

Microsoft Power BI

Power BI Dev Camp – Session 6

Developing Custom Visuals for Power BI


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>



Home | Camp Sessions ▾ | Camper Resources | COVID-19 | About

Home > Camp Sessions > Session 06: Developing Custom Visuals for Power BI

🔥 Session 06: Developing Custom Visuals for Power BI

In this camp session, campers will learn how to get started designing and implementing custom visuals for Power BI. The session examines the Power BI Visuals API that Microsoft created to assist in the development of custom visuals. Campers will learn how to define the capabilities and data mappings for a custom visual and how to program D3-style data binding using categorical data from a Power BI dataset. The session also demonstrates how to extend a visual with custom properties as well as how to take advantage of the powerful utility classes that are included along with the Power BI Visuals API. Along the way, campers will learn how to package a custom visual as a PBIVIZ file for distribution and how to add custom visuals to Power BI Desktop projects as well as how to publish custom visuals at organization scope.

📋 What Campers Will Learn:

- Getting Started with the Power BI Visual Dev Tools (PBIVIZ)
- Creating and Testing Custom Visual Projects
- Defining Data Roles and Data Mappings
- Extending a Visual with Custom Properties
- Programming DataViewObjectParser and VisualSettings
- Working with the Selection Manager

📖 Session Prerequisites

Campers attending this session should have prior experience developing with Node.js, TypeScript and Visual Studio Code.

📄 Session Info

Date	January 28, 2021
Time	2:00 PM Eastern - 11:00 AM Pacific
Attendee Link	https://aka.ms/PBIWebinar01282021

🔗 Session Links and Resources

🔗 GitHub Report with Tutorial and Samples

This is the GitHub repository that holds the tutorial and custom visual sample projects used in Ted's custom visual session.

🔗 Getting Started with Custom Visuals Tutorial

This tutorial provides hands-on lab exercises to get campers up to speed on creating and testing their first custom visual projects.

🔗 Microsoft Documentation on Custom Visuals

Microsoft's recently released documentation for developing custom visuals - an essential resource for developers.

🔗 Microsoft GitHub Repo for Custom Visuals

The original GitHub repository and wiki where Microsoft has maintained developer info for custom visual development. Much of what is here is now officially replaced with the Microsoft documentation link above this link.

Microsoft Custom Visual Documentation

- <https://docs.microsoft.com/en-us/power-bi/developer/visuals/>

The screenshot shows the Microsoft Custom Visual Documentation page for Power BI visuals. The page has a top navigation bar with links for Power BI, Get started, Work with data, Create & share, Admin & enterprise, Developer, and Resources. There are buttons for 'Buy now' and 'Open Power BI'. Below the navigation bar is a breadcrumb trail: Docs / Power Platform / Power BI / Developer / Power BI visuals. On the right side of the breadcrumb trail are links for Bookmark, Edit, and Share. The main content area is titled 'Power BI visuals documentation' and includes a sub-header: 'Develop your own Power BI visuals, to be used by you, your organization, or the entire Power BI community. Our documentation provides the information you need.' The left sidebar contains a search filter 'Filter by title' and a list of navigation items: Overview, Tutorials (with sub-items: Tutorial: Developing a Power BI circle card visual, Tutorial: Create a Power BI visual using React, Build a bar chart, Add formatting options, Add unit tests for visual project, Build a funnel plot from R script to R visual), Concepts, How to, Reference, and Resources. At the bottom of the sidebar is a 'Download PDF' link. The main content area features three columns of links: 'Get started' (with sub-links: GET STARTED, Visuals in Power BI, Power BI visual project structure, Frequently asked questions), 'Develop a Power BI visual' (with sub-links: TUTORIAL, Tutorial: Develop a Power BI visual, Tutorial: Adding formatting options to a Power BI visual, CONCEPT, Guidelines for Power BI visuals), and 'Publish Power BI visuals' (with sub-links: CONCEPT, Publish Power BI visuals to Partner Center, Get a Power BI visual certified). At the bottom of the main content area is a 'References and resources' section with a sub-link: REFERENCE, Samples of Power BI visuals.

Power BI Get started ▾ Work with data ▾ Create & share ▾ Admin & enterprise ▾ Developer ▾ Resources ▾ [Buy now](#) [Open Power BI](#)

[Docs](#) / [Power Platform](#) / [Power BI](#) / [Developer](#) / Power BI visuals [Bookmark](#) [Edit](#) [Share](#)

[Filter by title](#)

Power BI visuals documentation

> Overview

▾ Tutorials

- Tutorial: Developing a Power BI circle card visual
- Tutorial: Create a Power BI visual using React
- Build a bar chart
- Add formatting options
- Add unit tests for visual project
- Build a funnel plot from R script to R visual

> Concepts

> How to

> Reference

> Resources

[Download PDF](#)

Power BI visuals documentation

Develop your own Power BI visuals, to be used by you, your organization, or the entire Power BI community. Our documentation provides the information you need.

Get started

- [GET STARTED](#)
- [Visuals in Power BI](#)
- [Power BI visual project structure](#)
- [Frequently asked questions](#)

Develop a Power BI visual

- [TUTORIAL](#)
- [Tutorial: Develop a Power BI visual](#)
- [Tutorial: Adding formatting options to a Power BI visual](#)
- [CONCEPT](#)
- [Guidelines for Power BI visuals](#)

Publish Power BI visuals

- [CONCEPT](#)
- [Publish Power BI visuals to Partner Center](#)
- [Get a Power BI visual certified](#)

References and resources

- [REFERENCE](#)
- [Samples of Power BI visuals](#)

Agenda

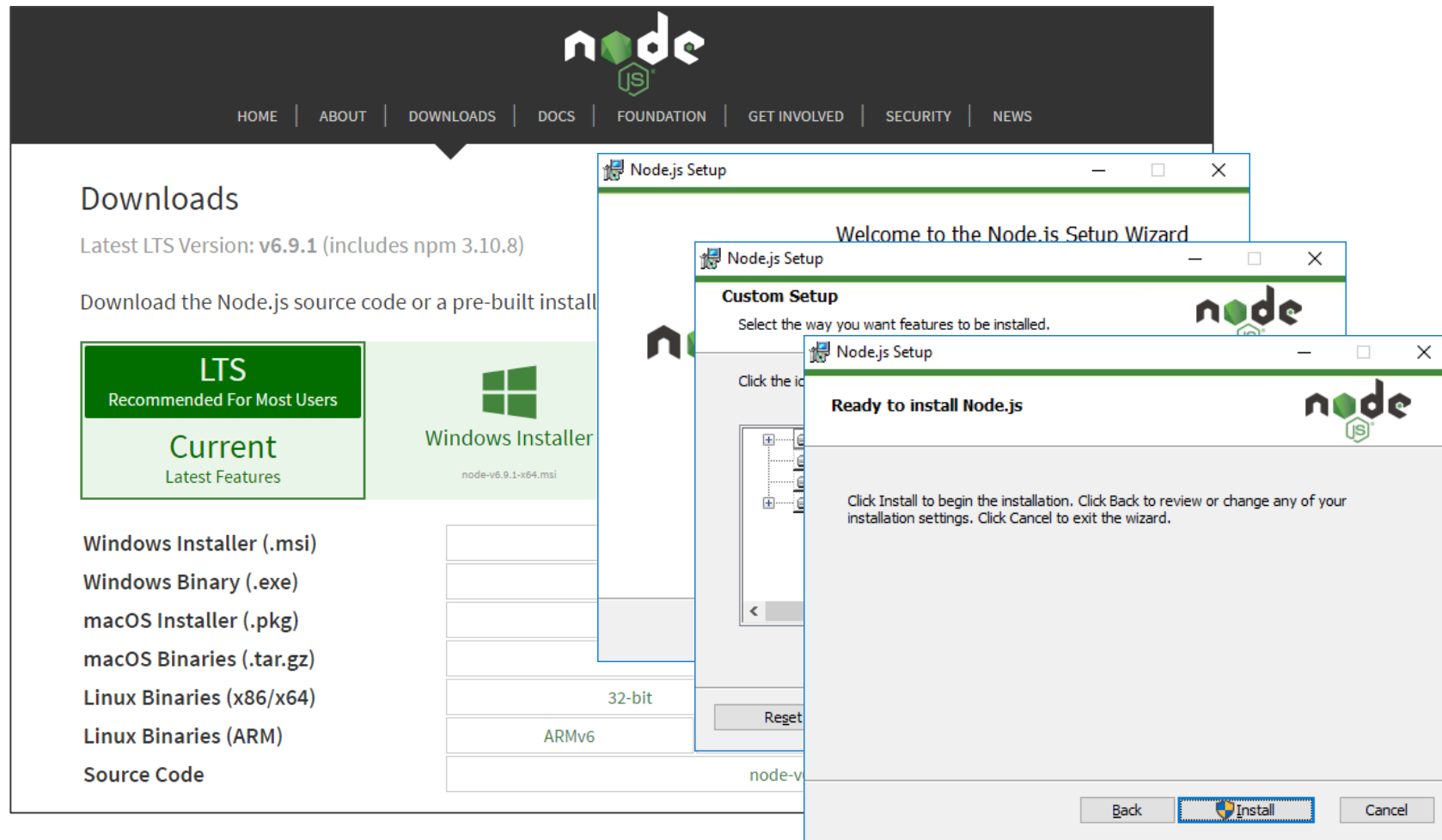
- Installing the Power BI Developer Tools
- Creating Your First Custom Visual
- Defining Data Roles and Data Mappings
- Extending a Visual with Custom Properties
- Implementing Highlighting with SelectionManager
- Custom Visual Packaging and Distribution

Installing the Power BI Developer Toolchain

- **Install Node.JS**
 - Installs Node Package Manager (npm)
- **Install Visual Studio Code**
 - Lightweight Alternative to Visual Studio for Node.js Development
- **Install the Power BI Developer Tools (pbiviz)**
 - Install using Node Package Manager (npm)
- **Create and install a local self-signed certificate**
 - Install using Power BI visuals CLI tool (pbiviz)

