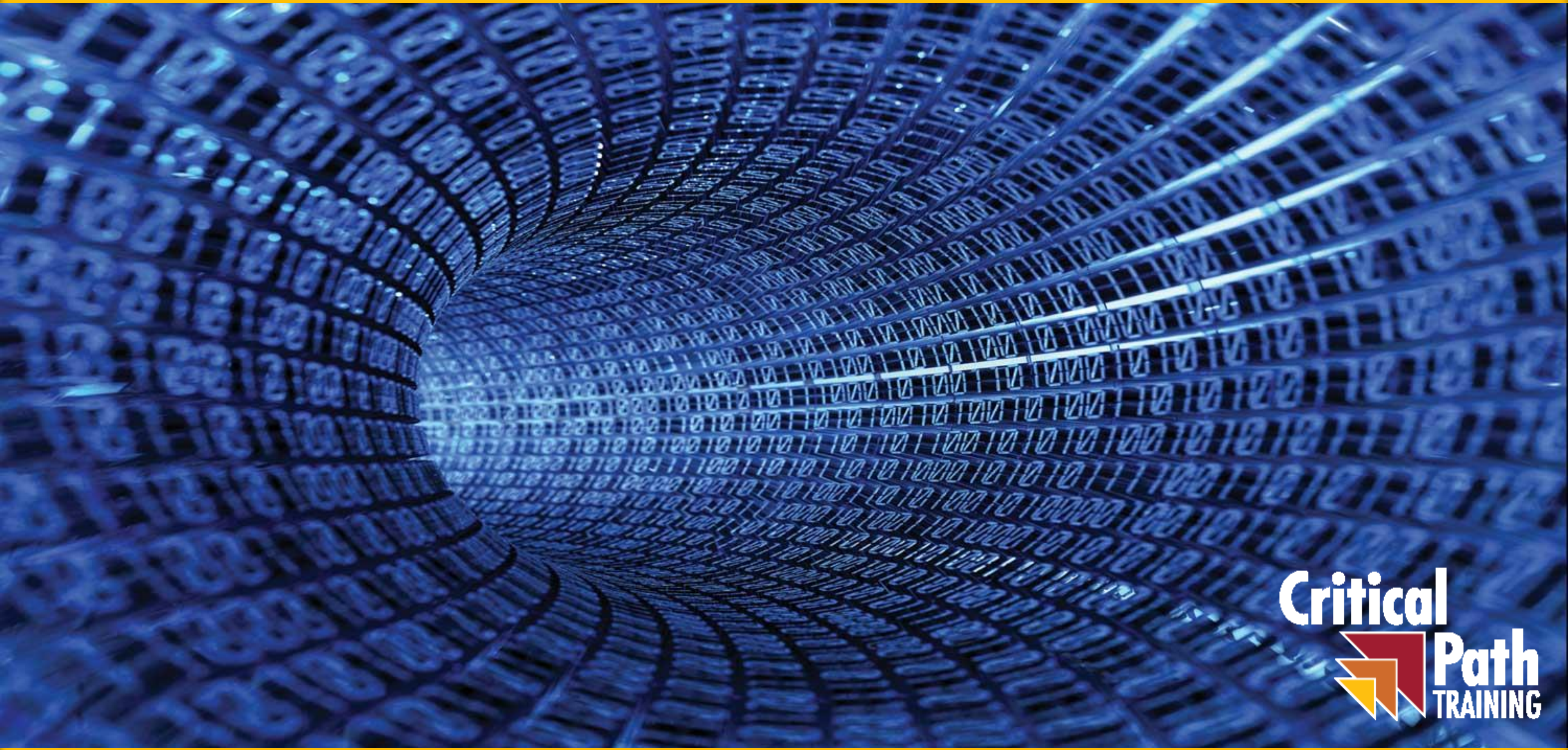
Developing and Distributing Custom Visuals

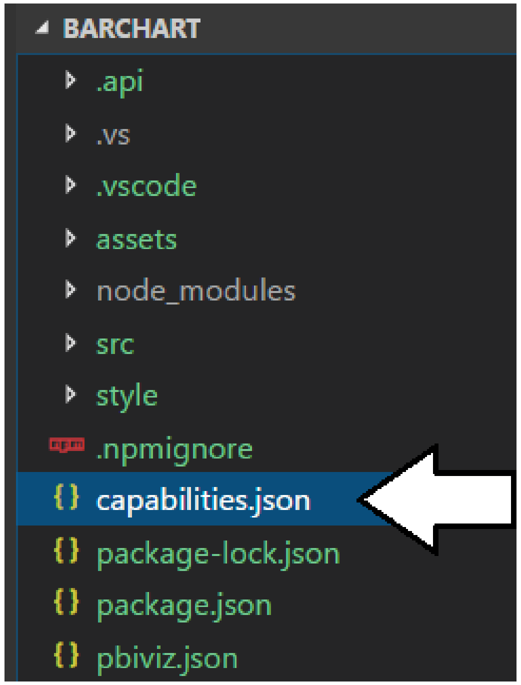


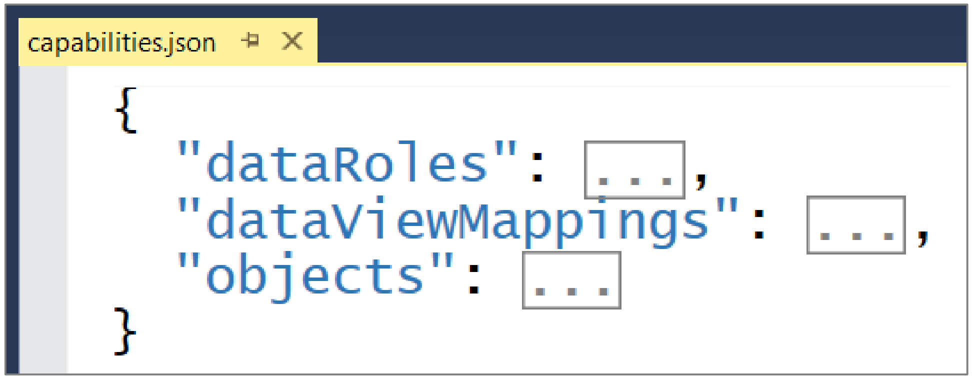
Agenda

- Defining Data Roles and Data Mappings
- Extending a Visual with Custom Properties
- Designing Custom Visuals using a View Model
- Advanced Custom Visual Design Features
- Packaging and Deploying Custom Visuals

Visual Capabilities

- Visual capabilities defined inside **capabilities.json**
  - **dataRoles** defines the field wells displayed on Fields pane
  - **dataViewMappings** defines the type of DataView used by visual
  - **objects** defines custom properties for visual



