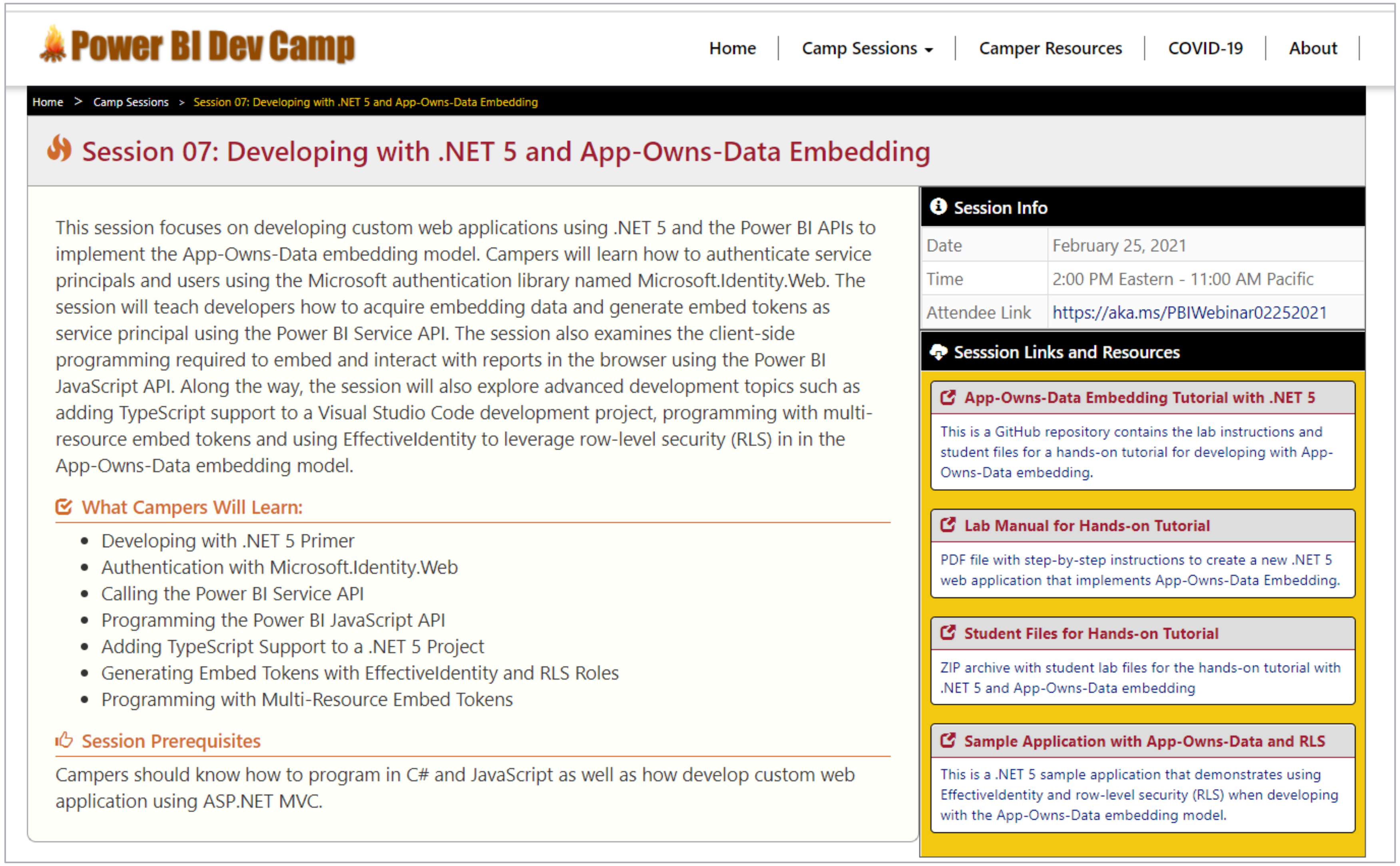
Intro to M Programming

Ted Pattison
Principal Program Manager
Customer Advisory Team (CAT) at Microsoft

Welcome to Power BI Dev Camp

- Power BI Dev Camp Portal - https://powerbidevcamp.net



Power BI Dev Camp
Home
Camp Sessions
Camper Resources
COVID-19
About
Home > Camp Sessions > Session 07: Developing with .NET 5 and App-Owns-Data Embedding

Session 07: Developing with .NET 5 and App-Owns-Data Embedding

This session focuses on developing custom web applications using .NET 5 and the Power BI APIs to implement the App-Owns-Data embedding model. Campers will learn how to authenticate service principals and users using the Microsoft authentication library named Microsoft.Identity.Web. The session will teach developers how to acquire embedding data and generate embed tokens as service principal using the Power BI Service API. The session also examines the client-side programming required to embed and interact with reports in the browser using the Power BI JavaScript API. Along the way, the session will also explore advanced development topics such as adding TypeScript support to a Visual Studio Code development project, programming with multi-resource embed tokens and using EffectiveIdentity to leverage row-level security (RLS) in in the App-Owns-Data embedding model.

What Campers Will Learn:

- Developing with .NET 5 Primer
- Authentication with Microsoft.Identity.Web
- Calling the Power BI Service API
- Programming the Power BI JavaScript API
- Adding TypeScript Support to a .NET 5 Project
- Generating Embed Tokens with EffectiveIdentity and RLS Roles
- Programming with Multi-Resource Embed Tokens

Session Prerequisites

Campers should know how to program in C# and JavaScript as well as how develop custom web application using ASP.NET MVC.

Session Info

Date	February 25, 2021
Time	2:00 PM Eastern - 11:00 AM Pacific
Attendee Link	https://aka.ms/PBIWebinar02252021

Sesssion Links and Resources

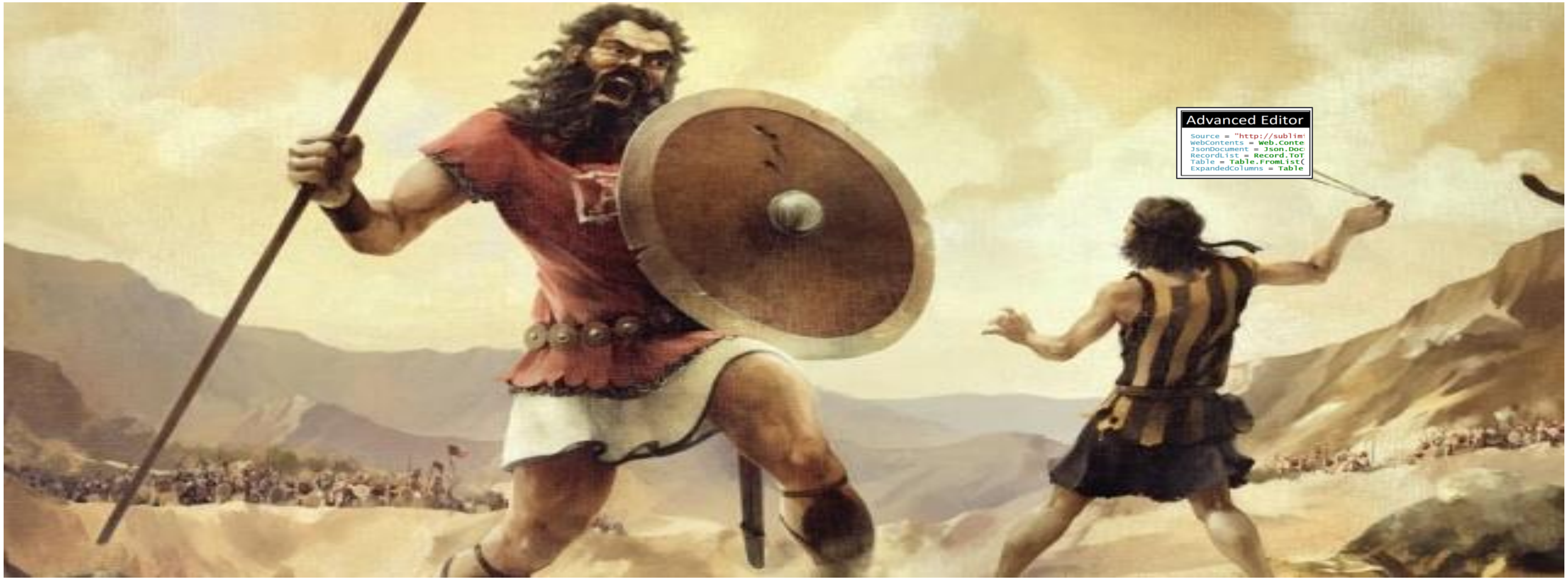
App-Owns-Data Embedding Tutorial with .NET 5
This is a GitHub repository contains the lab instructions and student files for a hands-on tutorial for developing with App-Owns-Data embedding.

Lab Manual for Hands-on Tutorial
PDF file with step-by-step instructions to create a new .NET 5 web application that implements App-Owns-Data Embedding.

Student Files for Hands-on Tutorial
ZIP archive with student lab files for the hands-on tutorial with .NET 5 and App-Owns-Data embedding

Sample Application with App-Owns-Data and RLS
This is a .NET 5 sample application that demonstrates using EffectiveIdentity and row-level security (RLS) when developing with the App-Owns-Data embedding model.

Using the Advanced Editor



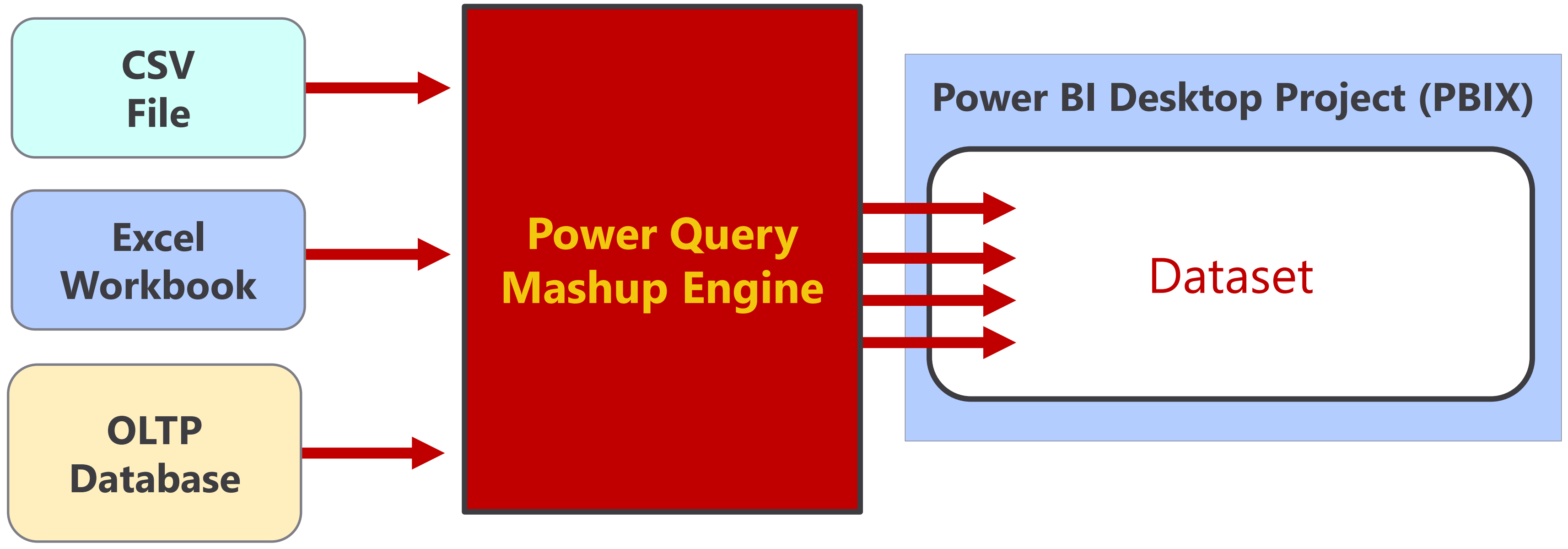
```
Source = "http://sublim
WebContents = Web.Conte
JsonDocument = Json.Doc
RecordList = Record.ToT
Table = Table.FromList(
ExpandedColumns = Table
```

