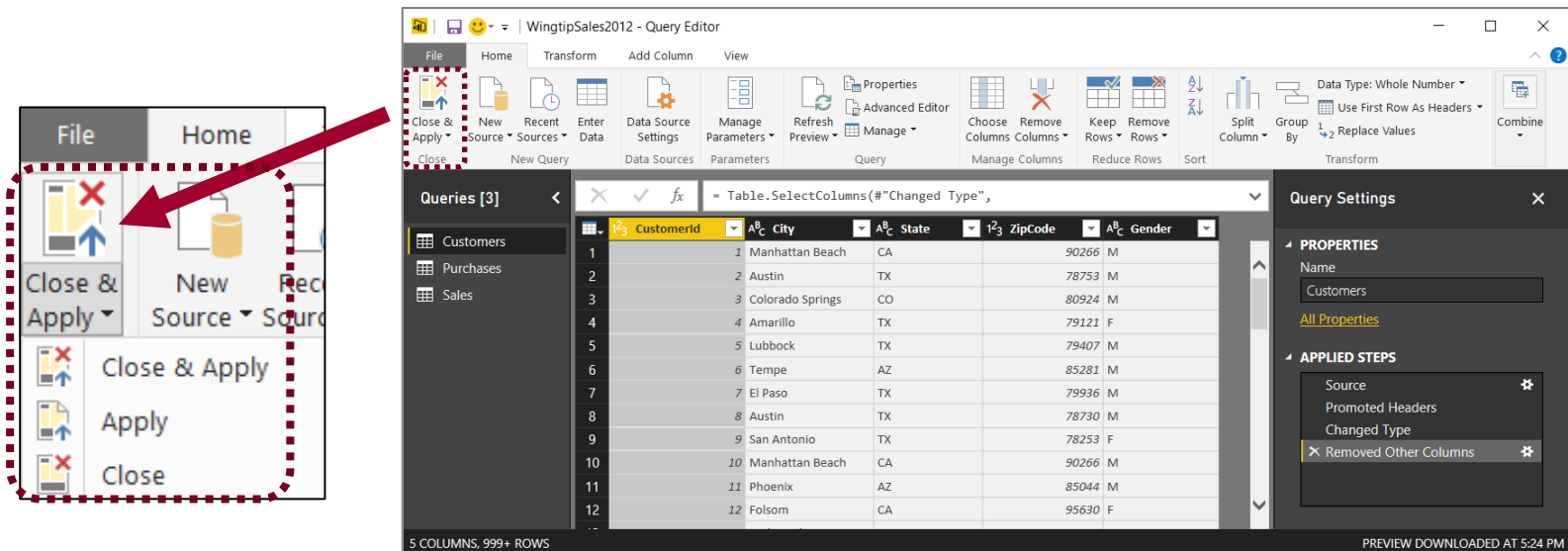
Power BI Desktop is an ETL Tool

- ETL process is essential part of any BI Project
 - **Extract** the data from wherever it lives
 - **Transform** the shape of the data for better analysis
 - **Load** the data into dataset for analysis and reporting



Query Editor Window

- Power BI Desktop provides separate Query Editor window
 - Provides easy-to-use UI experience for designing queries
 - Queries created by creating **Applied Steps**
 - Preview of table generated by query output shown in the middle
 - Query can be executed using **Apply** or **Close & Apply** command



Query Steps

- A query is created as a sequence of steps
 - Each step is a parameterized operation in data processing pipeline
 - Query starts with Source step to extract data from a data source
 - Additional steps added to perform transform operations on data
 - Each step is recorded using M (*aka Power Query Formula Language*)

The screenshot displays the Power Query Editor interface. At the top, the ribbon includes tabs for File, Home, Transform, Add Column, and View. The 'Transform' tab is active, showing options like 'Formula Bar', 'Monospaced', 'Always allow', 'Show whitespace', and 'Advanced Editor'. A red dashed box highlights the 'step formula bar' containing the formula: `= Table.ReplaceValue("#Replaced Female Values","M","Male",Replacer.ReplaceText,`. Below the formula bar is a data table with columns: Customerid, Customer, State, City, Zipcode, and Gender. The table contains 14 rows of customer data. On the right, the 'Query Settings' pane is open, showing the 'APPLIED STEPS' list. A red dashed box highlights this list, which includes: Source, Navigation, Removed Other Columns, Merged Columns, Reordered Columns, Replaced Female Values, Replaced Male Values (selected), Changed Type, and Added Conditional Column. A red arrow points from the 'step formula bar' to the 'APPLIED STEPS' list.

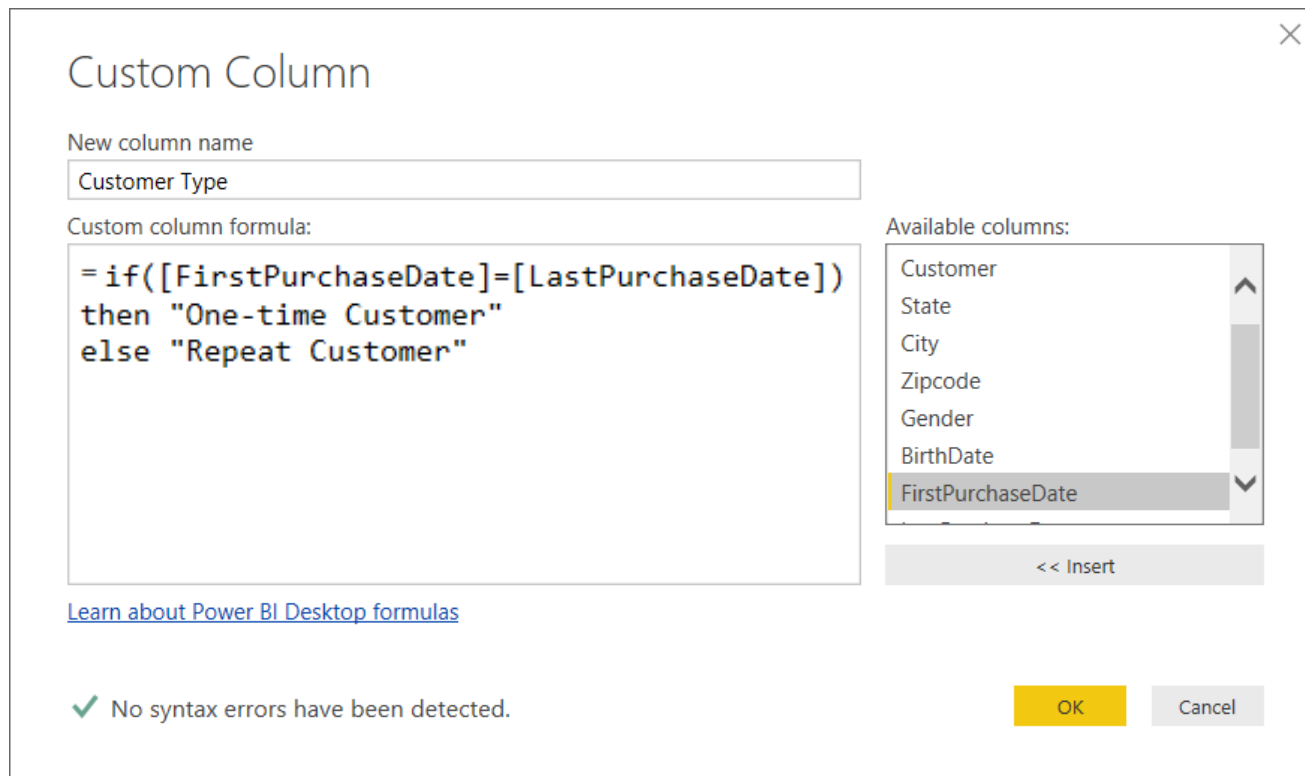
| Customerid | Customer | State | City | Zipcode | Gender |
|------------|-------------------|-------|------------|---------|--------|
| 1 | Nina Diaz | CA | Eureka | 95501 | Female |
| 2 | Melinda Carter | CA | Napa | 94558 | Female |
| 3 | Pam Miller | CA | Napa | 94558 | Female |
| 4 | Merle Blackwell | CA | Sacramento | 95823 | Female |
| 5 | Ariel Hale | CA | Sacramento | 95818 | Male |
| 6 | Randy Carter | CA | Sacramento | 95818 | Male |
| 7 | Lillie Hinton | CA | Eureka | 95501 | Female |
| 8 | Ladonna Moody | CA | Napa | 94559 | Female |
| 9 | Buddy McKay | OR | Bend | 97701 | Male |
| 10 | Warren Sykes | CA | Sacramento | 95818 | Male |
| 11 | Jan Rutledge | OR | Portland | 97216 | Female |
| 12 | Dallas Lester | OR | Eugene | 97402 | Male |
| 13 | Matthew Zimmerman | OR | Portland | 97220 | Male |
| 14 | Sheryl Hernandez | CA | Sacramento | 95823 | Female |

sequential list of steps for query



Custom Column Dialog

- You can write M code directly for custom column
 - The Custom Column dialog provides a simple M code editor



The screenshot shows the 'Custom Column' dialog box in Power BI Desktop. The dialog has a title bar with a close button (X). Inside, there's a section for 'New column name' with a text box containing 'Customer Type'. Below that is the 'Custom column formula:' section with a text area containing the M code: `= if([FirstPurchaseDate]=[LastPurchaseDate])
then "One-time Customer"
else "Repeat Customer"`. To the right of the formula editor is a list of 'Available columns:' including Customer, State, City, Zipcode, Gender, BirthDate, and FirstPurchaseDate (which is highlighted). Below the list is a '<< Insert' button. At the bottom left, there's a green checkmark and the text 'No syntax errors have been detected.' At the bottom right, there are 'OK' and 'Cancel' buttons. A link 'Learn about Power BI Desktop formulas' is located below the formula editor.

