

# Getting Up and Running with the Power BI Developer Tools

**Setup Time:** 60 minutes

**Lab Folder:** C:\Student\Modules\05\_PBIVIZ\Lab

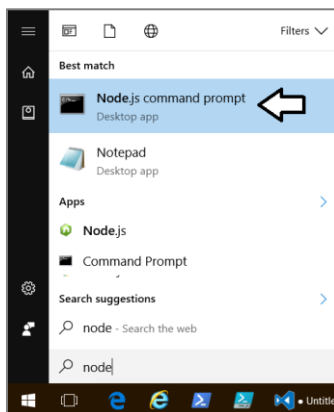
**Overview:** In this lab, you will begin by installing the Power BI Custom Visual Tool (PBIVIZ). After that, you will use PBIVIZ and Visual Studio Code to create and debug custom visual projects within the context of a Power BI workspace. You will also learn how to configure custom visual projects to use external libraries such as jQuery and the D3 library.

**Prerequisites:** This lab assumes you've already installed Node.JS and Visual Studio Code as described in setup.docx.

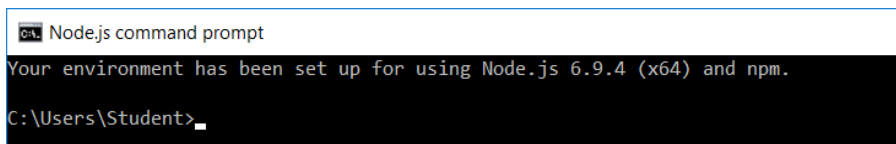
## Exercise 1: Prepare Your PC for Developing Custom Visuals

In this exercise you will install the Power BI Custom Visual Tool (PBIVIZ) and then install a self-signed SSL certificate which makes it possible to debug custom visual projects in Node.js using an address of <https://localhost>.

1. Install the Power BI Custom Visual Tool (PBIVIZ).
  - a) Using the Windows Start menu, launch the **Node.js command prompt**.



- b) You should now have an open Node.js command prompt.



- c) Type and execute the following command to install the Power BI Custom Visual Tool (PBIVIZ).

```
npm install -g powerbi-visuals-tools
```

- d) By inspecting the console output, you should be able to determine which version of the tools are being installed.

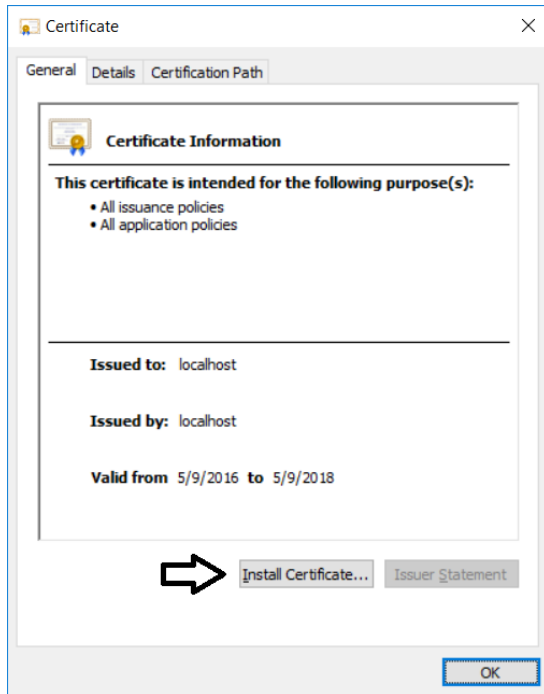
```
C:\Users\Student>npm install -g powerbi-visuals-tools
npm WARN deprecated minimatch@2.0.10: Please update to minimatch 3.0.2 or higher to avoid a RegEx
npm WARN deprecated minimatch@0.2.14: Please update to minimatch 3.0.2 or higher to avoid a RegEx
npm WARN deprecated graceful-fs@1.2.3: graceful-fs v3.0.0 and before will fail on node releases >
C:\Users\Student\AppData\Roaming\npm\pbiviz -> C:\Users\Student\AppData\Roaming\npm\node_modules\
lodash.isplainobject@4.0.6 node_modules\powerbi-visuals-tools\node_modules\gulp-powerbi-package-va
s\lodash.isplainobject
gulp-cli@1.2.2 node_modules\powerbi-visuals-tools\node_modules\gulp-powerbi-package-validator\nod
minimist@1.2.0 node_modules\powerbi-visuals-tools\node_modules\gulp\node_modules\minimist -> node
gulp@3.9.1 node_modules\powerbi-visuals-tools\node_modules\gulp -> node_modules\powerbi-visuals-t
- ms@0.7.2 node_modules\powerbi-visuals-tools\node_modules\send\node_modules\ms
C:\Users\Student\AppData\Roaming\npm
-- powerbi-visuals-tools@1.5.0
+-- connect@3.6.0
| +-- debug@2.6.1
```

2. install a self-signed SSL certificate for the SSL address of <https://localhost>.
  - a) In the Node.js command prompt, type and execute the following command.

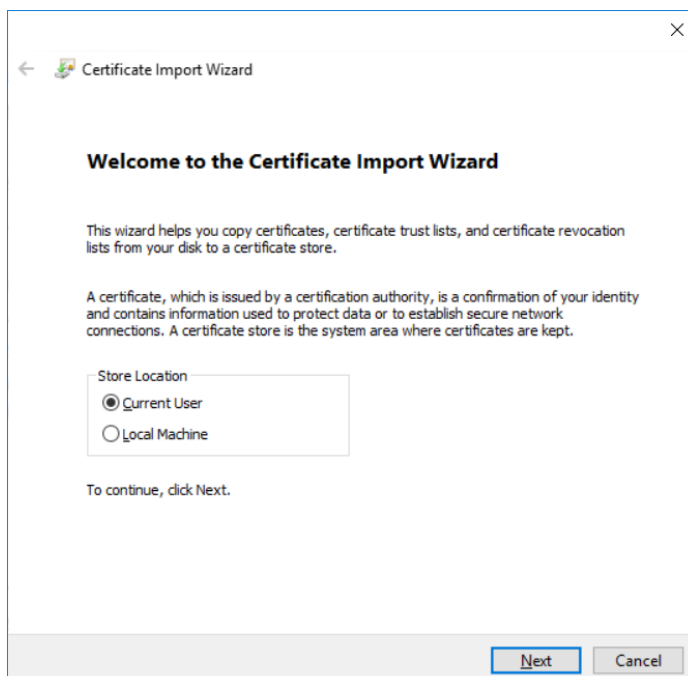
```
pbviz --install-cert
```

When you execute this command, you will be prompted with the Certificate dialog for a new self-signed SSL certificate for **localhost**.

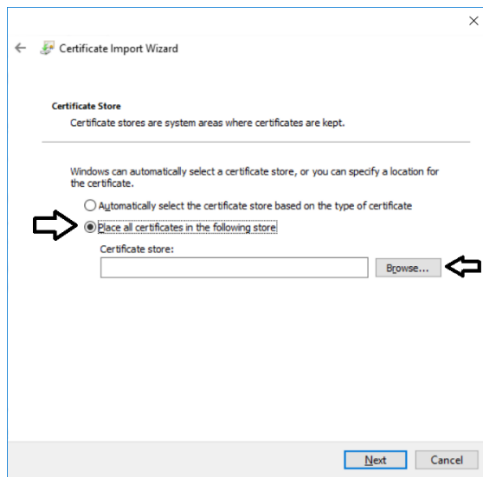
- b) When prompted by the **Certification** dialog, click the **Install Certificate...** button to start the Certificate Import Wizard.



