

# Developing Custom Visuals for Power BI

**Setup Time:** 60 minutes

**Lab Folder:** C:\Student\Modules\08\_CustomVisuals\Lab

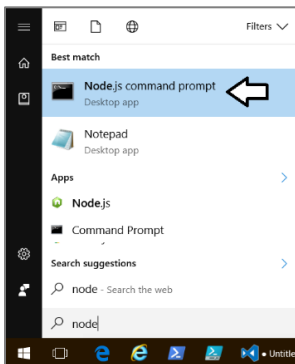
**Overview:** In this lab, you will begin by installing the Power BI Custom Visual Tool (PBIVIZ). After that, you will use Node.js, the PBIVIZ utility and Visual Studio Code to create and debug custom visual projects within the context of a Power BI workspace.

**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 the local 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.

```
Node.js command prompt
Your environment has been set up for using Node.js 10.15.3 (x64) and npm.
C:\Users\TedP>
```

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

```
npm install -g powerbi-visuals-tools@3
```

The **@3** at the end ensures that you install the version of these tools with the largest version number that starts with a 3 but not a 2.

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

```
+ powerbi-visuals-tools@3.1.2
updated 11 packages in 46.519s
C:\Users\TedP>_
```

2. Run the **pbiviz --help** command to discover what commands are available.
  - a) From the Node.js command prompt, type and execute the following command.

```
pbiviz --help
```

- b) You should see output in the console that lists the commands supported by the **pbviz** utility.

```
C:\Users\TedP>pbviz --help

Usage: pbviz [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 pbviz file
  update [version]    Updates the api definitions and schemas in the current visual. Changes the version if specified
  help [cmd]          display help for [cmd]

C:\Users\TedP>
```

- c) Check the version of **pbviz** by typing and executing the following command.

```
pbviz --version
```

- d) You should now see the version number of the **pbviz** utility as output in the console window.

```
Node.js command prompt

C:\Users\TedP>pbviz --version
3.1.2
```

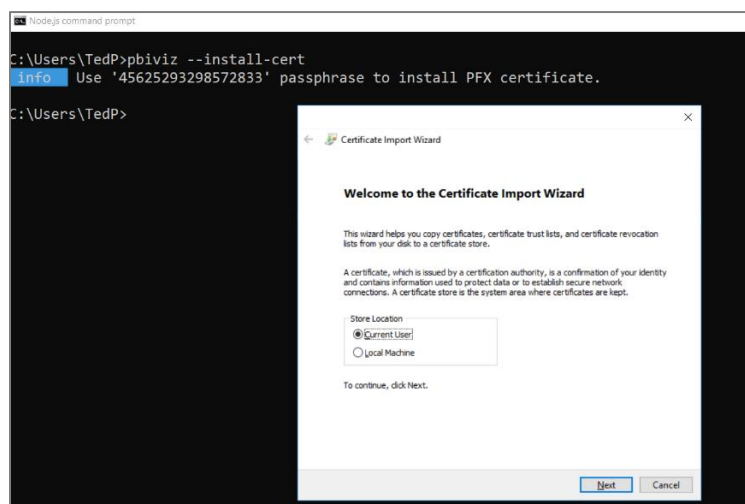
3. Install a self-signed SSL certificate to test and debug custom visuals using the <https://localhost> domain.

- 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 by the **Certificate Import Wizard**. This wizard provides an interactive experience in which you will work to complete the process of installing the self-signed SSL certificate for **localhost**.

- b) On the **Welcome to the Certificate Import Wizard** page, click **Next**.

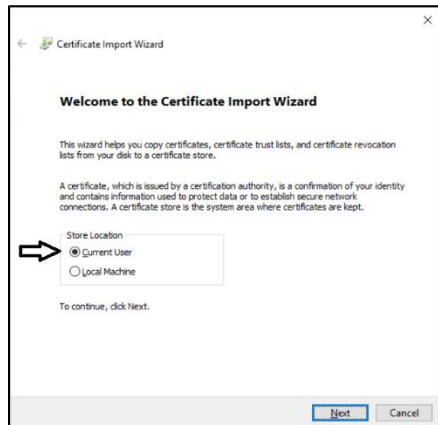


- c) Navigate back to the console and copy the passphrase into the Windows clipboard.