Custom Column

New column name
Customer Type

Custom column formula:
`= if([FirstPurchaseDate]=[LastPurchaseDate])
then "One-time Customer"
else "Repeat Customer"`

Available columns:
Customer
State
City
Zipcode
Gender
BirthDate
FirstPurchaseDate

<< Insert

[Learn about Power BI Desktop formulas](#)

✓ No syntax errors have been detected.

OK Cancel



Advanced Editor

or more correctly - The Simple Editor for Advanced Users

- Power BI Desktop based on "M" functional language
 - Query in Power BI Desktop saved as set of M statements in code
 - Query Editor generates code in M behind the scenes
 - Advanced users can view & modify query code in Advanced Editor

The screenshot displays the Power BI Desktop interface with the 'Advanced Editor' window open. The main window shows the 'Query Editor' for a query named 'Customers'. The 'Advanced Editor' window is titled 'Customers' and contains the following M code:

```
let
    Source = Sql.Database("cpt.database.windows.net", "WingtipSalesDB"),
    dbo_Customers = Source[Schema="dbo",Item="Customers"][Data],
    #"Removed Other Columns" = Table.SelectColumns(dbo_Customers,{"CustomerId", "FirstName", "LastName", "City", "State", "ZipCode", "Gender", "BirthDate", "FirstPurchaseDate", "LastPurchaseDate"}),
    #"Merged Columns" = Table.CombineColumns(#"Removed Other Columns",{"FirstName", "LastName"},Combiner.CombineTextByDelimiter(" ", QuoteStyle.None),{"Full Name"}),
    #"Reordered Columns" = Table.ReorderColumns(#"Merged Columns",{"CustomerId", "Customer", "State", "City", "ZipCode", "Gender", "BirthDate", "FirstPurchaseDate", "LastPurchaseDate"}),
    #"Replaced Female Values" = Table.ReplaceValue(#"Reordered Columns", "F", "Female", Replacer.ReplaceText, {"Gender"}),
    #"Replaced Male Values" = Table.ReplaceValue(#"Replaced Female Values", "M", "Male", Replacer.ReplaceText, {"Gender"}),
    #"Changed Type" = Table.TransformColumnTypes(#"Replaced Male Values",{{"BirthDate", type date}, {"FirstPurchaseDate", type date}, {"LastPurchaseDate", type date}}),
    #"Added Custom" = Table.AddColumn(#"Changed Type", "CustomerType", each if [FirstPurchaseDate] = [LastPurchaseDate] then "One-time customer" else "Repeat Customer"),
    #"Removed Columns" = Table.RemoveColumns(#"Added Custom",{"FirstPurchaseDate", "LastPurchaseDate"}),
    #"Renamed Columns" = Table.RenameColumns(#"Removed Columns",{{"CustomerType", "Customer Type"}})
in
    #"Renamed Columns"
```

The 'Advanced Editor' window also shows a status message: 'No syntax errors have been detected.' and buttons for 'Done' and 'Cancel'.

Why Learn M

- Accomplish things that cannot be done in query editor
 - Working with query functions
 - Performing calculations across rows
 - Navigate to SharePoint list by list title instead of GUID with the ID
- Author queries and check them into source control system
 - Add query logic in .m files and store them in GitHub, TFS, etc.
 - Ensure query logic is the same across PBIX projects
- Stay Ahead of the Pack and Win Admiration of Your Peers
 - People will think you are buddies with Chris Webb!



Agenda

- ✓ Power Query Mashup Engine
- M Programming Fundamentals
 - M Function Library
 - Query Functions
 - Query Parameters
 - Custom Data Connectors



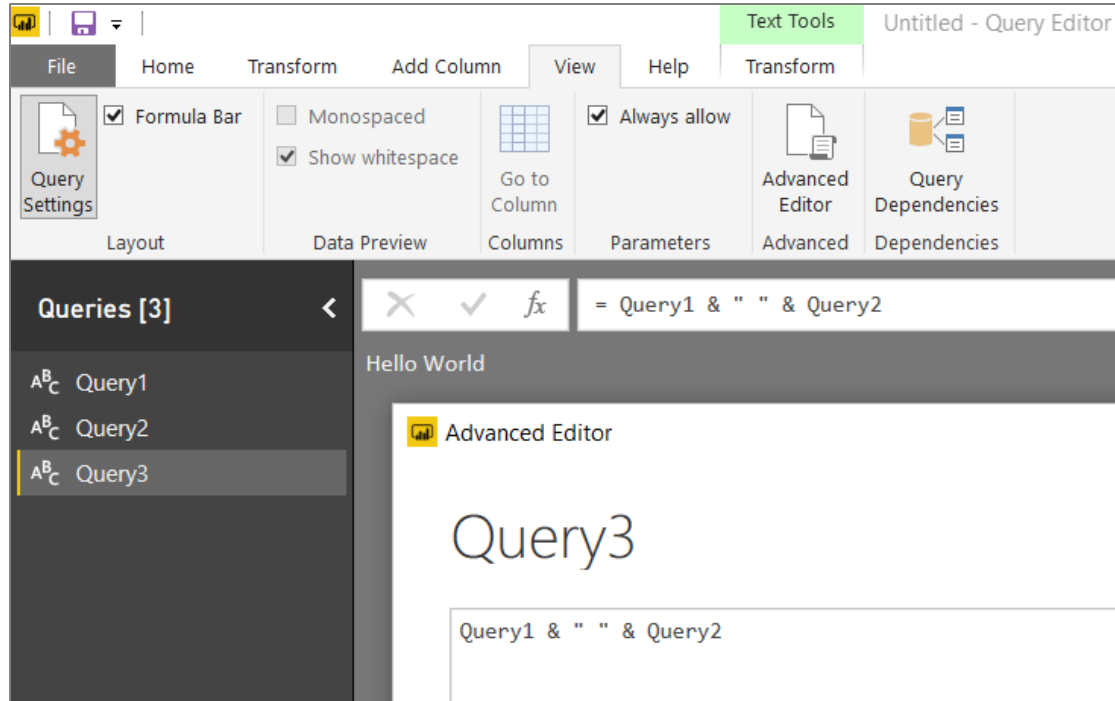
The M Programming Language

- M is a *functional* programming language
 - computation through evaluation of mathematical functions
 - Programming involves writing expressions instead of statements
 - M does not support changing-state or mutable data
 - Every query is a single expression that returns a single value
 - Every query has a return type
- Get Started with M
 - Language is case-sensitive
 - It's all about writing expressions
 - Query expressions can reference other queries by name



Referencing Other Queries

- Query can reference other queries by name
 - Every query is defined with a return type



Let Statement

- Queries usually created using **let** statement
 - Allows a single expressions to contain inner expressions
 - Each line in **let** block represents a separate expression
 - Each line in **let** block has variable which is named step
 - Each line in **let** block requires comma at end except for last line
 - Expression inside **in** block is returned as **let** statement value

The screenshot shows an 'Advanced Editor' window. The main text area displays 'Hello World' at the top. Below it, a 'let' block is defined with four lines of code, each enclosed in a red box and connected by a red arrow to the 'APPLIED STEPS' panel on the right. The code lines are: 'var1 = "Hello",', 'var2 = "World",', 'var3 = var1 & " " & var2,', and 'var4 = Text.Upper(var3)'. The 'in' block contains 'var4'. The 'APPLIED STEPS' panel on the right shows a list of variables: 'var1', 'var2', 'var3', and 'var4'. The 'var4' step is highlighted with a grey background and a red 'X' icon. The 'PROPERTIES' panel on the right shows the 'Name' property set to 'Hello World' and a link to 'All Properties'. At the bottom of the editor, a green checkmark and the text 'No syntax errors have been detected.' are visible.

```
let
  var1 = "Hello",
  var2 = "World",
  var3 = var1 & " " & var2,
  var4 = Text.Upper(var3)
in
  var4
```

✓ No syntax errors have been detected.