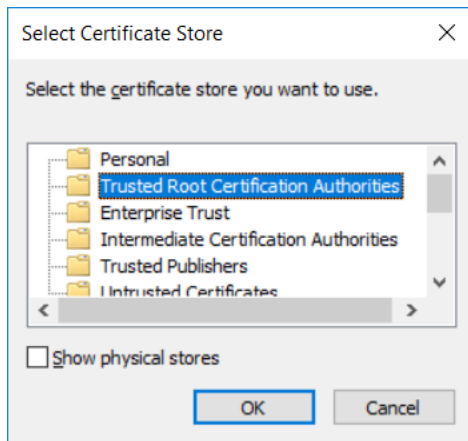
- c) On the **Welcome to the Certificate Import Wizard** page, select **Current user** and then click the **Next** button.



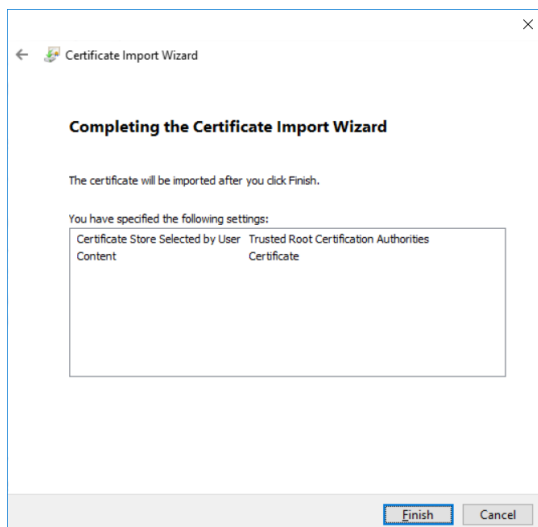
- d) On the **Certificate Store** page, select **Place all certificates in the following store** and then click the **Browse...** button.



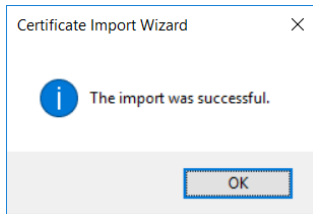
- e) In the **Select Certificate Store** dialog, select **Trusted Root Certificate Authorities** and click the **OK** button.



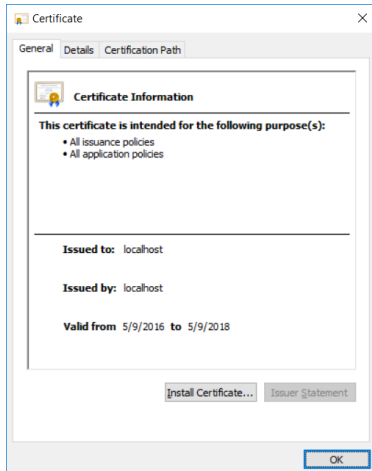
- f) On the **Certificate Store** page, click the **Next** button to continue.  
g) On the **Completing the Certification Import Wizard** page, click **Finish**.



- h) You should be prompted with a dialog that tells you the certificate import was successful.



- i) Click **OK** to close the **Certificate** dialog.



You have now installed the Power BI Custom Visual Tool (PBIVIZ) and a self-signed SSL certificate for the local debugging address of <https://localhost>. You can now begin to create and test custom visuals.

## Exercise 2: Create and Debug a Simple Custom Visual

In this exercise you will create and test out your first custom visual project using the PBIVIZ utility.

1. Create a new project for custom visual name viz01 using the PBIVIZ utility.
  - a) Return to the **Node.js** command prompt.
  - b) Type and execute `cd c:\Student` to switch the current path to the **c:\Student** folder.
  - c) Type and execute the command `md CustomVisuals` to create a new folder named **c:\Student\CustomVisuals**.
  - d) Type and execute `cd CustomVisuals` to switch the current path to the **c:\Student\CustomVisuals** folder.

```
Node.js command prompt

c:\Student>md CustomVisuals

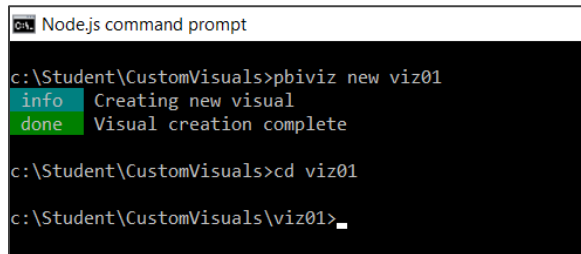
c:\Student>cd CustomVisuals

c:\Student\CustomVisuals>
```

- e) Type and execute the following command to create a new custom visual project named **viz01**.

```
pbiviz new viz01
```

- f) Once you have created the new project, type and execute **cd viz01** to make the folder for this project the current directory.



```
Node.js command prompt
c:\Student\CustomVisuals>pbviz new viz01
info Creating new visual
done Visual creation complete

c:\Student\CustomVisuals>cd viz01

c:\Student\CustomVisuals\viz01>
```

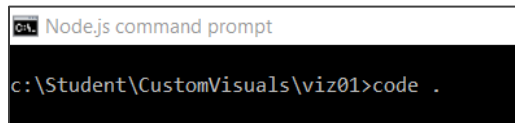
2. Run the **npm install** command on the new project to copy any missing package dependencies.

- Return to the Node.js command prompt.
- Type and execute the following command.

```
npm install
```

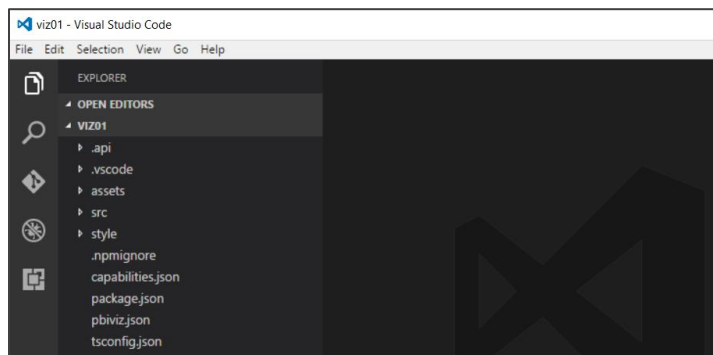
3. Open project using Visual Studio Code.

- In the Node.js command prompt, type and execute the command **code .** to open the project with Visual Studio Code.



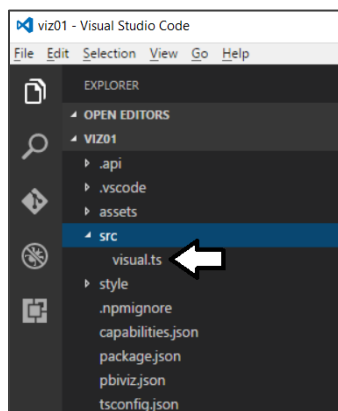
```
Node.js command prompt
c:\Student\CustomVisuals\viz01>code .
```

- Visual Studio Code should start and open the root folder of the **viz01** project.