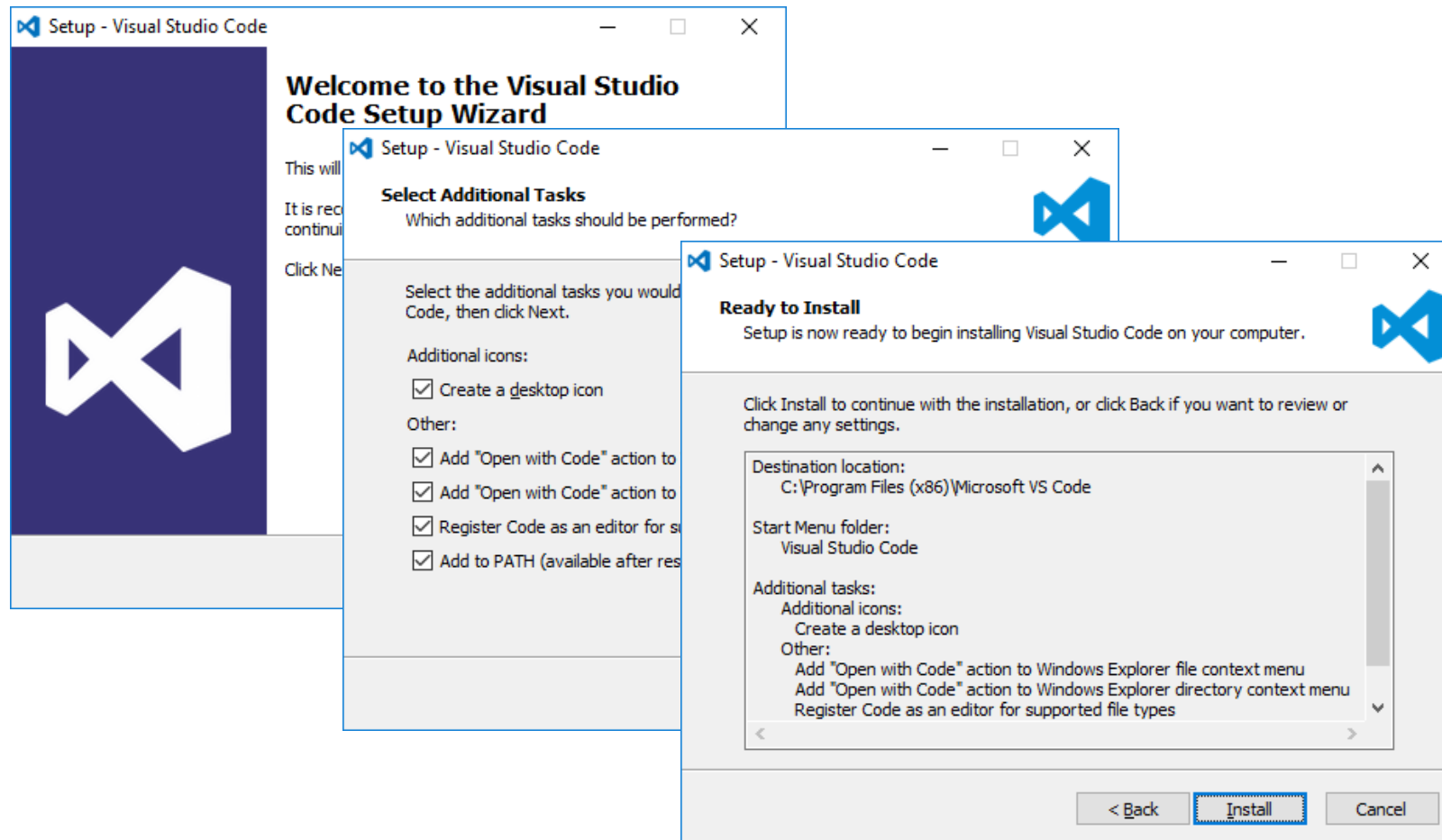
Installing node.js

- <https://nodejs.org/en/download/>



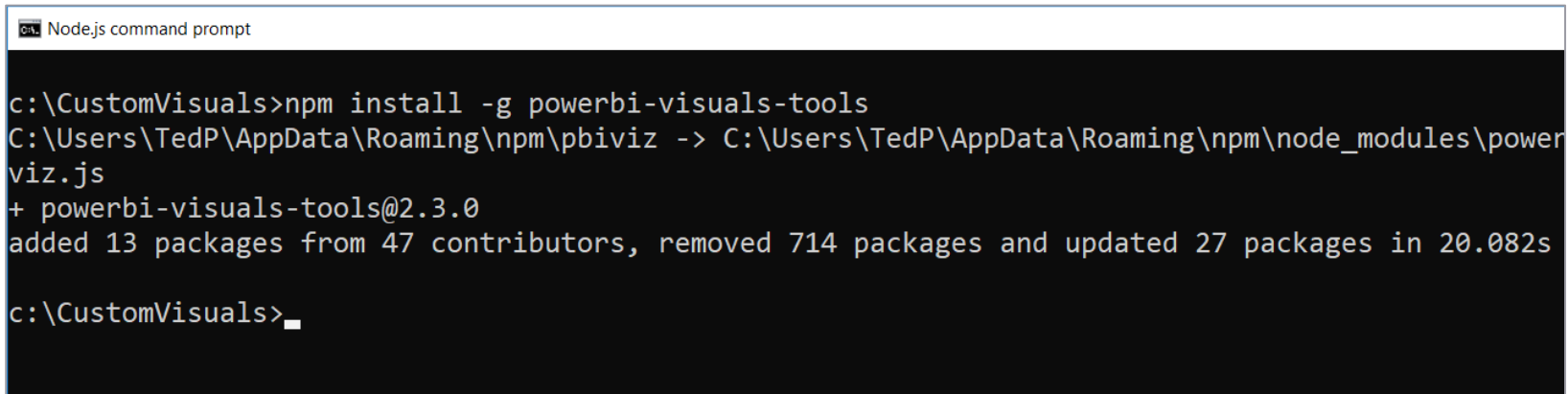
Install Visual Studio Code

- <http://code.visualstudio.com/>



Power BI Visual CLI Tool (PBIVIZ)

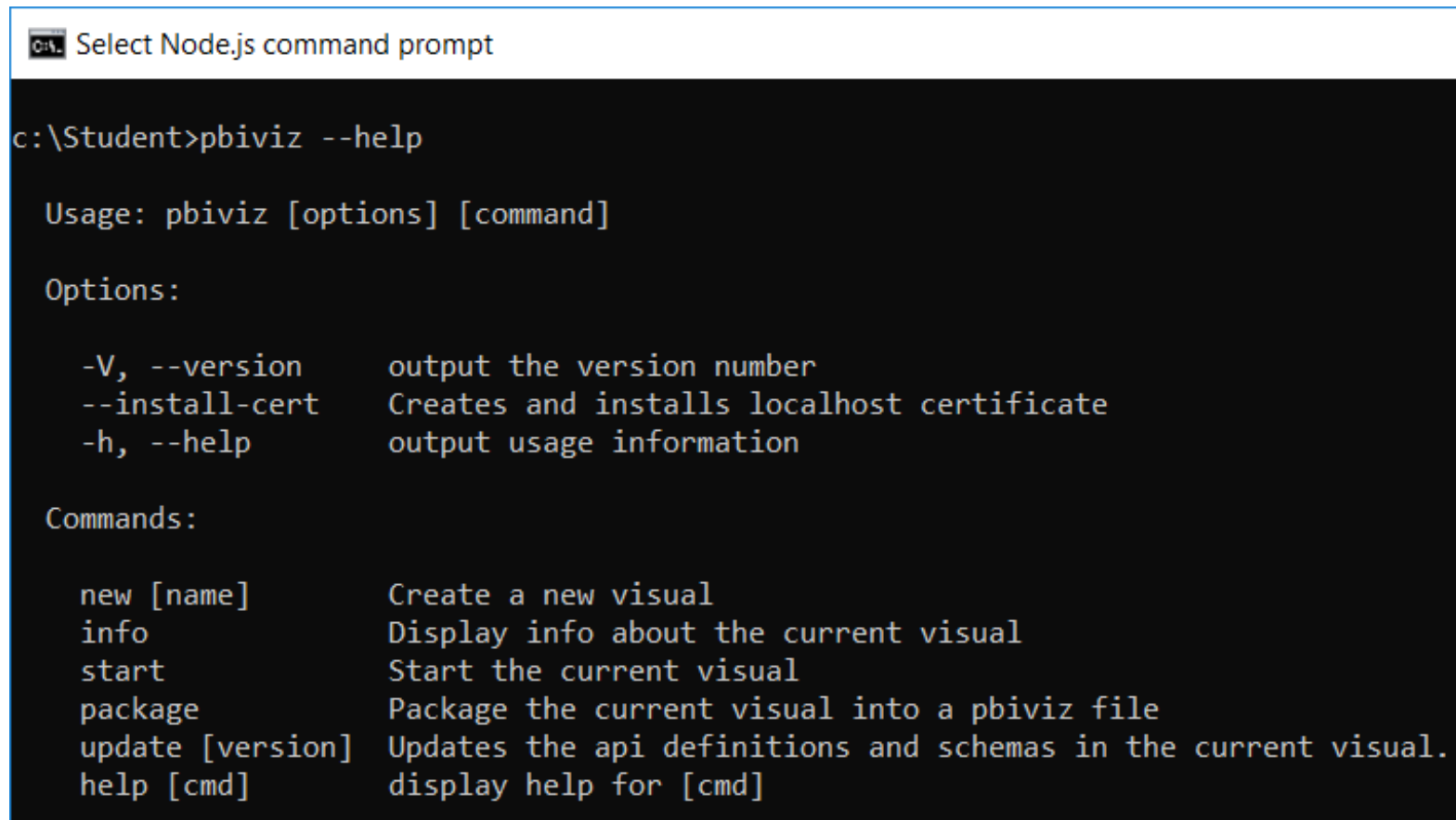
- What is the Power BI Custom Visual Tool?
 - Command-line utility for cross-platform dev
 - Use it with Visual Studio or Visual Studio Code
 - Requires that you first install node.js
 - Install by running command from node.js command prompt
npm install -g powerbi-visuals-tools



```
Node.js command prompt
c:\CustomVisuals>npm install -g powerbi-visuals-tools
C:\Users\TedP\AppData\Roaming\npm\pbiviz -> C:\Users\TedP\AppData\Roaming\npm\node_modules\powerbi-visuals-tools
+ powerbi-visuals-tools@2.3.0
added 13 packages from 47 contributors, removed 714 packages and updated 27 packages in 20.082s
c:\CustomVisuals>
```