PROPERTIES
Name
Hello World
[All Properties](#)

APPLIED STEPS
var1
var2
var3
✕ var4



Comments and Variable Names

- M supports using C-style comments
 - Multiline comments created using `/* */`
 - Single line comments created using `//`

```
/*  
  This is my most excellent query  
*/  
let  
  var1 = 42, // the secret of life
```

- Variable names with spaces must be enclosed in `#" "`
 - Variable names with spaces created automatically by query designer

```
let  
  var1 = "Spaces in ",  
  #"var 2" = "variable names ",  
  #"Bob's your uncle" = "are evil",  
  #"kitchen sink" = var1 & #"var 2" & #"Bob's your uncle"  
in  
  #"kitchen sink"
```

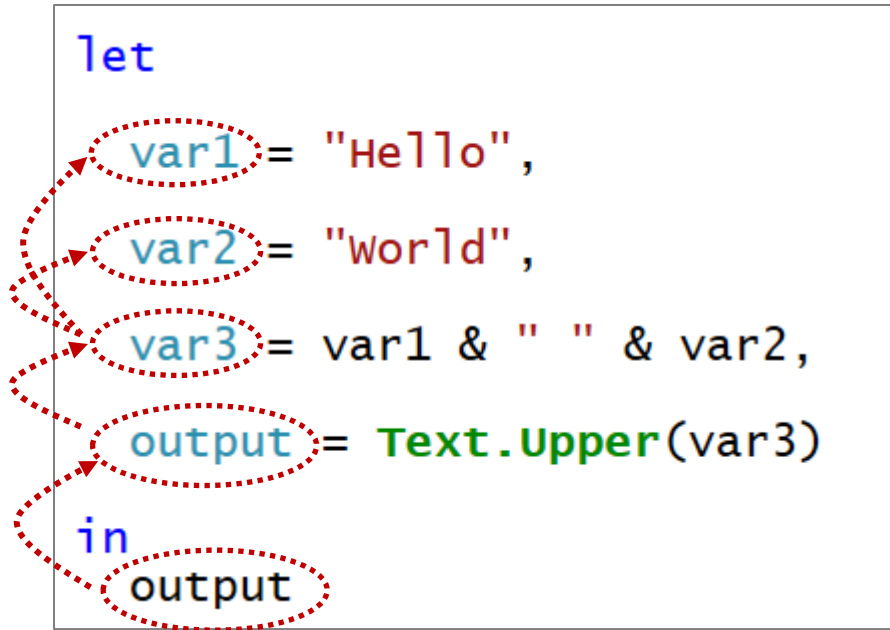
APPLIED STEPS

var1
var 2
Bob's your uncle
✕ Kitchen sink



Flow of Statement Evaluation

- Evaluation starts with expression inside **in** block
 - Expression evaluation triggers other expression evaluation



Will This M Code Work?

- Yes, the Mashup Engine has no problem with this
 - The order of expressions in **let** block doesn't matter
 - However, the visual designer might get confused

```
let
    var4 = Text.Upper(var3),
    var3 = var1 & " " & var2,
    var2 = "world",
    var1 = "Hello"
in
    var4
```

QUERY SETTINGS

▲ PROPERTIES

Name

Hello World Reversed

[All Properties](#)

▲ APPLIED STEPS

Hello World Reversed



Query Folding

- Mashup engine pushes work back to datasource when possible
 - Column selection and row filtering
 - Joins, Group By, Aggregate Operations
- Datasource that support folding
 - Relational database
 - Tabular and multidimensional databases
 - OData Web services
- What happens when datasource doesn't support query folding?
 - All work is done locally by the mashup engine
- Things that affect whether query folding occurs
 - The way you structure your M code
 - Privacy level of datasources
 - Native query execution



Query Folding Example

- When you execute this query in Power BI Desktop...

```
let
    Source = Sql.Database("ODYSSEUS", "WingtipSalesDB"),
    CustomersTable = Source{[Item="Customers"]}[Data],

    // select rows
    FilteredRows = Table.SelectRows(CustomersTable, each ([State] = "FL")),

    // select columns
    ColumnsToKeep = {"CustomerId", "FirstName", "LastName"} ,
    RemovedOtherColumns = Table.SelectColumns(FilteredReaders, ColumnsToKeep),

    // rename columns
    ColumnRenamingMap = { {"FirstName", "First Name"}, {"LastName", "Last Name"} },
    RenamedColumns = Table.RenameColumns(RemovedOtherColumns, ColumnRenamingMap)

in
    RenamedColumns
```

- Mashup Engine executes the following SQL query

```
execute sp_executesql
N'select [__].[CustomerId] as [CustomerId],
        [__].[FirstName] as [First Name],
        [__].[LastName] as [Last Name]
from [dbo].[Customers] as [__]
where [__].[State] = 'FL' and [__].[State] is not null'
```



Native Queries

- No query folding occurs after native query

```
let
    DatabaseServer = "cpt.database.windows.net",
    DatabaseName = "WingtipSalesDB",
    SQL = "SELECT CustomerId, FirstName, LastName" &
        " FROM Customers" &
        " WHERE CustomerId <= 10" &
        " ORDER BY LastName, FirstName" ,
    Source = Sql.Database( DatabaseServer, DatabaseName , [Query=SQL] ),
    output = Source
in
    output
```



M Type System

- Built-in types

any, none

null, logical, number, text, binary

time, date, datetime, datetimezone, duration

- Complex types

list, record, table, function

- User-defined types

- You can create custom types for records and tables



M Datatypes

```
let

  // primitives
  var1 = 123,      // number
  var2 = true,     // boolean
  var3 = "hello",  // text
  var4 = null,     // null

  // creating lists
  list1 = {1, 2, 3},      // list of three numbers

  // accessing list elements
  var5 = list1{1},

  // create records
  record1 = [ FirstName="Soupy", LastName="Sales", ID=3 ],

  // accessing records
  var6 = record1[FirstName],

  // table
  table1 = #table( {"A", "B"}, { {1, 2}, {3, 4} } ),

  // creating function
  function1 = (x) => x * 2,

  // calling function
  output = function1(var1)

in
  output
```



Initializing Dates and Times

```
// time
var1 = #time(09,15,00),

// date
var2 = #date(2013,02,26),

// date and time
var3 = #datetime(2013,02,26, 09,15,00),

// date and time in specific timezone
var4 = #datetimezone(2013,02,26, 09,15,00, 09,00),

// time durection
var5 = #duration(0,1,30,0),
```



Lists

- List is a single dimension array
 - Literal list can be created using `{ }` operators
 - List elements accessed using `{ }` operator and zero-based index

```
let
  RatPack = { "Frank", "Dean", "Sammy" } ,

  FirstRat  = RatPack{0} ,
  SecondRat = RatPack{1} ,
  ThirdRat  = RatPack{2} ,

  output = FirstRat & ", " & SecondRat & " and " & ThirdRat
in
  output
```

- Use `{ }?` to avoid error when index range is out-of-bounds

```
Rat4 = RatPack{4},    // error - index range out of bounds
Rat5 = RatPack{5}? ,  // no error - Rat5 equals null
```



Text.Select

- Text.Select can be used to clean up text value
 - You create a list of characters to include

```
// take a text value with unwanted characters
input = "!!My text has some @bad things !&^",

// get upper and lower case letters
set1 = {"A".. "Z"},
set2 = {"a".. "z"},

// get digits 0-9 and convert to text
set3 = List.Transform({0..9}, each Number.ToText(_)),

// add any other allowed characters
set4 = {" ", "-", "_", "."},

// combine all allowed characters in single list
allowedChars = set1 & set2 & set3 & set4,

// call Text.Select to strip out unwanted characters
output = Text.Select(input, allowedChars)
```



Records

- Record contains fields for single instance of entity

```
// create records by using [] and defining fields
Person1 = [FirstName="Chris", LastName="Webb"],
Person2 = [FirstName="Reza", LastName="Rad"],
Person3 = [FirstName="Matt", LastName="Masson"],

// access field inside a record using [] operator
FirstName1 = Person1[FirstName],
LastName2 = Person2[LastName],
```

- You must often create records to call M library functions

```
// create a record to define HTTP request headers
RequestHeaders = [ Accept="application/json",
                   #"OData-MaxVersion"="4.0" ],

// create a second record which contains the first record
OptionsRecord = [ Headers=RequestHeaders ],

// pass the second record as parameter to web.Contents
Response = web.Contents(url, OptionsRecord),
```



Combination Operator (&)

- Used to combine strings, arrays and records

```
// text concatenation: "ABC"  
var1 = "A" & "BC",  
  
// list concatenation: {1, 2, 3}  
var2 = {1} & {2, 3},  
  
// record merge: [ a = 1, b = 2 ]  
var3 = [ a = 1 ] & [ b = 2 ],
```



Table.FromRecords

- Table.FromRecords can be used to create table
 - Table columns are not strongly typed

```
let
    CustomersTable = Table.FromRecords({
        [ FirstName="Matt", LastName="Masson"],
        [ FirstName="Chris", LastName="Webb"],
        [ FirstName="Reza", LastName="Rad"],
        [ FirstName="Chuck", LastName="Sterling"]
    })
in
    CustomersTable
```

| | ABC 123 FirstName | ABC 123 LastName |
|---|-------------------------|------------------------|
| 1 | Matt | Masson |
| 2 | Chris | Webb |
| 3 | Reza | Rad |
| 4 | Chuck | Sterling |

ABC
123

Bad, Bad, Bad ☹️



Creating User-defined Types

- M allows you to create user-defined types
 - Here is a user-defined type for a record and a table

```
CustomerRecordType = type [FirstName = text, LastName = text],  
CustomerTableType = type table CustomerRecordType,
```

- User-defined table used to create table with strongly typed columns

```
let  
    CustomerRecordType = type [FirstName = text, LastName = text],  
    CustomerTableType = type table CustomerRecordType,  
    CustomerTable =  
        #table(CustomerTableType, {  
            { "Matt", "Masson" },  
            { "Chris", "Webb" },  
            { "Reza", "Rad" },  
            { "Chuck", "Sterilicious" }  
        })  
in  
    CustomerTable
```

| | AB C FirstName | AB C LastName |
|---|----------------|---------------|
| 1 | Matt | Masson |
| 2 | Chris | Webb |
| 3 | Reza | Rad |
| 4 | Chuck | Sterilicious |