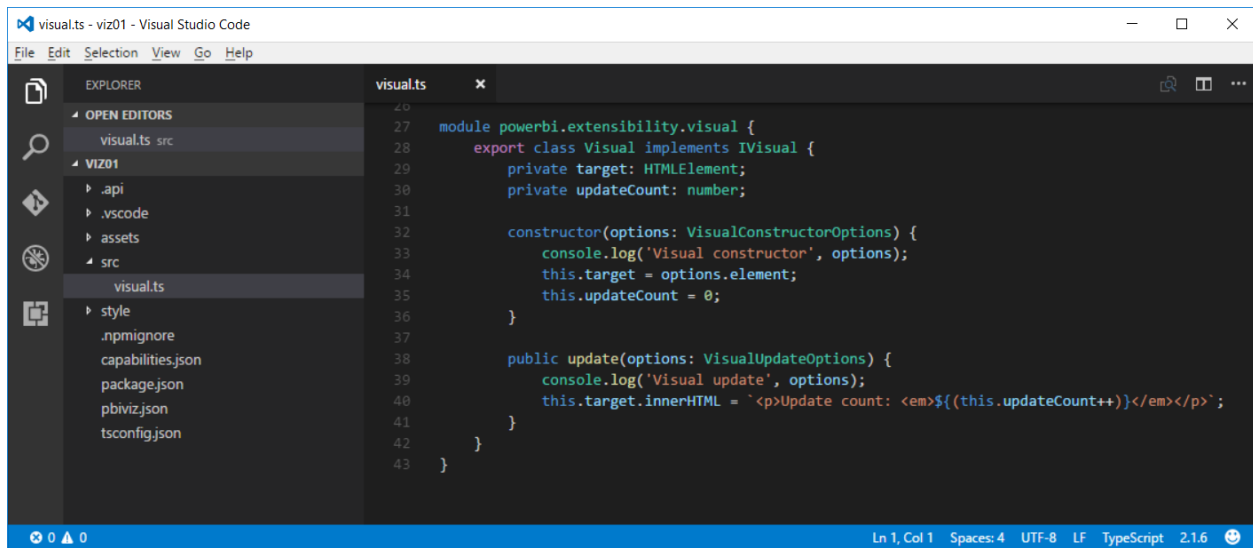


4. Inspect the TypeScript code in **visual.ts** to see the custom visual code that was added to the project by the PBIVIZ utility.

- In Visual Studio Code, expand the **src** folder and locate file named **visual.ts** and double-click this file to open it.



- b) Inside **visual.ts**, inspect the code for the class named **Visual**.



- c) Update the code in the **Visual** class with the following code.

```
module powerbi.extensibility.visual {
  export class Visual implements IVisual {

    private target: HTMLElement;
    private updateCount: number;

    constructor(options: VisualConstructorOptions) {
      console.log('visual constructor', options);
      this.target = options.element;
      this.updateCount = 0;
    }

    public update(options: VisualUpdateOptions) {
      console.log('visual update', options);
      this.target.innerHTML = `

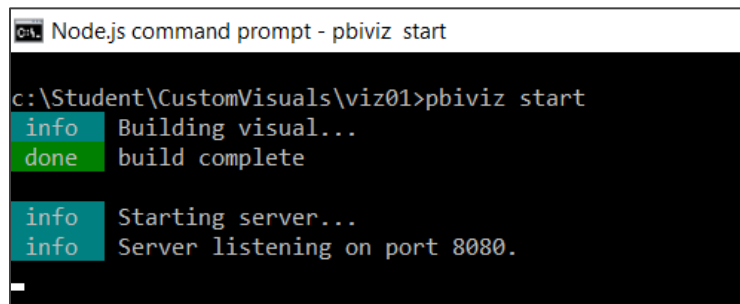
Update count: <em>${(this.updateCount++)}</em></p>`;
    }
  }
}


```

5. Use the PBIVIZ utility to start a debugging session for the **viz01** custom visual project.
- Return to the Node.js command prompt.
  - Type the following command to start a debugging session with the PBIVIZ utility and press ENTER

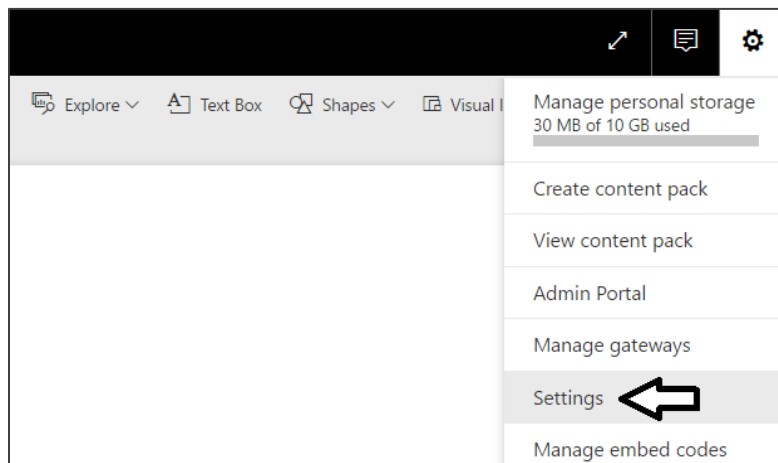
```
pbiviz start
```

- c) If you examine the console output, you can see that the web server provided by Node.js for debugging has started and is listening for incoming HTTP requests on **https://localhost:8080**.

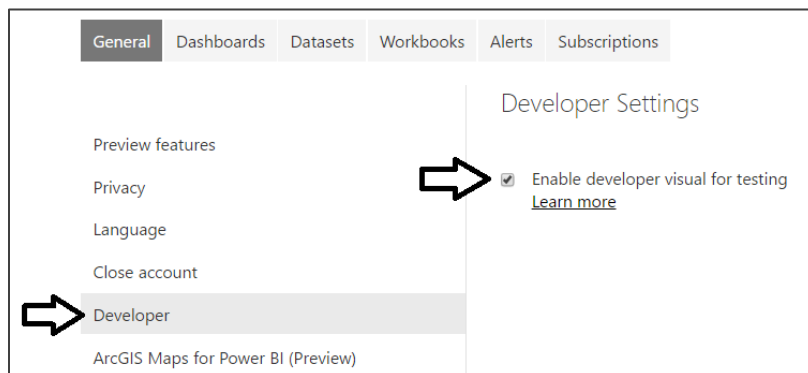


6. Test visual by loading it into the Power BI environment.

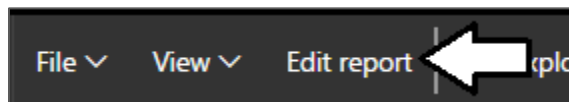
- Log into our personal workspace at <https://app.PowerBi.com>.
- Once you have logged into your personal workspace, drop down the Power BI Setting menu and select the **Settings** menu command as shown in the following dialog.



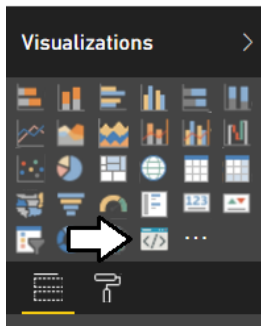
- Select **Developer** on the left and then select the **Enable developer visual for testing** checkbox.



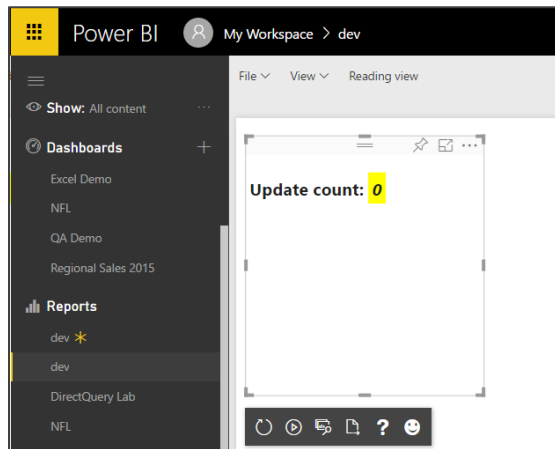
- Now navigate to any report in your workspace. It doesn't matter what report you use as long as you have a report.
- Move the report into edit mode by clicking the Edit report button.