Agenda

- The Power Query Mashup Engine
- M Programming Fundamentals
- Choosing Between OData.Feed & Web.Contents
- Introduction to Custom Connectors
- Importing Data from the Microsoft Graph API
- Signing and Deploying Custom Connectors

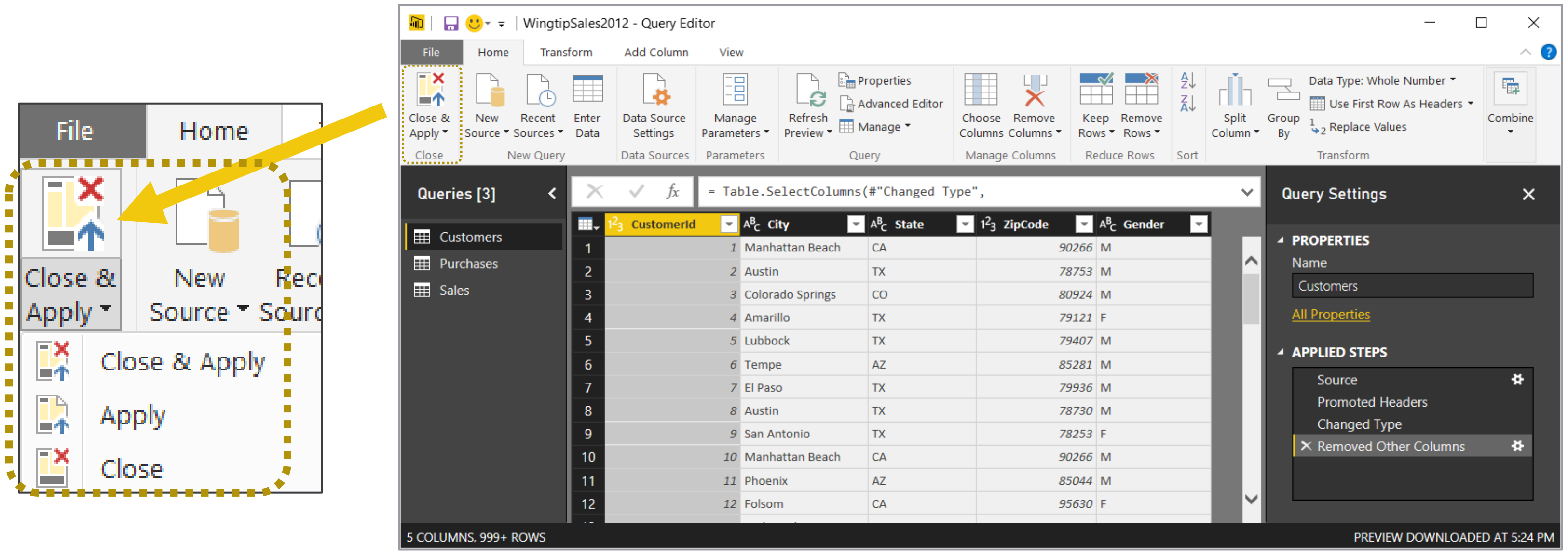
Power Query is an ETL Tool

- **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



```
= Table.SelectColumns(#"Changed Type",
```

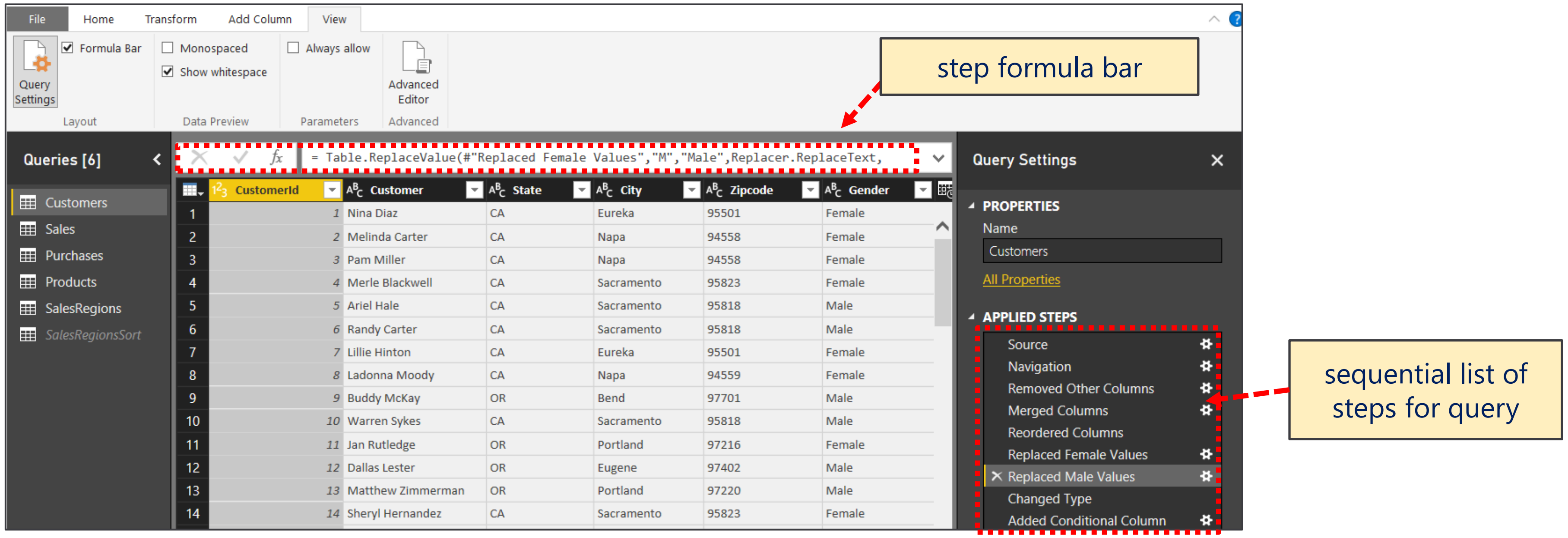
	CustomerId	City	State	ZipCode	Gender
1	1	Manhattan Beach	CA	90266	M
2	2	Austin	TX	78753	M
3	3	Colorado Springs	CO	80924	M
4	4	Amarillo	TX	79121	F
5	5	Lubbock	TX	79407	M
6	6	Tempe	AZ	85281	M
7	7	El Paso	TX	79936	M
8	8	Austin	TX	78730	M
9	9	San Antonio	TX	78253	F
10	10	Manhattan Beach	CA	90266	M
11	11	Phoenix	AZ	85044	M
12	12	Folsom	CA	95630	F

5 COLUMNS, 999+ ROWS

PREVIEW DOWNLOADED AT 5:24 PM

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)



Chosen settings: ☑ Formula Bar, ☐ Monospaced, ☑ Show whitespace, ☐ Always allow

Name: Customers

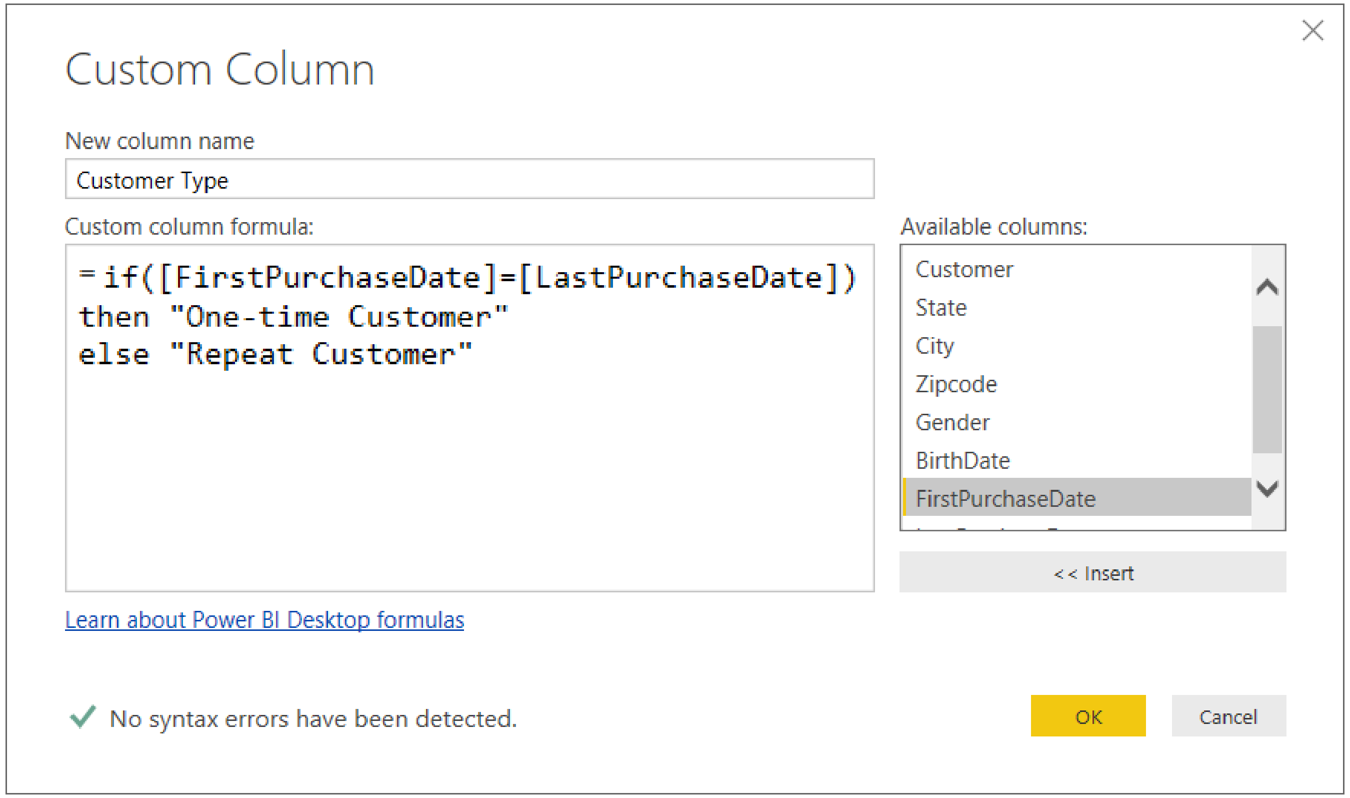
```
= Table.ReplaceValue(#"Replaced Female Values","M","Male",Replacer.ReplaceText,
```