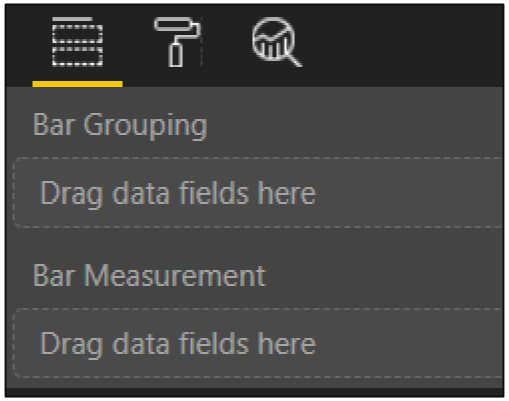


```
{
  "dataRoles": ...,
  "dataViewMappings": ...,
  "objects": ...
}
```

Data Roles

- DataRoles define how fields are associated with visual
  - Each dataRole is display as field well in the Field pane
  - dataRoles can be defined with conditions and data mappings

```
"dataRoles": [
  {
    "displayName": "Bar Grouping",
    "name": "myCategory",
    "kind": "Grouping"
  },
  {
    "displayName": "Bar Measurement",
    "name": "myMeasure",
    "kind": "Measure"
  }
]
```


Bar Grouping

Drag data fields here

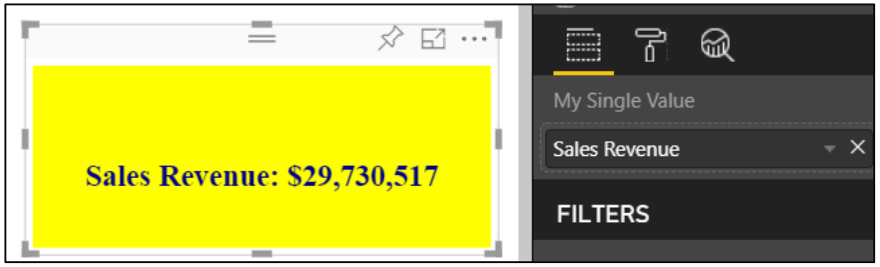
Bar Measurement

Drag data fields here

Data Mapping Modes

- Power BI visual API provides several mapping modes
  - Single
  - Table
  - Categorical
  - Matrix
  - Tree

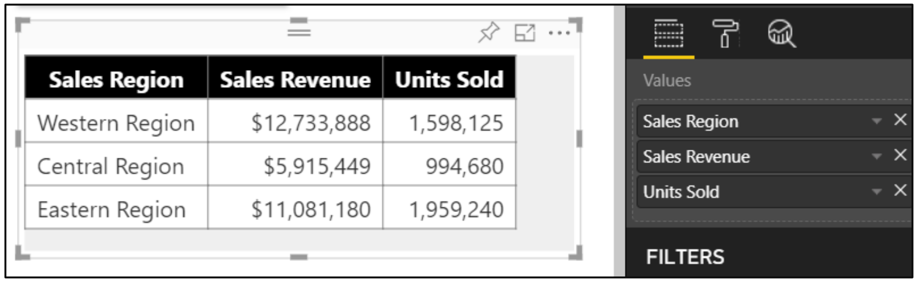
Single Mapping



Sales Revenue: $29,730,517

My Single Value: Sales Revenue

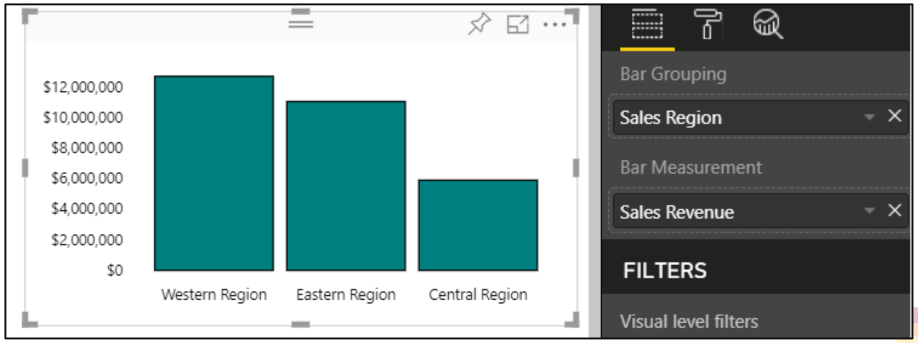
Table Mapping



Sales Region	Sales Revenue	Units Sold
Western Region	$12,733,888	1,598,125
Central Region	$5,915,449	994,680
Eastern Region	$11,081,180	1,959,240

Values: Sales Region, Sales Revenue, Units Sold

Categorical Mapping

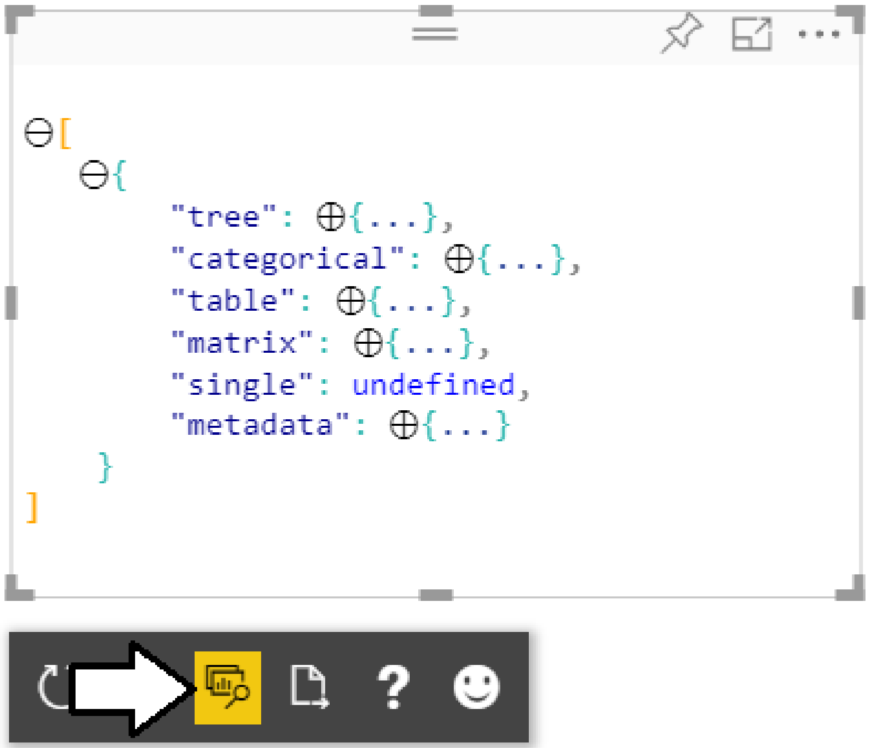


Bar Grouping: Sales Region

Bar Measurement: Sales Revenue

Developer Visual DataView

- Developer visual supports DataView mode
  - Allows you to see and explore data mapping
  - Allows you to see metadata for custom properties

