Microsoft Data Platform Business Intelligence Analytics Conference

Auckland, New Zealand
18-20 February 2019

www.difinity.co.nz

Power BI Custom Visuals

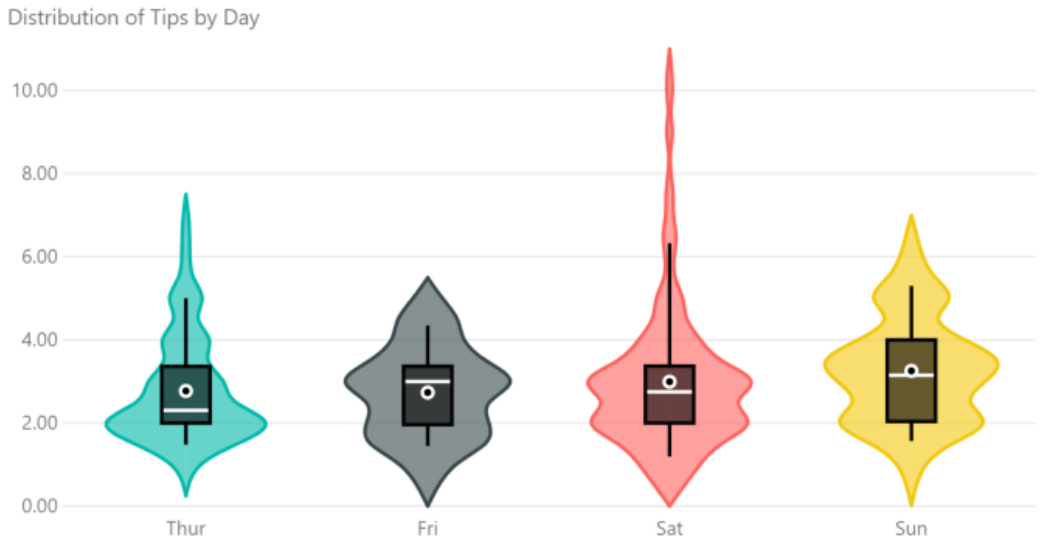
Getting Beyond the Boilerplate

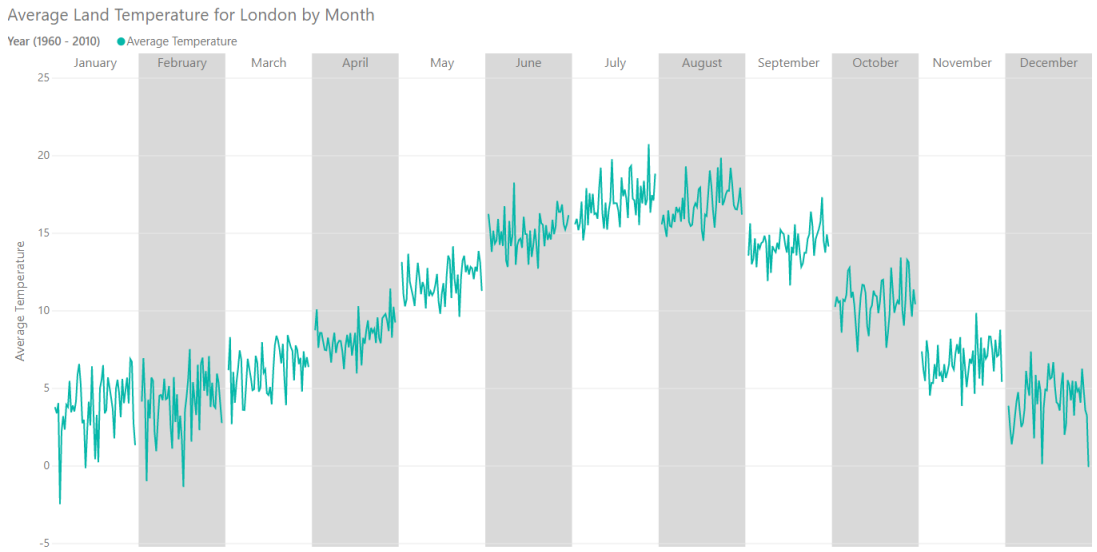
Daniel Marsh-Patrick

@the_d_mp daniel-m-p @run_dmp

By day: mild-mannered, Auckland-based Power BI implementation bloke

By night: author of Violin Plot & Small Multiple Line Chart Power BI Marketplace custom visuals





Agenda

Custom Visual Concepts

1. Toolchain
2. Creating a new visual
3. Anatomy

Making Our Visual

1. Visual Elements
2. Data Binding
3. Formatting Measures
4. Simple Properties
5. Additional Measures via Tooltip

Homework

Questions

Accompanying Materials

- GitHub repo: https://git.io/fhQxZ
- Key points are marked with tags so can be checked-out and compared
- Presentation slides will mention appropriate tag, e.g.:

`tag: 001`

- Checkout tag to view code as of that slide, e.g.:

```
git checkout 001
```

- If checking-out a tag that contains project package updates, run **`npm i`** just to be safe!