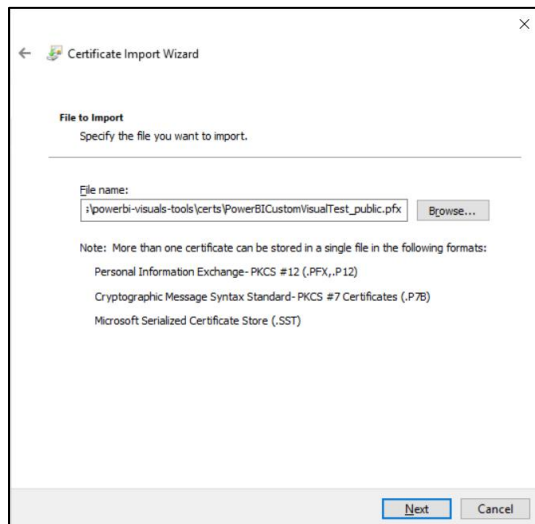
```
Node.js command prompt

C:\Users\TedP>pbviz --install-cert
info Use '45625293298572833' passphrase to install PFX certificate.
```

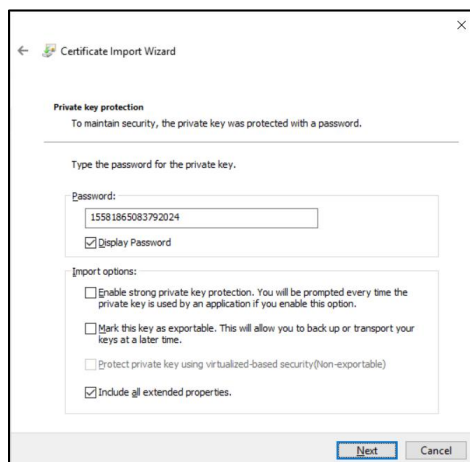
- d) On the **Welcome to the Certificate Import Wizard** page, select **Current User** and then click **Next**.



- e) On the **File to Import** page, accept the default **File name** value and click **Next** to continue.

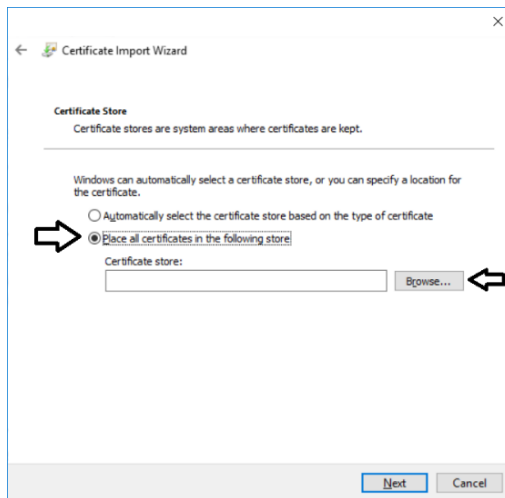


- f) On the **Private key protection** page, paste the passphrase from the clipboard into the **Password** textbox and click **Next**.

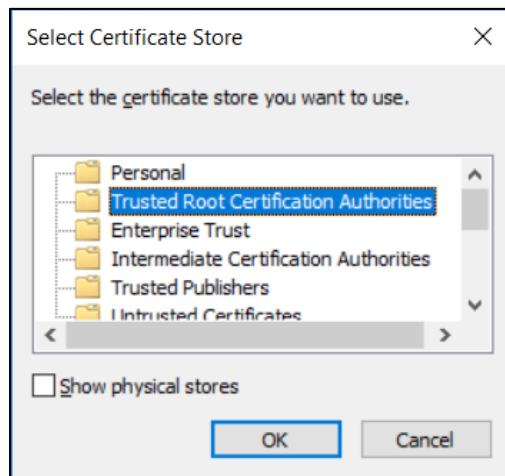


It can be helpful to check the **Display Password** option so you can make sure there are no quotes at the start or end of the password.

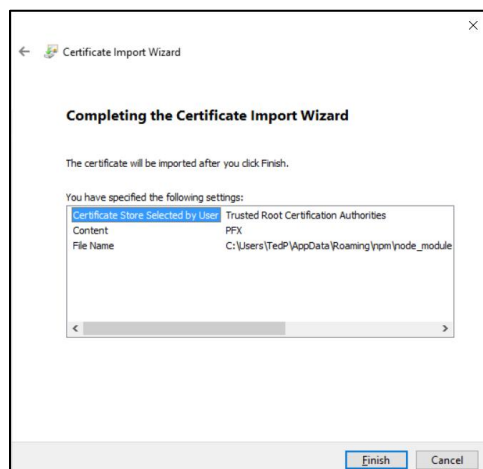
- g) On the **Certificate Store** page, select **Place all certificates in the following store** and then click the **Browse...** button.