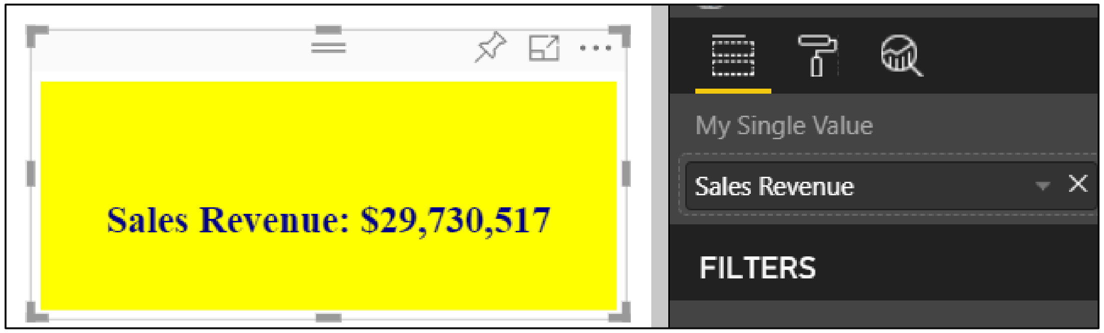


```
⊖[
   ⊖{
        "tree": ⊕{...},
        "categorical": ⊕{...},
        "table": ⊕{...},
        "matrix": ⊕{...},
        "single": undefined,
        "metadata": ⊕{...}
    }
]
```

Single Mapping Example: oneBigNumber

- dataRole can use dataViewMapping mode of single
  - For visuals like Card which only display single value
  - Condition can define that a dataRole requires exactly one measure

```
"dataRoles": [
  {
    "displayName": "My Single Value",
    "name": "myvalue",
    "kind": "Measure"
  }
],
"dataViewMappings": [
  {
    "conditions": [ { "myvalue": { "min": 1, "max": 1 } } ],
    "single": { "role": "myvalue" }
  }
]
```


Sales Revenue: $29,730,517

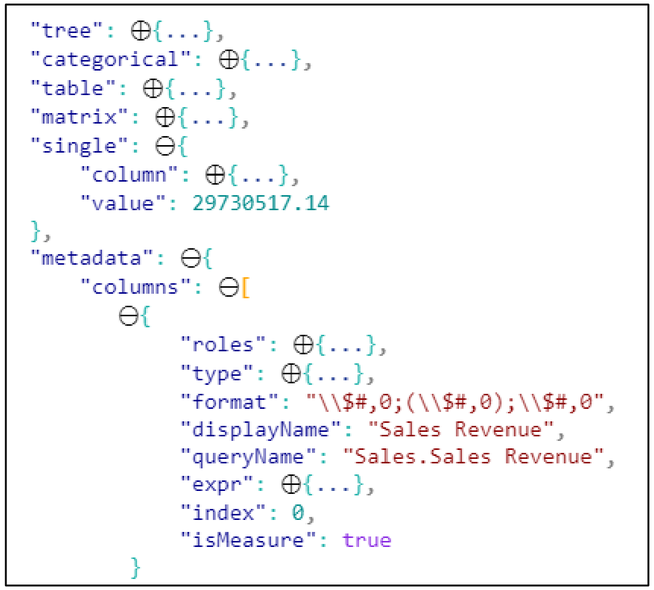
My Single Value

Sales Revenue

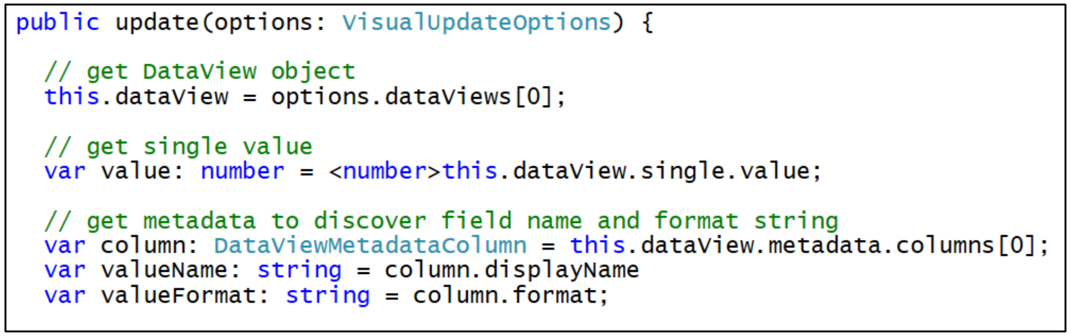
FILTERS

Programming in Single Mapping Mode

- Single mapping easy to access through visuals API
  - DataView object provides single.value property
  - value property defined as PrimativeValue { bool | number | string }
  - PrimativeValue must be explicitly cast
  - Other measure properties available through column metadata



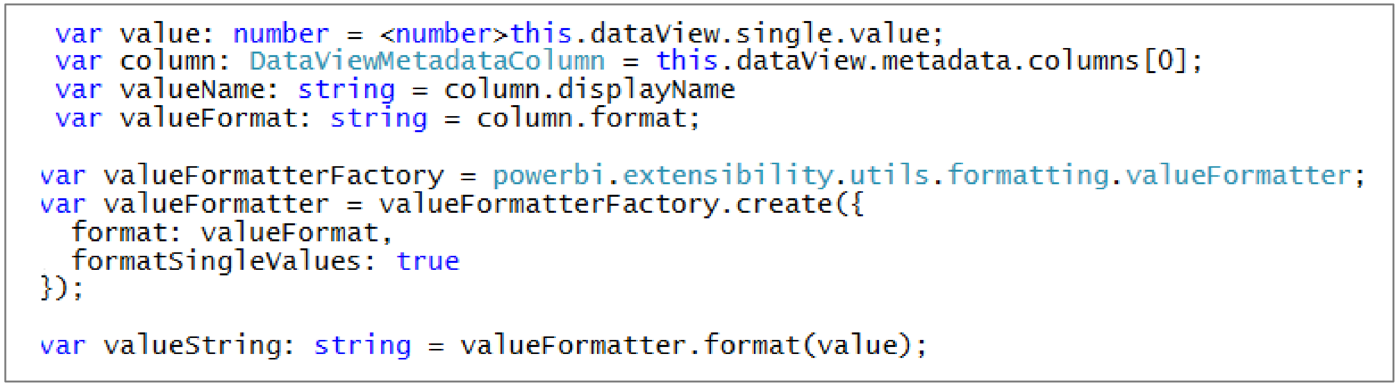
```
"tree": ⊕{...},
"categorical": ⊕{...},
"table": ⊕{...},
"matrix": ⊕{...},
"single": ⊖{
    "column": ⊕{...},
    "value": 29730517.14
},
"metadata": ⊖{
    "columns": ⊖[
      ⊖{
          "roles": ⊕{...},
          "type": ⊕{...},
          "format": "\\$#,0;(\\$#,0);\\$#,0",
          "displayName": "Sales Revenue",
          "queryName": "Sales.Sales Revenue",
          "expr": ⊕{...},
          "index": 0,
          "isMeasure": true
      }
```