Using Each with Unary Functions

- Many library functions take function as parameters
 - Function parameters are often unary (*e.g. they accept 1 parameter*)

```
FilteredRows = Table.SelectRows(CustomersTable, (row) => row[CustomerId]<=10 ),
```

- M provides **each** syntax to make code easier to read/write
 - Unary parameter passed implicitly using **_** variable

```
FilteredRows = Table.SelectRows(CustomersTable, each _[CustomerId]<=10 ),
```




- You can omit **_** variable when accessing fields inside record

```
FilteredRows = Table.SelectRows(CustomersTable, each [CustomerId]<=10 ),
```

```
AddedColumn =Table.AddColumn(FilteredRows, "Display Name", each [FirstName] & " " & [LastName])
```

- You must use **_** variable when using **each** with a list

```
MyList = { "Item 1", "Item 2", "Item 3" },  
MyUpperCaseList = List.Transform(MyList, each Text.Upper(_) )
```



Performing Calculations Across Rows

- Requires adding an index column

| | Quarter | \$ Sales | 1.2 Index | \$ Running Total |
|---|---------|----------|-----------|------------------|
| 1 | 2016-Q1 | 124 | 0 | 124 |
| 2 | 2016-Q2 | 154 | 1 | 278 |
| 3 | 2016-Q3 | 167 | 2 | 445 |
| 4 | 2016-Q4 | 188 | 3 | 633 |
| 5 | 2017-Q1 | 150 | 4 | 783 |
| 6 | 2017-Q2 | 193 | 5 | 976 |
| 7 | 2017-Q3 | 208 | 6 | 1184 |
| 8 | 2017-Q4 | 234 | 7 | 1418 |

PROPERTIES

Name

Sales Running Total

[All Properties](#)

APPLIED STEPS

Source

AddedIndex

✕ AddedCustom

Custom Column

New column name

Running Total

Custom column formula:

= List.Sum(List.Range(AddedIndex[Sales], 0, [Index]+1))



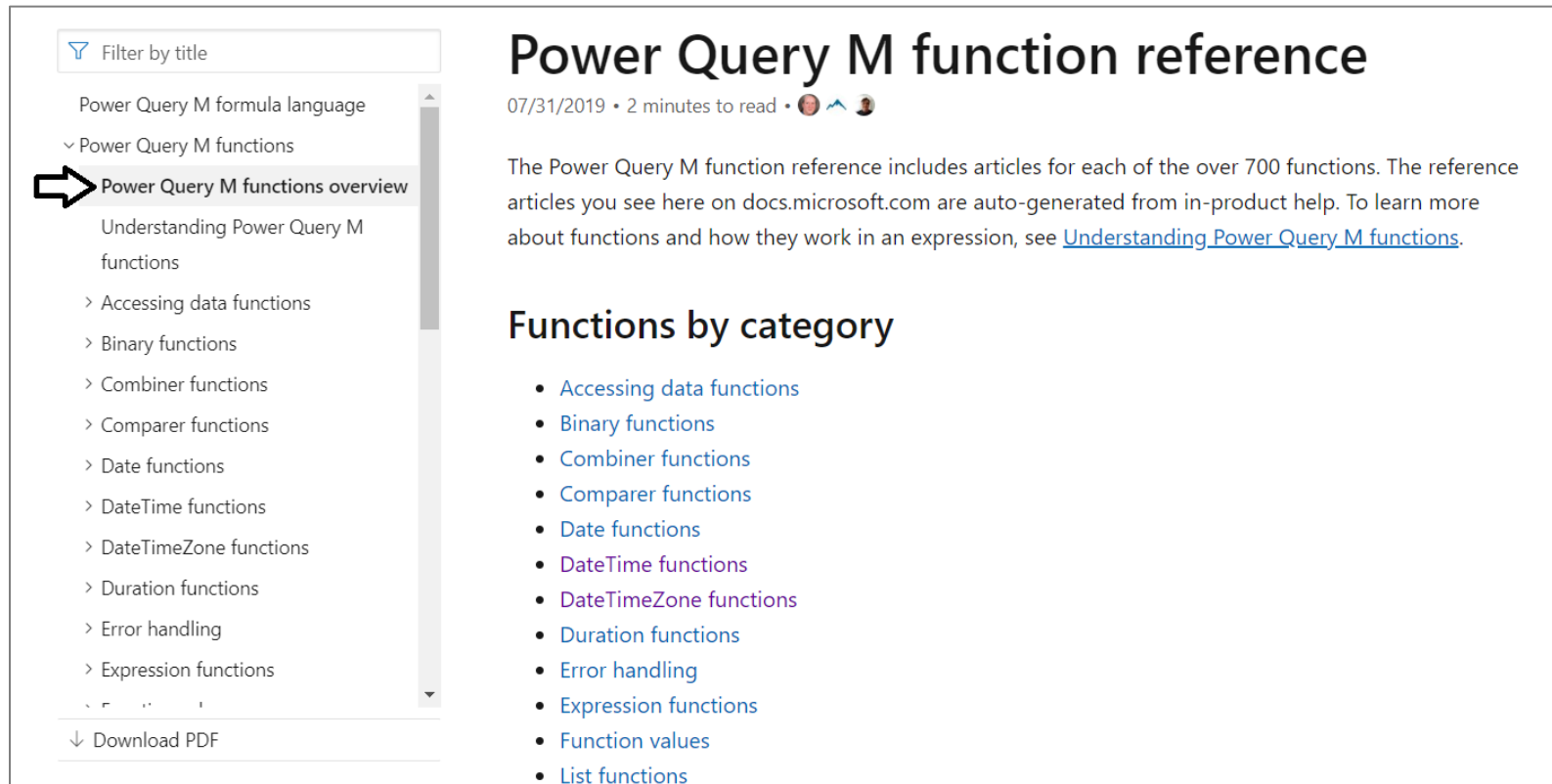
Agenda

- ✓ Power Query Mashup Engine
- ✓ M Programming Fundamentals
- M Function Library
 - Query Functions
 - Query Parameters
 - Custom Data Connectors



M Function Library

- Check out the Power Query M function reference
 - <https://docs.microsoft.com/en-us/powerquery-m/power-query-m-function-reference>



Filter by title

Power Query M formula language

▼ Power Query M functions

➡ Power Query M functions overview

Understanding Power Query M functions

› Accessing data functions

› Binary functions

› Combiner functions

› Comparer functions

› Date functions

› DateTime functions

› DateTimeZone functions




› Duration functions

› Error handling

› Expression functions

Download PDF

Power Query M function reference

07/31/2019 • 2 minutes to read •   

The Power Query M function reference includes articles for each of the over 700 functions. The reference articles you see here on docs.microsoft.com are auto-generated from in-product help. To learn more about functions and how they work in an expression, see [Understanding Power Query M functions](#).

Functions by category

- [Accessing data functions](#)
- [Binary functions](#)
- [Combiner functions](#)
- [Comparer functions](#)
- [Date functions](#)
- [DateTime functions](#)
- [DateTimeZone functions](#)
- [Duration functions](#)
- [Error handling](#)
- [Expression functions](#)
- [Function values](#)
- [List functions](#)



Accessing Data using OData.Feed

- OData.Feed can pull data from OData web service
 - OData connector assists with navigation through entities
 - OData connector support query folding

```
let
    Source = OData.Feed("http://subliminalsystems.com/api/"),

    // get Customers table
    CustomersTable = Source[{Name="Customers",Signature="table"}][Data],

    // select columns
    ColumnsToKeep = {"CustomerId", "FirstName", "LastName", "City", "State", "Zipcode", "Gender", "BirthDate"},
    RemovedOtherColumns = Table.SelectColumns(CustomersTable, ColumnsToKeep),

    // select rows
    FilteredRows = Table.SelectRows(RemovedOtherColumns, each [CustomerId] <= 10),

    // perform other transforms
    ReplacedValue = Table.ReplaceValue(FilteredRows,"F","Female",Replacer.ReplaceText,{"Gender"}),
    ReplacedValue1 = Table.ReplaceValue(ReplacedValue,"M","Male",Replacer.ReplaceText,{"Gender"}),
    ChangedType = Table.TransformColumnTypes(ReplacedValue1,{{"BirthDate", type date}}),
    MergedColumns = Table.CombineColumns(ChangedType,{"FirstName", "LastName",
                                                    Combiner.CombineTextByDelimiter(" ", QuoteStyle.None),
                                                    "Customer"})

in
    MergedColumns
```

- OData makes extra calls to acquire metadata
 - Let's look at the execution of this query using Fiddler



Web.Contents

- Can be more efficient than OData.Feed
 - You can pass OData query string parameters (e.g. \$select)

```
let
    // create REST URI for OData source
    Source = "http://subliminalsystems.com/api/Customers?" &
        "?$select=CustomerId,FirstName,LastName,City,State,Zipcode,Gender,BirthDate" &
        "&filter=(CustomerId+1e+10)",

    // create options record for calling Web.Contents
    OptionsRecord = [Headers=[Accept="application/json;odata=nometadata",
        #"OData-MaxVersion"="4.0"]],

    // call Web.Content to make call across network
    WebContents = Web.Contents(Source, OptionsRecord),

    // deal with JSON dataset return by Web.Contents
    JsonDocument = Json.Document(WebContents),
    RecordList = Record.ToTable(JsonDocument){1}[Value],
    Table = Table.FromList(RecordList, Splitter.SplitByNothing(), null, null, ExtraValues.Error),
    ColumnsToExpand = {"CustomerId", "FirstName", "LastName", "City", "State", "Zipcode", "Gender", "BirthDate"},
    ExpandedColumns = Table.ExpandRecordColumn(Table, "Column1", ColumnsToExpand, ColumnsToExpand),
```