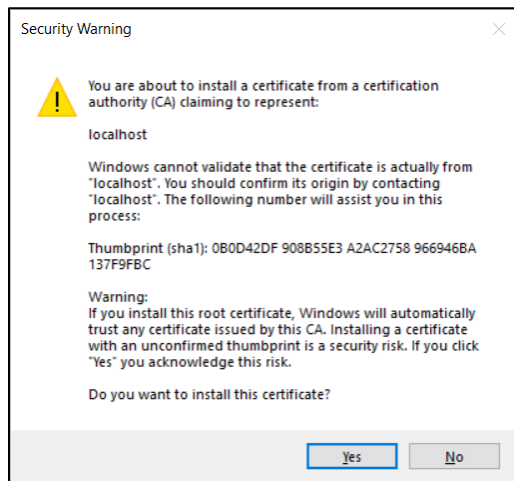
- h) In the **Select Certificate Store** dialog, select **Trusted Root Certificate Authorities** and click the **OK** button.



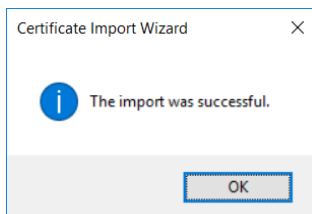
- i) On the **Certificate Store** page, click the **Next** button to continue.  
j) On the **Completing the Certification Import Wizard** page, click **Finish**.



- k) If you see the following **Security Warning** dialog, click **Yes** to continue with the certificate installation.



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

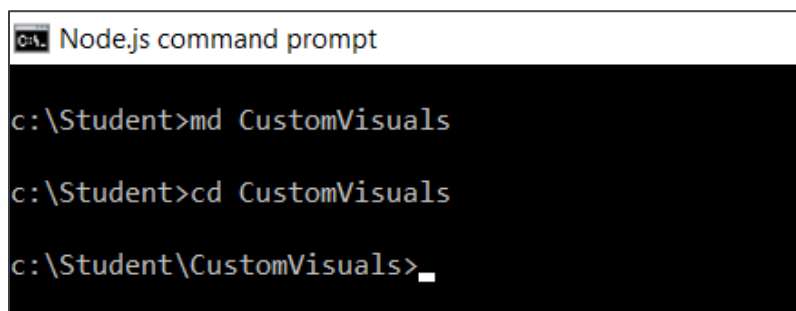


You have now installed the Power BI Custom Visual Tool (PBIVIZ) and a self-signed SSL certificate for the local debugging address of <https://localhost>. You can now begin to create, test and debug custom visuals.

## 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.

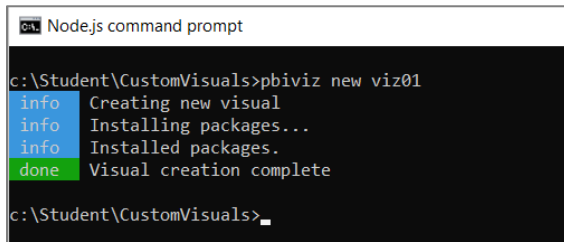
1. Create a new project for custom visual name **viz01** using the **pbiviz** utility.
  - a) Return to the **Node.js** command prompt.
  - b) Type and execute **cd c:\Student** to switch the current path to the **c:\Student** folder.
  - c) Type and execute the command **md CustomVisuals** to create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



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

```
pbiviz new viz01
```

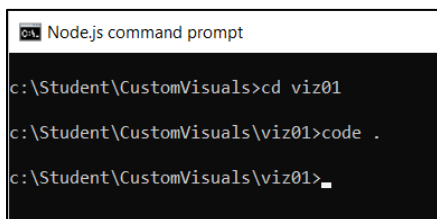
- f) You should see output in the console window that the visual has been created successfully.