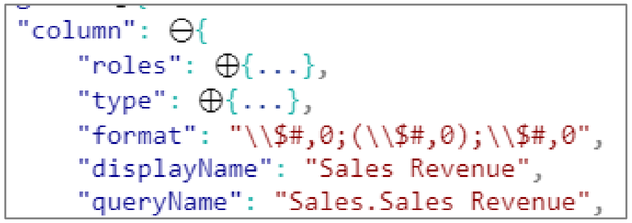
```
public update(options: VisualUpdateOptions) {

  // get DataView object
  this.dataView = options.dataViews[0];

  // get single value
  var value: number = <number>this.dataView.single.value;

  // get metadata to discover field name and format string
  var column: DataViewMetadataColumn = this.dataView.metadata.columns[0];
  var valueName: string = column.displayName
  var valueFormat: string = column.format;
```

Using the Power BI Formatting Utilities

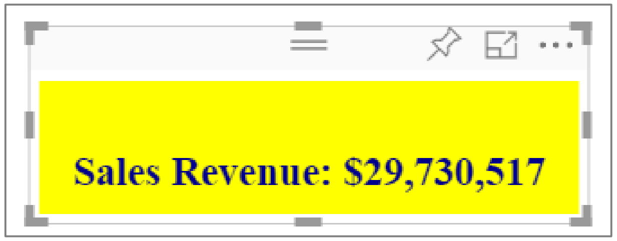
- Used to format values using Power BI formatting strings
  - Requires installing powerbi-visuals-utils-formattingutils package



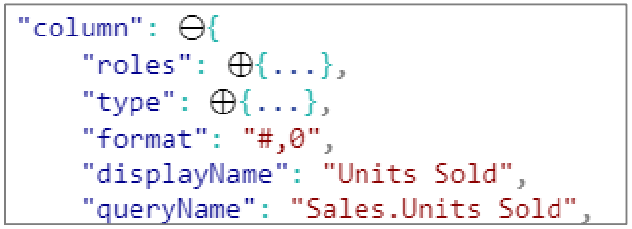
```
  var value: number = <number>this.dataView.single.value;
  var column: DataViewMetadataColumn = this.dataView.metadata.columns[0];
  var valueName: string = column.displayName
  var valueFormat: string = column.format;

 var valueFormatterFactory = powerbi.extensibility.utils.formatting.valueFormatter;
 var valueFormatter = valueFormatterFactory.create({
   format: valueFormat,
   formatSingleValues: true
 });

 var valueString: string = valueFormatter.format(value);
```



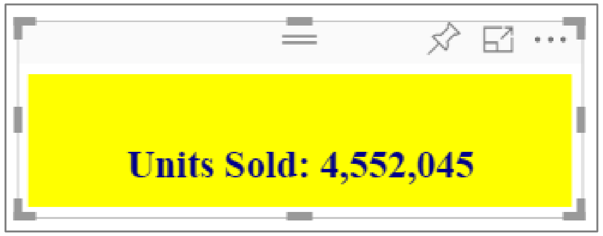
```
"column": ⊖{
    "roles": ⊕{...},
    "type": ⊕{...},
    "format": "\\$#,0;(\\$#,0);\\$#,0",
    "displayName": "Sales Revenue",
    "queryName": "Sales.Sales Revenue",
```



Sales Revenue: $29,730,517



```
"column": ⊖{
    "roles": ⊕{...},
    "type": ⊕{...},
    "format": "#,0",
    "displayName": "Units Sold",
    "queryName": "Sales.Units Sold",
```

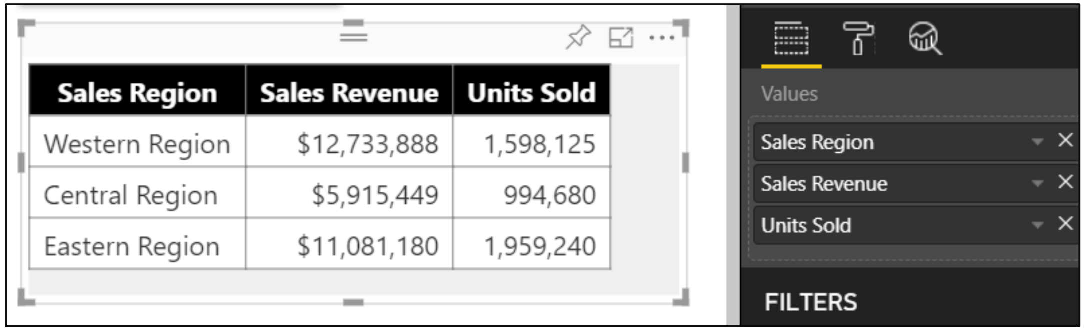


Units Sold: 4,552,045

Table Mapping Example: Snazzy Table

- dataRole can use dataViewMapping mode of table
  - For visuals which display rows & columns for ordered set of fields
  - condition can define number of fields that can be added

```
"dataRoles": [
  {
    "displayName": "Values",
    "name": "values",
    "kind": "GroupingOrMeasure"
  }
],
"dataViewMappings": [
  {
    "conditions": [ {"values": { "min": 1, "max": 5 } } ],
    "table": { "rows": { "for": { "in": "values" } } }
  }
]
```



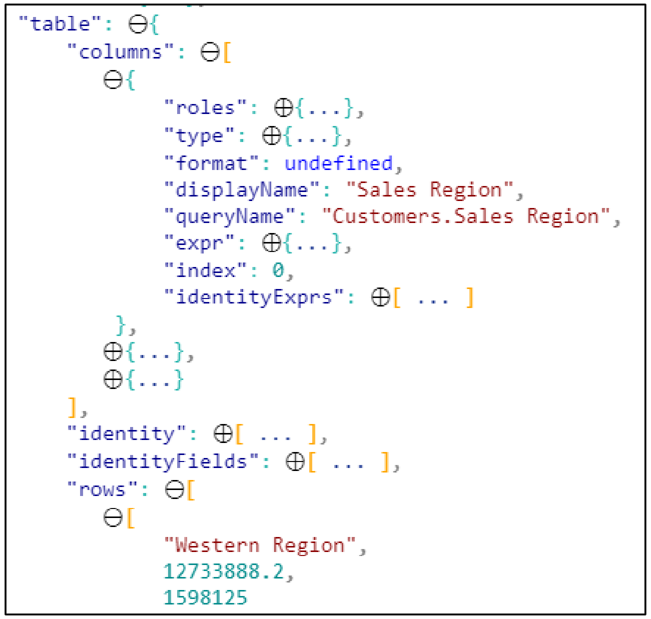
Sales Region	Sales Revenue	Units Sold
Western Region	$12,733,888	1,598,125
Central Region	$5,915,449	994,680
Eastern Region	$11,081,180	1,959,240

