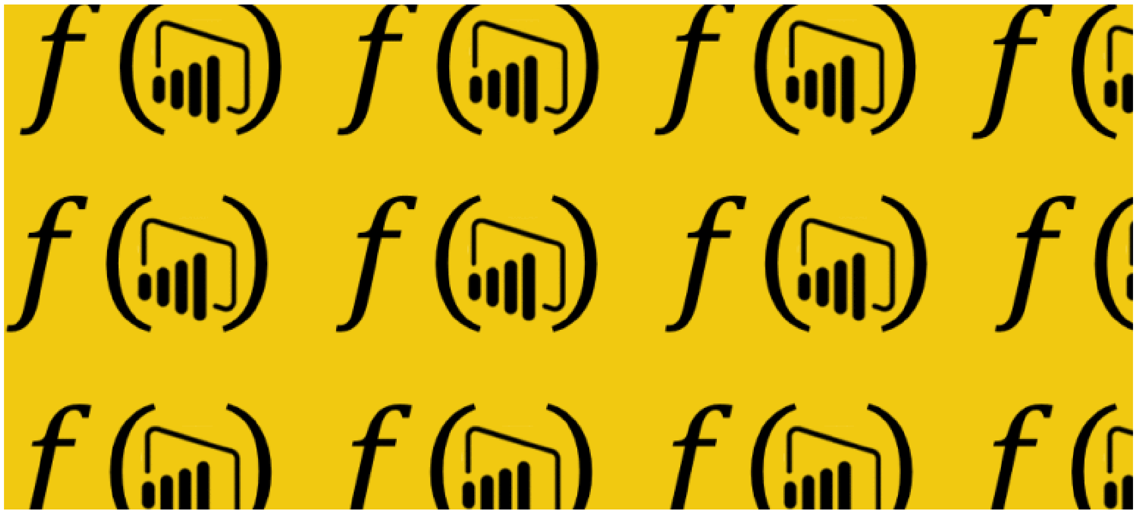
Query Power 📊

Power Query, Power Pivot, Power BI, PowerShell and other Powerful Tools



Navigating over 600+ M funcitons

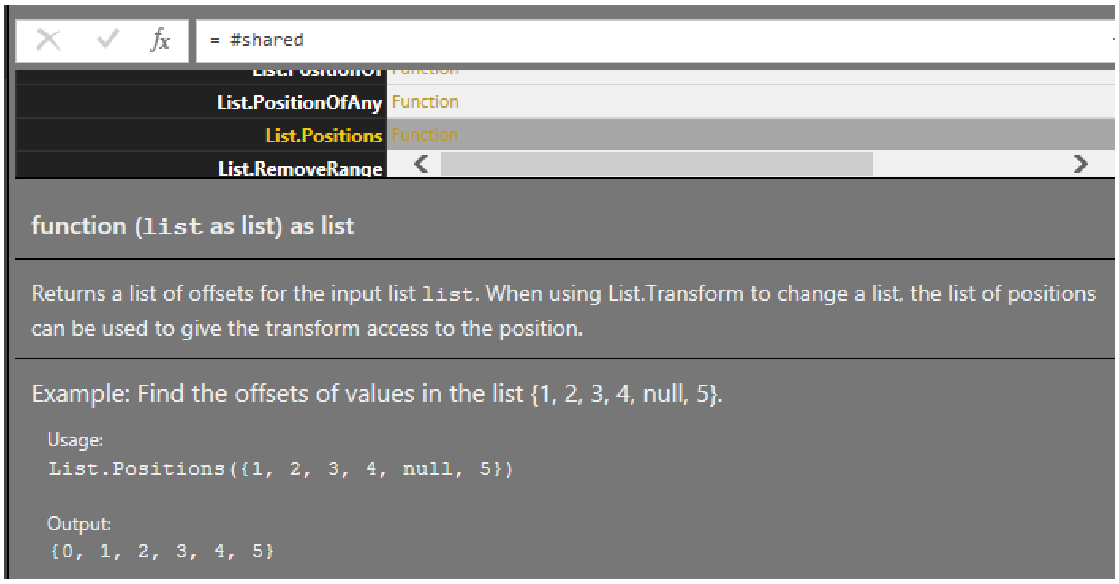
2017-05-01 Power Query

Whenever I'm faced with a data mashup problem in Power BI, I try to check if it can be resolved with a standard M functions exposed by the intrinsic variable *#shared*. Navigating this data structure when you don't know what you're looking for seems to be tedious task. Up until recently my function discovery workflow included to turn the output of *#shared* into a table and then search for keywords in the name of the functions. Hoping that the developers called the functions accordingly. However, after discovering where the documentation for the functions is stored I've come up with more flexible ways of navigating the 600+ definitions.

Function documentation

A while ago I've stumbled across a wonderful GitHub repository — Power BI Desktop Query Extensions written by Taylor Clark. This is one of those hidden treasures that is filled with golden nuggets. I recommend it to anyone interested in taking their Power Query skills to the next level. Although the repository is actually a single file with 545 LOC, in those lines you'll see some of the best examples of text, list and table manipulation. My favorite finds are those on testing and documenting M functions.

Apparently the documentation that you see while browsing *#shared* is stored as a *metadata record* on the *type* of the functions.



```
= #shared