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

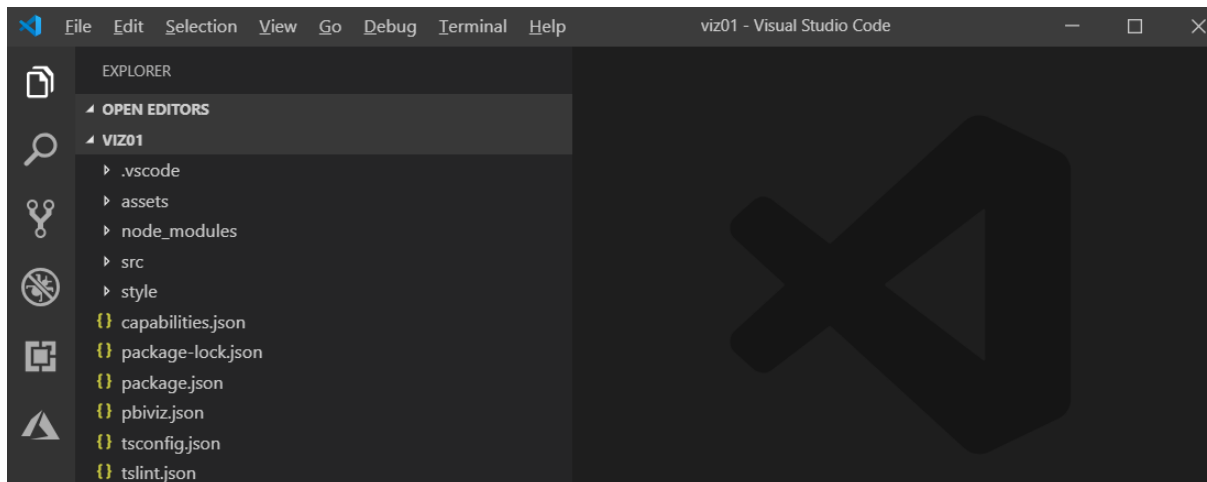
2. Open project using Visual Studio Code.

- a) In the Node.js command prompt, type and execute **cd viz01** to make the folder for this project the current directory  
b) In the Node.js command prompt, type and execute **code .** to open the project with Visual Studio Code.



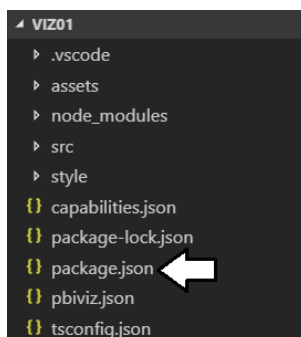
```
Node.js command prompt
c:\Student\CustomVisuals>cd viz01
c:\Student\CustomVisuals\viz01>code .
c:\Student\CustomVisuals\viz01>
```

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



3. Inspect the **package.json** file for your new project.

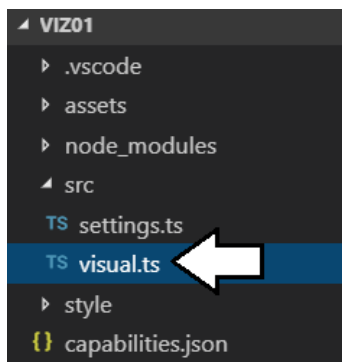
- a) Open the the **package.json** file.



- b) Inspect the **dependencies** and **devDependencies** sections to see what other packages have been installed.

```
1 package.json > ...
2 {
3   "name": "visual",
4   "scripts": {
5     "pbiviz": "pbiviz",
6     "start": "pbiviz start",
7     "package": "pbiviz package",
8     "lint": "tslint -r \"node_modules/tslint-microsoft-contrib\" \"
9   },
10  "dependencies": {
11    "@babel/runtime": "^7.4.5",
12    "@babel/runtime-corejs2": "^7.4.5",
13    "@types/d3": "5.5.0",
14    "d3": "5.5.0",
15    "powerbi-visuals-utils-dataviewutils": "^2.2.0",
16    "powerbi-visuals-api": "~2.6.0",
17    "core-js": "3.1.3"
18  },
19  "devDependencies": {
20    "ts-loader": "5.2.2",
21    "typescript": "3.0.1"
22  }
23 }
```

- c) After you have inspected **package.json**, close this file without saving any changes.
4. Inspect what's inside **visual.ts** to see the TypeScript code automatically added by **pbiviz** when creating a new custom visual.
- a) In Visual Studio Code, expand the **src** folder and locate file named **visual.ts** and double-click this file to open it.



- b) Look inside the source file **visual.ts** and inspect the pre-provided code for the class named **Visual**.

```
src > TS visual.ts > ...
1  "use strict";
2
3  import "core-js/stable";
4  import "../style/visual.less";
5  import powerbi from "powerbi-visuals-api";
6  import VisualConstructorOptions = powerbi.extensibility.visual.VisualConstructorOptions;
7  import VisualUpdateOptions = powerbi.extensibility.visual.VisualUpdateOptions;
8  import IVisual = powerbi.extensibility.visual.IVisual;
9  import EnumerateVisualObjectInstancesOptions = powerbi.EnumerateVisualObjectInstancesOptions;
10 import VisualObjectInstance = powerbi.VisualObjectInstance;
11 import DataView = powerbi.DataView;
12 import VisualObjectInstanceEnumerationObject = powerbi.VisualObjectInstanceEnumerationObject;
13
14 import { VisualSettings } from "../settings";
15 export class Visual implements IVisual {
16   private target: HTMLElement;
17   private updateCount: number;
```