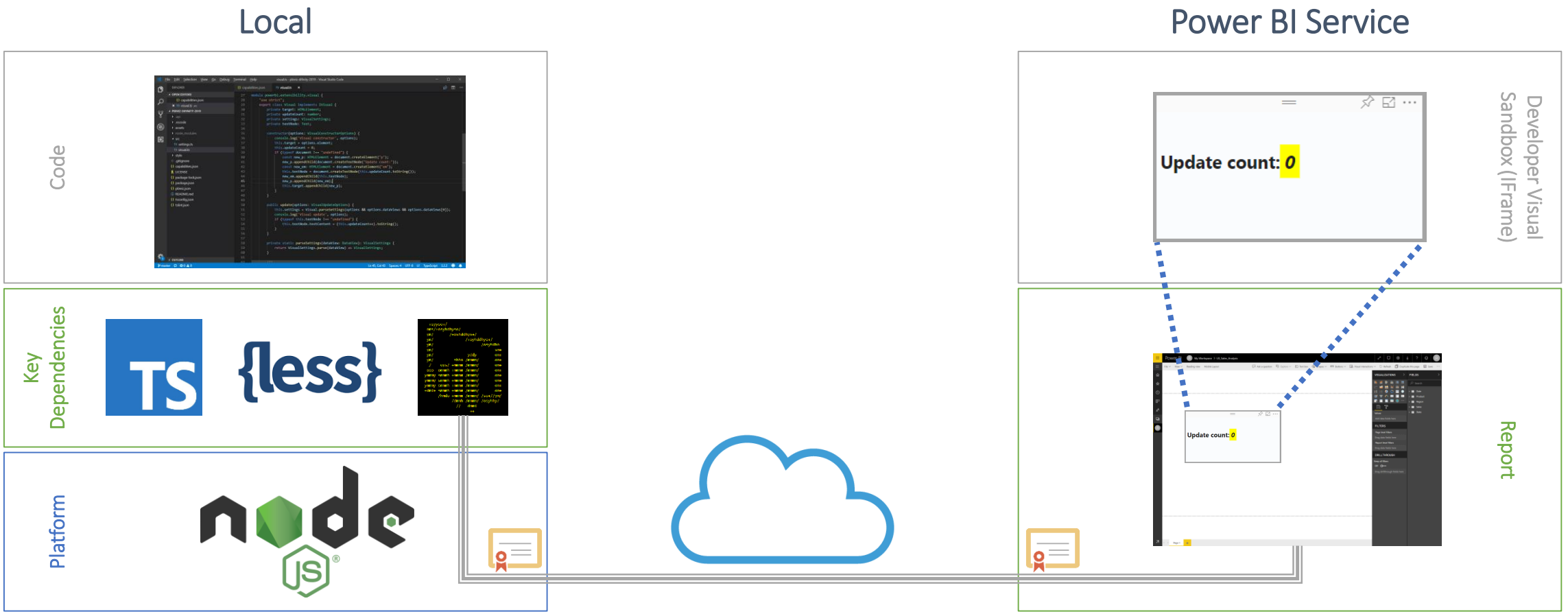
Custom Visual Concepts

An Overview for The Uninitiated… and perhaps also The Initiated

So, You Want to Develop A Custom Visual…

- Requires custom visuals SDK & supporting tools
- Develop locally w/TypeScript (visuals & slicers) or R (visuals only)
- Test/debug via Power BI Service & local development environment
- Alternatives to SDK:
  - R / Python (preview) visuals
  - D3.js custom visual - 'lift and shift' existing D3.js examples
  - Charticulator [gallery] – can export chart to Power BI Custom Visual