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

APPLIED STEPS:
- Source
- Navigation
- Removed Other Columns
- Merged Columns
- Reordered Columns
- Replaced Female Values
- Replaced Male Values
- Changed Type
- Added Conditional Column

Custom Column Dialog

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



Custom Column

New column name

Customer Type

Custom column formula:

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

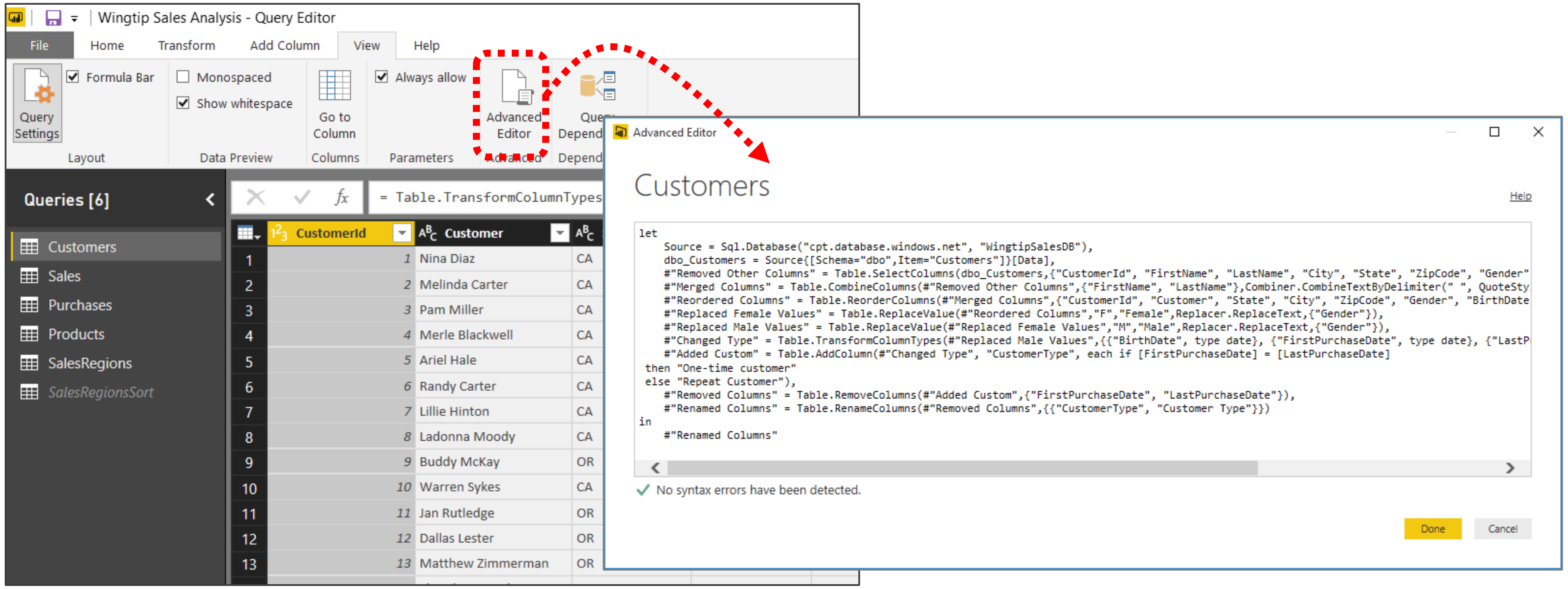
Available columns:

No syntax errors have been detected.

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



	CustomerId	Customer	
1	1	Nina Diaz	CA
2	2	Melinda Carter	CA
3	3	Pam Miller	CA
4	4	Merle Blackwell	CA
5	5	Ariel Hale	CA
6	6	Randy Carter	CA
7	7	Lillie Hinton	CA
8	8	Ladonna Moody	CA
9	9	Buddy McKay	OR
10	10	Warren Sykes	CA
11	11	Jan Rutledge	OR
12	12	Dallas Lester	OR
13	13	Matthew Zimmerman	OR

```
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"
    #"Merged Columns" = Table.CombineColumns(#"Removed Other Columns",{"FirstName", "LastName"},Combiner.CombineTextByDelimiter(" ", QuoteSty
    #"Reordered Columns" = Table.ReorderColumns(#"Merged Columns",{"CustomerId", "Customer", "State", "City", "ZipCode", "Gender", "BirthDate
    #"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}, {"LastP
    #"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"
```

✓ No syntax errors have been detected.

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

- ✓The Power Query Mashup Engine
- ➢M Programming Fundamentals
- Choosing Between `OData.Feed` & `Web.Contents`
- Introduction to Custom Connectors
- Importing Data from the Microsoft Graph API
- Signing and Deploying Custom Connectors

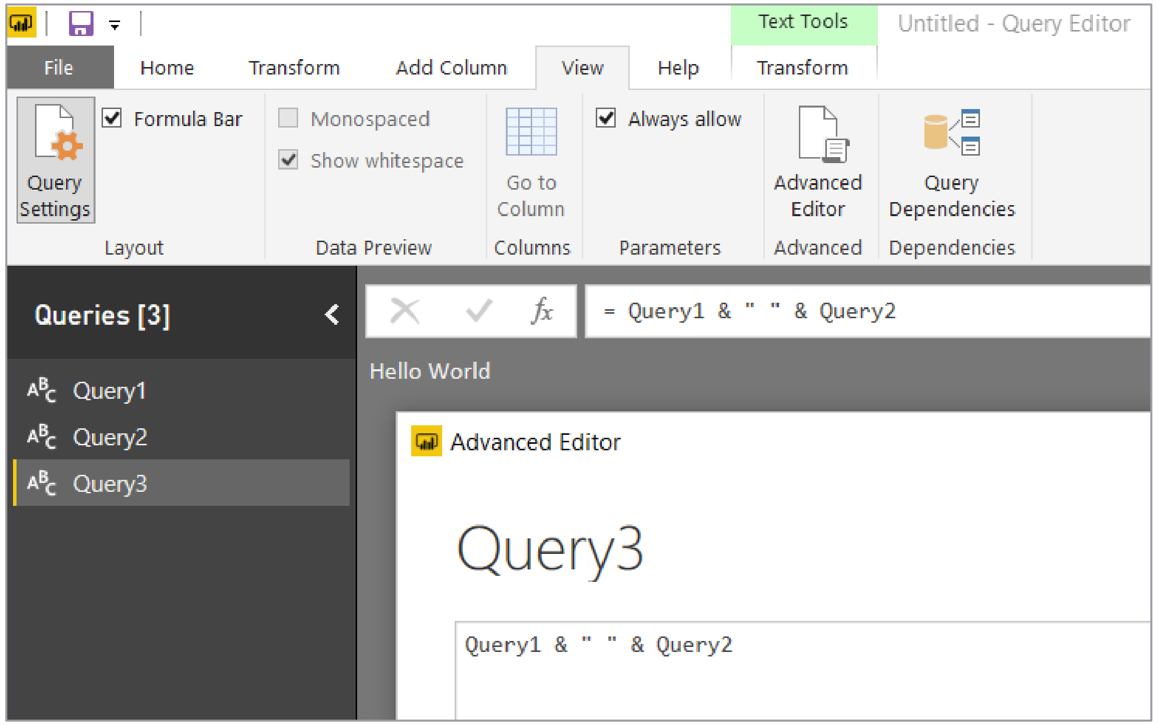
The M Programming Language

- **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



```
= Query1 & " " & Query2
```

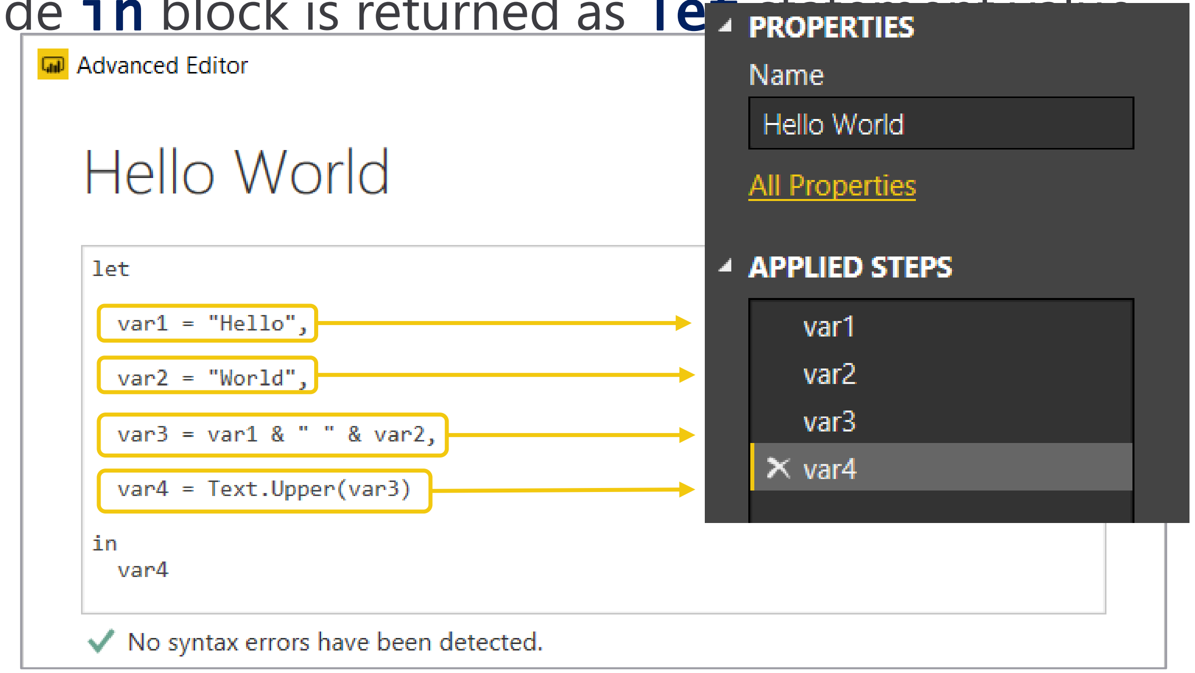
Hello World

Query3

```
Query1 & " " & Query2
```

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 `le`



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

No syntax errors have been detected.