- c) Delete all the existing code from **visual.ts** and replace it with the following code.

```
import powerbi from "powerbi-visuals-api";
import VisualConstructorOptions = powerbi.extensibility.visual.VisualConstructorOptions;
import VisualUpdateOptions = powerbi.extensibility.visual.VisualUpdateOptions;
import IVisual = powerbi.extensibility.visual.IVisual;

import "../style/visual.less"

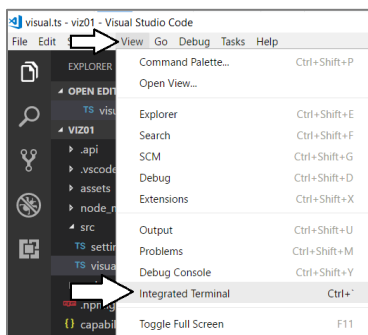
export class Visual implements IVisual {
    private target: HTMLElement;
    private updateCount: number;

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

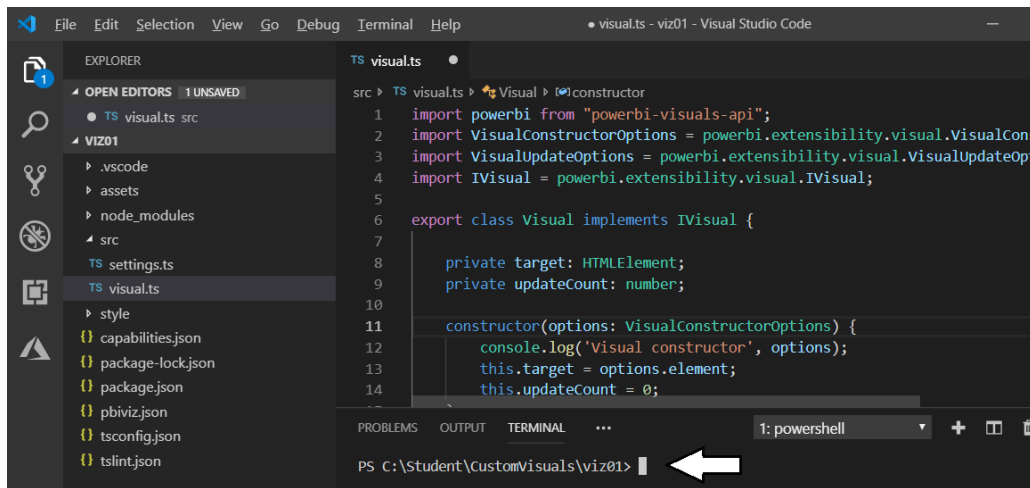
    public update(options: VisualUpdateOptions) {
        console.log('visual update', options);
        this.target.innerHTML = `<p>Update count: <em>${(this.updateCount++)}</em></p>`;
    }
}
```

If it's easier, you can copy and paste this code from **C:\Student\Modules\08\_CustomVisuals\Lab\Snippets\Exercise2-Visual-Starter.ts.txt**.

5. Open the **Integrated Terminal** so you can execute **pbviz** commands from within **Visual Studio Code**.
- a) In Visual Studio Code, select the **View > Integrated Terminal** command to display the Integrated Terminal.



You should now see a command line prompt within Visual Studio Code.

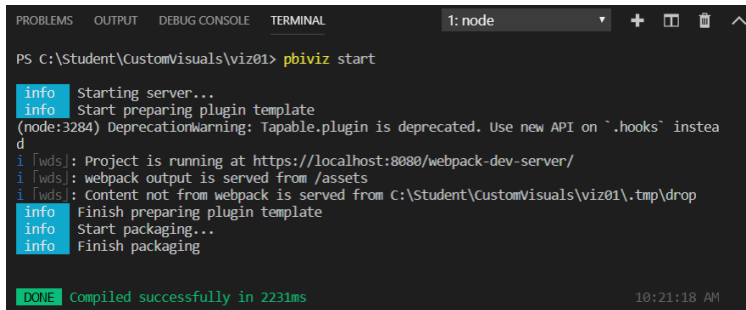




6. Use the **start** command of the **pbviz** utility to start a debugging session for the **viz01** custom visual project.
  - a) Place your cursor at the command prompt in the Integrated Terminal.
  - b) Type and execute the following command to start a debugging session for the **viz01** project.

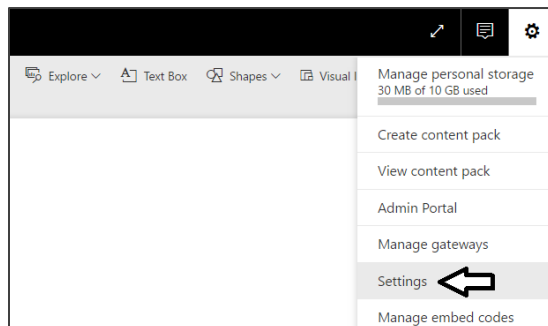
```
pbviz start
```

- 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**.