Getting Started with PBIVIZ

- PBIVIZ.EXE is a command-line utility
 - You execute PBIVIZ commands from the NODE.JS command line



```

c:\Student>pbiviz --help

Usage: pbiviz [options] [command]

Options:

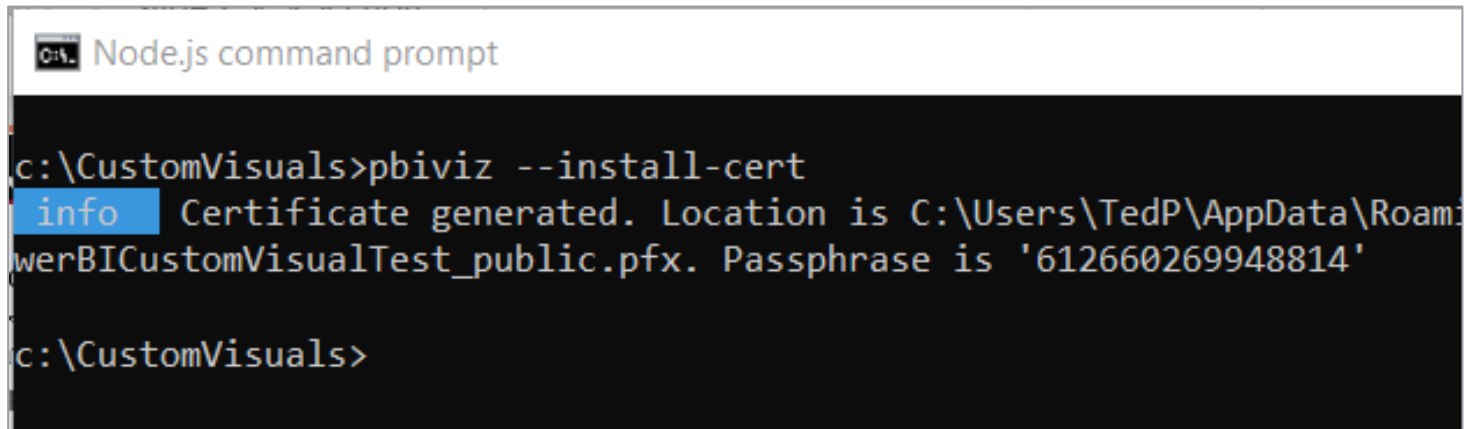
  -V, --version      output the version number
  --install-cert     Creates and installs localhost certificate
  -h, --help         output usage information

Commands:

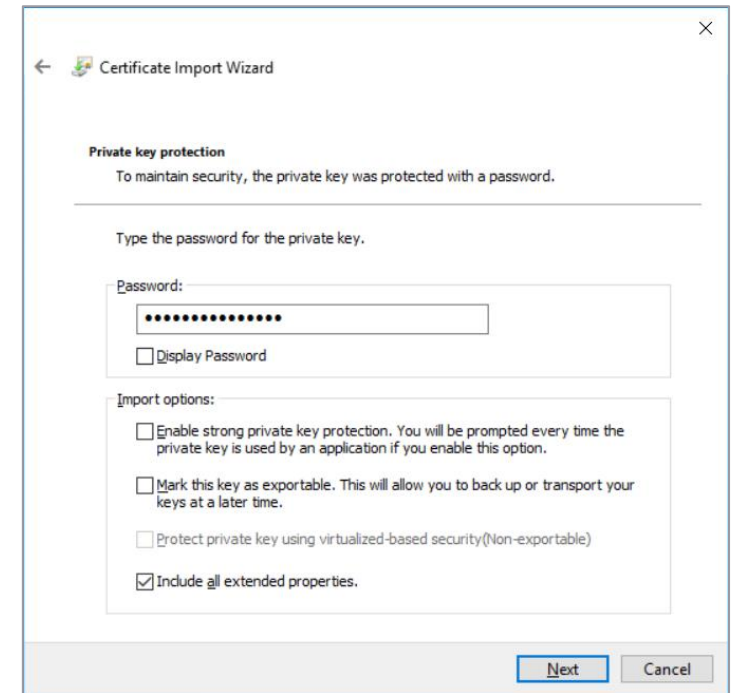
  new [name]         Create a new visual
  info               Display info about the current visual
  start              Start the current visual
  package            Package the current visual into a pbiviz file
  update [version]   Updates the api definitions and schemas in the current visual.
  help [cmd]         display help for [cmd]
```

Creating a Certificate for Local Testing

- PBIVIZ provide local web server for testing & debugging
 - Web server runs locally on developer's workstation in Node.js
 - Makes it possible to test custom visuals in Power BI Service
 - Custom visual resources served up from <https://localhost>
 - Setup requires creating self-signed SSL certificate
 - SSL certificate created using pbiviz --install-cert command
 - You must copy a passphrase to properly install the certificate

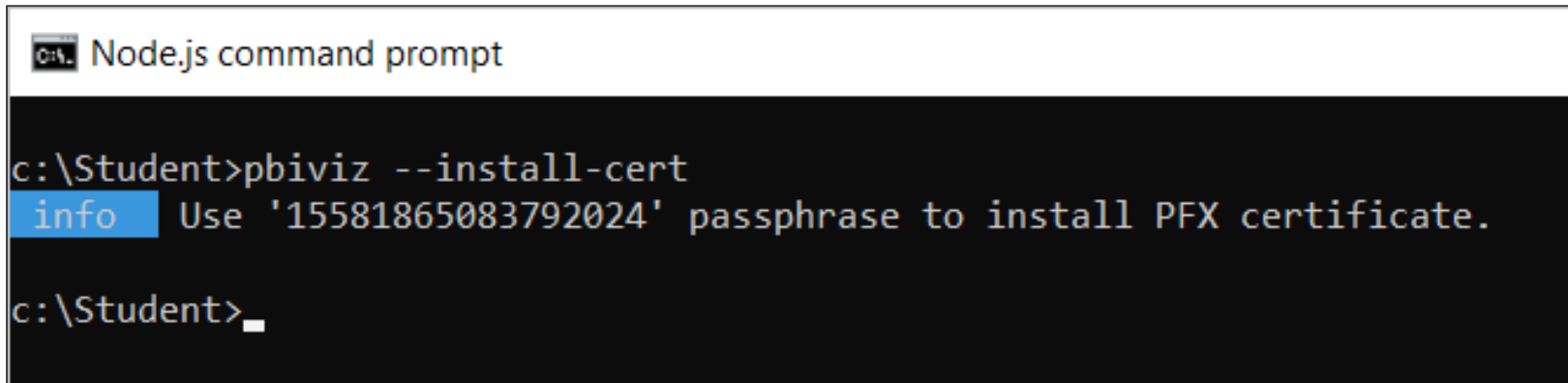


```
C:\CustomVisuals>pbiviz --install-cert  
info Certificate generated. Location is C:\Users\TedP\AppData\Roaming\PowerBICustomVisualTest_public.pfx. Passphrase is '612660269948814'  
C:\CustomVisuals>
```



Installing the SSL Certificate

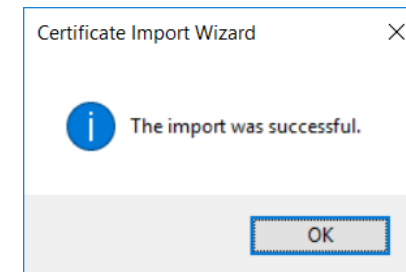
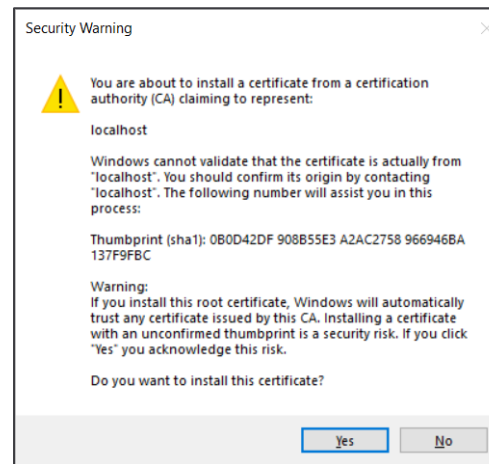
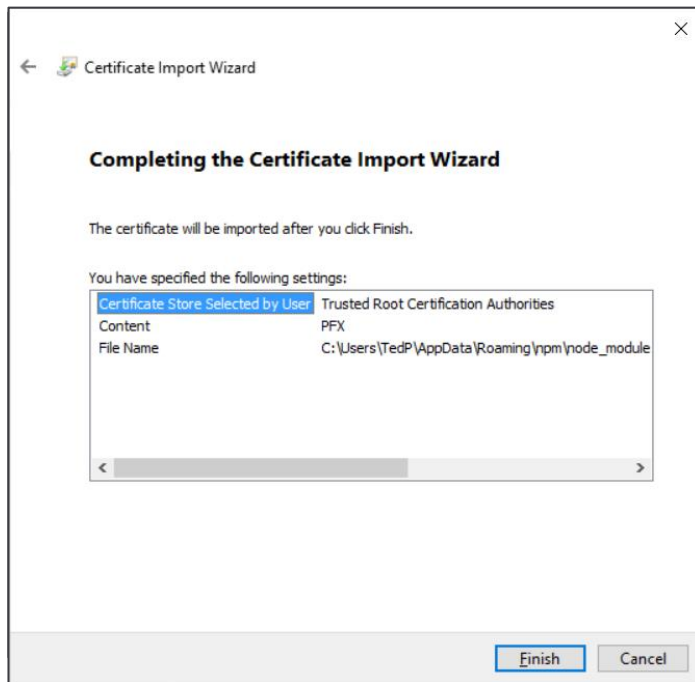
- Installing certificate enables SSL through <https://localhost>
 - Installing certificate is a one time operation – not once per project
 - SSL certificate installed using pbiviz --install-cert command
 - Running --install-cert command starts Certificate Import Wizard



```
Node.js command prompt
c:\Student>pbiviz --install-cert
info Use '15581865083792024' passphrase to install PFX certificate.
c:\Student>_
```

The Certificate Import Wizard

- Wizards steps you through process of installing certificate
 - You enter certificate passphrase as part of installation process