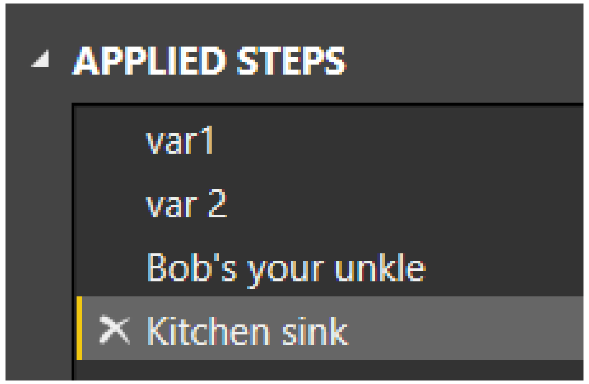
Comments and Variable Names

- **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
```

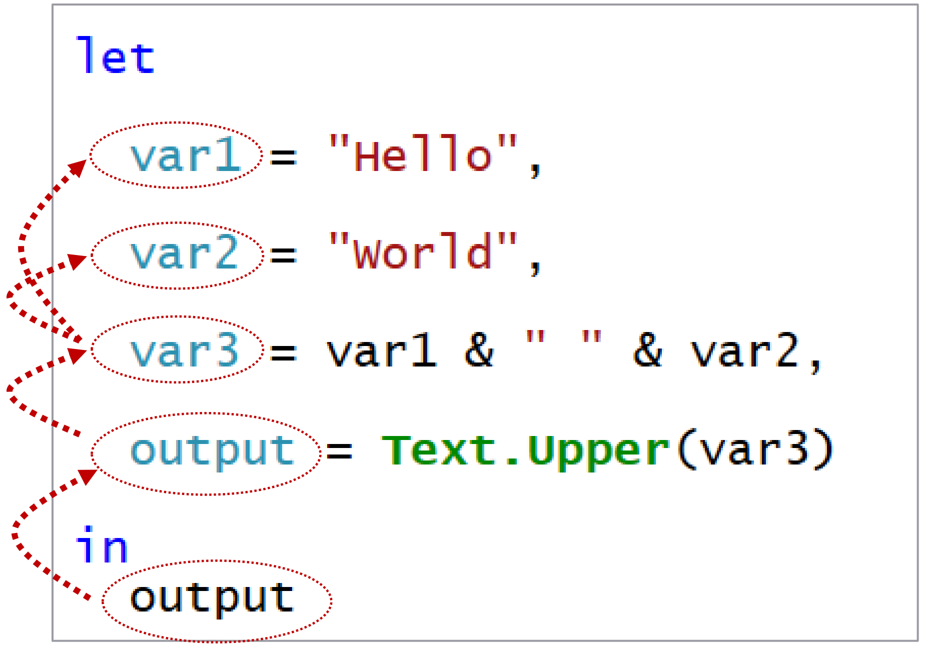
- **Variable names with spaces must be enclosed in #" "**
  - Variable names with spaces created automatically by query designer

```
let

  var1 = "Spaces in ",

  #"var 2" = "variable names ",

  #"Bob's your unkle" = "are evil",

  #"Kitchen sink" = var1 & #"var 2" & #"Bob's your unkle"

in
  #"Kitchen sink"
```


APPLIED STEPS
var1
var 2
Bob's your unkle
Kitchen sink

Flow of Statement Evaluation

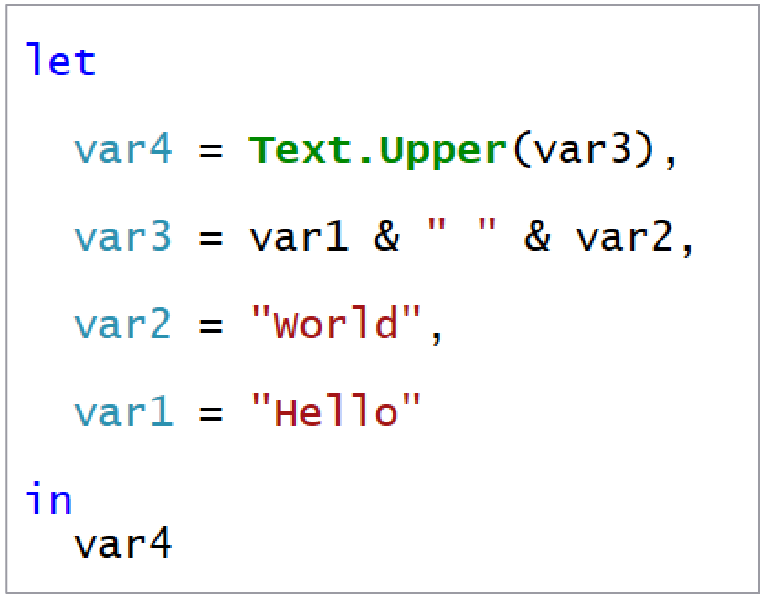
- **Evaluation starts with expression inside `in` block**
  - Expression evaluation triggers other expression evaluation



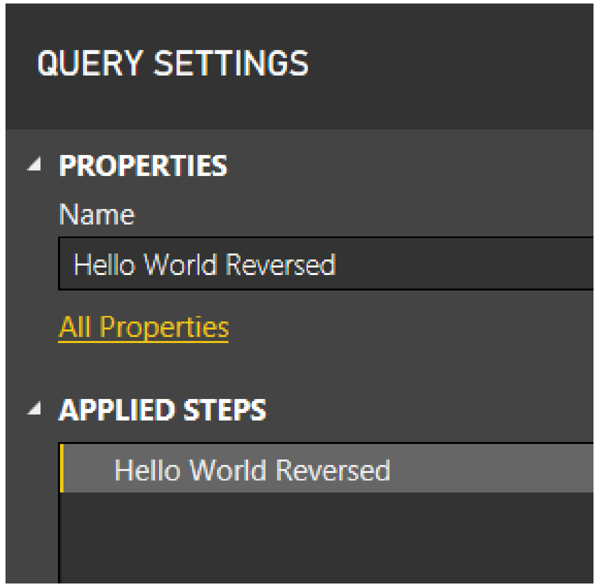
```
let
    var1 = "Hello",
    var2 = "World",
    var3 = var1 & " " & var2,
    output = Text.Upper(var3)
in
    output
```

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

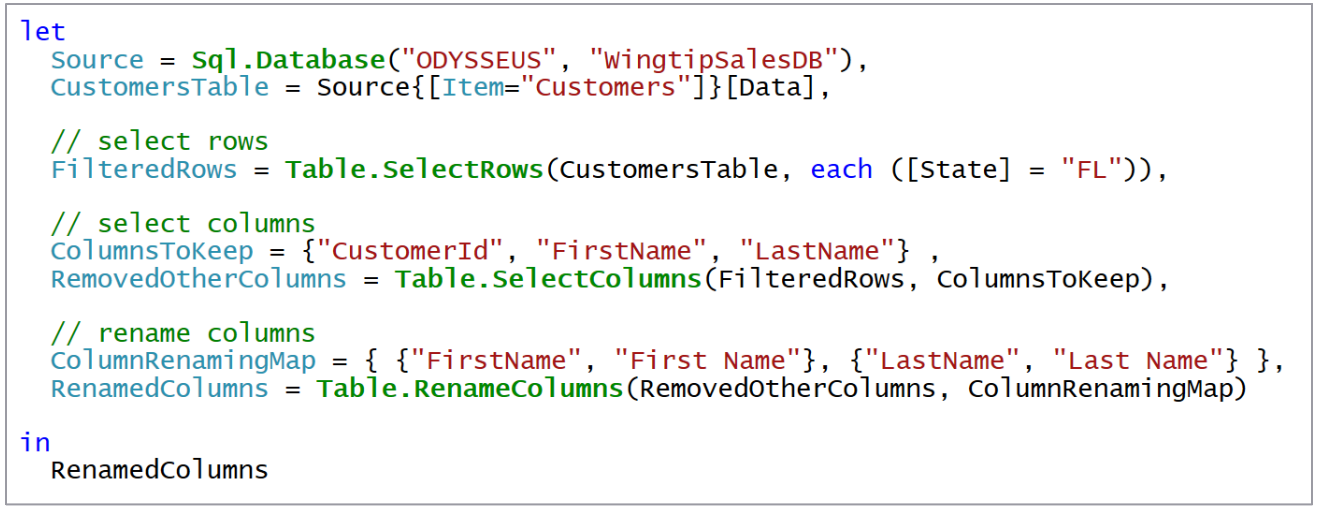
Hello World Reversed

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

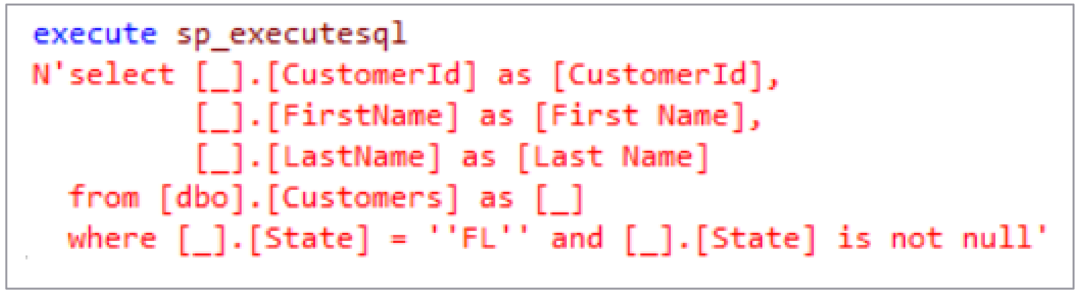
Query Folding Example

- 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(FilteredRows, 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

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

Examples of programming with 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 charactors
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 charactors
set4 = {" ", "-", "_", "."},

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

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

Records

- Record contains fields for single instance of entity

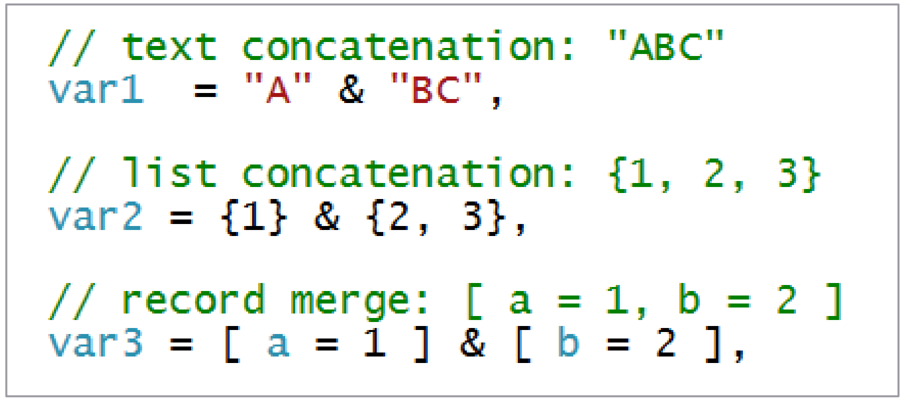
```
// create records by using [] and defining fields
Person1 = [FirstName="Chris", LastName="Webb"],
Person2 = [FirstName="Reza", LastName="Rad"],
Person3 = [FirstName="Matt", LastName="Masson"],