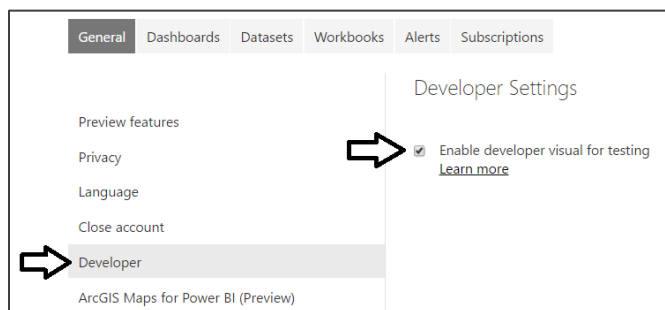


```
PS C:\Student\CustomVisuals\viz01> pbviz start
info Starting server...
info Start preparing plugin template
(node:3284) DeprecationWarning: Tapable.plugin is deprecated. Use new API on `hooks` instead
i [wds]: Project is running at https://localhost:8080/webpack-dev-server/
i [wds]: webpack output 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
DONE Compiled successfully in 2231ms
```

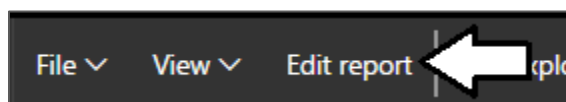
7. Test visual by loading it into the Power BI environment.
  - a) Log into our personal workspace at <https://app.powerbi.com>.
  - b) Once you have logged into your personal workspace, drop down the Power BI Setting menu (*it's the one with the gear icon*) and then select the **Settings** menu command as shown in the following dialog.



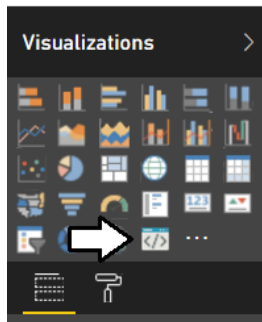
- c) Select **Developer** on the left and then select the **Enable developer visual for testing** checkbox.



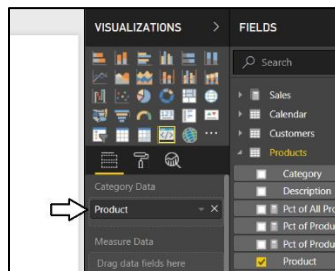
- d) Now navigate to any report in your workspace. It doesn't matter what report you use as long as you have a report.
  - e) Move the report into edit mode by clicking the **Edit report** button.



- f) Add a new page to the report so you have a blank report page to work with.
- g) Add an instance of the **Developer Visual** to the page.

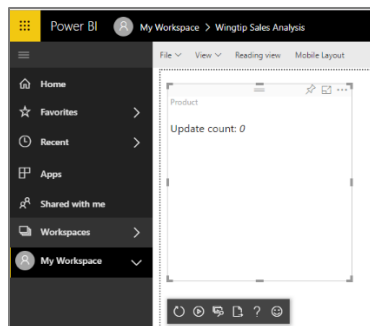


- h) Select the developer visual and then add a field into the **Category Data** well in the **Fields** pane.

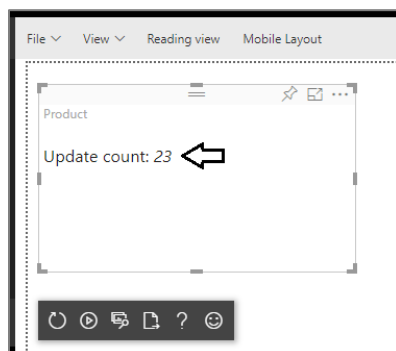


It doesn't really matter which field you add. However, you must add at least one field for the visual rendering to occur properly.

- i) You should now see the custom visual for **viz01** on the report page render inside the developer visual.



- j) 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.