Agenda

- ✓ Power Query Mashup Engine
- ✓ M Programming Fundamentals
- ✓ M Function Library
- Query Functions
 - Query Parameters
 - Custom Data Connectors



Understanding Function Queries

- Query can be converted into reusable function
 - Requires editing query M code in Advanced Editor
 - Function query defined with one or more parameters

```
GetExpensesFromFile

(filePath as text) =>

let
    Source = Csv.Document(Web.Contents(filePath),
        #"Changed Type" = Table.TransformColumnTypes
```

- Function query can be called from other queries
- Function query can be called using Invoke Custom Function
- Function query can't be edited with visual designer



List.Generate

- **List.Generate** accepts 3 function parameters

```
MyList = List.Generate( ()=>1, (item)=>(item<=10), (item)=>(item+1) )
```

- You can use **each** syntax for 2nd and 3rd parameter

```
MyList = List.Generate( ()=>1, each _<=10, each _+1 )
```

- You can optionally split functions out into separate expressions

```
let
    StartFunction = ()=>1,
    TestFunction = each _ <= 10,
    IncrementFunction = each _ + 1,
    MyList = List.Generate( StartFunction, TestFunction, IncrementFunction)
in
    MyList
```

| List | |
|------|----|
| 1 | 1 |
| 2 | 2 |
| 3 | 3 |
| 4 | 4 |
| 5 | 5 |
| 6 | 6 |
| 7 | 7 |
| 8 | 8 |
| 9 | 9 |
| 10 | 10 |



Agenda

- ✓ Power Query Mashup Engine
- ✓ M Programming Fundamentals
- ✓ M Function Library
- ✓ Query Functions
- Query Parameters
- Custom Data Connectors



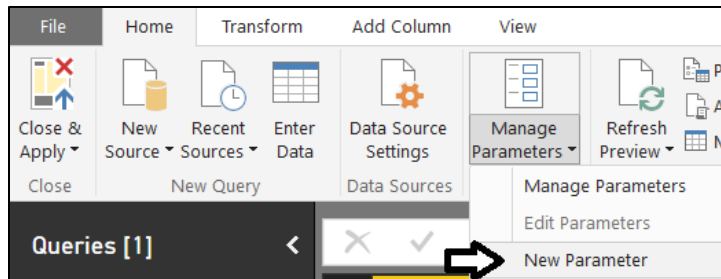
Query Parameters

- What is a Query Parameter?
 - Configurable setting with project scope
 - Strongly-typed value to which you can apply restrictions
 - Can be referenced from a query
 - Selected values can be populated using list
- Where are Parameters commonly used
 - To parameterize data source connection details
 - To filter rows when importing data



Creating Query Parameters

- Parameters can be created using **Manager Parameters** menu



- Parameter properties
 - Name
 - Description
 - Required
 - Allowed Values
 - Default Value
 - Current Value

A screenshot of the 'Parameters' dialog box in Power BI. The dialog box has a title bar 'Parameters' and a close button. It contains a list of parameters on the left, with 'Customer State' selected. The right pane shows the properties for 'Customer State':

- Name:** Customer State
- Description:** This parameter is used in the Customers query to filter the customer rows which are loaded into the dataset for the Power BI Desktop project.
- Required:** ☒
- Type:** Text
- Allowed Values:** List of values
- Table of Allowed Values:**

| | |
|---|----|
| 1 | CA |
| 2 | OR |
| 3 | WA |
| 4 | AZ |
| 5 | TX |
| * | |
- Default Value:** CA
- Current Value:** CA

At the bottom right, there are 'OK' and 'Cancel' buttons.



Referencing Parameters in a Query

- Parameters can be referenced inside query
 - Next query execution uses current parameter value

The screenshot shows a 'Filter Rows' dialog box with a close button (X) in the top right corner. It has two tabs: 'Basic' (unselected) and 'Advanced' (selected). Below the tabs, it says 'Show rows where:'. There are two columns: 'And/Or' and 'Column'. The 'And/Or' column has a dropdown menu with 'And' selected. The 'Column' column has a dropdown menu with 'State' selected. The 'Operator' column has a dropdown menu with 'equals' selected. The 'Value' column has a dropdown menu with 'Customer State' selected. There is a '...' button to the right of the 'Value' dropdown. Below the filter rules, there is an 'Add Clause' button. At the bottom right, there are 'OK' and 'Cancel' buttons.

| And/Or | Column | Operator | Value |
|--------|--------|----------|----------------|
| | State | equals | Customer State |
| And | State | | ABC |

Buttons: Add Clause, OK, Cancel



Creating a Project Template File

The screenshot displays the Power BI Desktop application window. The title bar reads 'Untitled - Power BI Desktop'. The ribbon includes tabs for File, Home, View, Modeling, and Help. The Home tab is active, showing various tool groups: Clipboard (Paste, Cut, Copy, Format Painter), External data (Get Data, Recent Sources, Enter Data, Edit Queries, Refresh), Insert (New Page, New Visual, Ask A Question, Text box, Image, Shapes), Custom visuals (From Marketplace, From File), Themes (Switch Theme), Relationships (Manage Relationships), and Calculations (New Measure, New Column, New Quick Measure).

The main content area shows a report titled 'Giants Team Roster'. It features a table with columns: City, Conference, Division, and Roster. The data row shows: East Rutherford, NJ | NFC | NFC-East. Below the table is a map titled 'City and Team' showing the United States with a green dot indicating the location of East Rutherford, NJ. To the right of the map are two tables: 'Offensive Players' and 'Defensive Players'.

| Number | Player | Position | College |
|--------|------------------|------------------|---------------------|
| 2 | Aldrick Rosas | Kicker | Southern Oregon |
| 3 | Geno Smith | Quarterback | West Virginia |
| 5 | Davis Webb | Quarterback | California |
| 9 | Brad Wing | Punter | LSU |
| 10 | Eli Manning | Quarterback | Mississippi |
| 12 | Tavarres King | Wide Receiver | Georgia |
| 13 | Odell Beckham Jr | Wide Receiver | LSU |
| 15 | Brandon Marshall | Wide Receiver | Central Florida |
| 17 | Dwayne Harris | Wide Receiver | East Carolina |
| 18 | Roger Lewis | Wide Receiver | Bowling Green State |
| 19 | Travis Rudolph | Wide Receiver | Florida State |
| 22 | Wayne Gallman | Running Back | Clemson |
| 26 | Orleans Darkwa | Running Back | Tulane |
| 28 | Paul Perkins | Running Back | UCLA |
| 34 | Shane Vereen | Running Back | California |
| 43 | Shane Smith | Tight End | San Jose State |
| 51 | Zak DeOssie | Long Snapper | Brown |
| 61 | Nick Becton | Offensive Tackle | Virginia Tech |
| 63 | Chad Wheeler | Offensive Tackle | USC |
| 65 | Jessamen Dunker | Offensive Tackle | Tennessee State |
| 66 | Adam Bisnowaty | Offensive Tackle | Pittsburgh |
| 67 | Justin Pugh | Guard | Syracuse |
| 69 | Brett Jones | Center | Regina (Canada) |

| Number | Player | Position |
|--------|-----------------------------|--------------------|
| 20 | Janoris Jenkins | Cornerback |
| 21 | Landon Collins | Safety |
| 23 | Darryl Morris | Defensive Back |
| 24 | Eli Apple | Cornerback |
| 25 | Brandon Dixon | Cornerback |
| 27 | Darian Thompson | Safety |
| 29 | Nat Berhe | Safety |
| 33 | Andrew Adams | Safety |
| 36 | Ryan Murphy | Safety |
| 37 | Ross Cockrell | Cornerback |
| 38 | Donte Deayon | Cornerback |
| 39 | Derrick Mathews | Linebacker |
| 41 | Dominique Rodgers-Cromartie | Cornerback |
| 44 | Mark Herzlich | Linebacker |
| 46 | Calvin Munson | Linebacker |
| 47 | Kelvin Sheppard | Middle Linebacker |
| 48 | Akeem Ayers | Linebacker |
| 52 | Jonathan Casillas | Linebacker |
| 54 | Olivier Vernon | Defensive End |
| 55 | Ray-Ray Armstrong | Outside Linebacker |
| 57 | Keenan Robinson | Linebacker |
| 58 | Curtis Grant | Linebacker |
| 59 | Devon Kennard | Linebacker |



The Template File Implementation

- Solution required advanced query design

Queries [8]

Data [3]

Teams

Team List

Position Codes

Parameters [2]

Team (Eagles)

RosterUrl

Extract [1]

GetPlayers

Data Model [2]