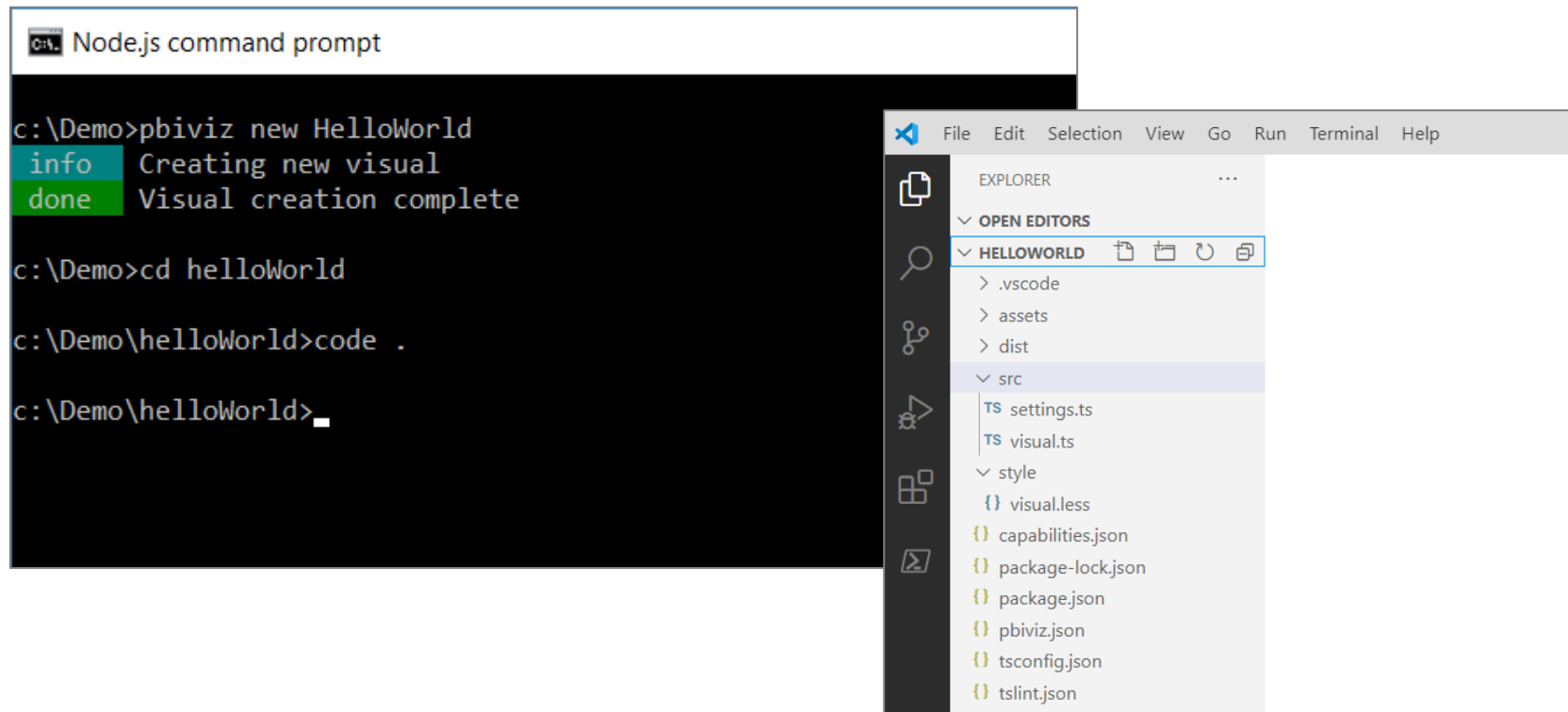


Agenda

- ✓ Installing the Power BI Developer Tools
- Creating Your First Custom Visual
 - Defining Data Roles and Data Mappings
 - Extending a Visual with Custom Properties
 - Implementing Highlighting with SelectionManager
 - Custom Visual Packaging and Distribution

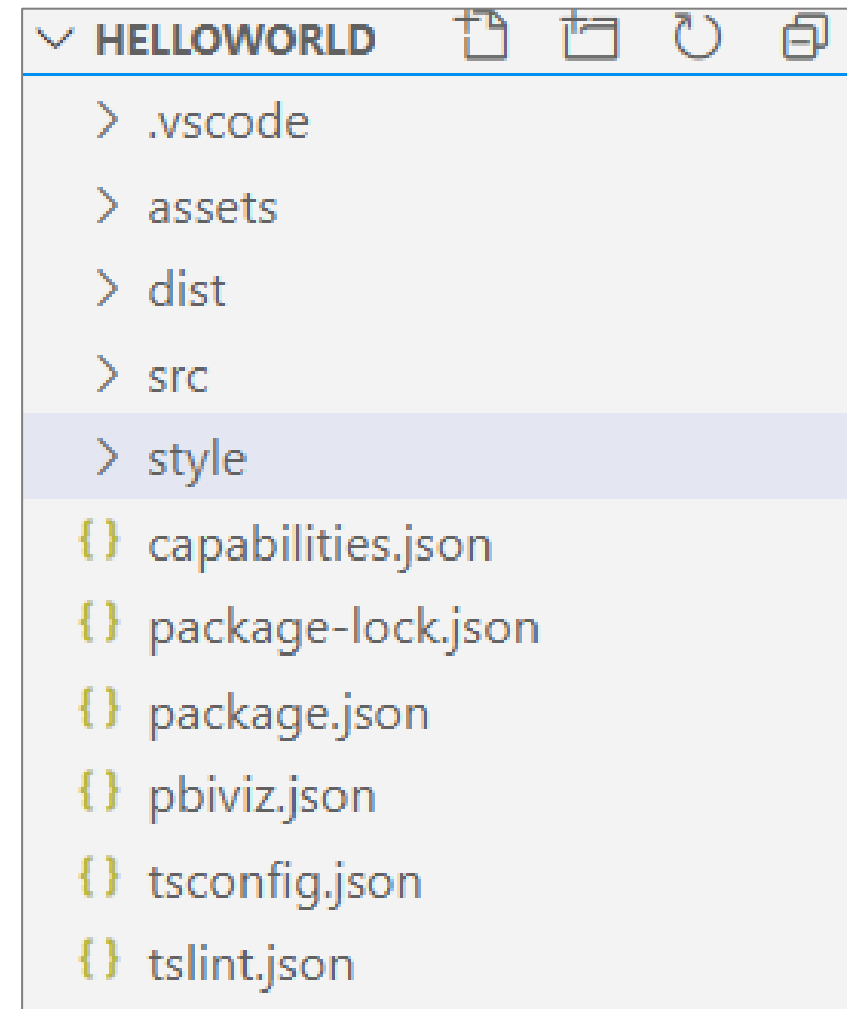
Creating a New Custom Visual Project

- Creating a new project
`pbiviz new <ProjectName>`
- Open the Project with Visual Studio Code
`code .`



Top-level project files

- **package.json**
 - Used by npm to manage packages
- **pbiviz.json**
 - Main manifest file for your custom visual project
- **capabilities.json**
 - File used to define visual capabilities
- **tsconfig.json & tslint.json**
 - Typescript compiler settings



Package.json

✓ VIZ01

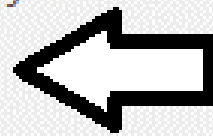
- > .vscode
- > assets
- > node_modules
- > src
- > style

{ } capabilities.json

{ } package-lock.json

{ } package.json

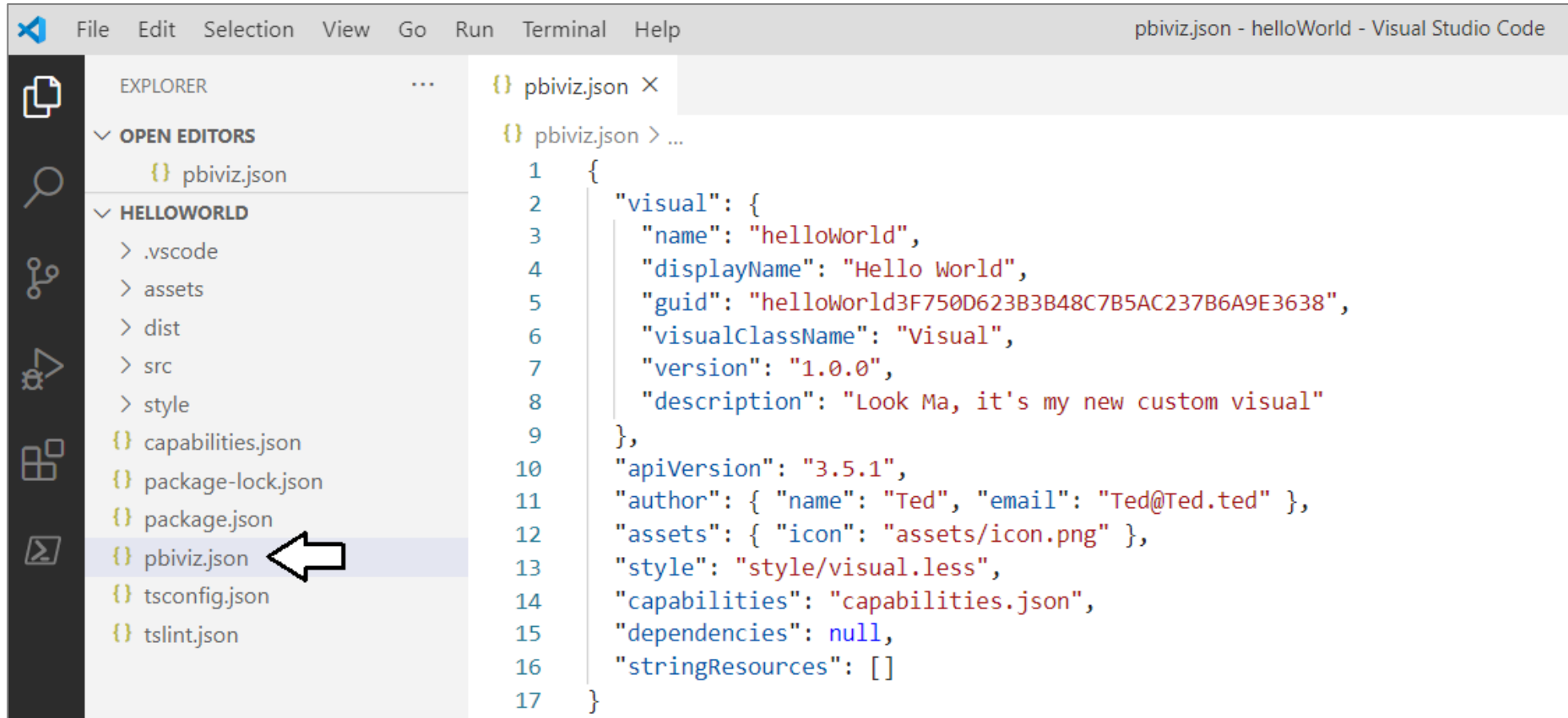
{ } pbiviz.json



```
{ } package.json > ...
1  {
2    "name": "visual",
   ▶ Debug
3    "scripts": {
4      "pbiviz": "pbiviz",
5      "start": "pbiviz start",
6      "package": "pbiviz package",
7      "lint": "tslint -c tslint.json -p tsconfig.json"
8    },
9    "dependencies": {
10     "@babel/runtime": "7.6.0",
11     "@babel/runtime-corejs2": "7.6.0",
12     "@types/d3": "5.7.2",
13     "d3": "5.12.0",
14     "powerbi-visuals-utils-dataviewutils": "2.2.1",
15     "powerbi-visuals-api": "~2.6.1",
16     "core-js": "3.2.1"
17   },
18   "devDependencies": {
19     "ts-loader": "6.1.0",
20     "tslint": "^5.18.0",
21     "tslint-microsoft-contrib": "^6.2.0",
22     "typescript": "3.6.3"
23   }
24 }
```