SDK Toolchain - 1,000m View



Toolchain - Development Environment

- Pre-requisites:
  - Power BI subscription (free is OK)
  - Visual Studio Code (or editor of choice)
  - PowerShell v4+ (Windows) / Terminal (OSX)
- Setup:
  - Node.js
  - `powerbi-visuals-tools`: `npm i powerbi-visuals-tools -g`
  - Create and install certificate: `pbiviz --create-cert` / `pbiviz --install-cert`
  - Create project: `pbiviz new`

Creating the Visual Project

tag: 001

- Create with tools - `pbiviz new [visualname]`,
- Only letters and numbers (will camel-case names)

- New visual with default template: `pbiviz new difinity`

- Other templates (using the `-t` switch):
  - `table`
  - `slicer`
  - `rvisual`
  - `rhtml`

Verifying Connection & Developer Visual

tag: 001

- Start the visual host (from root directory), e.g.:

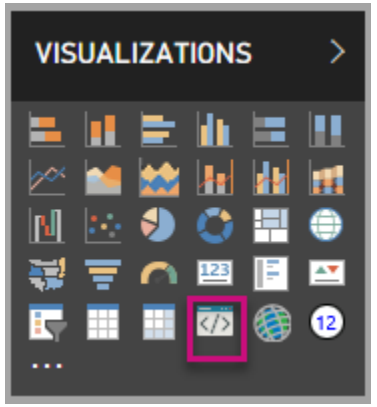
```
cd difinity
pbiviz start
```

- Create report in Power BI Service workspace
(we need data & canvas)

- Enable developer visual in Power BI Service workspace:

  Settings > Developer > Enable developer visual for testing

- Edit report and add developer visual



Anatomy – Key Files

tag: 001

- Build path:
  - `package.json` – Node.js packages
  - `tsconfig.json` – root files and TS compiler options

- Visual project:
  - `pbiviz.json` – project metadata
  - `capabilities.json` – describes visual to host
  - `src/visual.ts` – main module
  - `src/settings.ts` – classes to manage properties (settings)
  - `style/visual.less` – less + CSS style sheet

visual.ts

tag: 001

- (Default) file containing `IVisual` class and boilerplate code
- Matches specified class name in `pbiviz.json`
- Events & methods:

	Invoked	Purpose
`constructor`	When visual is added to the canvas	Initial setup of your visual
`update`	When specific events occur, and data fields are present (e.g. data change, property change, resize, etc.)	Manage supplied data, settings and subsequent display of your visual
`destroy`	[optional] When visual is deleted	Cleanup tasks, if required
`enumerateObjectInstances`	[optional] When properties pane selected [optional] When custom properties are changed	Read and process visual object data (custom properties) from configuration and render them in the properties pane

pbiviz.json

tag: 001

- Identifying metadata – name, author, version etc.
- Specifies API version
- Ensure `guid` is unique (particularly if you started working with another visual)
- Specifies any packages your visual needs for publication
- Some packages have dependencies that need to be explicitly declared (e.g. `formattingUtils`)

capabilities.json

tag: 001

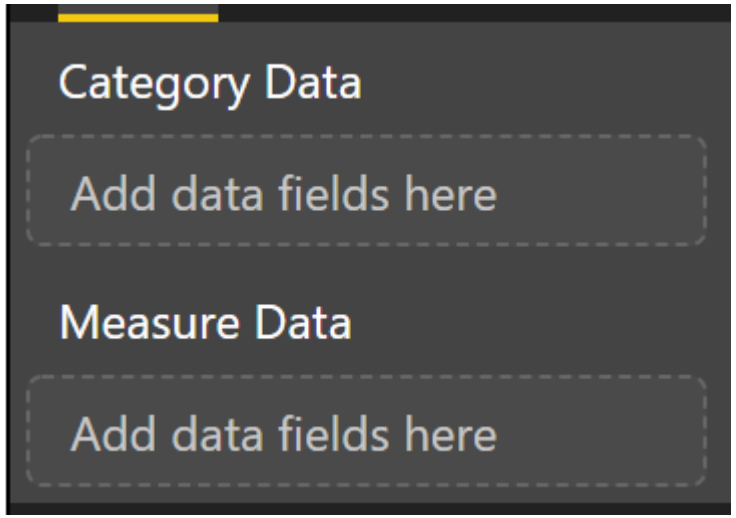
- Basics / essential learning:
  - Data Roles – expected fields
  - Data View Mappings – how data roles relate to each other
  - Objects – property pane options

- Additional features (not for today!):
  - Partial Highlight Support
  - Advanced Edit Mode
  - Sorting
  - + others