// 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

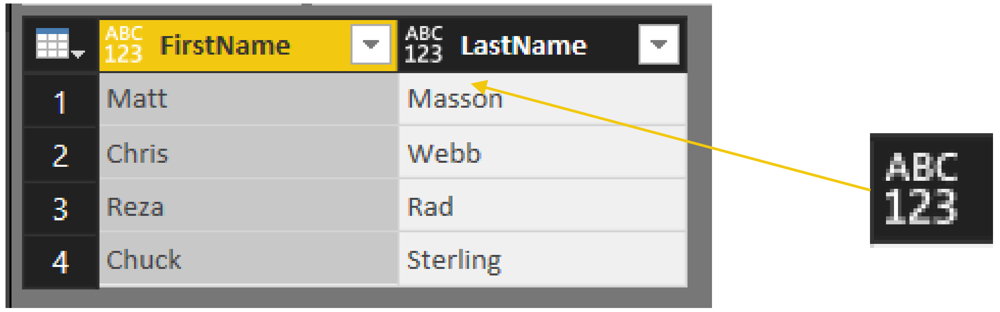


```
// text concatenation: "ABC"
var1  = "A" & "BC",

// 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

Bad, Bad, Bad ☹

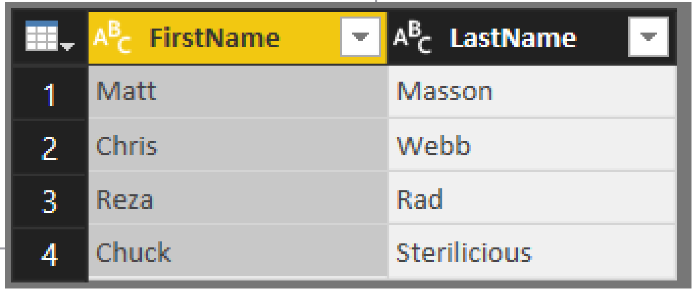
Creating User-defined Types

- **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,
  CustomersTable =
    #table(CustomerTableType, {
      { "Matt", "Masson" },
      { "Chris", "Webb" },
      { "Reza", "Rad" },
      { "Chuck", "Sterilicious"}
    }
  )
in
  CustomersTable
```



	FirstName	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  ),
```

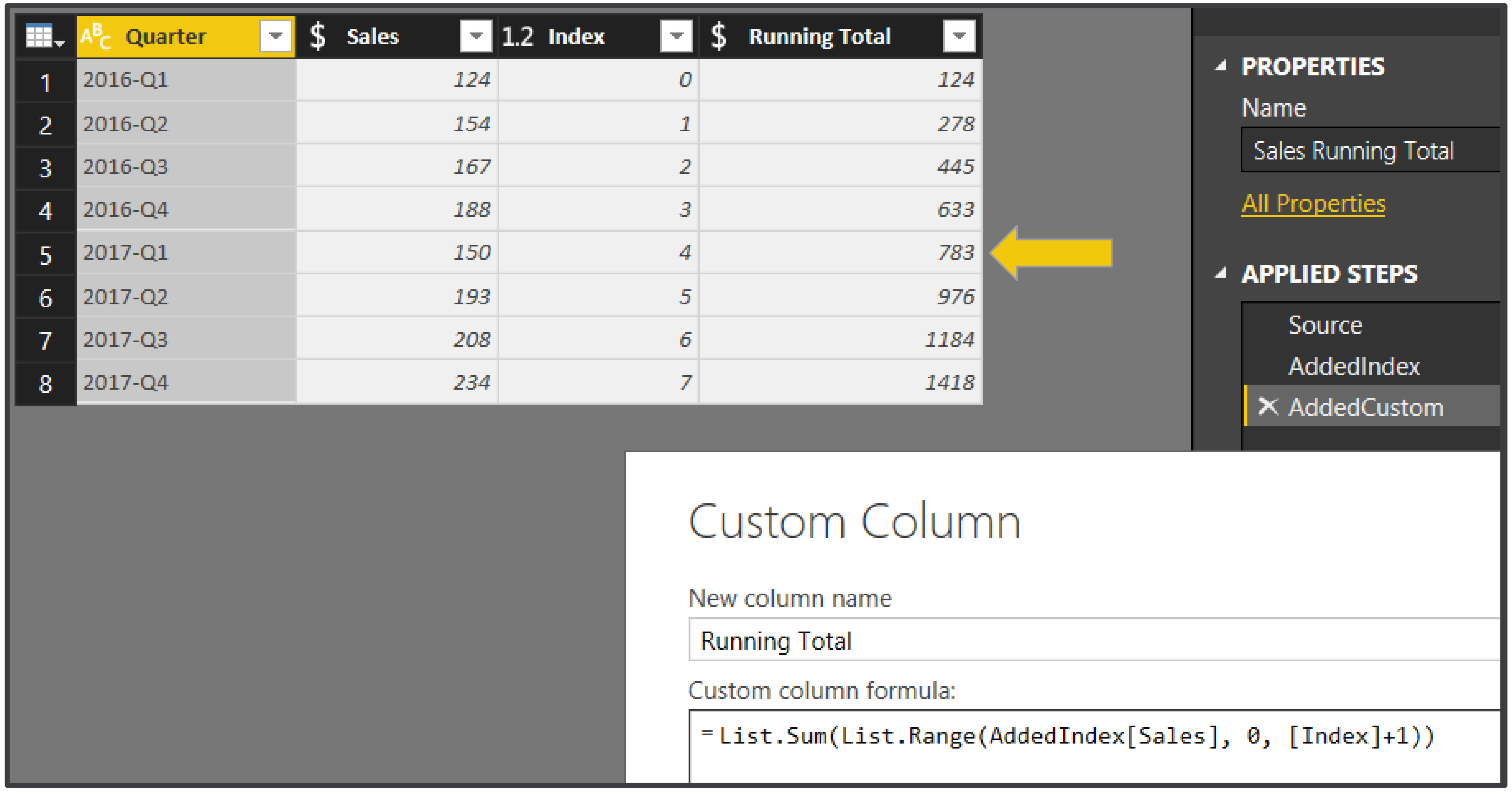
```
AddedColumn =Table.AddColumn(FilteredRows, "Display Name", each [FirstName] & " " & [LastName])
```

  - 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	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

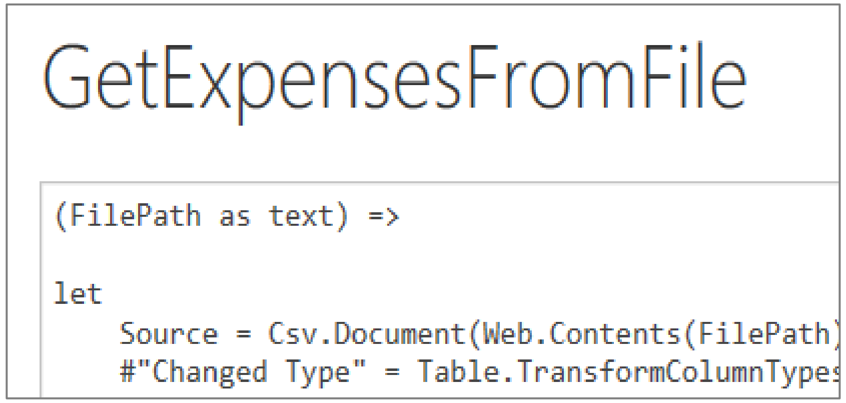
New column name: Running Total

Custom column formula:

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

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
```

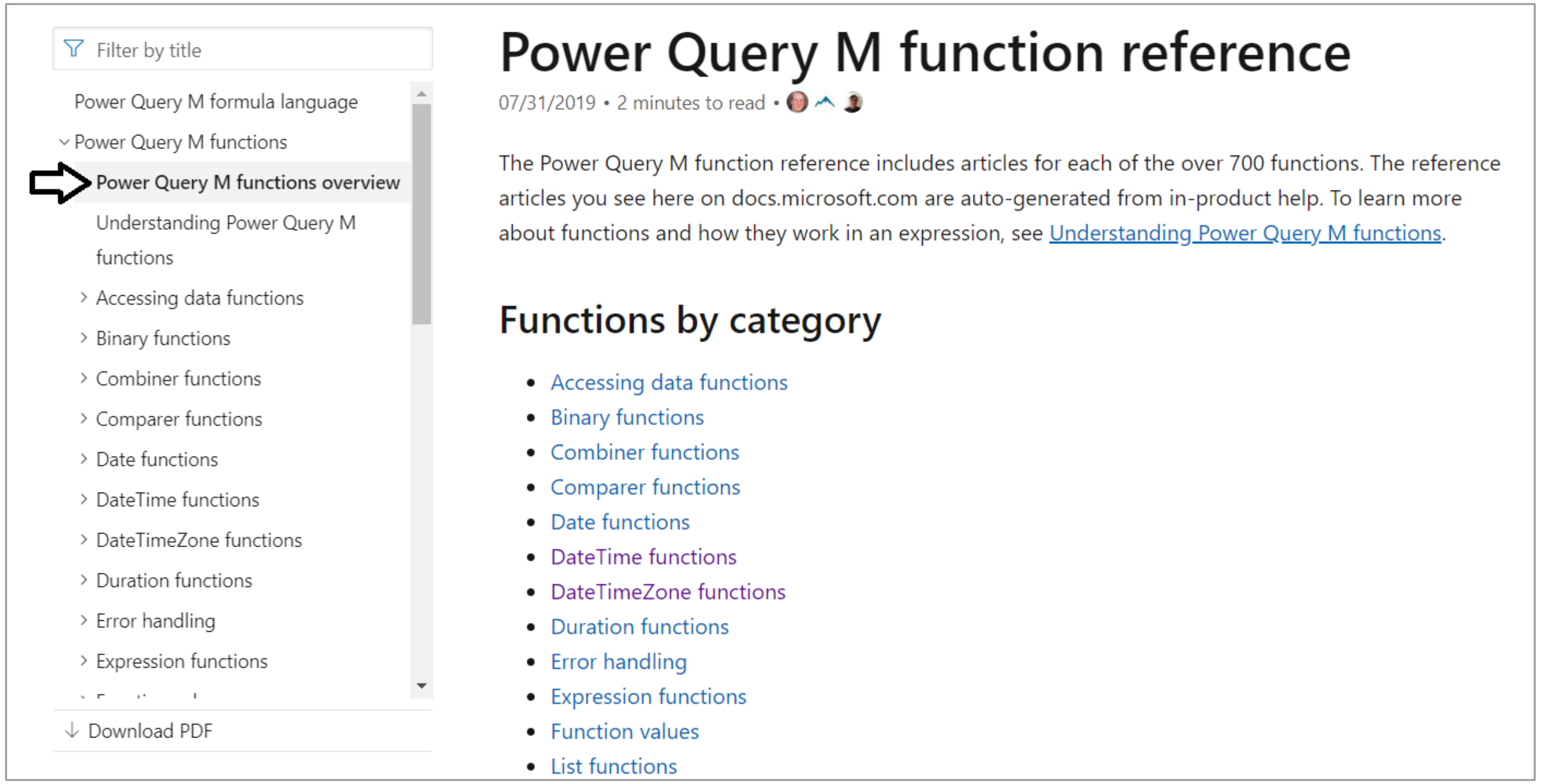
  - Function query can be called from other queries
  - Function query can be called using Invoke Custom Function
  - Function query can't be edited with visual designer