The pbiviz.json File

- Acts as top-level manifest file for custom visual project
 - Contains information used in final visual PBIVIZ packaging process

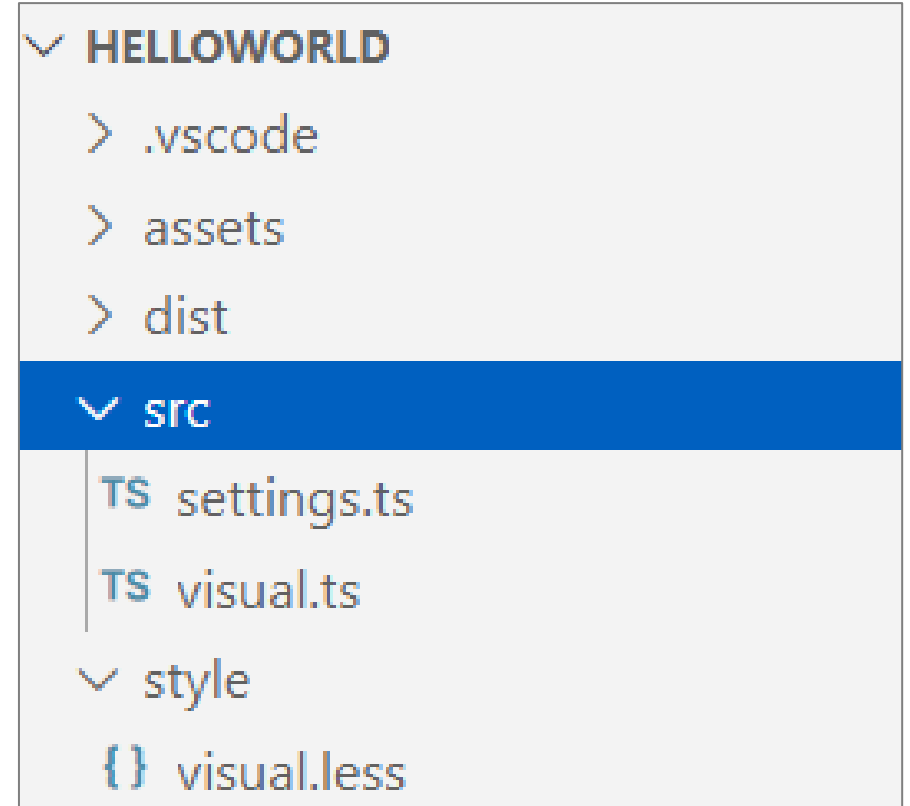


The screenshot shows the Visual Studio Code interface. On the left, the Explorer sidebar displays the file structure of a project named 'HELLOWORLD'. The files listed are: .vscode, assets, dist, src, style, capabilities.json, package-lock.json, package.json, pbiviz.json (highlighted with a white arrow), tsconfig.json, and tslint.json. The main editor area shows the content of pbiviz.json, which is a JSON manifest file. The file content is as follows:

```
1 {
2   "visual": {
3     "name": "helloWorld",
4     "displayName": "Hello World",
5     "guid": "helloWorld3F750D623B3B48C7B5AC237B6A9E3638",
6     "visualClassName": "Visual",
7     "version": "1.0.0",
8     "description": "Look Ma, it's my new custom visual"
9   },
10  "apiVersion": "3.5.1",
11  "author": { "name": "Ted", "email": "Ted@Ted.ted" },
12  "assets": { "icon": "assets/icon.png" },
13  "style": "style/visual.less",
14  "capabilities": "capabilities.json",
15  "dependencies": null,
16  "stringResources": []
17 }
```

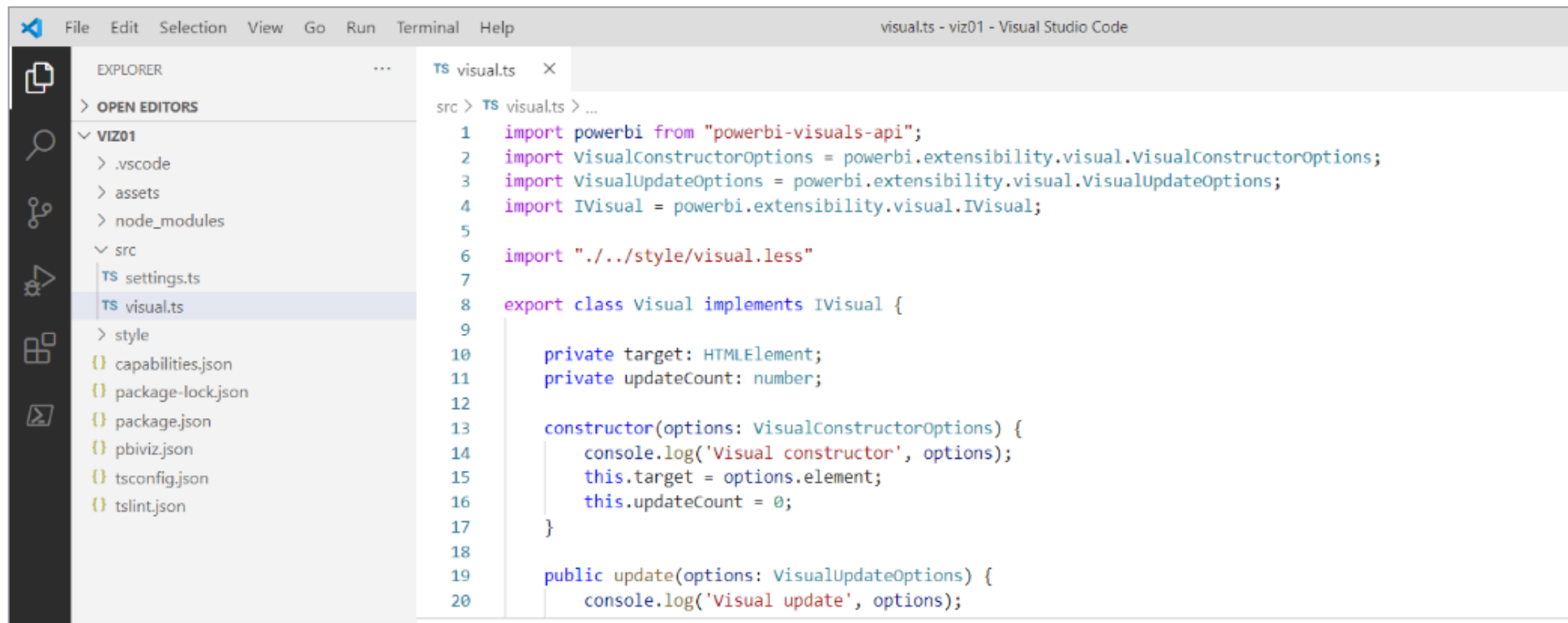
Visual Source Files

- **src/visual.ts**
 - visual class definition
- **src/settings.ts**
 - helper class to manage visual properties
- **style/visual.less**
 - CSS used to style custom visual



Authoring a Custom Visual Class

- Custom visual is a class that implements `IVisual`
 - Minimum visual class must implement `IVisual` interface and provide **update** method
 - Parameterized constructor used to create visual elements
 - **update** method performs visual rendering