Data Roles - Initial

tag: 001

```
"dataRoles": [
    {
        "displayName": "Category Data",
        "name": "category",
        "kind": "Grouping"
    },
    {
        "displayName": "Measure Data",
        "name": "measure",
        "kind": "Measure"
    }
],
```



Data View Mapping - Initial

tag: 001

```
"dataViewMappings": [
    {
        "categorical": {
            "categories": {
                "for": {
                    "in": "category"
                },
                "dataReductionAlgorithm": {
                    "top": {}
                }
            },
            "values": {
                "select": [
                    {
                        "bind": {
                            "to": "measure"
                        }
                    }
                ]
            }
        }
    }
]
```

- How your data looks canonically
- One type supported per visual

  `single, categorical, table, matrix`

- Visual `update` method only fires if there's data added
- Specify valid mappings using `conditions`
- Use `dataReductionAlgorithm` to manage amount of data

  `top, bottom, sample, window`

Objects - Initial

tag: 001

capabilities.json

```
"objects": {
    "dataPoint": {
        "displayName": "Data colors",
        "properties": {
            "defaultColor": {…
            },
            "showAllDataPoints": {…
            },
            "fill": {…
            },
            "fillRule": {…
            },
            "fontSize": {…
            }
        }
    }
},
```

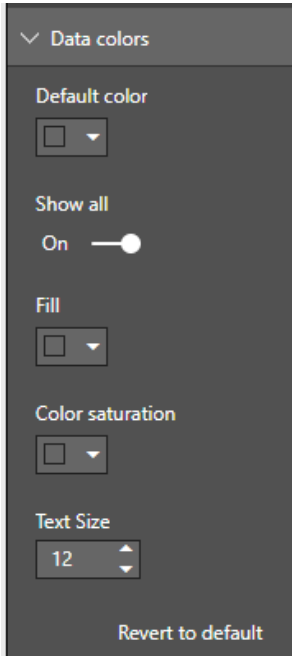
+

settings.ts (VisualSettings class)

```
export class VisualSettings extends DataViewObjectsParser {
  public dataPoint: dataPointSettings = new dataPointSettings();
  }

export class dataPointSettings {
 // Default color
  public defaultColor: string = "";
 // Show all
  public showAllDataPoints: boolean = true;
 // Fill
  public fill: string = "";
 // Color saturation
  public fillRule: string = "";
 // Text Size
  public fontSize: number = 12;
 }
```

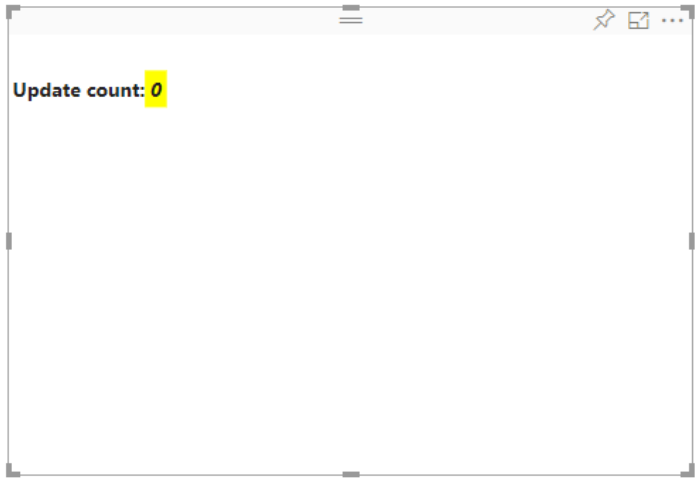
=



Let's Quickly Explore the Boilerplate Visual

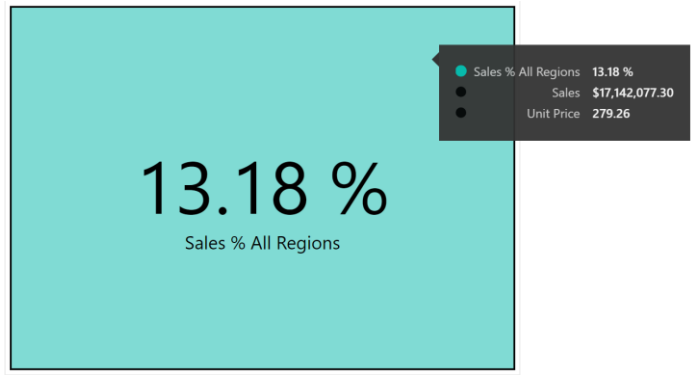
tag: 001

- Source code
- In-browser demo
- Events and console logging
- Data view (categorical mapping)