Values
- Sales Region
- Sales Revenue
- Units Sold

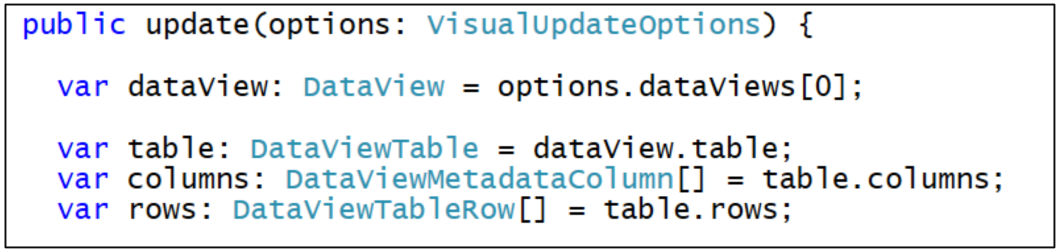
FILTERS

Programming in Table Mapping Mode

- Table mapping data accessible through visuals API
  - DataView object provides table property
  - table property provides columns property and rows property



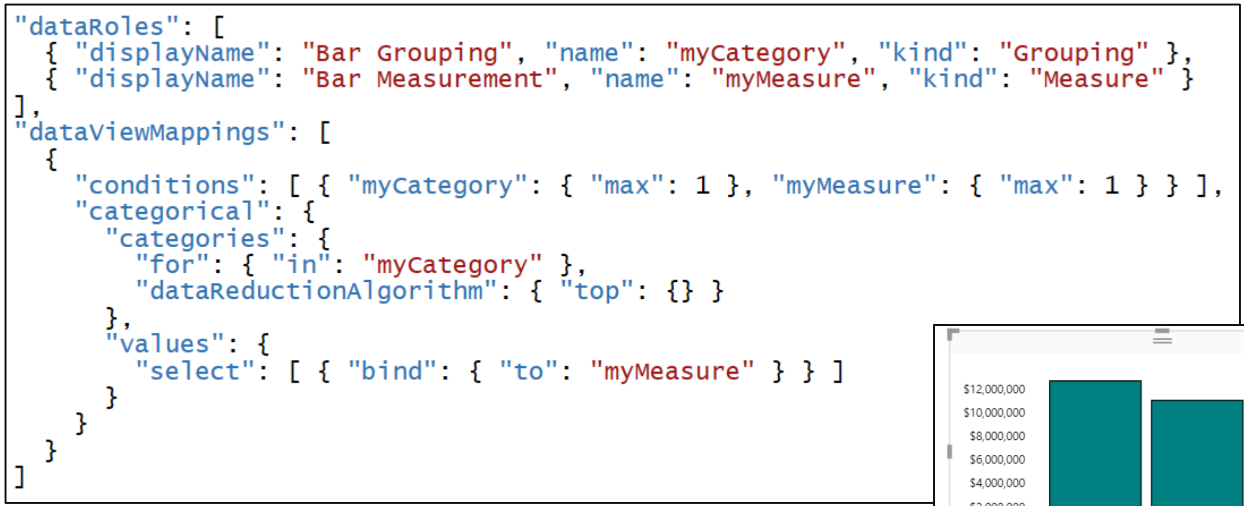
```
"table": ⊖{
    "columns": ⊖[
      ⊖{
          "roles": ⊕{...},
          "type": ⊕{...},
          "format": undefined,
          "displayName": "Sales Region",
          "queryName": "Customers.Sales Region",
          "expr": ⊕{...},
          "index": 0,
          "identityExprs": ⊕[ ... ]
       },
      ⊕{...},
      ⊕{...}
    ],
    "identity": ⊕[ ... ],
    "identityFields": ⊕[ ... ],
    "rows": ⊖[
      ⊖[
          "Western Region",
          12733888.2,
          1598125
```



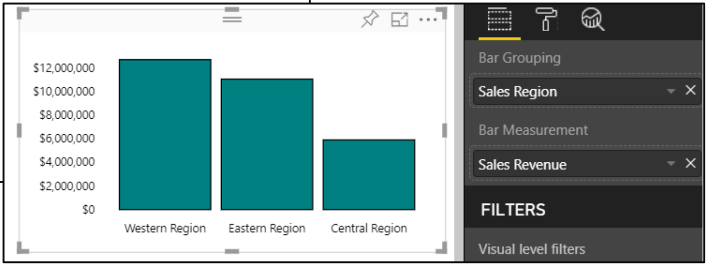
```
public update(options: VisualUpdateOptions) {

  var dataView: DataView = options.dataViews[0];

  var table: DataViewTable = dataView.table;
  var columns: DataViewMetadataColumn[] = table.columns;
  var rows: DataViewTableRow[] = table.rows;
```

Categorical Mapping Example: Barchart

- dataRole can use dataViewMapping mode of categorical
  - This is the most common type of data mapping
  - For visuals which divide data into groups for analysis
  - Groups defined as columns and values defined as measures



```
"dataRoles": [
  { "displayName": "Bar Grouping", "name": "myCategory", "kind": "Grouping" },
  { "displayName": "Bar Measurement", "name": "myMeasure", "kind": "Measure" }
],
"dataViewMappings": [
  {
    "conditions": [ { "myCategory": { "max": 1 }, "myMeasure": { "max": 1 } } ],
    "categorical": {
      "categories": {
        "for": { "in": "myCategory" },
        "dataReductionAlgorithm": { "top": {} }
      },
      "values": {
        "select": [ { "bind": { "to": "myMeasure" } } ]
      }
    }
  }
]
```



Bar Grouping: Sales Region

Bar Measurement: Sales Revenue

FILTERS

Visual level filters

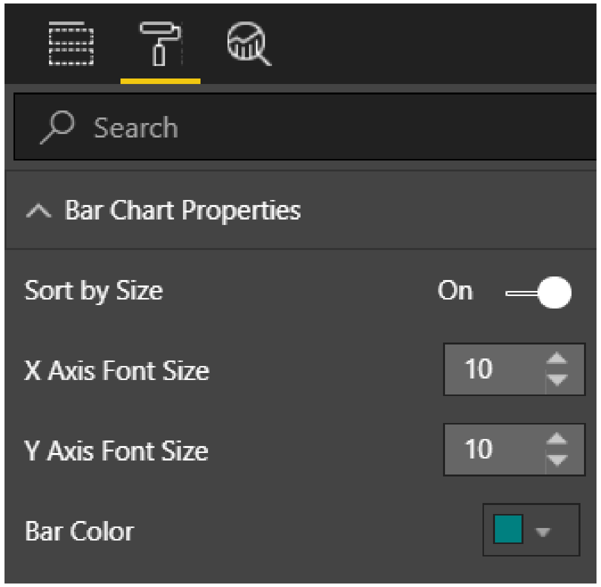
Agenda

- ✓ Defining Data Roles and Data Mappings
- ➢ Extending a Visual with Custom Properties
- Designing Custom Visuals using a View Model
- Advanced Custom Visual Design Features
- Packaging and Deploying Custom Visuals