Players

Team Details

✕

✓

f_x

= #table(type table

| | ^A _C Team | ^A _C Division | ^A _C Conference | ^A _C City | ^A _C RosterUrl |
|----|--------------------------------|------------------------------------|--------------------------------------|--------------------------------|-------------------------------------|
| 1 | Bills | AFC-East | AFC | Buffalo, NY | http://www.buffalo |
| 2 | Dolphins | AFC-East | AFC | Miami, FL | http://www.miami |
| 3 | Jets | AFC-East | AFC | East Rutherford, NJ | http://www.newyork |
| 4 | Patriots | AFC-East | AFC | Foxboro, MA | http://www.patriots |
| 5 | Bengals | AFC-North | AFC | Cincinnati, OH | http://www.bengals |
| 6 | Browns | AFC-North | AFC | Cleveland, OH | http://www.cleveland |
| 7 | Ravens | AFC-North | AFC | Baltimore, MD | http://www.baltimore |
| 8 | Steelers | AFC-North | AFC | Pittsburg, PA | http://www.steelers |
| 9 | Colts | AFC-South | AFC | Indianapolis, IN | http://www.colts.com |
| 10 | Jaguars | AFC-South | AFC | Jacksonville, FL | http://www.jaguars |
| 11 | Texans | AFC-South | AFC | Houston, TX | http://www.houston |
| 12 | Titans | AFC-South | AFC | Nashville, TN | http://www.titans.com |
| 13 | Broncos | AFC-West | AFC | Denver, CO | http://www.denver |



Agenda

- ✓ Power Query Mashup Engine
- ✓ M Programming Fundamentals
- ✓ M Function Library
- ✓ Query Functions
- ✓ Query Parameters
- Custom Data Connectors

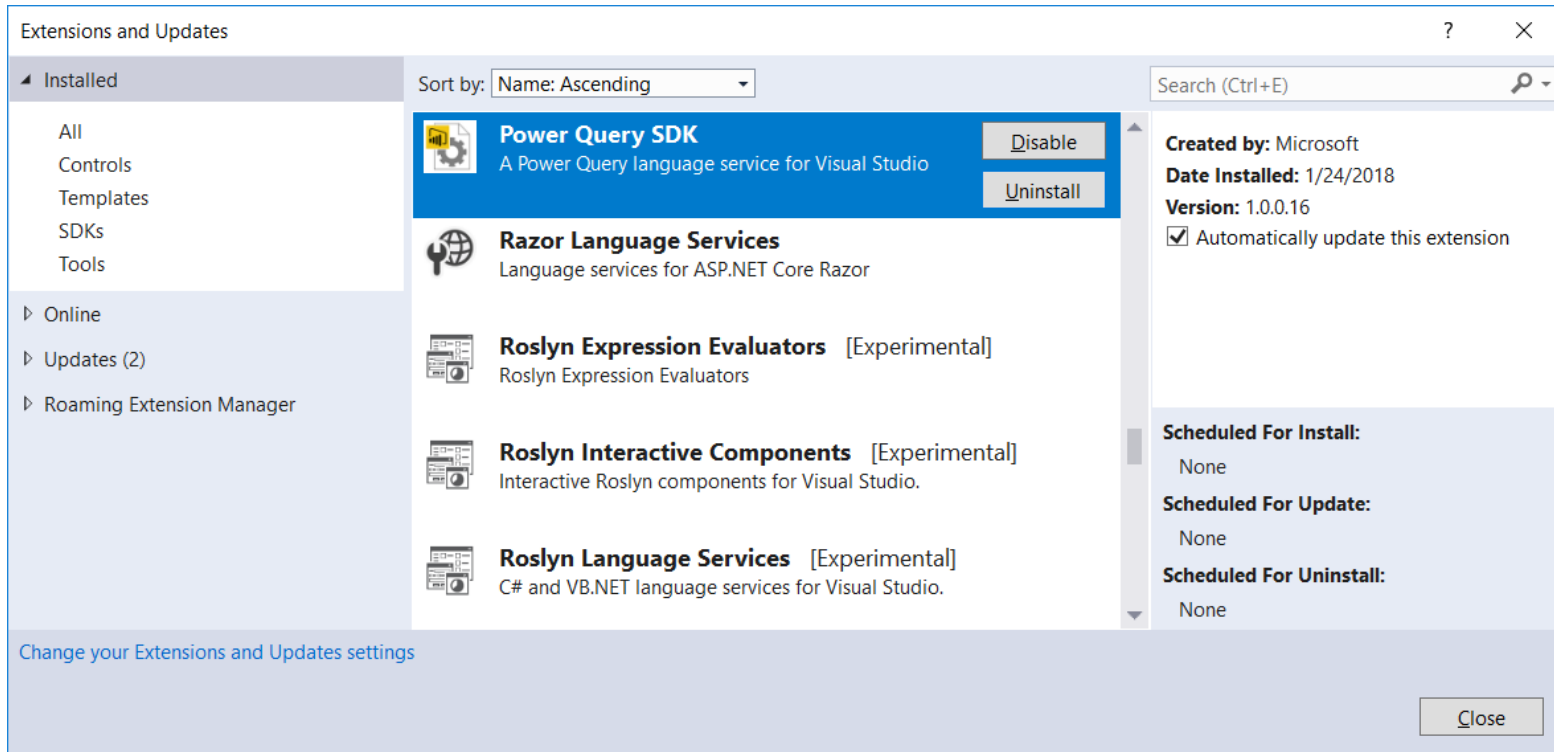


Developing Custom Data Connectors

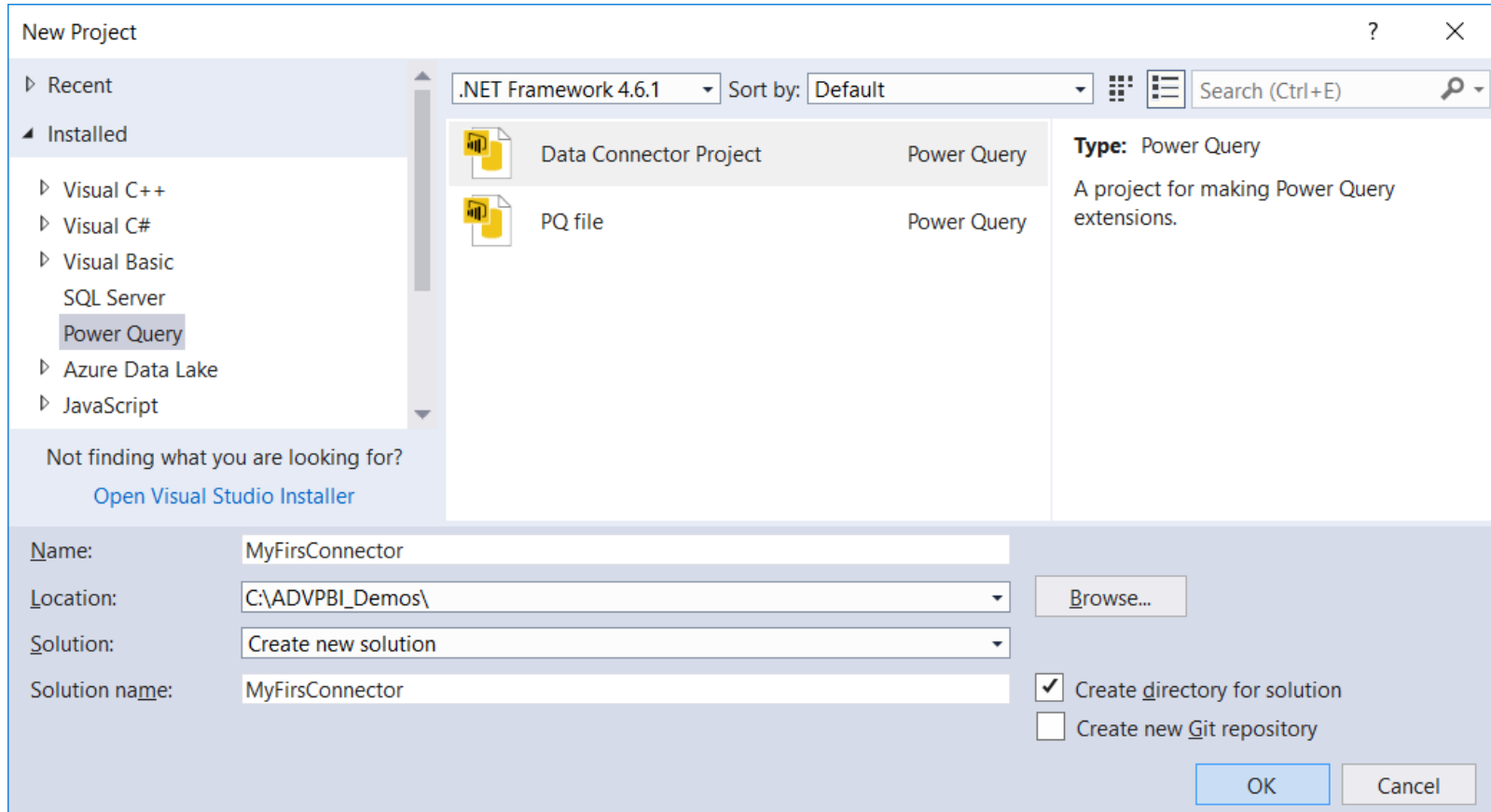
- Custom Connectors let you write reusable query logic
 - Custom connector is written using M programming language
 - Custom connector can be used across PBIX project files
- Common motivation for developing a custom connector
 - Creating a friendly view of a REST API for business analyst
 - Providing branding on top of existing connector
 - Exposing a limited/filtered view over your data source
 - Control how mashup engine authenticates against datasource
 - Implementing OAuth v2 authentication flow for a SaaS offering
 - Enabling Direct Query for a data source via an ODBC driver`



Power Query SDK



Creating a New Data Connector Project



The screenshot shows the 'New Project' dialog box in Visual Studio. The 'Installed' section on the left lists various project types, with 'Power Query' selected. The main pane displays two options: 'Data Connector Project' and 'PQ file', both categorized as 'Power Query'. The 'Type: Power Query' description on the right states: 'A project for making Power Query extensions.' The bottom section contains fields for project configuration: 'Name' (MyFirsConnector), 'Location' (C:\ADVPBI_Demos\), 'Solution' (Create new solution), and 'Solution name' (MyFirsConnector). There are also checkboxes for 'Create directory for solution' (checked) and 'Create new Git repository' (unchecked), along with 'Browse...', 'OK', and 'Cancel' buttons.