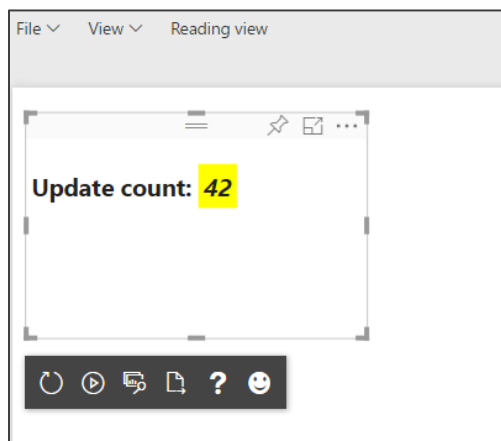
- Add a new page to the report so you have a blank report page to work with.
- Add an of the Developer Visual to the page.



h) You should see the custom visual for **viz01** on the report page.



i) Resize the visual by dragging and dropping the lower right corner of the visual with the mouse. You should see the **Update count** total increase whenever you resize the visual.



7. Make a change to the code in the visual.

- a) Return to Visual Studio Code.
- b) Open the source file **visual.ts**.
- c) Inside **visual.ts**, locate the **update** method.

```
public update(options: VisualUpdateOptions) {  
    console.log('Visual update', options);  
    this.target.innerHTML = `

Update count: <em>${(this.updateCount++)}</em></p>`;   
}

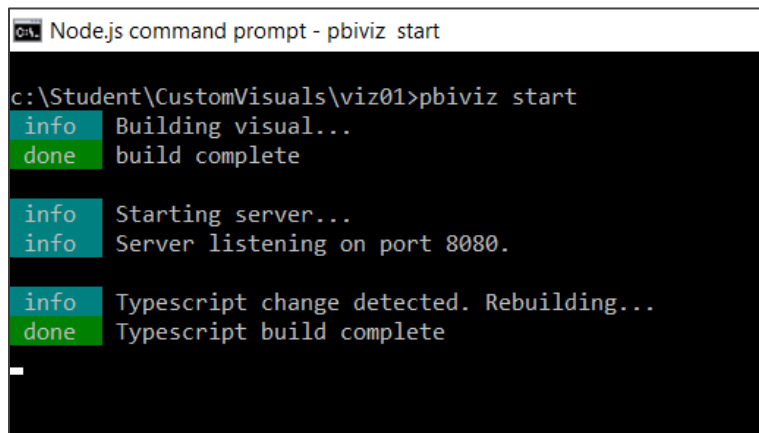

```



- d) Replace the code in the **update** method by copying and pasting the following code.

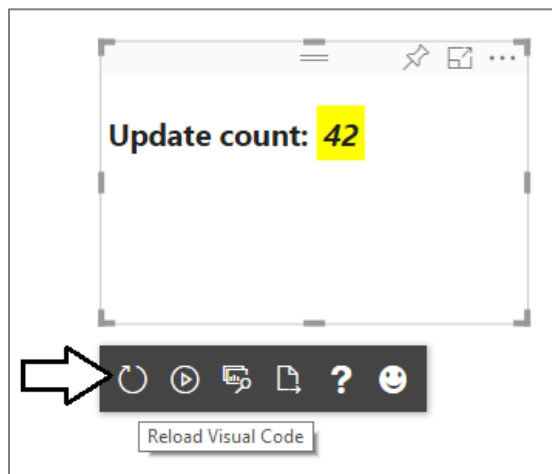
```
public update(options: VisualUpdateOptions) {  
    console.log('Visual update', options);  
    this.target.innerHTML =  
        <table>  
            <tr><td>Width:</td><td>${options.viewport.width}</td></tr>  
            <tr><td>Height:</td><td>${options.viewport.height}</td></tr>  
        </table>;  
}
```

- e) After you have updated the **update** method, save your changes to the file **visual.ts**.  
f) When you save **visual.ts**, you will notice that the Node.js command prompt will run to recompile the code for your visual..

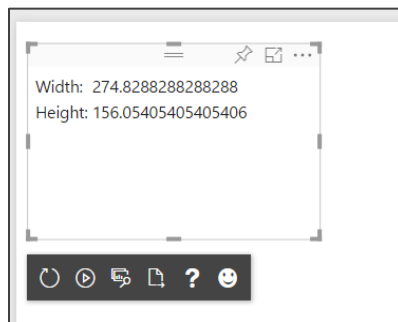


```
Node.js command prompt - pbviz start  
  
c:\Student\CustomVisuals\viz01>pbviz start  
info Building visual...  
done build complete  
  
info Starting server...  
info Server listening on port 8080.  
  
info Typescript change detected. Rebuilding...  
done Typescript build complete
```

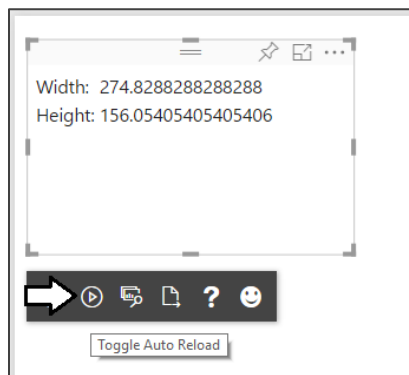
- g) Return to the browser when you instantiated the visual for testing.  
h) In the visual developer panel, click the **Reload Visual Code** button as shown in the following screenshot.



- i) You should now see that the visual output as shown in the following screenshot which shows the visual width and height.



- j) Click the **Toggle Auto Reload** button so that visual automatically reloads when you make additional updates.

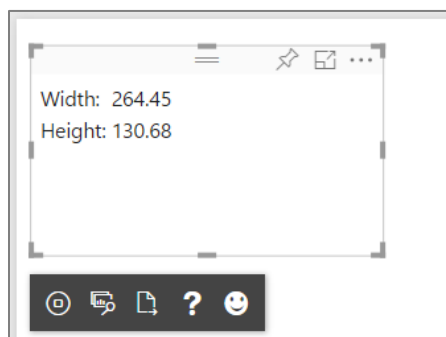


8. Make another change to the code in your custom visual.

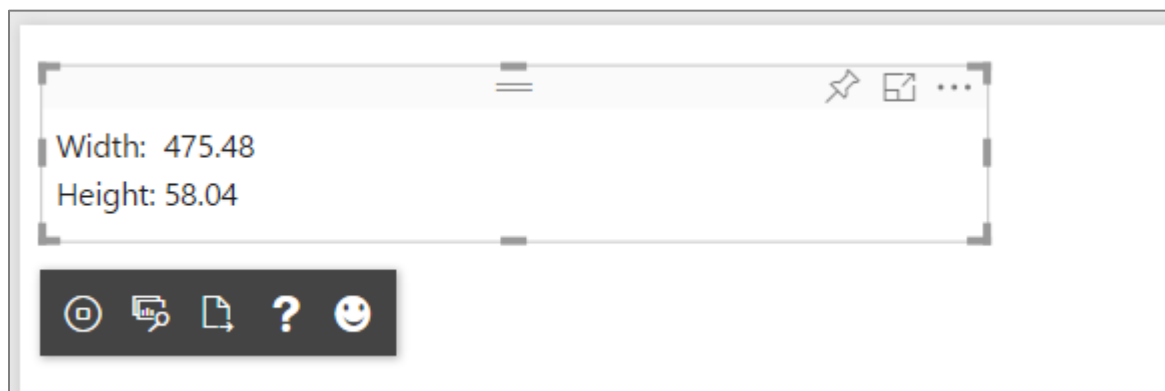
- Return to Visual Studio Code.
- Move down to the **update** method.
- Modify the code that retrieve the visual width and height by adding the **toFixed(2)** method to configure the output to only show two digits of accuracy after the decimal.

```
public update(options: VisualUpdateOptions) {  
    console.log('Visual update', options);  
    this.target.innerHTML =  
        `<table>  
          <tr><td>width:</td><td>${options.viewport.width.toFixed(2)}</td></tr>  
          <tr><td>Height:</td><td>${options.viewport.height.toFixed(2)}</td></tr>  
        </table>`;  
}
```

- Save your changes to **visual.ts**.
- Return to the browser and you should see the impact of the change you just made.