```

List.PositionOfAny	Function
List.Positions	Function
List.RemoveRange	

function (list as list) as list

Returns a list of offsets for the input list list. When using List.Transform to change a list, the list of positions can be used to give the transform access to the position.

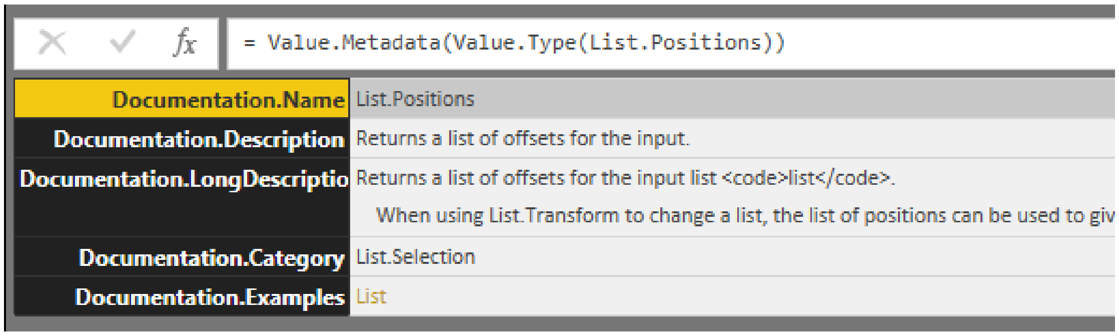
Example: Find the offsets of values in the list {1, 2, 3, 4, null, 5}.

Usage:
```
List.Positions({1, 2, 3, 4, null, 5})
```

Output:
```
{0, 1, 2, 3, 4, 5}
```

In order to view it you need to first check the *type* of the function and then retrieve the *metadata*. For example:

```
= Value.Metadata(Value.Type(List.Positions))
```



```
= Value.Metadata(Value.Type(List.Positions))
```

Documentation.Name	List.Positions
Documentation.Description	Returns a list of offsets for the input.
Documentation.LongDescriptio	Returns a list of offsets for the input list <code>list</code>. When using List.Transform to change a list, the list of positions can be used to giv
Documentation.Category	List.Selection
Documentation.Examples	List

Something that looks like a record in the query view, can easily be transformed into a table. And there are so many ways you can render the information stored as a table.

Function signature

Before proceeding any further there is still one piece of the puzzle missing —the signature of the functions. I couldn't find it neither in the documentation metadata, nor via other standard functions. So I had to roll out a custom query that generates a text like:

```
"function (list as list) as list"
```