New Project



Recent

Installed

- Visual C++
- Visual C#
- Visual Basic
- SQL Server
- Power Query**
- Azure Data Lake
- JavaScript

Not finding what you are looking for?
[Open Visual Studio Installer](#)

.NET Framework 4.6.1 Sort by: Default Search (Ctrl+E)

| Icon | Project Name | Type |
|---|------------------------|-------------|
|  | Data Connector Project | Power Query |
|  | PQ file | Power Query |

Type: Power Query
A project for making Power Query extensions.

Name: MyFirsConnector

Location: C:\ADVPBI_Demos\ Browse...

Solution: Create new solution

Solution name: MyFirsConnector

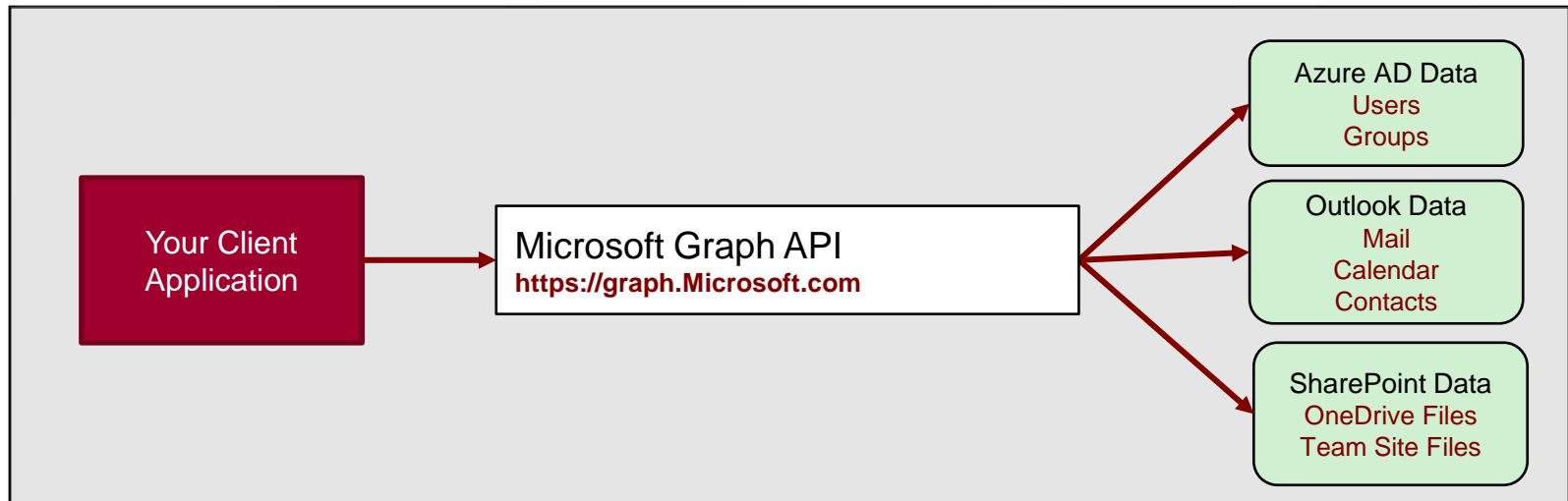
☒ Create directory for solution
☐ Create new Git repository

OK Cancel



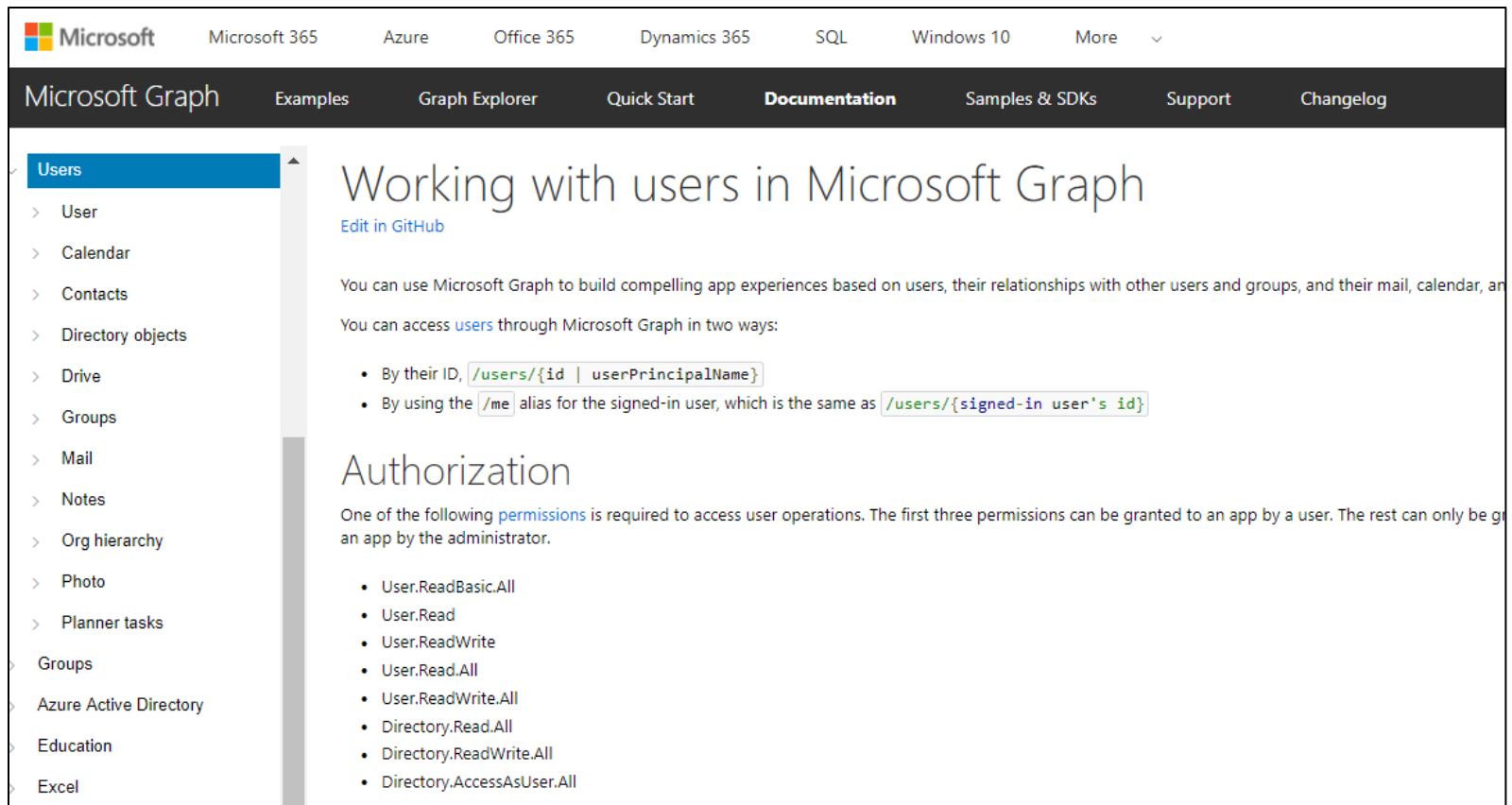
The Microsoft Graph API

- Designed as a one-stop-shopping kind of service
 - Abstracts away divisions between AD, Exchange and SharePoint
 - No need to discover endpoints using the Discovery Service
 - You can acquire and cache a single access token per user