- f) Experiment by resizing the visual and you should see the width and height automatically update.



At this point, you are done testing your first custom visual.

9. Stop visual debugging session.
  - a) Return to Node.js command prompt.
  - b) Hold down the **Ctrl** key on the keyboard and then press **C** to interrupt the Node.js debugging session.

```
info Typescript change detected. Rebuilding...
done Typescript build complete

info Typescript change detected. Rebuilding...
done Typescript build complete

info Stopping server...
^CTerminate batch job (Y/N)?
```

10. When prompted to terminate the session, type Y and press ENTER.

```
info Typescript change detected. Rebuilding...
done Typescript build complete

info Stopping server...
^CTerminate batch job (Y/N)? y

c:\Student\CustomVisuals\viz01>
```

### Exercise 3: Create a Custom Visual using jQuery

In this exercise, you will create a new visual named **viz02** which uses jQuery to implement a simple Power BI visual.

1. Create a new visual project named **viz02**.
  - a) Return to the Node.js command prompt.
  - b) Type and execute the following command to make the current directory back to **C:\Student\CustomVisuals**

```
Cd ..
```

```
Node.js command prompt

c:\Student\CustomVisuals>pbviz new viz02
info Creating new visual
done Visual creation complete

c:\Student\CustomVisuals>
```

- c) Type and execute the following three commands to create a new visual project and open it in Visual Studio Code.

```
pbviz new viz02  
cd viz02
```

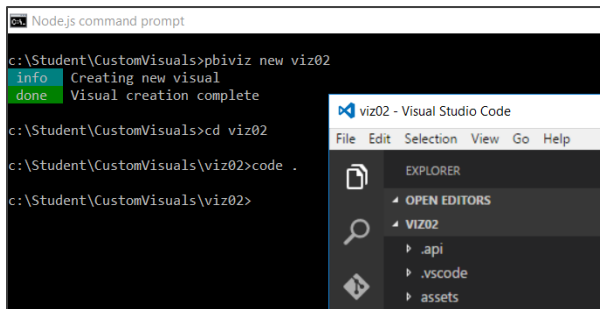
- d) Run the **npm install** command on the new project to copy any missing package dependencies.

```
npm install
```

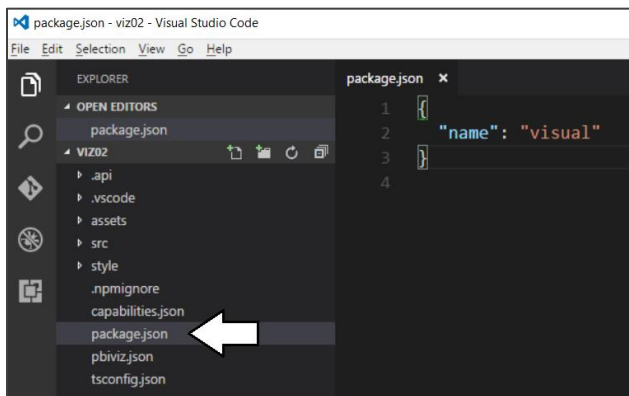
- e) Type and execute the following line to open Visual Studio Code

```
Code .
```

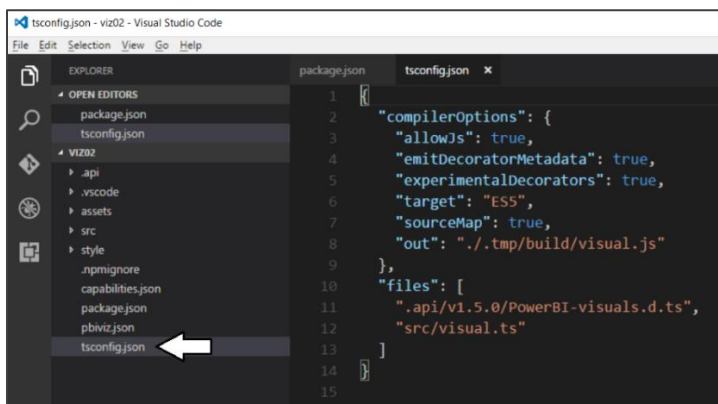
- f) You should now have a new project named viz02 that is open in Visual Studio Code.



- g) Open the **package.json** file and inspect its contents.



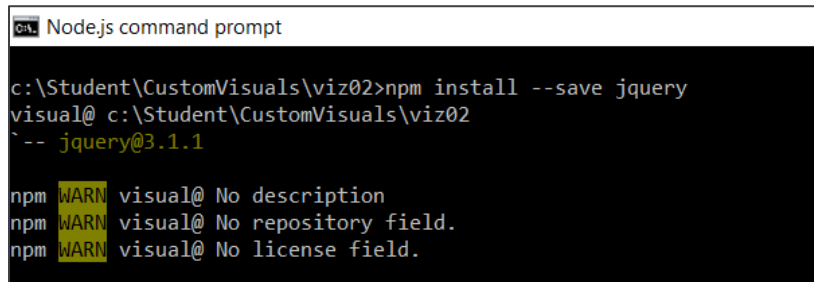
- h) Open the **tsconfig.json** file and inspect its contents.



2. Add support to the **viz02** project to program against jQuery
  - a) Return to the Node.js command prompt.
  - b) Type and execute the following command.

```
npm install jquery --save-dev
```

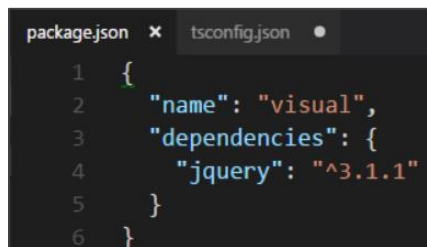
- c) When you execute this command, you should see the following output in the console.



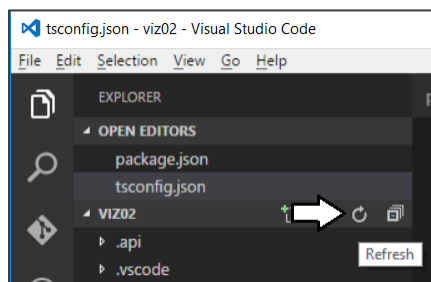
```
Node.js command prompt
c:\Student\CustomVisuals\viz02>npm install --save jquery
visual@ c:\Student\CustomVisuals\viz02
`-- jquery@3.1.1

npm WARN visual@ No description
npm WARN visual@ No repository field.
npm WARN visual@ No license field.
```

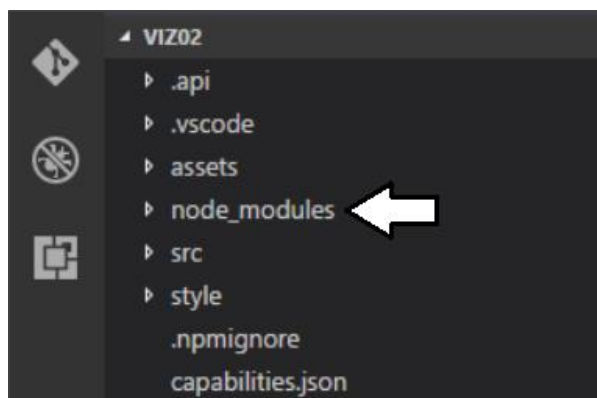
- d) Return to Visual Studio Code and inspect the **package.json** file to see how it has changed.



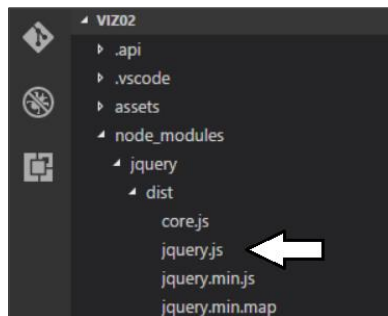
- e) In the left navigation of Visual Studio Code, click the Refresh button to show any new folders.