8. Modify the code in the **update** function of the custom visual.

- a) Return to Visual Studio Code.
- b) Open the source file **visual.ts**.
- c) Inside **visual.ts**, locate the **update** method which should currently match the following code listing.

```
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.

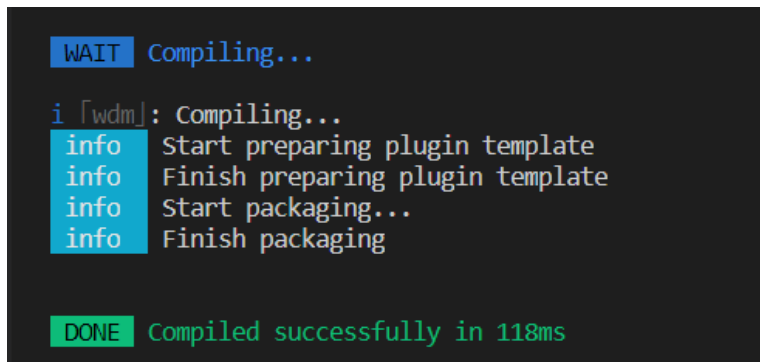
```
public update(options: VisualUpdateOptions) {  
    console.log('Visual update', options);  
    this.target.innerHTML =  
        `



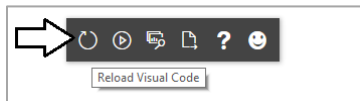

```

You can copy and paste this code from **C:\Student\Modules\08\_CustomVisuals\Lab\Snippets\Exercise2-Replace-Update-Function.ts.txt**.

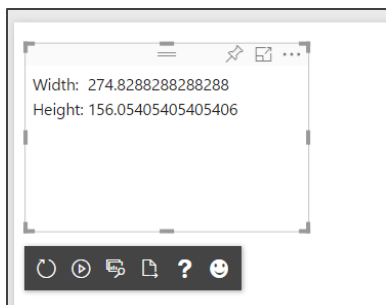
- e) After you have updated the **update** method, save your changes to the file **visual.ts**.
- f) When you save **visual.ts**, you will notice that the Node.js command prompt will run to recompile the code for your visual.



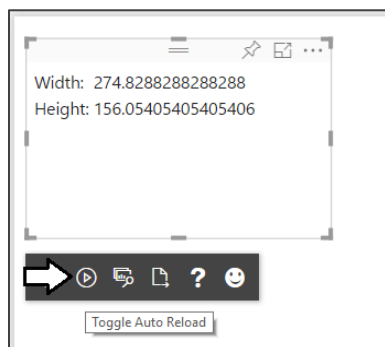
- g) Return to the browser and the page where you instantiated the custom visual for testing.
- h) In the visual developer panel, click the **Reload Visual Code** button as shown in the following screenshot.



- 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.



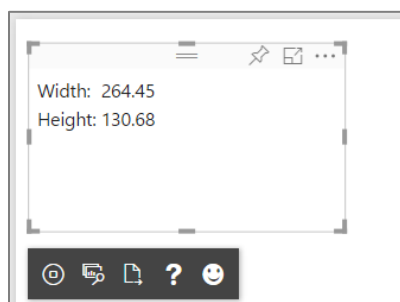
9. Make another change to the code in your custom visual.

- 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 id='myTable'>  
          <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>`;  
}
```

You can copy & paste this from **C:\Student\Modules\08\_CustomVisuals\Lab\Snippets\Exercise2-Replace-Update-Function-Part2.ts.txt**.

- Save your changes to **visual.ts**.
- Return to the browser and you should see the effect of your changes in that the values now show only two points of precision.



- 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.

10. Stop the visual debugging session.

- Return to Node.js command prompt.
- Hold down the **Ctrl** key on the keyboard and then press **C** to interrupt the Node.js debugging session.
- When prompted to terminate the session, type **Y** and press **ENTER**.

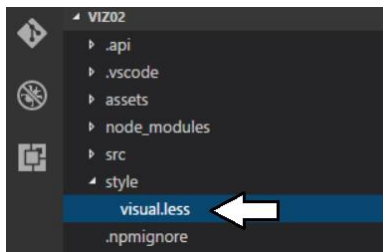
```
[2] ../tmp/precompile/visualPlugin.ts 788 bytes {0} [built]
[3] external "Function('return this')()" 42 bytes {0} [built]
i | wdm | Compiled successfully.

Terminate batch job (Y/N)? y
PS C:\Student\CustomVisuals\viz01>
```

Now you have created and tested a simple custom visual project. Next you will create another custom visual project that uses jQuery.

11. Add some CSS styles to your visual.

- Expand the style folder and open the file named **visual.less**.