Extending Visuals with Custom Properties

- Custom properties defined using **objects**
  - You can define one or more objects in **capabilities.json**
  - Each object defined with name, display name and properties
  - object properties automatically persistent inside visual metadata
  - properties can be seen and modified by user in Format pane
  - Custom properties require extra code to initialize Format pane

```
"objects": {
  "barchartProperties": {
    "displayName": "Bar Chart Properties",
    "properties": {
      "sortBySize": {
        "displayName": "Sort by Size",
        "type": { "bool": true }
      },
      "xAxisFontSize": {
        "displayName": "X Axis Font Size",
        "type": { "integer": true }
      },
      "yAxisFontSize": {
        "displayName": "Y Axis Font Size",
        "type": { "integer": true }
      },
      "barColor": {
        "displayName": "Bar Color",
        "type": { "fill": { "solid": { "color": true } } }
      }
    }
  }
}
```



Bar Chart Properties

Sort by Size: On

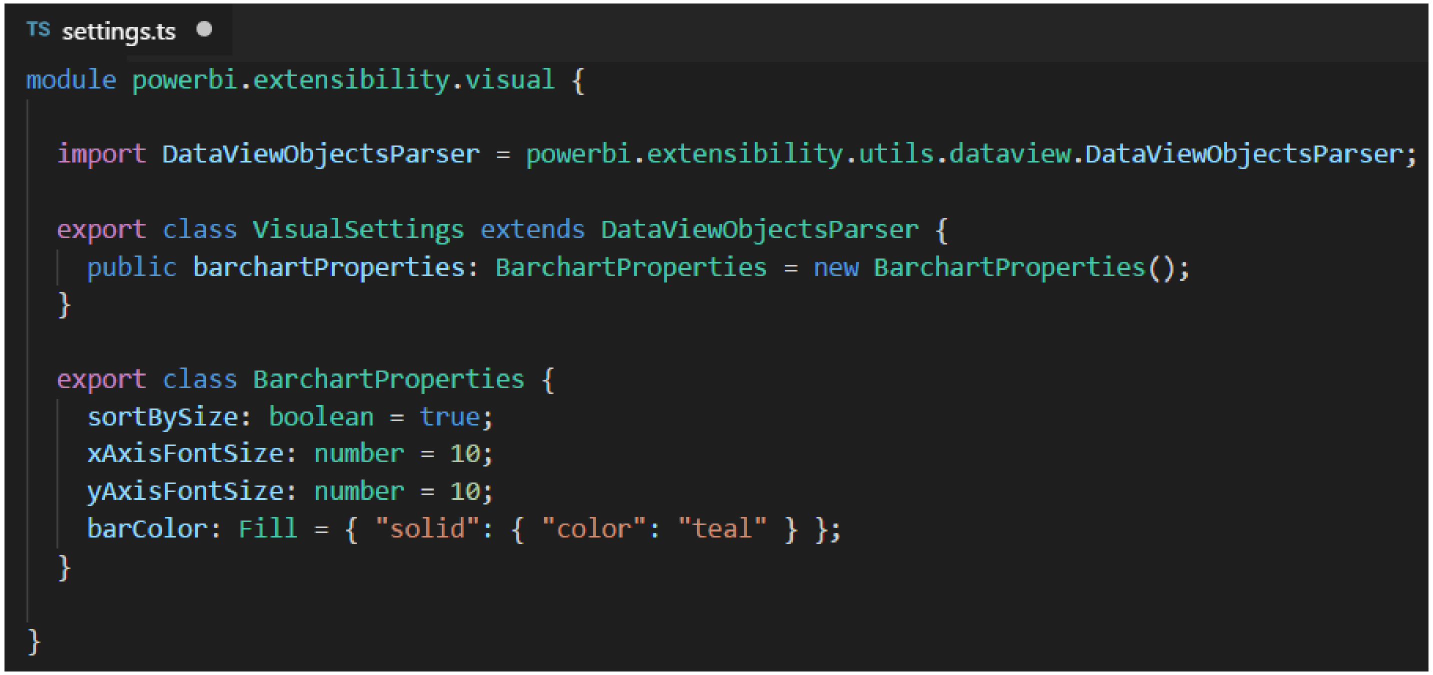
X Axis Font Size: 10

Y Axis Font Size: 10

Bar Color: 🟦

DataViewObjectParser and VisualSettings

- Power BI visual utilities provide DataViewObjectParser
  - Abstracts away tricky code to initialize and read property values