Agenda

- ✓The Power Query Mashup Engine
- ✓M Programming Fundamentals
- ➢Choosing Between `OData.Feed` & `Web.Contents`
- Introduction to Custom Connectors
- Importing Data from the Microsoft Graph API
- Signing and Deploying Custom 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 •

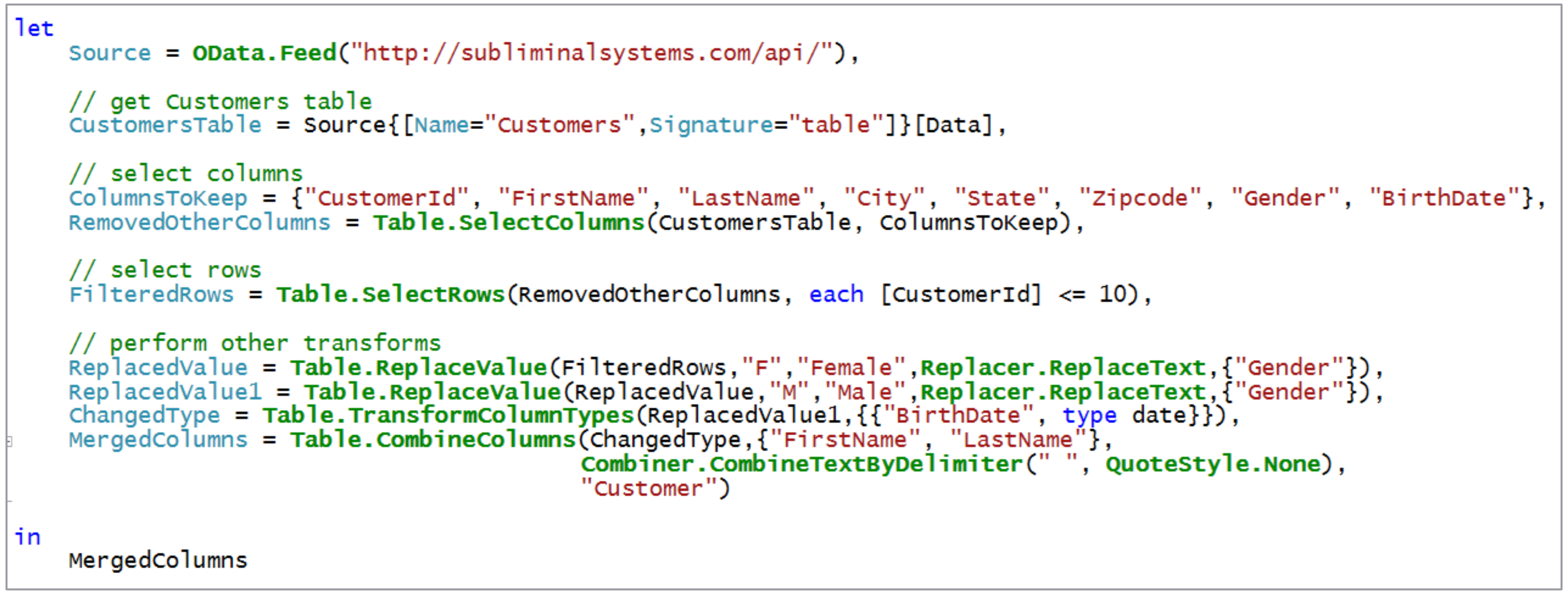
The Power Query M function reference includes articles for each of the over 700 functions. The reference articles you see here on docs.microsoft.com are auto-generated from in-product help. To learn more about functions and how they work in an expression, see Understanding Power Query M functions.

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

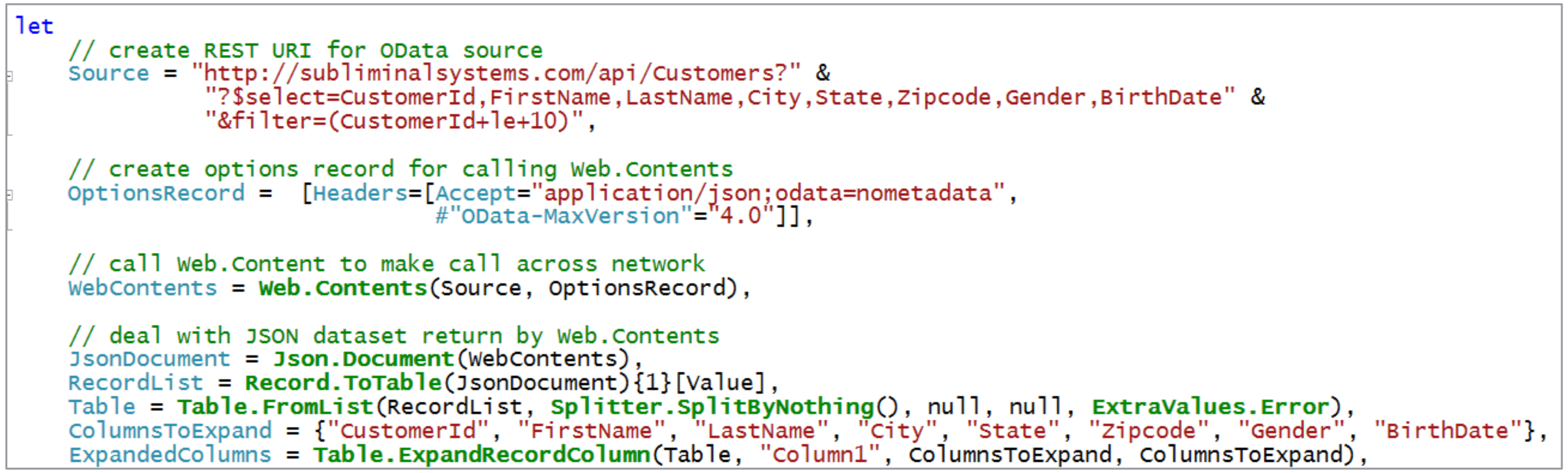


```
let
    Source = OData.Feed("http://subliminalsystems.com/api/"),

    // 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+le+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),
```

List.Generate

- List.Generate accepts 3 function parameters

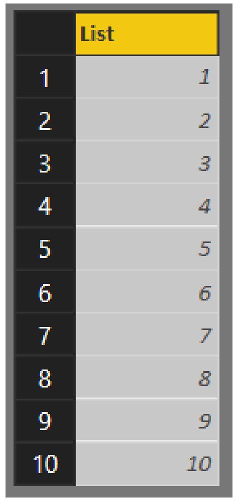
```
MyList = List.Generate( ()=>1, (item)=>(item<=10), (item)=>(item+1) )
```

- 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 =  ()=>10,
  TestFunction = each (_ <= 70),
  IncrementFunction = each (_ + 10),

  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

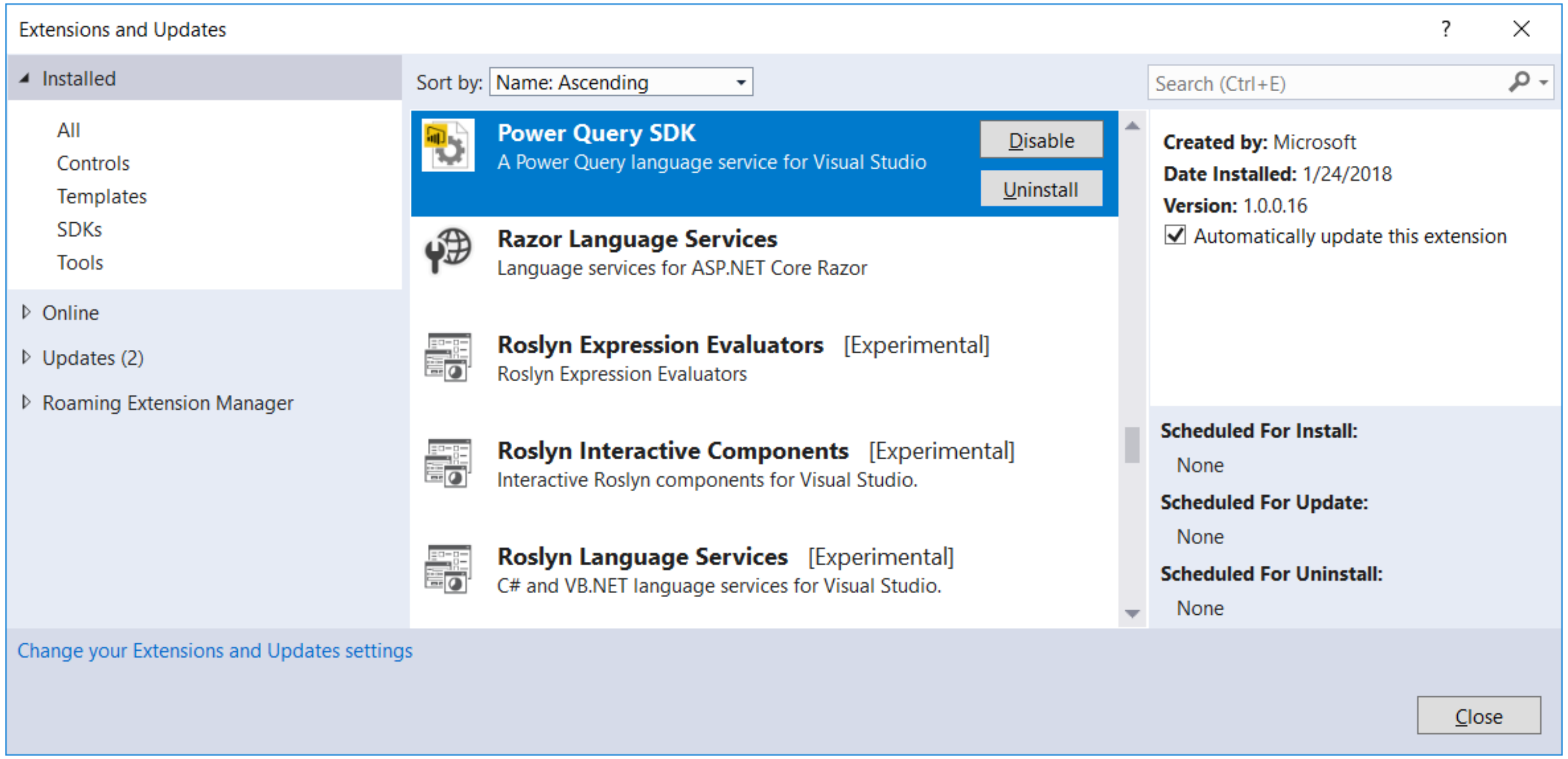
- The Power Query Mashup Engine
- M Programming Fundamentals
- Choosing Between OData.Feed & Web.Contents
- Introduction to Custom Connectors
- Importing Data from the Microsoft Graph API
- Signing and Deploying Custom Connectors

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



Power Query SDK
A Power Query language service for Visual Studio

Razor Language Services
Language services for ASP.NET Core Razor

Roslyn Expression Evaluators [Experimental]
Roslyn Expression Evaluators

Roslyn Interactive Components [Experimental]
Interactive Roslyn components for Visual Studio.