- f) After you refresh the project, you should see a new folder named **node\_modules**.

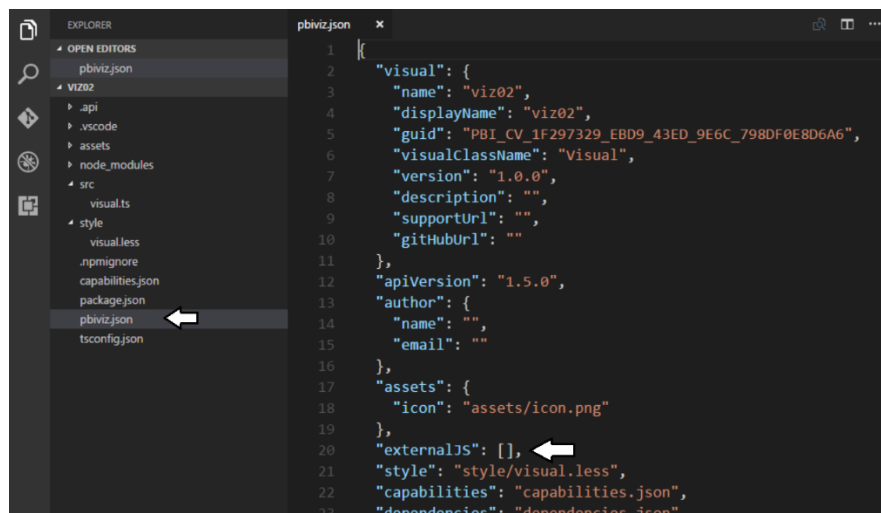


- g) Expand the **node\_modules** folder and locate the main jQuery source file located at **node\_modules/jquery/dist/jquery.js**.



3. Add the external file reference for **jquery.js** into the **pbviz.json** file.

- a) Open the **pbviz.json** file and locate the **externalJS** setting as shown in the following screenshot.



- b) Update the array for the **externalJS** setting by adding the path to **jquery.js**.

```
"externalJS": [  
  "node_modules/jquery/dist/jquery.js"  
],
```

- c) When the **externalJS** setting matches the following screenshot, save and close **pbviz.json**.



4. Add DefinitelyTyped package for the jQuery library to enable strongly-typed programming
  - a) Return to the Node.js command prompt.
  - b) Type and execute the following command to add the typed definition files for jQuery into your project.

```
npm install @types/jquery --save-dev
```

- c) As the command executes, you should see the following output in the console window.

```
c:\Student\CustomVisuals\viz02>npm install @types/jquery --save
visual@ c:\Student\CustomVisuals\viz02
~-- @types/jquery@2.0.40

npm WARN visual@ No description
npm WARN visual@ No repository field.
npm WARN visual@ No license field.
```

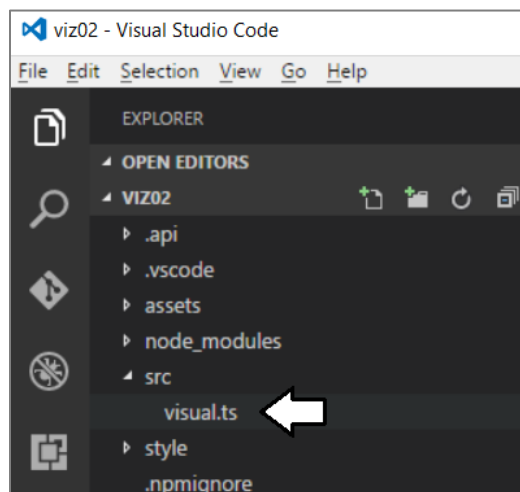
- d) Return to Visual Studio Code and inspect the **package.json** file. You should see the new reference to **@types/jquery**.
5. Update the **tsconfig.json** file.
  - a) Open the **tsconfig.json** file.
  - b) Move to the bottom of **tsconfig.json** and locate the **files** property.
  - c) Add an entry for **node\_modules/@types/jquery/index.d.ts** just before **src/visual.ts** as shown in the following screenshot.

```
  },
  "files": [
    ".api/v1.5.0/PowerBI-visuals.d.ts",
    "node_modules/@types/jquery/index.d.ts",
    "src/visual.ts"
  ]
}
```

- d) Save and close **tsconfig.js**.

Now you are finally ready to begin programming with TypeScript code that uses the jQuery library.

6. Modify the **visual.ts** file.
  - a) Open **visual.ts** if it is not already open.



- b) Modify the class definition named **Visual** to match the following code listing.

```
module powerbi.extensibility.visual {  
    export class Visual implements IVisual {  
        constructor(options: VisualConstructorOptions) { }  
        public update(options: VisualUpdateOptions) { }  
    }  
}
```

- c) Modify the **Visual** class to match the following code listing,

```
module powerbi.extensibility.visual {  
    export class Visual implements IVisual {  
        private container: JQuery;  
        constructor(options: VisualConstructorOptions) {  
            this.container = $(options.element);  
        }  
        public update(options: VisualUpdateOptions) {}  
    }  
}
```

- d) Modify the **update** method to match the following code listing.

```
public update(options: VisualUpdateOptions) {  
    var table: JQuery = $("

|                                                                                                                                                                                                                                                                                                                                                                                  |                                                                                                                                                                                                                                                                                                                |                                                                                                                                                                                |                                                                                                      |
|----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|--------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|------------------------------------------------------------------------------------------------------|
| ").text("width") )<br>.append( \$(" <td>").text(options.viewport.width.toFixed(0))<br/>            ))<br/>        .append( \$("<tr>")<br/>            .append( \$("<td>").text("Height") )<br/>            .append( \$("<td>").text(options.viewport.height.toFixed(0))<br/>            )<br/>        );<br/>    this.container.empty().append(table);<br/>}</td></td></tr></td> | ").text(options.viewport.width.toFixed(0))<br>))<br>.append( \$(" <tr>")<br/>            .append( \$("<td>").text("Height") )<br/>            .append( \$("<td>").text(options.viewport.height.toFixed(0))<br/>            )<br/>        );<br/>    this.container.empty().append(table);<br/>}</td></td></tr> | ").text("Height") )<br>.append( \$(" <td>").text(options.viewport.height.toFixed(0))<br/>            )<br/>        );<br/>    this.container.empty().append(table);<br/>}</td> | ").text(options.viewport.height.toFixed(0))<br>)<br>);<br>this.container.empty().append(table);<br>} |
| ").text("Height") )<br>.append( \$(" <td>").text(options.viewport.height.toFixed(0))<br/>            )<br/>        );<br/>    this.container.empty().append(table);<br/>}</td>                                                                                                                                                                                                   | ").text(options.viewport.height.toFixed(0))<br>)<br>);<br>this.container.empty().append(table);<br>}                                                                                                                                                                                                           |                                                                                                                                                                                |                                                                                                      |


```

- e) Save our changes to **visual.ts**.

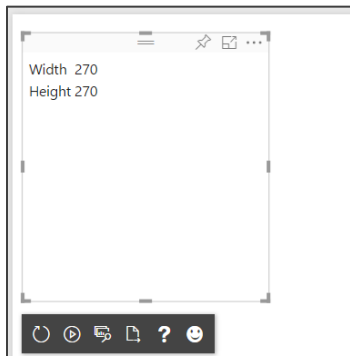
7. Test out your new visual on a Power BI report.

- a) Return to the Node.js command prompt and run the following command to start a new debugging session.