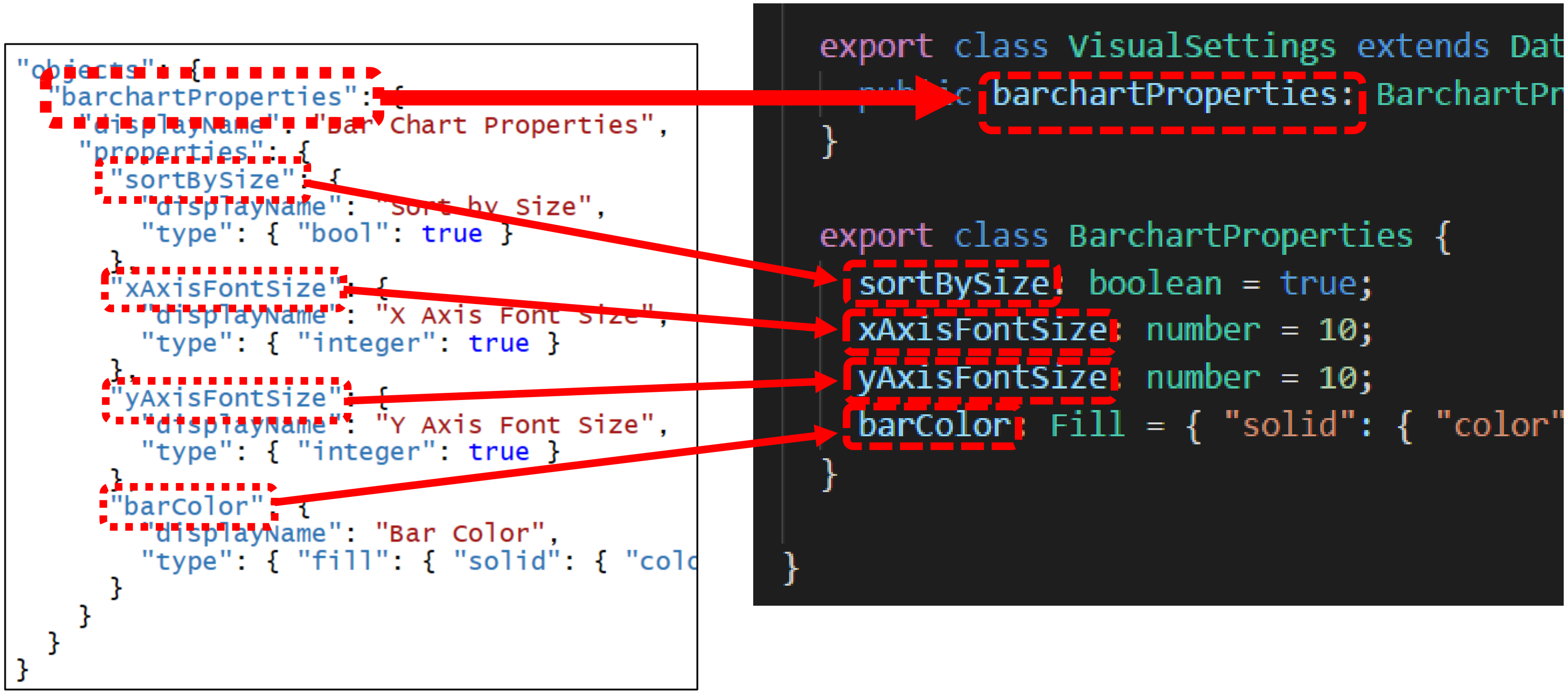


settings.ts

```
module powerbi.extensibility.visual {

  import DataViewObjectsParser = powerbi.extensibility.utils.dataview.DataViewObjectsParser;

  export class VisualSettings extends DataViewObjectsParser {
    public barchartProperties: BarchartProperties = new BarchartProperties();
  }

  export class BarchartProperties {
    sortBySize: boolean = true;
    xAxisFontSize: number = 10;
    yAxisFontSize: number = 10;
    barColor: Fill = { "solid": { "color": "teal" } };
  }

}
```

Mapping Object Properties to VisualSettings

- VisualSettings class must map to named objectnamed
  - VisualSetting class contains named field that maps to object name
  - Named field based on custom class with mapped properties
  - Object & property names must match what's in capabilities.json



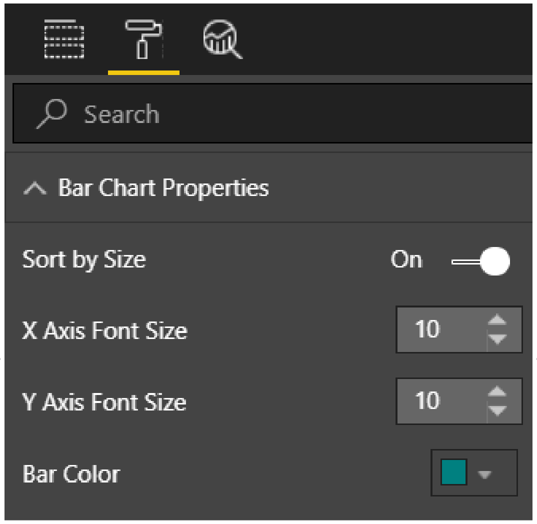
```
"objects": {
  "barchartProperties": {
    "displayName": "Bar Chart Properties",
    "properties": {
      "sortBySize": {
        "displayName": "Sort by Size",
        "type": { "bool": true }
      },
      "xAxisFontSize": {
        "displayName": "X Axis Font Size",
        "type": { "integer": true }
      },
      "yAxisFontSize": {
        "displayName": "Y Axis Font Size",
        "type": { "integer": true }
      },
      "barColor": {
        "displayName": "Bar Color",
        "type": { "fill": { "solid": { "colo
      }
    }
  }
}
```

```
export class VisualSettings extends Dat
  public barchartProperties: BarchartPr
}

export class BarchartProperties {
  sortBySize: boolean = true;
  xAxisFontSize: number = 10;
  yAxisFontSize: number = 10;
  barColor: Fill = { "solid": { "color"
}

}
```

Initializing Objects in the Format Pane

- Visual must initialize properties in Format pane
  - Visual must implement enumerateObjectInstances
  - VisualSettings makes this relatively easy
  - Extra code required to make property appear as spinner

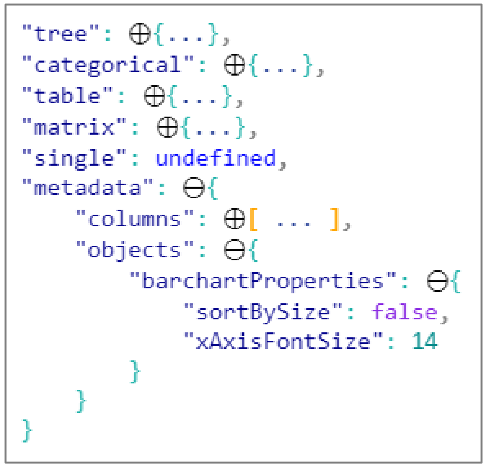
```
public enumerateObjectInstances(options: EnumerateVisualObjectInstancesOptions): VisualObjectInstanceEnumeration {

  // register object properties
  var visualObjects: VisualObjectInstanceEnumerationObject =
    <VisualObjectInstanceEnumerationObject>VisualSettings
      .enumerateObjectInstances(this.settings, options);

  // configure spinners for integers properties
  visualObjects.instances[0].validValues = {
    xAxisFontSize: { numberRange: { min: 10, max: 36 } },
    yAxisFontSize: { numberRange: { min: 10, max: 36 } },
  };

  // return visual object collection
  return visualObjects;
}
```



Sort by Size On
X Axis Font Size 10
Y Axis Font Size 10
Bar Color 🟦

Retrieving Property Values

- Property values persisted into visual metadata
  - Properties not persisted white they still retain default values



```
"tree": ⊕{...},
"categorical": ⊕{...},
"table": ⊕{...},
"matrix": ⊕{...},
"single": undefined,
"metadata": ⊖{
    "columns": ⊕[ ... ],
    "objects": ⊖{
        "barchartProperties": ⊖{
            "sortBySize": false,
            "xAxisFontSize": 14
        }
    }
}
```

  - Property values retrieved using VisualSettings object

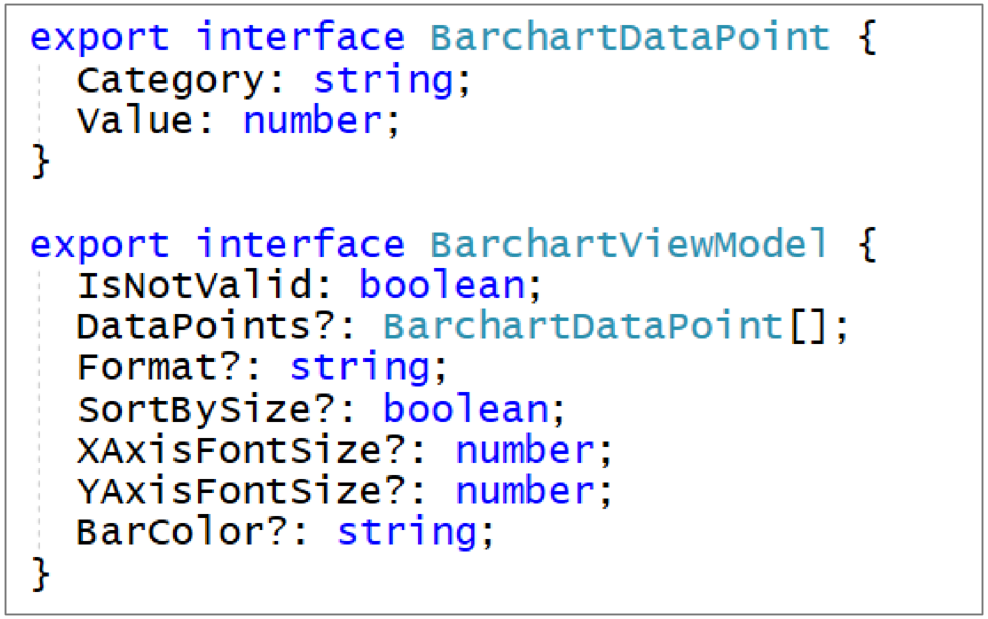
```
public update(options: VisualUpdateOptions) {

  if (options.dataViews[0]) {

    // create VisualSettings object
    this.settings = VisualSettings.parse(options.dataViews[0]) as VisualSettings;

    // retrieve property values
    var sortBySize: boolean = this.settings.barchartProperties.sortBySize
    var xAxisFontSize: number = this.settings.barchartProperties.xAxisFontSize;
```