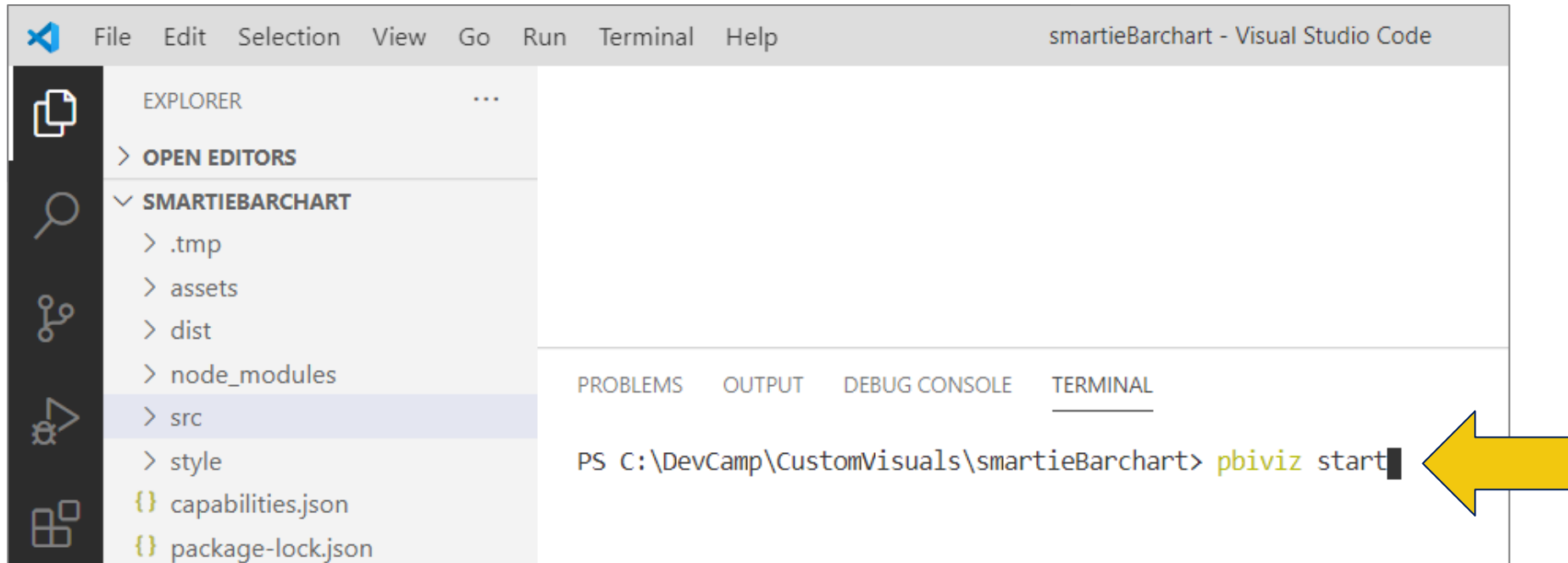


The screenshot shows the Visual Studio Code editor with a file explorer on the left and a code editor on the right. The file explorer shows a project structure with a folder named 'VIZ01' containing subfolders like '.vscode', 'assets', 'node_modules', 'src', and 'style'. The 'src' folder is expanded, showing files like 'settings.ts', 'visual.ts', and 'style'. The 'visual.ts' file is selected and its content is displayed in the code editor. The code defines a class 'Visual' that implements the 'IVisual' interface. It includes imports for 'powerbi' and 'powerbi-visuals-api', and defines a constructor and an 'update' method.

```
src > TS visual.ts > ...  
1  import powerbi from "powerbi-visuals-api";  
2  import VisualConstructorOptions = powerbi.extensibility.visual.VisualConstructorOptions;  
3  import VisualUpdateOptions = powerbi.extensibility.visual.VisualUpdateOptions;  
4  import IVisual = powerbi.extensibility.visual.IVisual;  
5  
6  import "../style/visual.less"  
7  
8  export class Visual implements IVisual {  
9  
10     private target: HTMLElement;  
11     private updateCount: number;  
12  
13     constructor(options: VisualConstructorOptions) {  
14         console.log('Visual constructor', options);  
15         this.target = options.element;  
16         this.updateCount = 0;  
17     }  
18  
19     public update(options: VisualUpdateOptions) {  
20         console.log('Visual update', options);
```

Visual Studio Code Terminal

- Use the Terminal to execute commands with **npm** and **pbiviz**



Running a Custom Visual Project

- Visual projects run & tested using pbiviz start command
 - Run pbiviz start from Visual Studio Code from Integrated console
 - Command starts local debugging session in node.js

```
PS C:\Student\CustomVisuals\viz01> pbiviz start
```

```
info Starting server...
```

```
info Start preparing plugin template
```

```
i | wds|: Generating SSL Certificate
```

```
i | wds|: Project is running at https://localhost:8080/webpack-dev-server/
```

```
i | wds|: webpack output is served from /assets
```

```
i | wds|: Content not from webpack is served from C:\Student\CustomVisuals\viz01\.tmp\drop
```

```
info Finish preparing plugin template
```

```
info Start packaging...
```

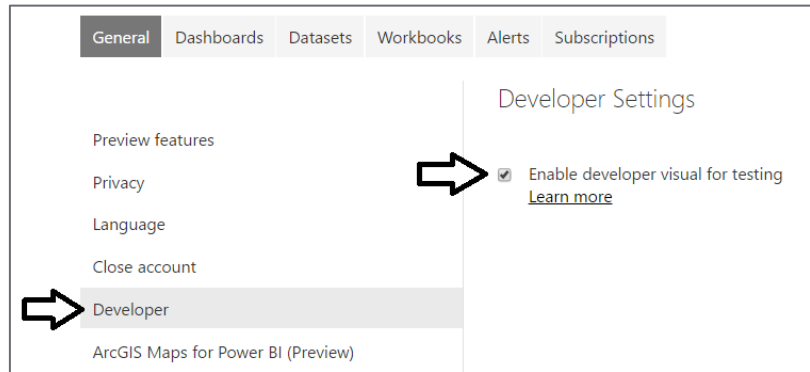
```
info Finish packaging
```

```
Webpack Bundle Analyzer saved report to C:\Student\CustomVisuals\viz01\webpack.statistics.dev.html
```

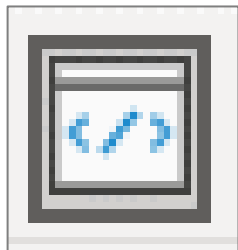
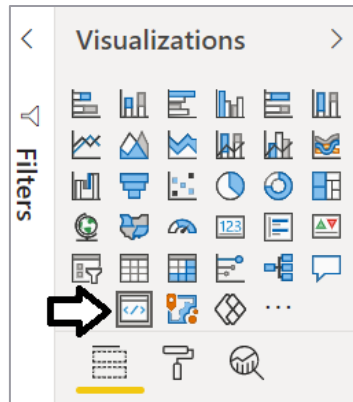
```
DONE Compiled successfully in 1966ms
```

The Developer Visual

- Must be enabled on Developer Settings page



- Provides new Developer visual for testing and debugging custom visuals



Working with the Developer Visual

- Developer visual loads custom visual from node.js
 - Makes it possible to test custom visual inside Power BI Service
 - Developer visual provides toolbar with development utilities

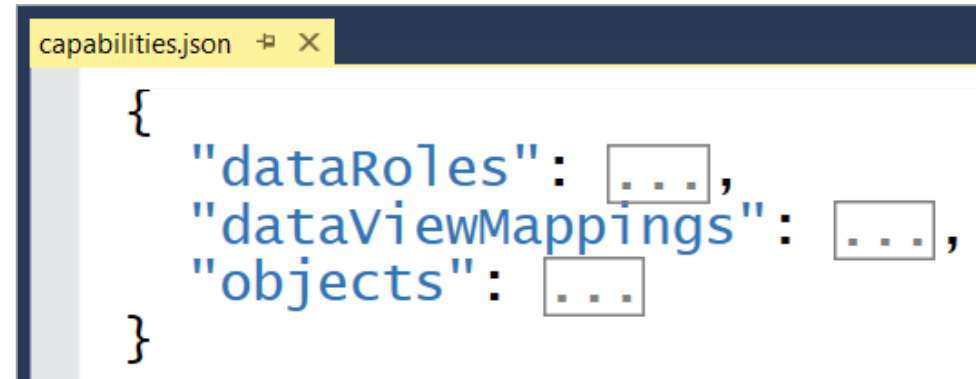
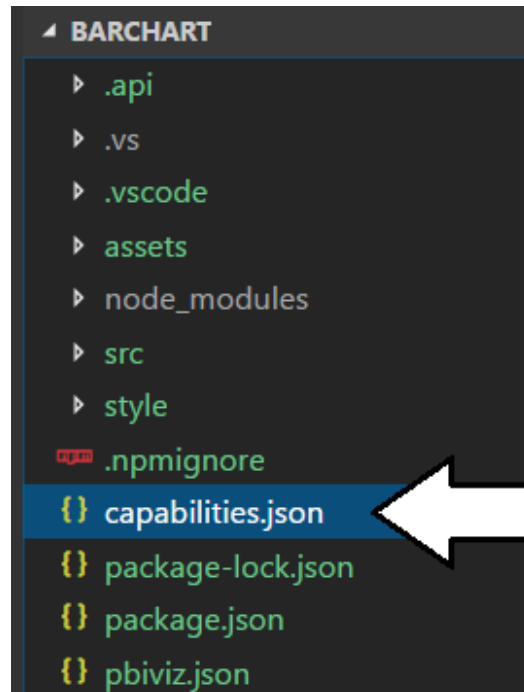


Agenda

- ✓ Installing the Power BI Developer Tools
- ✓ Creating Your First Custom Visual
- Defining Data Roles and Data Mappings
 - Extending a Visual with Custom Properties
 - Implementing Highlighting with SelectionManager
 - Custom Visual Packaging and Distribution

Visual Capabilities

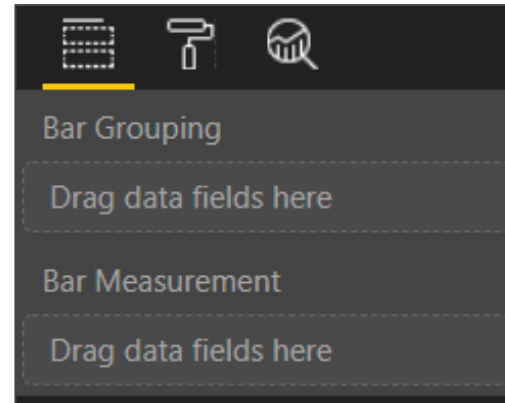
- Visual capabilities defined inside capabilities.json
 - dataRoles defines the field wells displayed on Fields pane
 - dataViewMappings defines the type of DataView used by visual
 - objects defines custom properties for visual