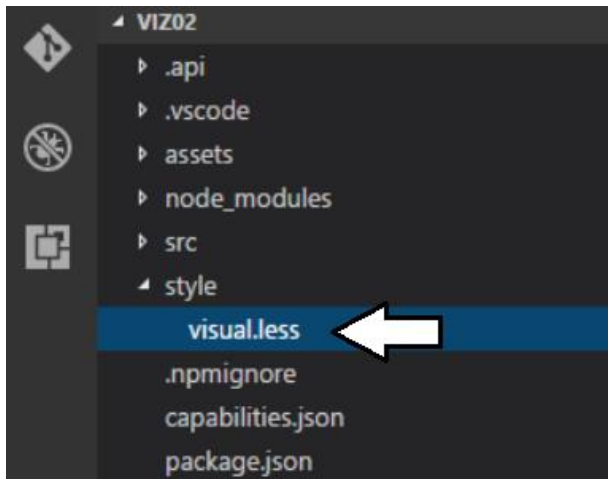
```
pbiviz start
```

- b) Move back to the browser and return to the Power BI report you used in the previous exercise.  
c) Make sure the report is in edit mode.  
d) Add a Developer Visual to the page to see your visual. It should match the following screenshot.





8. Add some CSS styles to your visual.
  - a) Return to the **viz02** project in Visual Studio Code.
  - b) Expand the style folder and open the file named **visual.less**.



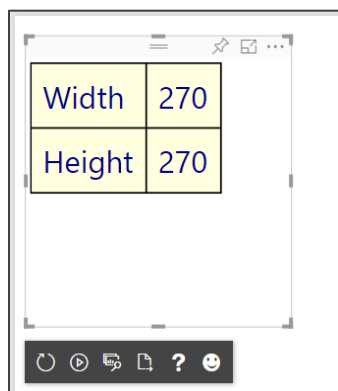
- c) Replace the contents of **visual.less** with the following CSS code.

```
#myTable{  
    background-color: black;  
    font-size: 32px;  
    td{  
        background-color: lightyellow;  
        color:darkblue;  
        padding: 12px;  
    }  
}
```

- d) Save your changes to **visual.less**.

When you save your changes to **visual.less**, the PBIVIZ utility will recompile the entire **viz02** project.

- e) Return to the browser and refresh your visual. You should see the effects of the CSS styles in your visual.



9. Add scaling font behavior to your visual.
  - a) Return to Visual Studio Code.
  - b) Navigate to the **visual.ts** file and locate the **update** method.
  - c) Place your cursor in the update method just before the last line that looks like this.

```
this.container.empty().append(table);
```

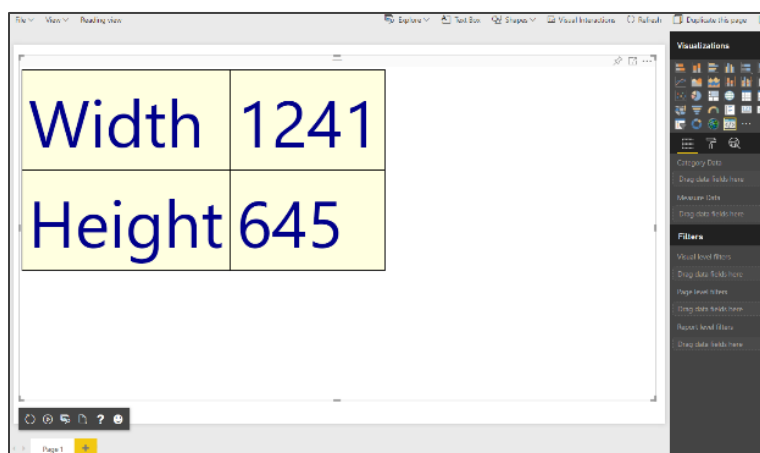
- d) Copy and paste the following code into the **update**.

```
var scaledFontSizeWidth: number = Math.round(options.viewport.width / 7);
var scaledFontSizeHeight: number = Math.round(options.viewport.height / 5);
var scaledFontSize: number = Math.min(...[scaledFontSizeWidth, scaledFontSizeHeight]);
var scaledFontSizeCss: string = scaledFontSize + "px";

$("td", table).css({
  "font-size": scaledFontSizeCss
});

this.container.empty().append(table);
```

- e) Save your changes to **visual.ts**.
- f) Return to the browser and refresh your visual.
- g) As you change the size of the visual, the font size should now change along with it.



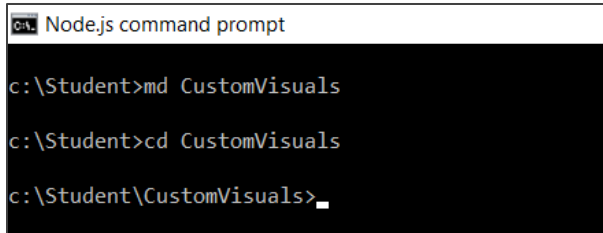
## Exercise 4: Create a Custom Visual using the D3 Library

In this exercise, you will create a new visual named **viz03** which uses D3 to implement a simple Power BI visual.

1. Create a new visual project named **viz03**.

- a) Return to the Node.js command prompt.
- b) Type and execute the following command to make the current directory back to **C:\Student\CustomVisuals**

```
Cd ..
```

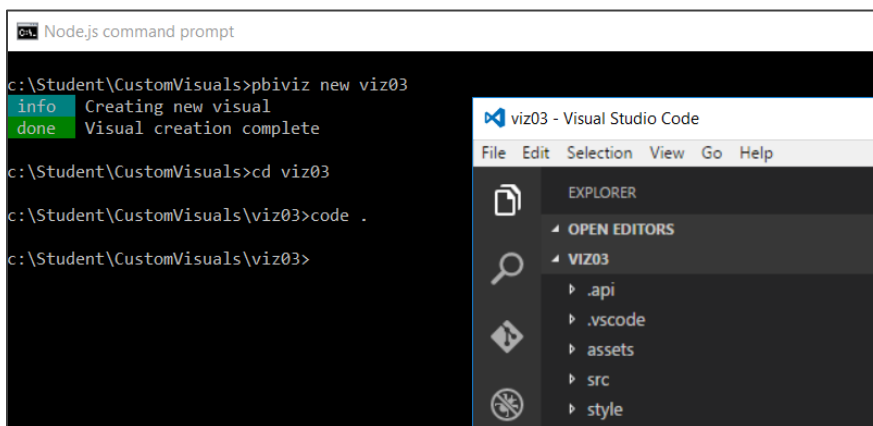


```
Node.js command prompt
c:\Student>md CustomVisuals
c:\Student>cd CustomVisuals
c:\Student\CustomVisuals>
```

- c) Type and execute the following three commands to create a new visual project and open it in Visual Studio Code.

```
pbviz new viz03
cd viz03
code .
```

- d) You should now have a new project named **viz03** that is open in Visual Studio Code.



2. Run the **npm install** command on the new project to copy any missing package dependencies.

- a) Return to the Node.js command prompt.
- b) Type and execute the following command.

```
npm install
```

3. Add the NPM package for the D3 library.

- a) Type and execute the following command.

```
npm install d3@3 --save-dev
```

- b) Wait for the install command to complete.
- c) Type and execute the following command to install the typed definition files for the D3 library.

```
npm install @types/d3@3 --save
```

- d) Wait for the **npm install** command to complete.
- e) Return to Visual Studio Code.
- f) Open the **package.json** file to see the new dependency on the D3 library and the D3 typed definition files.

You don't need to make any changes to **package.json** as the npm install command does that for you when you use the **--save** option.

4. Add the external reference for **d3.js** into the **pbviz.json** file.
  - a) Open the **pbviz.json** file and locate the **externalJS** setting as shown in the following screenshot.