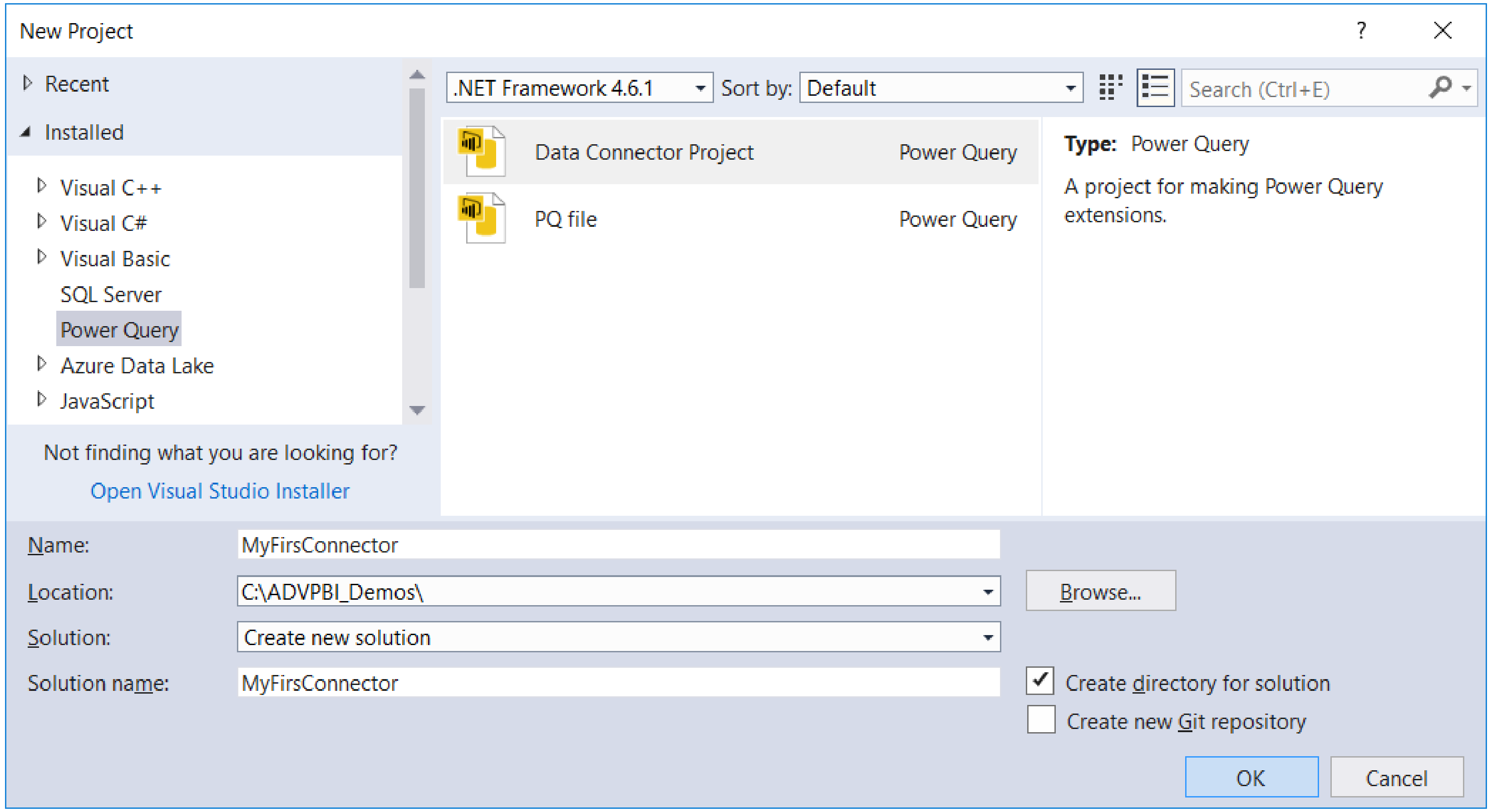
Roslyn Language Services [Experimental]
C# and VB.NET language services for Visual Studio.

Created by: Microsoft
Date Installed: 1/24/2018
Version: 1.0.0.16
[✓] Automatically update this extension

Scheduled For Install:
None
Scheduled For Update:
None
Scheduled For Uninstall:
None

Change your Extensions and Updates settings

Creating a New Data Connector Project


New Project

Not finding what you are looking for?
Open Visual Studio Installer

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

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\
Solution:	Create new solution
Solution name:	MyFirsConnector

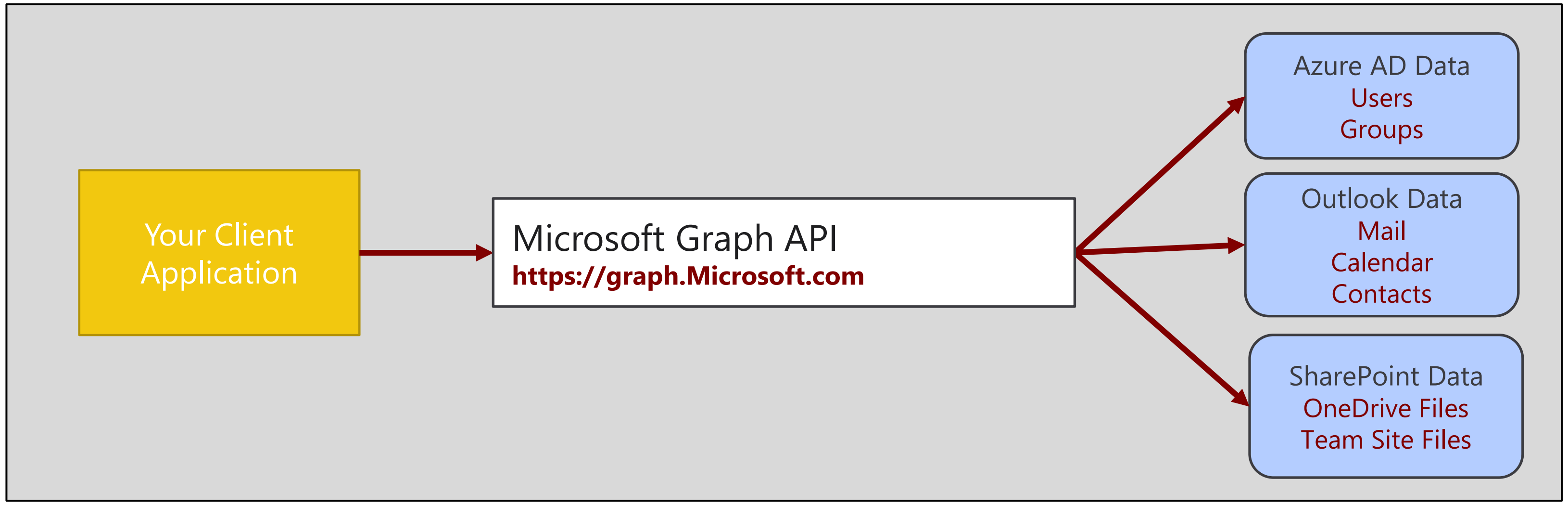
[✓] Create directory for solution
[ ] Create new Git repository

Agenda

- The Power Query Mashup Engine
- M Programming Fundamentals
- Choosing Between OData.Feed & Web.Contents
- Introduction to Custom Connectors
- Importing Data from the Microsoft Graph API
- Signing and Deploying Custom Connectors