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

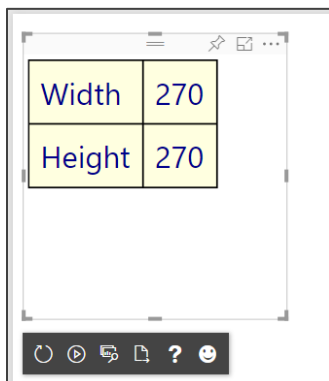
```
#myTable{
  background-color: black;

  td{
    background-color: lightyellow;
    color:darkblue;
    padding: 8px;
  }
}
```

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

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

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



12. Add scaling font behavior to your visual.

- Return to Visual Studio Code.
- Navigate to the **visual.ts** file and locate the **update** method.
- Copy and paste the following code into the **update**.

```
public update(options: VisualUpdateOptions) {  
    console.log('Visual update', options);  
  
    var scaledFontSizeWidth: number = Math.round(options.viewport.width / 8);  
    var scaledFontSizeHeight: number = Math.round(options.viewport.height / 5);  
    var scaledFontSize: number = Math.min(...[scaledFontSizeWidth, scaledFontSizeHeight]);  
    var scaledFontSizeCss: string = scaledFontSize + "px";  
  
    this.target.innerHTML =  
        `



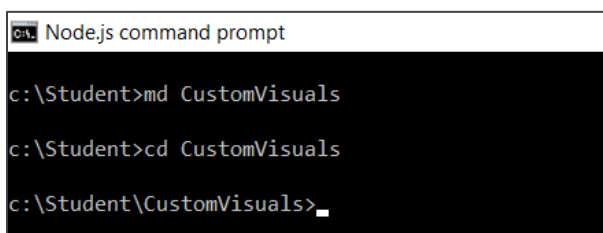

```

### Exercise 3: 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.

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

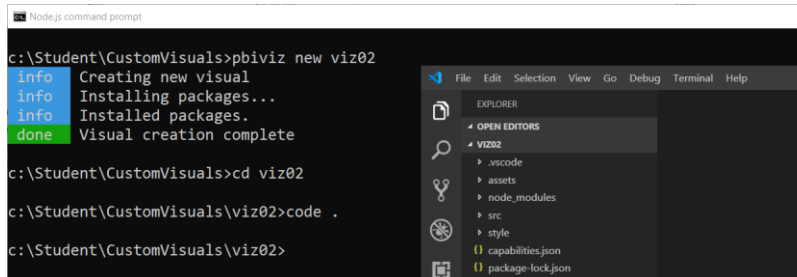
```
cd ..
```



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

```
pbiviz new viz02
cd viz02
code .
```

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



2. Open the package.json file and see what's inside.

```
{
  "name": "visual",
  "scripts": {
    "pbiviz": "pbiviz",
    "start": "pbiviz start",
    "package": "pbiviz package",
    "lint": "tslint -r \"node_modules/tslint-microsoft-contrib\" \",
  },
  "dependencies": {
    "@babel/runtime": "^7.4.5",
    "@babel/runtime-corejs2": "^7.4.5",
    "@types/d3": "5.5.0",
    "d3": "5.5.0",
    "powerbi-visuals-utils-dataviewutils": "^2.2.0",
    "powerbi-visuals-api": "^2.6.0",
    "core-js": "3.1.3"
  },
  "devDependencies": {
    "ts-loader": "5.2.2",
    "typescript": "3.0.1"
  }
}
```

There is no need to add the Node.js package for d3 and @types/d3 because they have been automatically added by the template used to create a new custom visual project.

- a) Close package.json without saving any changes.
3. Open the Integrated Terminal so you can execute command-line instructions from within Visual Studio Code.
- a) Inside Visual Studio Code, select the **View > Terminal** menu command.
- b) You should see the **Terminal** window with a command prompt inside Visual Studio Code.

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

4. 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.

```
import powerbi from "powerbi-visuals-api";
import VisualConstructorOptions = powerbi.extensibility.visual.VisualConstructorOptions;
import VisualUpdateOptions = powerbi.extensibility.visual.VisualUpdateOptions;
import IVisual = powerbi.extensibility.visual.IVisual;
import * as d3 from "d3";

export class Visual implements IVisual {

  constructor(options: VisualConstructorOptions) { }

  public update(options: VisualUpdateOptions) { }

}
```

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

```
export class Visual implements IVisual {  
  
    private svgRoot: d3.Selection<SVGElement, {}, HTMLElement, any>;  
    private ellipse: d3.Selection<SVGElement, {}, HTMLElement, any>;  
    private text: d3.Selection<SVGElement, {}, HTMLElement, any>;  
    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);  
}
```

You can copy & paste all this code from **C:\Student\Modules\08\_CustomVisuals\Lab\Snippets\Exercise4-Visual-Starter.ts.txt**

5. 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 and then add a field into the **Data Category** well inside the **Fields** pane.  
e) You should now see your custom visual render on the page and it should match the following screenshot.



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



Congratulations. You have now completed this lab.