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"  
  }  
]
```



Data Mapping Modes

- Power BI visual API provides several mapping modes

- Single
- Table
- Categorical
- Matrix
- Tree

Single Mapping

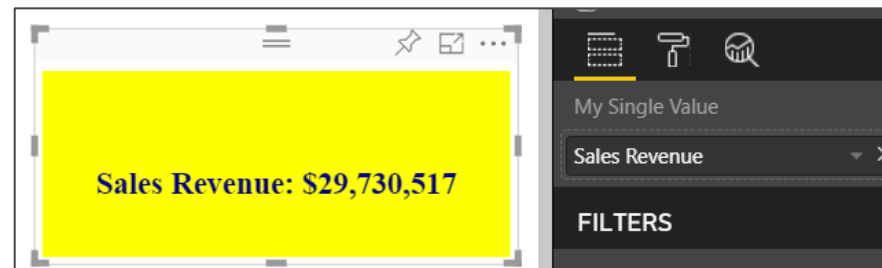
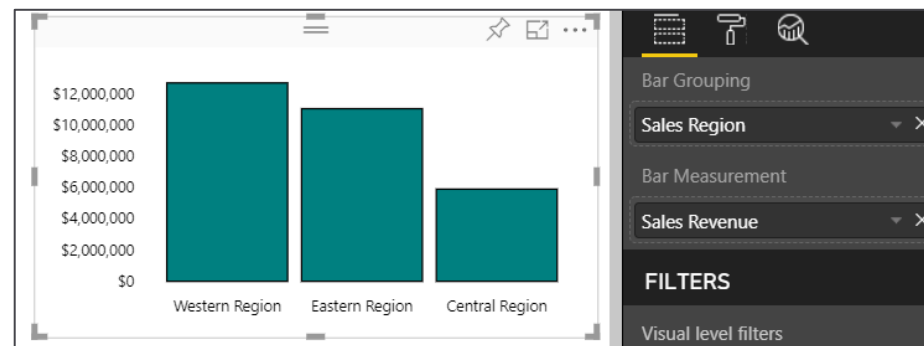


Table Mapping

A Power BI visual in Table Mapping mode. The main area displays a table with three columns: Sales Region, Sales Revenue, and Units Sold. The right-hand pane shows the configuration: "Values" is selected, and "Sales Region", "Sales Revenue", and "Units Sold" are all added to the list. The "FILTERS" section is empty.

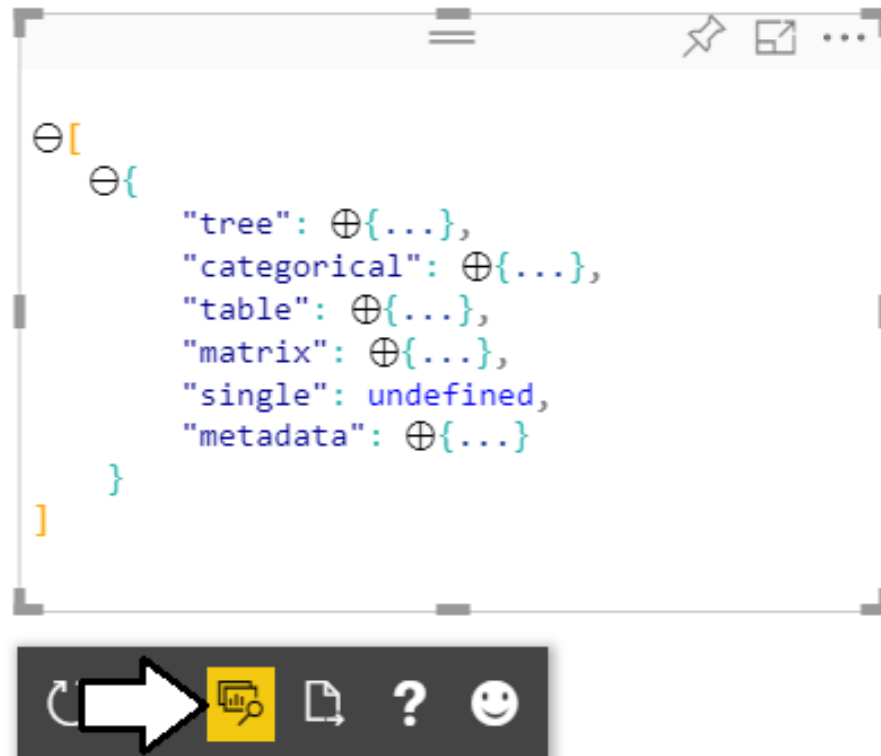
Sales Region	Sales Revenue	Units Sold
Western Region	\$12,733,888	1,598,125
Central Region	\$5,915,449	994,680
Eastern Region	\$11,081,180	1,959,240

Categorical Mapping



Developer Visual DataView

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



Designing with View Model

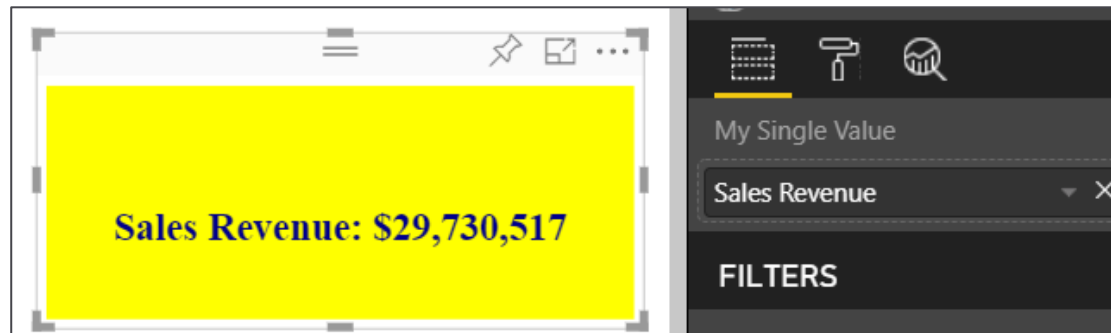
- Best practice involves creating view model for each visual
 - View model defines data required for rendering
 - createViewModel method gets data to generate view model
 - update method calls createViewModel to get view model

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

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" }  
  }  
]
```



Programming in Single Mapping Mode

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

```
"tree": ⊕{...},
"category": ⊕{...},
"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

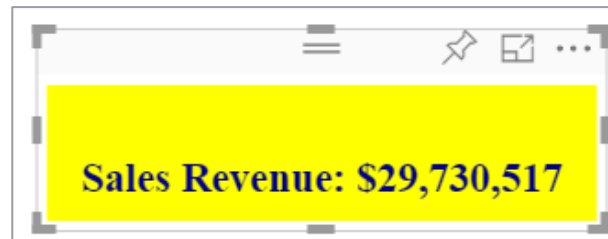
- Used to format values using Power BI formatting strings
 - Requires installing powerbi-visuals-utils-formattingutils package

```
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",
```



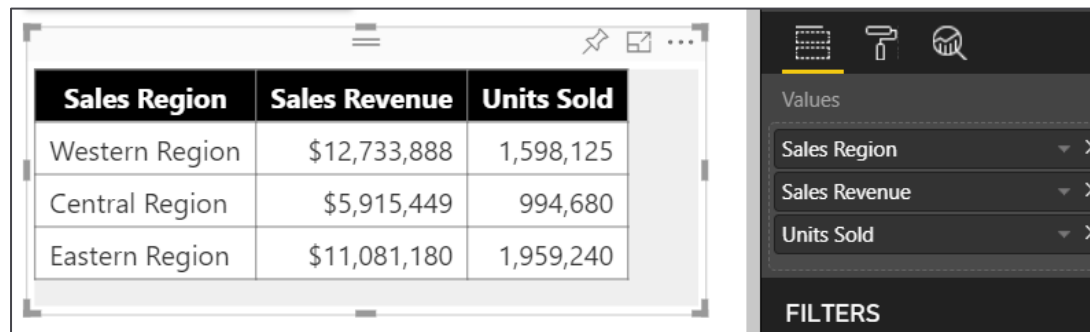
```
"column": ⊖{
  "roles": ⊕{...},
  "type": ⊕{...},
  "format": "#,0",
  "displayName": "Units Sold",
  "queryName": "Sales.Units Sold",
```



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" } } }  
  }  
]
```



The screenshot displays a Power BI report interface. On the left, a table visual is shown with three columns: 'Sales Region', 'Sales Revenue', and 'Units Sold'. The table contains three rows of data. On the right, a filter pane is visible, showing the 'Values' section with three items: 'Sales Region', 'Sales Revenue', and 'Units Sold', each with a dropdown arrow and a close button. Below the 'Values' section, there is a 'FILTERS' section.

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

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

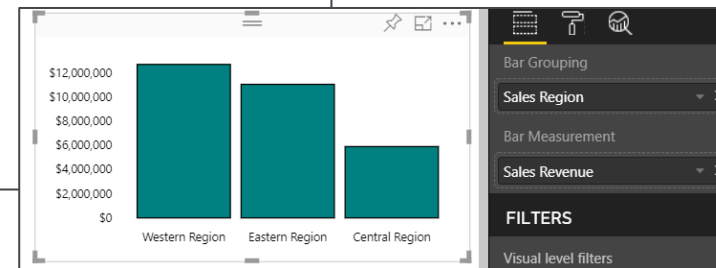
```
"table": ⊖{  
  "columns": ⊖[  
    ⊖{  
      "roles": ⊖{...},  
      "type": ⊖{...},  
      "format": undefined,  
      "displayName": "Sales Region",  
      "queryName": "Customers.Sales Region",  
      "expr": ⊖{...},  
      "index": 0,  
      "identityExprs": ⊖[ ... ]  
    },  
    ⊖{...},  
    ⊖{...}  
  ],  
  "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" } } ]  
      }  
    }  
  }  
]
```



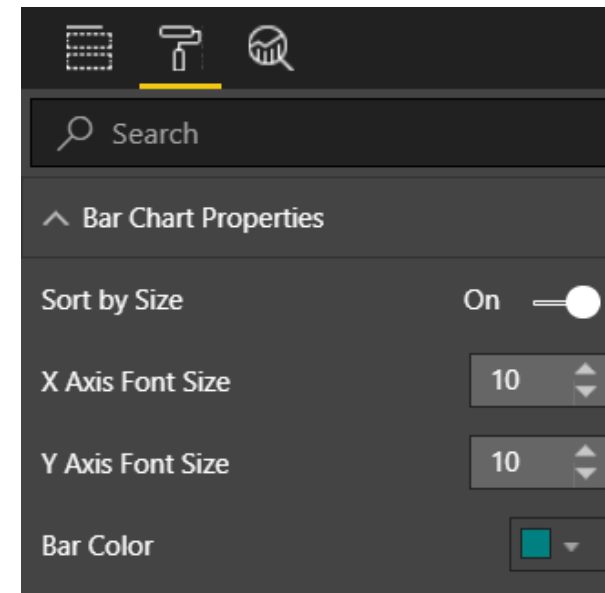
Agenda

- ✓ Installing the Power BI Developer Tools
- ✓ Creating Your First Custom Visual
- ✓ Defining Data Roles and Data Mappings
- Extending a Visual with Custom Properties
 - Implementing Highlighting with SelectionManager
 - Custom Visual Packaging and Distribution

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 } } }
      }
    }
  }
}
```



DataViewObjectParser and VisualSettings

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