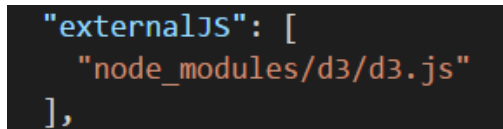
A screenshot of a code editor showing the **pbviz.json** file. The file is open, and the **externalJS** property is highlighted with a white arrow pointing to its value, which is an empty array `[]`. The file structure is as follows:

```
11 },
12 "apiVersion": "1.5.0",
13 "author": {
14   "name": "",
15   "email": ""
16 },
17 "assets": {
18   "icon": "assets/icon.png"
19 },
20 "externalJS": [],
21 "style": "style/visual.less",
22 "capabilities": "capabilities.json",
23 "dependencies": "dependencies.json"
24 }
```

- b) Update the array for the **externalJS** setting by adding the path to **d3.js**.

```
"externalJS": [
  "node_modules/d3/d3.js"
],
```

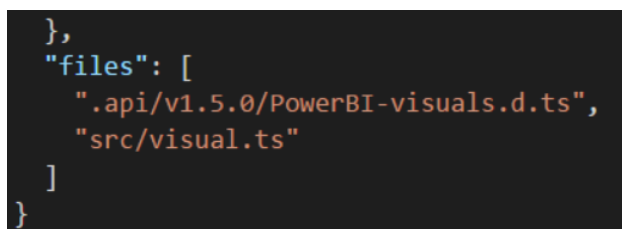
- c) When the **externalJS** setting matches the following screenshot, save and close **pbviz.json**.



A screenshot of the **pbviz.json** file showing the **externalJS** property updated with the path `"node_modules/d3/d3.js"`. The code is as follows:

```
"externalJS": [
  "node_modules/d3/d3.js"
],
```

5. Modify the **files** setting in the project's **tsconfig.json** file to include a reference to the D3 typed definition file for named **index.d.ts**.
  - a) Open the **tsconfig.json** file.
  - b) Move to the bottom of **tsconfig.json** and locate the **files** property.



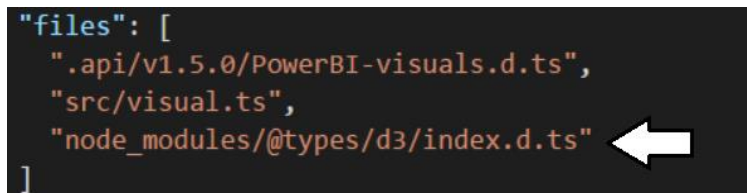
A screenshot of the **tsconfig.json** file showing the **files** property. The file structure is as follows:

```
},
"files": [
  ".api/v1.5.0/PowerBI-visuals.d.ts",
  "src/visual.ts"
]
```

- c) Add a new entry into the array for the **files** property using the following project-relative path to the D3 typed definition file.

```
node_modules/@types/d3/index.d.ts
```

- d) Your update to **tsconfig.json** should match the following screenshot.



A screenshot of the **tsconfig.json** file showing the **files** property updated with the path `"node_modules/@types/d3/index.d.ts"`. The code is as follows:

```
"files": [
  ".api/v1.5.0/PowerBI-visuals.d.ts",
  "src/visual.ts",
  "node_modules/@types/d3/index.d.ts"
]
```

- e) Save and close **tsconfig.json**.

Now you are now ready to begin programing your TypeScript code using the D3 library.

6. Modify the **visual.ts** file.

- a) Open **visual.ts** if it is not already open.
- b) Modify the class definition named **Visual** to match the following code.

```
module powerbi.extensibility.visual {  
    export class Visual implements IVisual {  
        constructor(options: VisualConstructorOptions) { }  
        public update(options: VisualUpdateOptions) { }  
    }  
}
```

- c) Modify the Visual class as shown in the following code listing.

```
module powerbi.extensibility.visual {  
    export class Visual implements IVisual {  
        private svgRoot: d3.Selection<SVGElementInstance>;  
        private ellipse: d3.Selection<SVGElementInstance>;  
        private text: d3.Selection<SVGElementInstance>;  
        private padding: number = 20;  
        constructor(options: VisualConstructorOptions) {  
            this.svgRoot = d3.select(options.element).append("svg");  
        }  
        public update(options: VisualUpdateOptions) {}  
    }  
}
```

- d) Modify the constructor using the following code.

```
constructor(options: VisualConstructorOptions) {  
    this.svgRoot = d3.select(options.element).append("svg");  
  
    this.ellipse = this.svgRoot.append("ellipse")  
        .style("fill", "rgba(255, 255, 0, 0.5)")  
        .style("stroke", "rgba(0, 0, 0, 1.0)")  
        .style("stroke-width", "4");  
  
    this.text = this.svgRoot.append("text")  
        .text("Hello D3")  
        .attr("text-anchor", "middle")  
        .attr("dominant-baseline", "central")  
        .style("fill", "rgba(255, 0, 0, 1.0)")  
        .style("stroke", "rgba(0, 0, 0, 1.0)")  
        .style("stroke-width", "2");  
}
```

- e) Modify the update method by adding code to append the **svgRoot** element.

```
public update(options: VisualUpdateOptions) {  
    this.svgRoot  
        .attr("width", options.viewport.width)  
        .attr("height", options.viewport.height);  
}
```

- f) Add the following code to create a **plot** variable.

```
this.svgRoot
  .attr("width", options.viewport.width)
  .attr("height", options.viewport.height);

var plot = {
  xoffset: this.padding,
  yoffset: this.padding,
  width: options.viewport.width - (this.padding * 2),
  height: options.viewport.height - (this.padding * 2),
};
```

- g) Add the following code to resize the ellipse.

```
this.ellipse
  .attr("cx", plot.xoffset + (plot.width * 0.5))
  .attr("cy", plot.yoffset + (plot.height * 0.5))
  .attr("rx", (plot.width * 0.5))
  .attr("ry", (plot.height * 0.5))
```

- h) Add the following code to scale the font size.

```
var fontSizeForWidth: number = plot.width * .20;
var fontSizeForHeight: number = plot.height * .35;
var fontSize: number = d3.min([fontSizeForWidth, fontSizeForHeight]);
```

- i) Add the following code to resize the text element.

```
this.text
  .attr("x", plot.xoffset + (plot.width / 2))
  .attr("y", plot.yoffset + (plot.height / 2))
  .attr("width", plot.width)
  .attr("height", plot.height)
  .attr("font-size", fontSize);
```

- j) Your implementation of **update** should now match the following code listing.

```
public update(options: VisualUpdateOptions) {

  this.svgRoot
    .attr("width", options.viewport.width)
    .attr("height", options.viewport.height);

  var plot = {
    xoffset: this.padding,
    yoffset: this.padding,
    width: options.viewport.width - (this.padding * 2),
    height: options.viewport.height - (this.padding * 2),
  };

  this.ellipse
    .attr("cx", plot.xoffset + (plot.width * 0.5))
    .attr("cy", plot.yoffset + (plot.height * 0.5))
    .attr("rx", (plot.width * 0.5))
    .attr("ry", (plot.height * 0.5))

  var fontSizeForWidth: number = plot.width * .20;
  var fontSizeForHeight: number = plot.height * .35;
  var fontSize: number = d3.min([fontSizeForWidth, fontSizeForHeight]);

  this.text
    .attr("x", plot.xoffset + (plot.width / 2))
    .attr("y", plot.yoffset + (plot.height / 2))
    .attr("width", plot.width)
    .attr("height", plot.height)
    .attr("font-size", fontSize);
}
```

7. Test out your new visual on a Power BI report.
- a) Return to the Node.js command prompt and run the following command to start a new debugging session.

```
pbviz start
```

- b) Move back to the browser and return to the Power BI report you used in the previous exercise.
- c) Make sure the report is in edit mode.
- d) Add a Developer Visual to the page to see your visual. It should match the following screenshot.



- e) Experiment by resizing the visual and seeing how it scales to various sizes.