Making Our Visual

Our Visual



- Card displays single measure
- Configurable background colour & border (stroke) width
- Measure formatted according to data model
- Additional measures as tooltips

Before we Start… Let's Tear Down

The boilerplate visual confirms everything's working but we want to start with a clean slate (including d3.js):

- Clear down code and capabilities `tag: 002`
- Add d3.js and type definitions: `tag: 003`

```
npm i d3@3.5 --save
npm i @types/d3@3.5 --save-dev
```

Add to `externalJS` in `pbiviz.json`

Objective 1: Visual Elements

- Add properties for SVG elements to class — `tag: 004`
- Add elements to `constructor` — `tag: 005`
- Add single data role & view mapping (to trigger `update` method) — `tag: 006`
- Manage sizing/styling of elements in `update` method — `tag: 007`

Objective 2: Bind Data

- Inspect data view and measure formatting
- Restrict measure count — `tag: 008`
- Add `dataView` extraction to `update` method — `tag: 009`
- Replace placeholder value with value extracted from `dataView` — `tag: 010`
- Replace placeholder label with column `metadata` — `tag: 011`

Objective 3: Format the Measure

- Our measure displays, but doesn't look very good!
- The metadata contains formatting from the data model (if set)
- We can format this by using the Power BI Formatting Utilities:

```
npm i powerbi-visuals-utils-formattingutils --save
```

tag: 012

Add declarations to `tsconfig.json` build flow

tag: 013

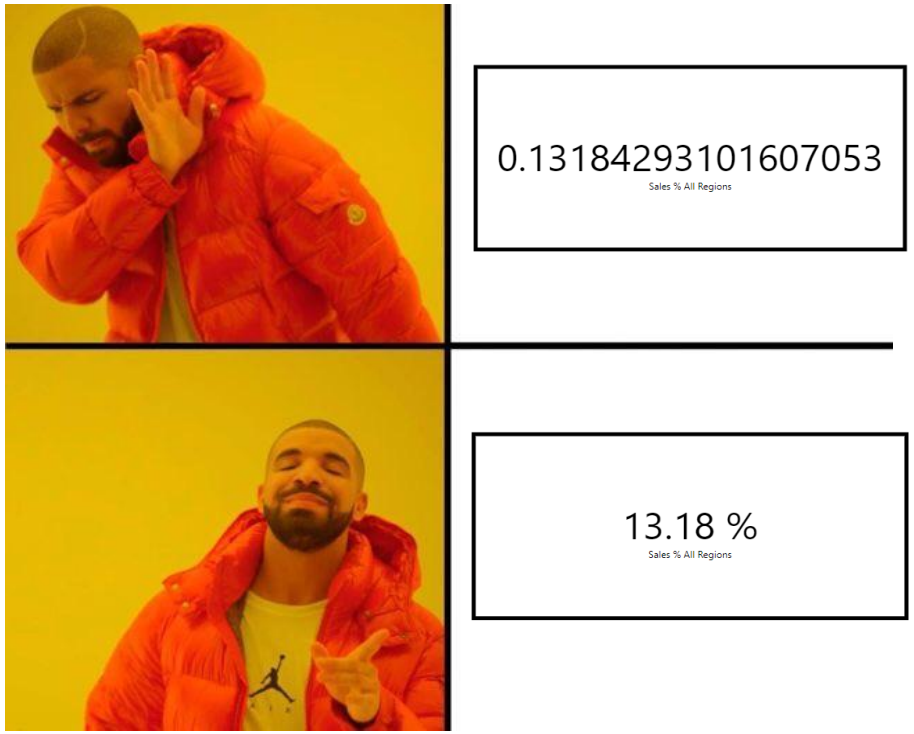
Add JavaScript artefacts to `pbiviz.json` `externalJS` section

tag: 014

Add CSS artefacts to `style/visual.less`

tag: 015

Objective 3 (Continued)



- Add the `valueFormatter` as accessible object in `visual.ts` `tag: 016`
- Modify the `text` method to apply formatting to the value `tag: 017`