More Info on the Microsoft Graph API

- <https://developer.microsoft.com/en-us/graph/docs/api-reference/v1.0>



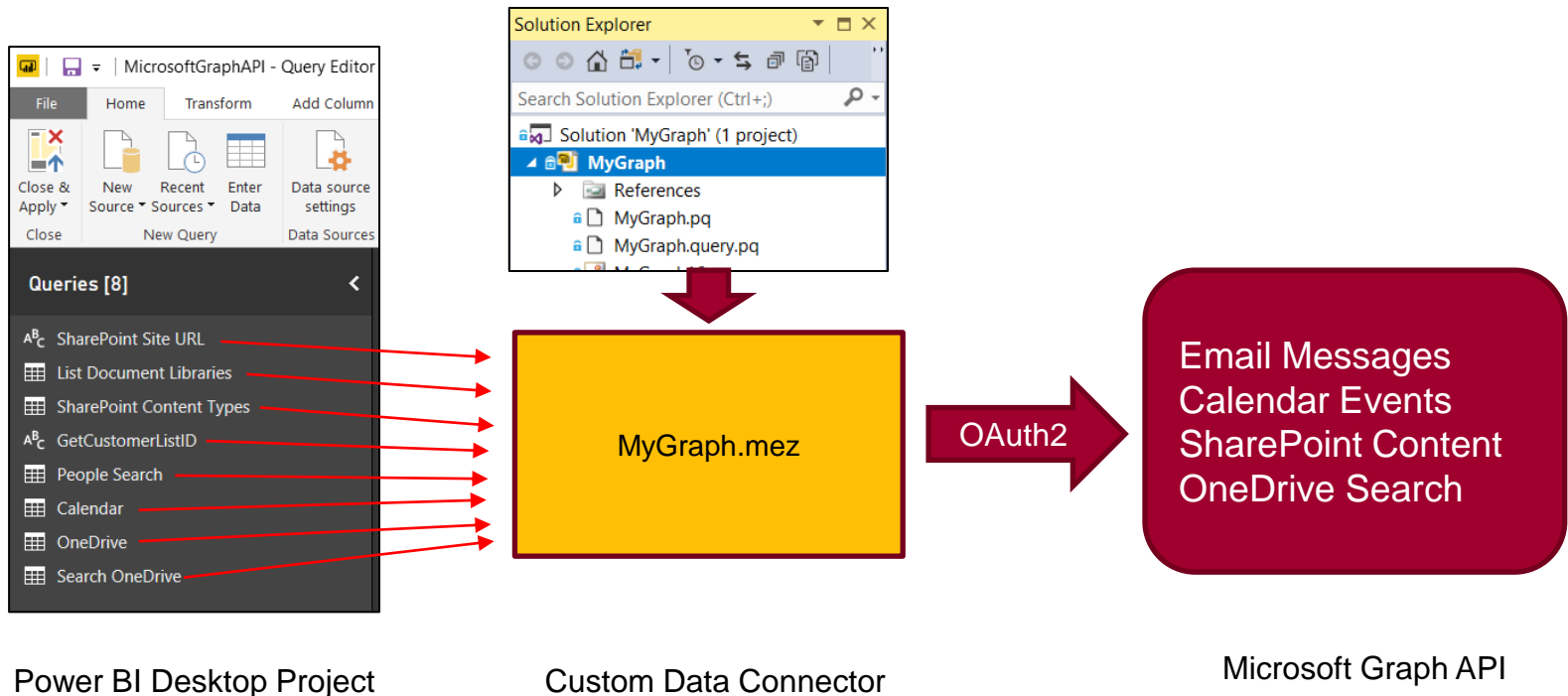
The screenshot shows the Microsoft Graph API documentation page. The top navigation bar includes links for Microsoft 365, Azure, Office 365, Dynamics 365, SQL, Windows 10, and a 'More' dropdown. Below this is a secondary navigation bar with 'Microsoft Graph', 'Examples', 'Graph Explorer', 'Quick Start', 'Documentation' (highlighted), 'Samples & SDKs', 'Support', and 'Changelog'. On the left, a sidebar lists various API endpoints: Users (highlighted), User, Calendar, Contacts, Directory objects, Drive, Groups, Mail, Notes, Org hierarchy, Photo, Planner tasks, Groups, Azure Active Directory, Education, and Excel. The main content area is titled 'Working with users in Microsoft Graph' and includes a link to 'Edit in GitHub'. It explains that Microsoft Graph can be used to build app experiences based on users and their relationships. It then lists two ways to access users: by their ID using the path `/users/{id | userPrincipalName}`, or by using the `/me` alias for the signed-in user, which is the same as `/users/{signed-in user's id}`. Below this, the 'Authorization' section states that one of the following permissions is required to access user operations. The first three permissions can be granted to an app by a user, while the rest can only be granted by an administrator. The permissions listed are:

- User.ReadBasic.All
- User.Read
- User.ReadWrite
- User.Read.All
- User.ReadWrite.All
- Directory.Read.All
- Directory.ReadWrite.All
- Directory.AccessAsUser.All



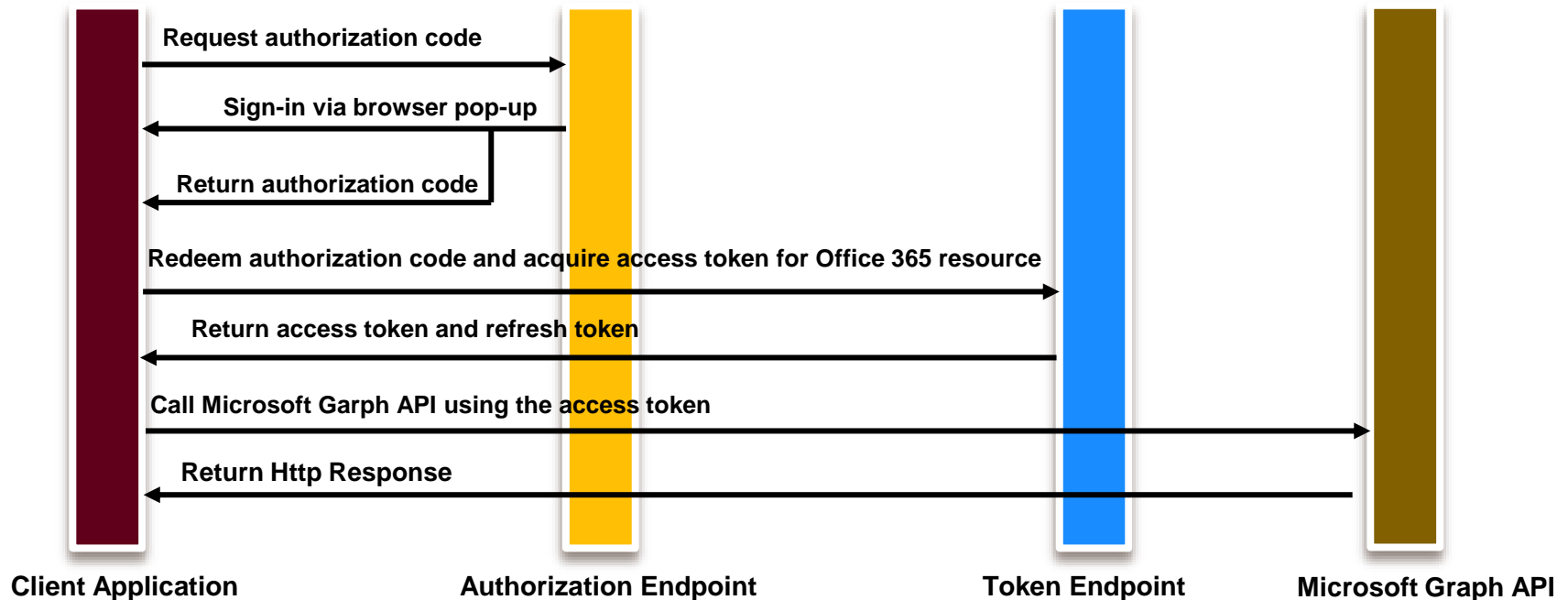
MyGraph Demo

- Project originally created by Matt Masson
 - Connector designed to query Microsoft Graph API
 - Connector provides code to authenticate with OAuth2



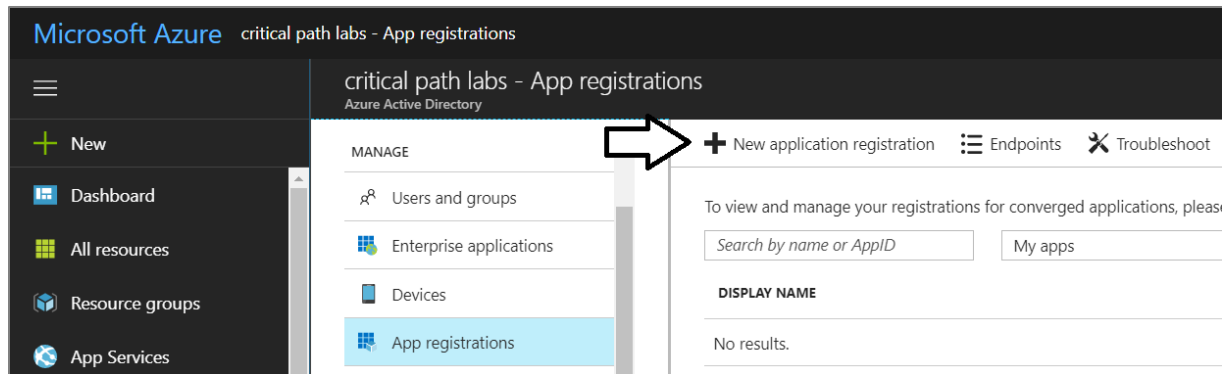
Authorization Code Grant Flow

- Sequence of Requests in Authorization Code Grant Flow
 - Application redirects to AAD authorization endpoint
 - User prompted to log on at Windows logon page
 - User prompted to consent to permissions (first access)
 - AAD redirects to application with authorization code
 - Application redirects to AAD access token endpoint



Registering an Azure Application

- Can be done using Azure portal



- Details you need for the custom data connector
 - Client ID
 - Client Secret
 - Redirect URL



Summary

- ✓ Power Query Mashup Engine
- ✓ M Programming Fundamentals
- ✓ M Function Library
- ✓ Query Functions
- ✓ Query Parameters
- ✓ Custom Data Connectors