```
TS settings.ts •
module powerbi.extensibility.visual {

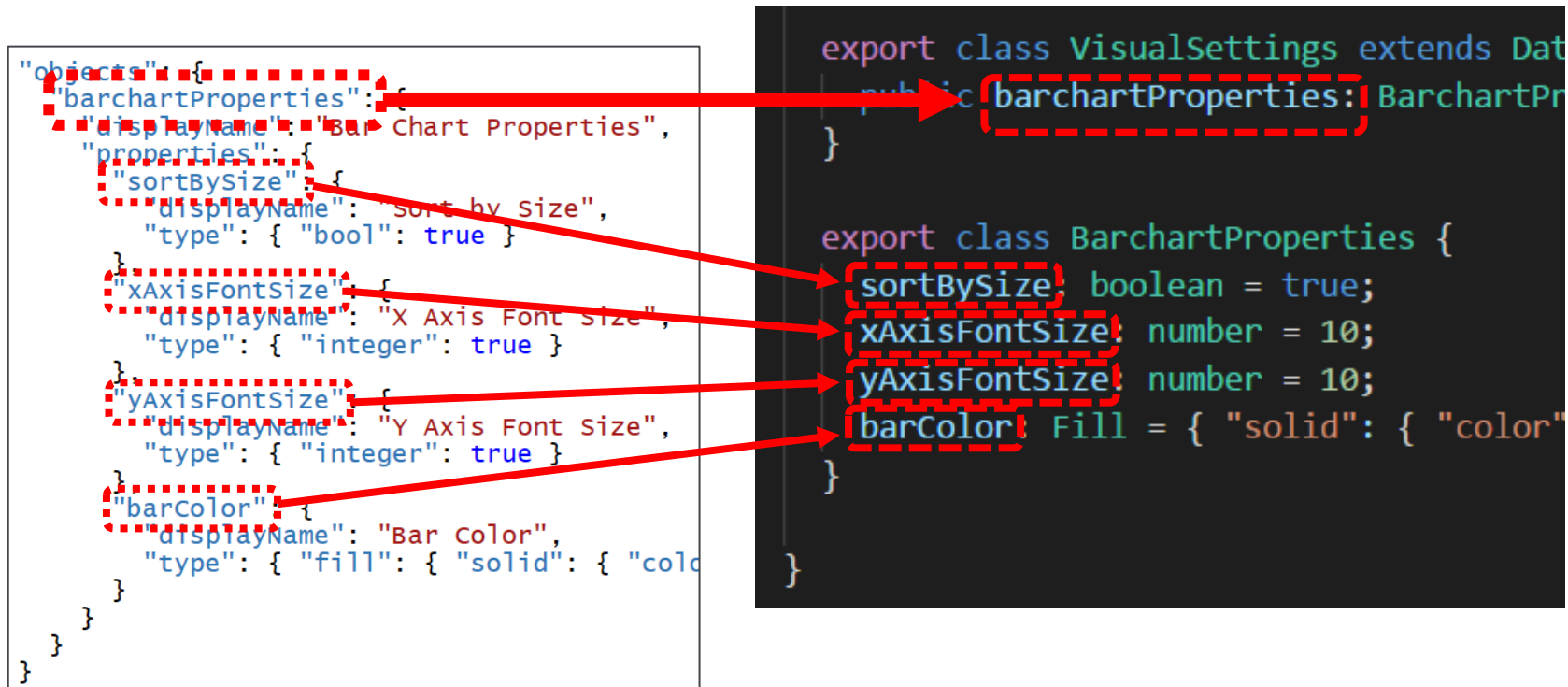
  import DataViewObjectsParser = powerbi.extensibility.utils.dataview.DataViewObjectsParser;

  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

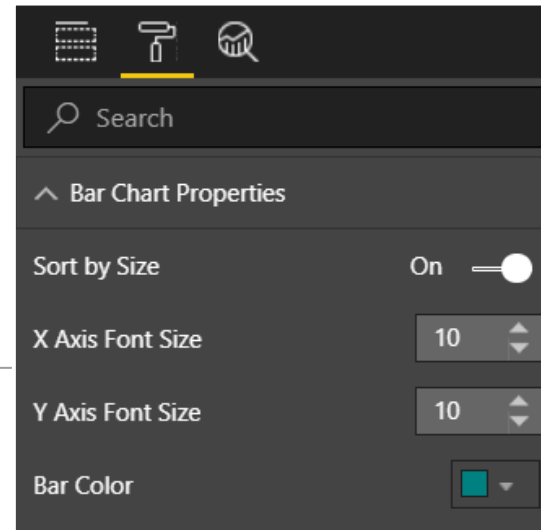
- VisualSettings class must map to named objectnamed
 - VisualSetting class contains named field that maps to object name
 - Named field based on custom class with mapped properties
 - Object & property names must match what's in capabilities.json



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;  
}
```



Retrieving Property Values

- Property values persisted into visual metadata
 - Properties not persisted while 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

- ✓ Installing the Power BI Developer Tools
- ✓ Creating Your First Custom Visual
- ✓ Defining Data Roles and Data Mappings
- ✓ Extending a Visual with Custom Properties
- Implementing Highlighting with SelectionManager
 - Custom Visual Packaging and Distribution

Agenda

- ✓ Installing the Power BI Developer Tools
- ✓ Creating Your First Custom Visual
- ✓ Defining Data Roles and Data Mappings
- ✓ Extending a Visual with Custom Properties
- ✓ Implementing Highlighting with SelectionManager
- Custom Visual Packaging and Distribution

Packaging A Custom Visual for Deployment

- Use the **pbiviz package** command to build PBIVIZ file for distribution
 - Build versioned package for distribution
 - Build version for testing in Power BI Desktop

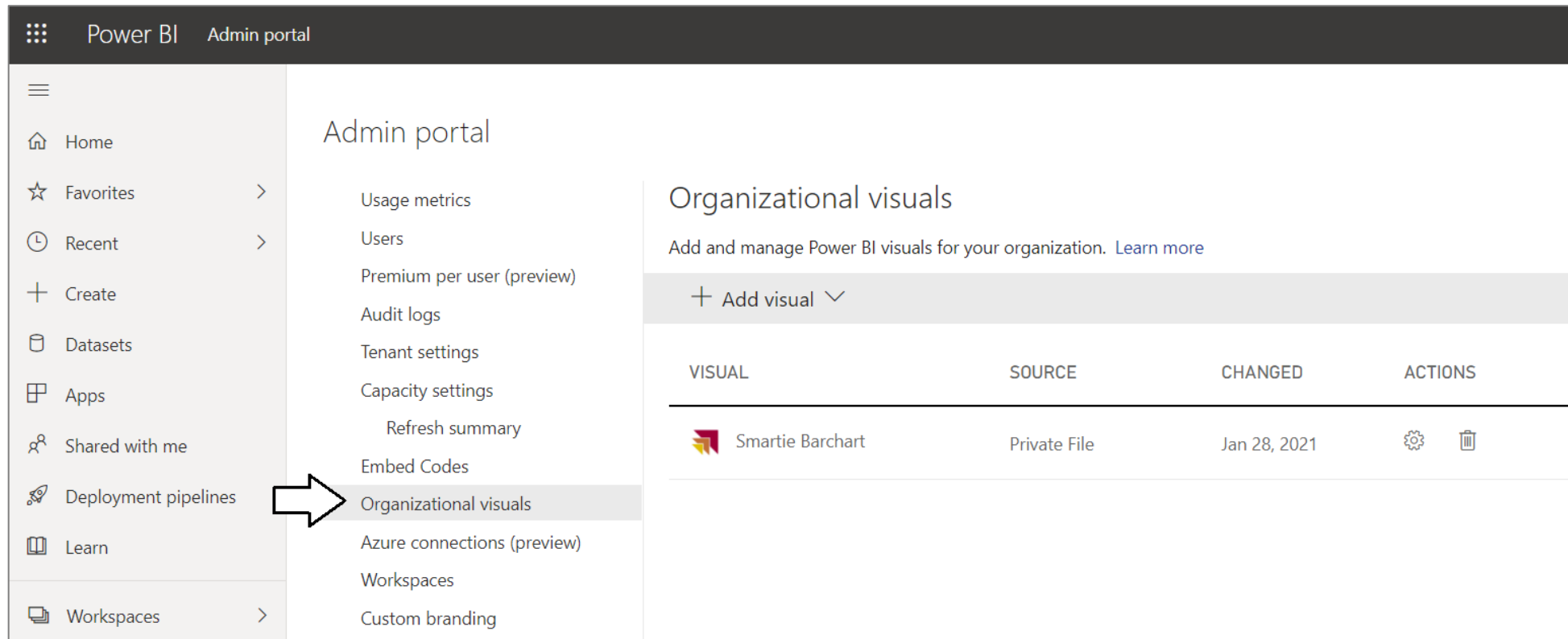
Usage: pbiviz package [options]

Options:




-t, --target [target]	Enable babel loader to compile JS into ES5 standart (default: "es5")
--resources	Produces a folder containing the pbiviz resource files (js, css, json) (default: false)
--no-pbiviz	Doesn't produce a pbiviz file (must be used in conjunction with resources flag)
--no-minify	Doesn't minify the js in the package (useful for debugging)
--no-plugin	Doesn't include a plugin declaration to the package
-c, --compression <compressionLevel>	Enables compression of visual package (default: "6")
-h, --help	output usage information

Organizational Visuals Gallery

- Make custom visuals available on organization-wide basis



The screenshot displays the Power BI Admin portal interface. On the left is a navigation sidebar with options: Home, Favorites, Recent, Create, Datasets, Apps, Shared with me, Deployment pipelines, Learn, and Workspaces. The main content area is titled 'Admin portal' and lists various administrative settings: Usage metrics, Users, Premium per user (preview), Audit logs, Tenant settings, Capacity settings, Refresh summary, Embed Codes, Organizational visuals (highlighted with a white arrow), Azure connections (preview), Workspaces, and Custom branding. The right-hand pane is titled 'Organizational visuals' and includes the text 'Add and manage Power BI visuals for your organization. [Learn more](#)'. Below this is a '+ Add visual' button and a table listing existing organizational visuals.

VISUAL	SOURCE	CHANGED	ACTIONS
 Smartie Barchart	Private File	Jan 28, 2021	 

Summary

- ✓ Installing the Power BI Developer Tools
- ✓ Creating Your First Custom Visual
- ✓ Defining Data Roles and Data Mappings
- ✓ Extending a Visual with Custom Properties
- ✓ Implementing Highlighting with SelectionManager
- ✓ Custom Visual Packaging and Distribution

Get Ready for Next Month...

[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 applications in .NET 5 using the Power BI APIs and the App-Owns-Data embedding model. Campers will then learn how to program Azure AD authentication using Microsoft's most recent Authentication Library named Microsoft.Identity.Web. This session will teach developers how to implement app-owns-data embedding using the Power BI Service API combined together with the Power BI JavaScript API.

The session will also explore advanced development topics such as adding TypeScript support to a Visual Studio Code development project and programming the Power BI Service API to generate multi-resource embed tokens.

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
- 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

Questions