Agenda

- ✓ Defining Data Roles and Data Mappings
- ✓ Extending a Visual with Custom Properties
- ➢ Designing Custom Visuals using a View Model
- Advanced Custom Visual Design Features
- Packaging and Deploying Custom Visuals

Designing with View Model

- Best practice involves creating view model for each visual
  - View model defines data required for rendering
  - createViewModel method gets data to generate view model
  - update method calls createViewModel to get view model



```
export interface BarchartDataPoint {
    Category: string;
    Value: number;
}

export interface BarchartViewModel {
    IsNotValid: boolean;
    DataPoints?: BarchartDataPoint[];
    Format?: string;
    SortBySize?: boolean;
    XAxisFontSize?: number;
    YAxisFontSize?: number;
    BarColor?: string;
}
```

DEMO

Examining the View Model in the Barchart Visual

Agenda

- ✓ Defining Data Roles and Data Mappings
- ✓ Extending a Visual with Custom Properties
- ✓ Designing Custom Visuals using a View Model
- ➢ Advanced Custom Visual Design Features
- Packaging and Deploying Custom Visuals

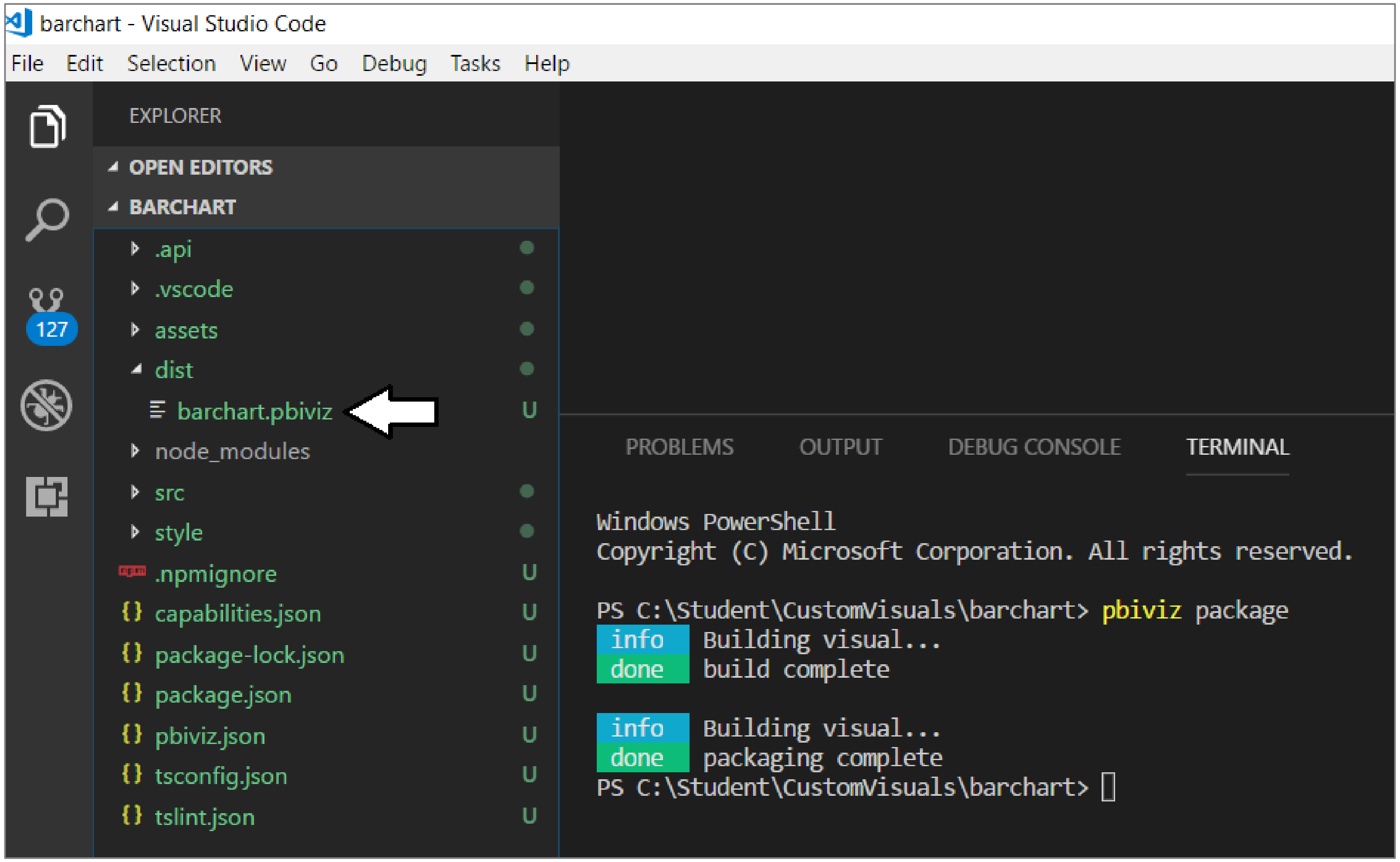
smartieBarChart Demo

- Advanced Visual Features
  - Support for Visual Highlighting
  - Selection Manager
  - TooltipsServiceWrapper

Agenda

- ✓ Defining Data Roles and Data Mappings
- ✓ Extending a Visual with Custom Properties
- ✓ Designing Custom Visuals using a View Model
- ✓ Advanced Custom Visual Design Features
- ➢ Packaging and Deploying Custom Visuals

Packaging and Deploying Custom Visuals



```
Windows PowerShell
Copyright (C) Microsoft Corporation. All rights reserved.

PS C:\Student\CustomVisuals\barchart> pbiviz package
 info  Building visual...
 done  build complete

 info  Building visual...
 done  packaging complete
PS C:\Student\CustomVisuals\barchart>
```

Summary

- ✓ Defining Data Roles and Data Mappings
- ✓ Extending a Visual with Custom Properties
- ✓ Designing Custom Visuals using a View Model
- ✓ Advanced Custom Visual Design Features
- ✓ Packaging and Deploying Custom Visuals