It uses a combination of *Type.FunctionParameters*, *Type.FunctionReturn* and a looooong list of type to text conversion.

```
(placeholder as function)=>
let
//Serialize type to text
```

```
  TypeAsText = (value as any) =>
  let
    prefix = if Type.IsNullable(value) then "nullable " else ""
  in
    prefix&(
    if Type.Is(value, type binary) then "binary" else
    if Type.Is(value, type date) then "date" else
    if Type.Is(value, type datetime) then "datetime" else
    if Type.Is(value, type datetimezone) then "datetimezone" else
    if Type.Is(value, type duration) then "duration" else
    if Type.Is(value, type function) then "function" else
    if Type.Is(value, type list) then "list" else
    if Type.Is(value, type logical) then "logical" else
    if Type.Is(value, type none) then "none" else
    if Type.Is(value, type null) then "null" else
    if Type.Is(value, type number) then "number" else
    if Type.Is(value, type record) then "record" else
    if Type.Is(value, type table) then "table" else
    if Type.Is(value, type text) then "text" else
    if Type.Is(value, type time) then "time" else
    if Type.Is(value, type type) then "type" else
    if Type.Is(value, type any) then "any"
    else error "unknown"),
//if parameter is Optional set prefix
  OptionalPrefix = (_)=>if Type.IsNullable(_) then "optional " else "",
//get list of function parameters
  parameters = Type.FunctionParameters(Value.Type(placeholder)),
//create a text list of parameters and associate types "[optional] paramname as
  parametersWithTypes = List.Accumulate(Record.FieldNames(parameters),{},
                              (state,cur)=>state&{
                                     OptionalPrefix(Record.Field(parameters,
                                     cur&" as "&TypeAsText(Record.Field(para
in
//merge parameter list and prefix with "function (" and suffix with function re
  "function ("&
  Text.Combine(parametersWithTypes,", ")&
  ") as "&
  TypeAsText(Type.FunctionReturn(Value.Type(placeholder)))
```

Signature.cs hosted with ❤ by GitHub view raw

Stiching everything together

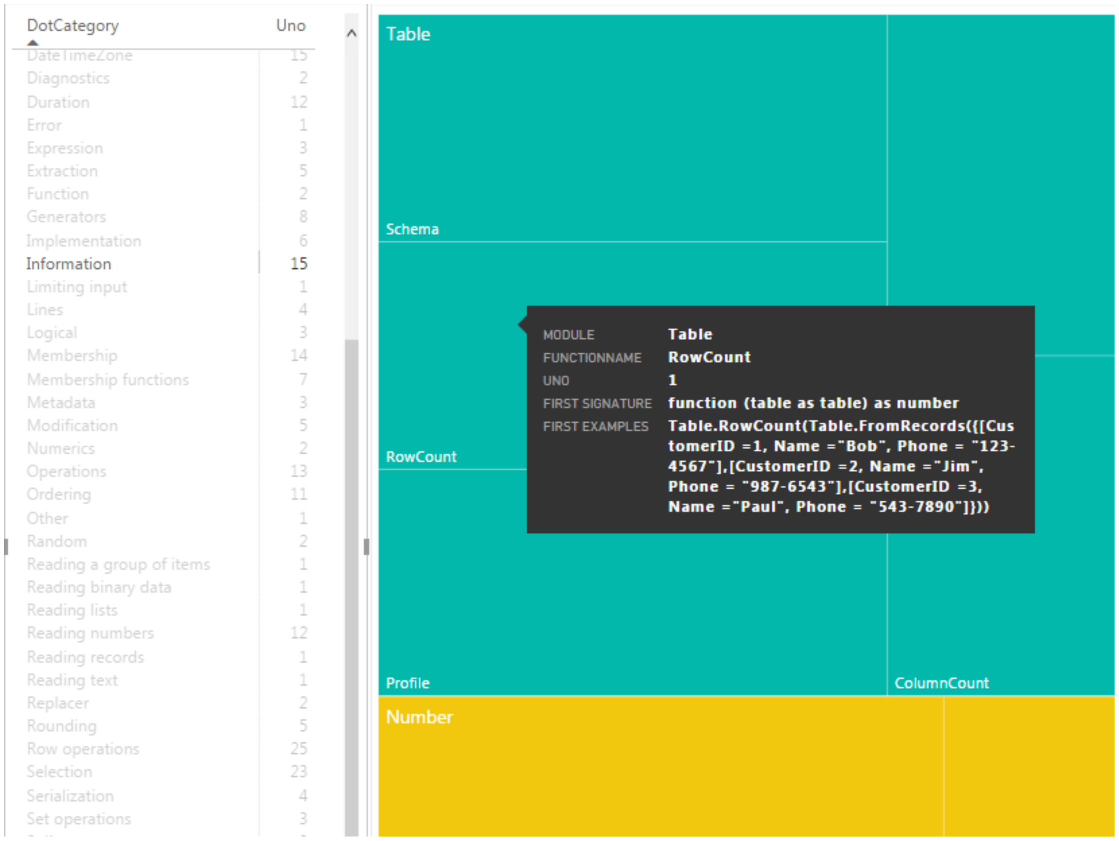
I've started from *#shared* record, transforming it to table, then filtering only on function definitions and then added columns one by one for *Category*, *Description*, function *Examples* and *Signature*. After some string manipulation to parse the *Module* names I ended up with the query below.

```
let
    sharedTab = Record.ToTable(#shared),
//select only functions from #shared
    functions = Table.SelectRows(sharedTab, each Type.Is(Value.Type([Value]),ty
//parse Module from function name
    modules = Table.AddColumn(functions, "Module", each Text.Split([Name], ".")
    functionNames = Table.AddColumn(modules, "FunctionName", each List.Last(Te
//get category from documentation
    categories = Table.AddColumn(functionNames, "Category", each try Value.Meta
//parse only the first code example from documentation
    examples = Table.AddColumn(categories, "Examples", each
                                            let eg = Value.Metadata(Value.Type(
                                            in if Type.Is(Value.Type(eg),type r
//get the short description from the documentation
    descriptions = Table.AddColumn(examples, "Description", each Value.Metadata
//parse subcategories
    subcategories = Table.AddColumn(descriptions, "DotCategory", each List.Last
//adding the signature of the functions
    out = Table.AddColumn(subcategories, "Signature", each Signature(Record.Fie
in
    out
```