Objective 4: Simple Properties

- Add objects to `capabilities.json` — tag: 018
- Add `CardSettings` class and `card` instance to `settings.ts` — tag: 019
- Verify that `objects` available in Developer Visual `metadata`
- Amend attributes for `rect` in `visual.ts`: — tag: 020
  - `fill`
  - `stroke-width`

Objective 5: Additional Measures as Tooltip

- We're going to add a Tooltip role to the visual
- Add more fields, displayed when we hover over
- Our `single` data role doesn't allow more fields to be added, so:
  - Add a new role for `tooltip` (with no `conditions`)
  - Change the data view mapping to `categorical` and map roles

`tag: 021`

- Update our existing code to use the correct data view

`tag: 022`

- Make our `measure` lookup more safe (if we add fields to `tooltip`, and remove the `measure`, it'll show the wrong thing!)

`tag: 023`

Objective 5 (Continued)

- Add [Power BI Tooltip Utilities]:

  `npm i powerbi-visuals-utils-tooltiputils --save` tag: 024

  Add declarations to `tsconfig.json` build flow tag: 025

  Add JavaScript artefacts to `pbiviz.json` `externalJS` section tag: 026

- Add declarations to `visual.ts` tag: 027

Objective 5 (Continued)

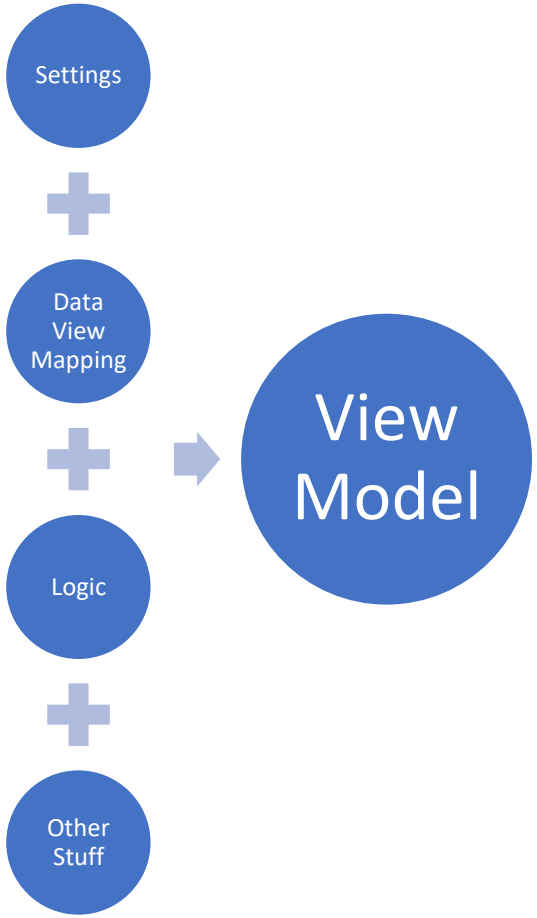
- Add `tooltipServiceWrapper` property to `Visual` class — tag: 028
- Instantiate `tooltipServiceWrapper` in the `constructor` — tag: 029
- Create `tooltips` array for `measure` field and `addTooltip` call — tag: 030
- Populate tooltip with additional fields — tag: 031

Homework

If you're still keen!

View Models

- Data view mappings surface data from your dataset to the visual
- Your visual will have its own presentation logic
- VM introduces *presentation separation*
- Keeps nuances of the DVM separate
- Try to structure logically to the rendering of your visual, which DVMs don't do
- Implement via TypeScript interfaces



Simple Interfaces Example

`powerbi.extensibility.VisualTooltipDataItem`

(developed and provided by MS in Visuals API)

```
interface VisualTooltipDataItem {
    displayName: string;
    value: string;
    color?: string;
    header?: string;
    opacity?: string;
}
```

- The ‘shape’ of data needed to produce a tooltip
- Used by our `tooltips` array, e.g.:

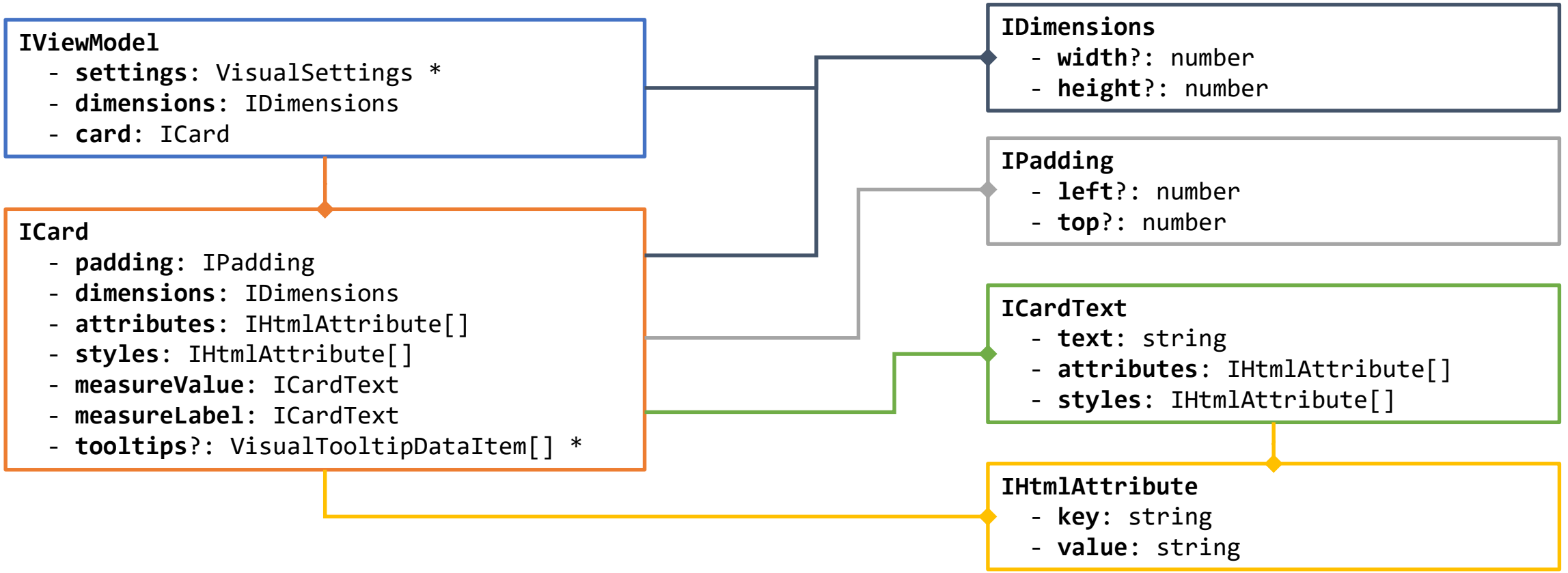
```
/** Add the measure data */
    tooltips.push({
        displayName: measureDisplayLabel,
        value: measureFormatted,
        color: this.visualSettings.card.fillColour
    });
```

- Note mandatory vs. optional properties

View Model for Our Visual

- Logic and rendering all currently in the `update` method:
  - Lots of repeated code
  - Adding more code will make method harder to manage
- Separation of duties:
  - View model to handle logic required to manage visual state
  - `update` method manages view model mapping and subsequent rendering
- Probably the biggest change we'll make so far!
  - Seems like a lot of effort, but makes it easier to scale changes
  - Will simplify testing if logic is as consolidated as possible
  - Makes sense to think about defining one up-front

View Model Structure



* provided by API/external MS packages

Implementing the VM – Supporting Interfaces

- Create `src/interfaces.ts` — `tag: 032`
- Add to `tsconfig.json` build flow — `tag: 033`
- Add interface declarations — `tag: 034`

Implementing the VM – Initial View Model

- This will handle the default behaviour and verify design
- Create `src/viewModel.ts` & add to `tsconfig.json` build flow

tag: 035

- Add view model interface declaration (`IViewModel`)

tag: 036

- Create initial `visualTransform` function in `viewModel.ts`

tag: 037

- Because we changed the build flow, restart the visual host

Implementing the VM – Separation of Logic

- Modify the `update` function in `visual.ts` to use view model — `tag: 038`
- All logic now being derived in `visualTransform` and `update` takes care of the rendering
- Verify visual looks okay – should behave similar to Objective #1
- Add condition checks for valid data view — `tag: 039`
- Update view model with measure and tooltip data — `tag: 040`
- Verify functionality is as before (it should be!)

Thanks for Tuning In!

Hopefully we have time for questions!

Where's that GitHub repo again? https://git.io/fhQxZ
Remember to `npm i` when exploring!

Thanks to our sponsors

Platinum Sponsors

Microsoft

RADACAD

Silver

Wardy it solutions™

Exhibitor

TIMEXTENDER

Evaluate Sessions and Win a Prize!

https://www.surveymonkey.com/r/5LR9LFB