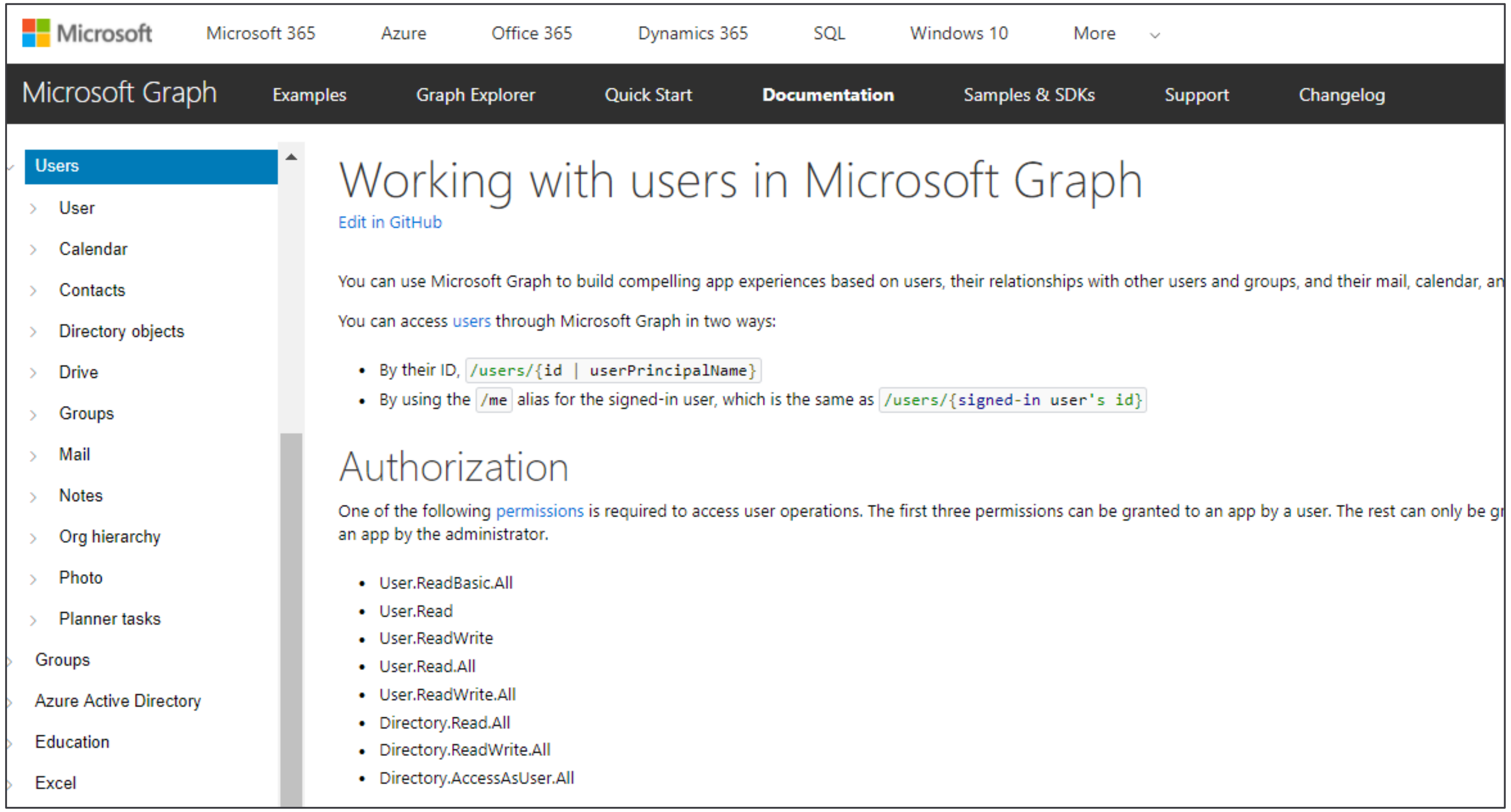
The Microsoft Graph API

- **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



Working with users in Microsoft Graph

Edit in GitHub

You can use Microsoft Graph to build compelling app experiences based on users, their relationships with other users and groups, and their mail, calendar, an

You can access users through Microsoft Graph in two ways:

- By their ID, `/users/{id | userPrincipalName}`
- By using the `/me` alias for the signed-in user, which is the same as `/users/{signed-in user's id}`

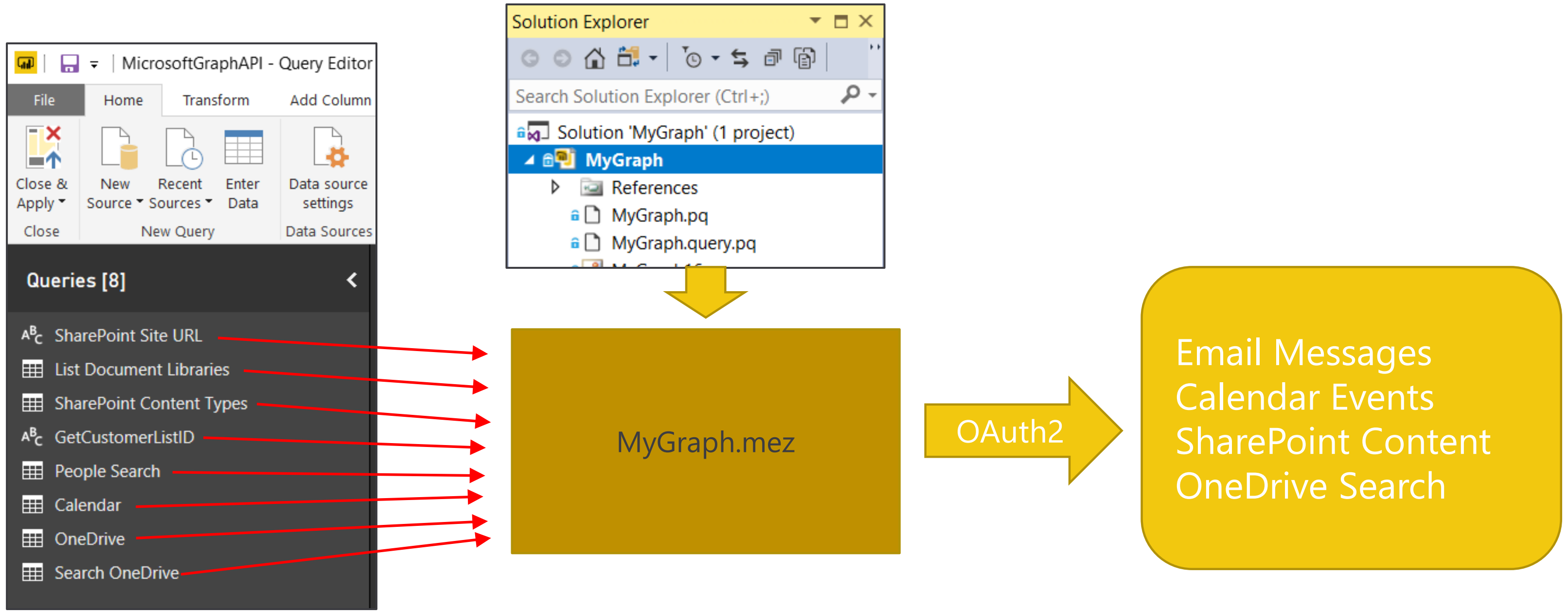
Authorization

One of the following permissions is required to access user operations. The first three permissions can be granted to an app by a user. The rest can only be gr
an app by the administrator.

- 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



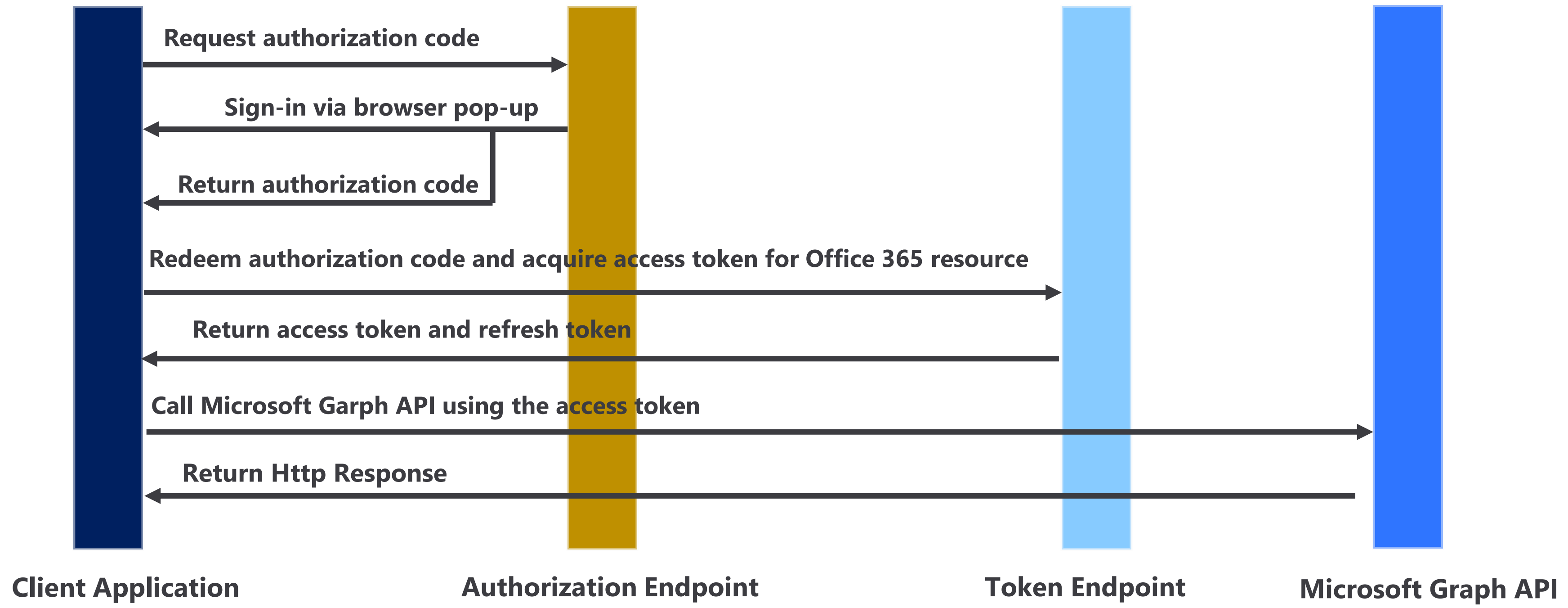
Power BI Desktop Project

Custom Data Connector

Microsoft Graph API

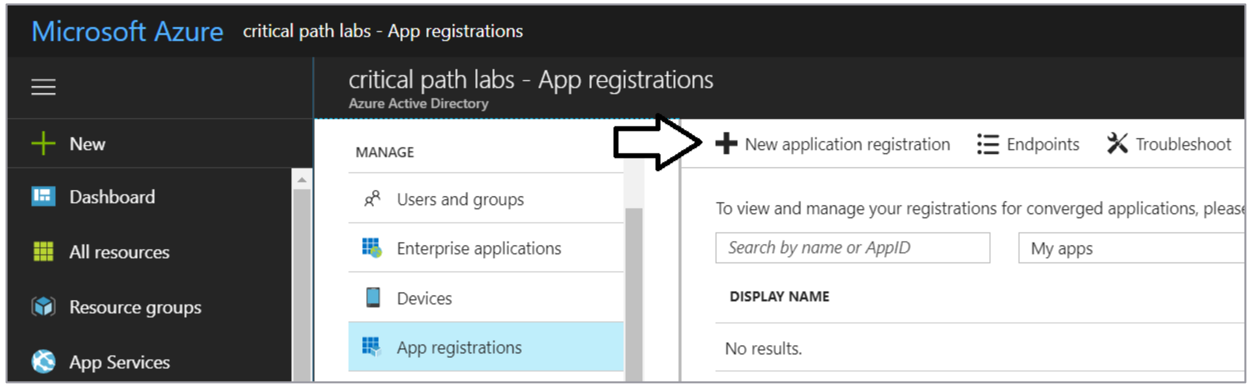
Authorization Code Grant Flow

- **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


Request authorization code
Sign-in via browser pop-up
Return authorization code
Redeem authorization code and acquire access token for Office 365 resource
Return access token and refresh token
Call Microsoft Garph API using the access token
Return Http Response
Client Application
Authorization Endpoint
Token Endpoint
Microsoft Graph API

Registering an Azure Application

- Can be done using Azure portal



To view and manage your registrations for converged applications, please

No results.

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

Agenda

- The Power Query Mashup Engine
- M Programming Fundamentals
- Choosing Between OData.Feed & Web.Contents
- Introduction to Custom Connectors
- Importing Data from the Microsoft Graph API
- Signing and Deploying Custom Connectors