NavigateShared.cs hosted with ❤ by GitHub view raw

I was planning to represent this data in a tile chart. Similar to a periodic table. However after experiencing performance problems while rendering 600 tiles with Infographic Designer 1.5.2, I gave up on the whole idea and opted to visualize all of this information in a treemap.

I've added a dummy column(*[Uno]*) with value 1 and used as a Value field in the treemap chart. Then I've added column *[Module]* to the Group field and the *[FunctionName]* in Details section. All of the remaining columns: *[Description]*, *[Signature]* and *[Examples]*, I've added to the Tooltips section of the chart. To control the treemap, I've used a matrix which acts as a filter on *[Category]* column.



DotCategory	Uno
DateTimeZone	15
Diagnostics	2
Duration	12
Error	1
Expression	3
Extraction	5
Function	2
Generators	8
Implementation	6
Information	15
Limiting input	1
Lines	4
Logical	3
Membership	14
Membership functions	7
Metadata	3
Modification	5
Numerics	2
Operations	13
Ordering	11
Other	1
Random	2
Reading a group of items	1
Reading binary data	1
Reading lists	1
Reading numbers	12
Reading records	1
Reading text	1
Replacer	2
Rounding	5
Row operations	25
Selection	23
Serialization	4
Set operations	3

MODULE	Table
FUNCTIONNAME	RowCount
UNO	1
FIRST SIGNATURE	function (table as table) as number
FIRST EXAMPLES	Table.RowCount(Table.FromRecords({[CustomerID =1, Name ="Bob", Phone = "123-4567"],[CustomerID =2, Name ="Jim", Phone = "987-6543"],[CustomerID =3, Name ="Paul", Phone = "543-7890"]}))

Click to view the embedded version of the report ➡️

Share this:

Print Twitter Facebook 4 Google LinkedIn WhatsApp

Like

Be the first to like this.

Some Power Query internals
In "Power Query"

R.Execute() the Swiss Army knife of Power Query
In "Power Query"

Closures, metadata and cascading parameters
In "Power Query"

One thought on "Navigating over 600+ M funcitons"

Pingback: Funciones de Power Query y la facilidad de #shared – Power BI y Business Intelligence