Getting Up and Running with the Power BI Developer Tools

Setup Time: 60 minutes

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

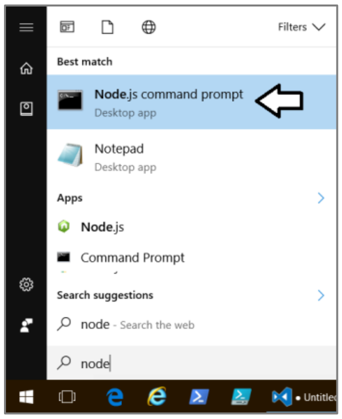
Overview: In this lab, you will begin be installing the Power BI Custom Visual Tool (PBIVIZ). After that, you will use Node.js, the PBIVIZ utility and Visual Studio Code to create and debug custom visual projects within the context of a Power BI workspace. You will also learn how to configure custom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

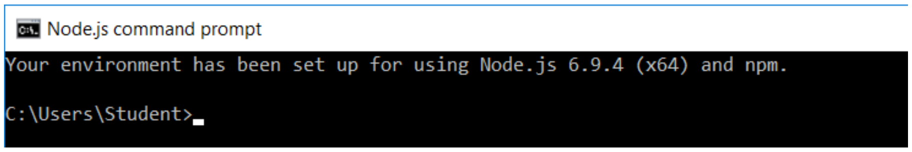
Exercise 1: Prepare Your PC for Developing Custom Visuals

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 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.

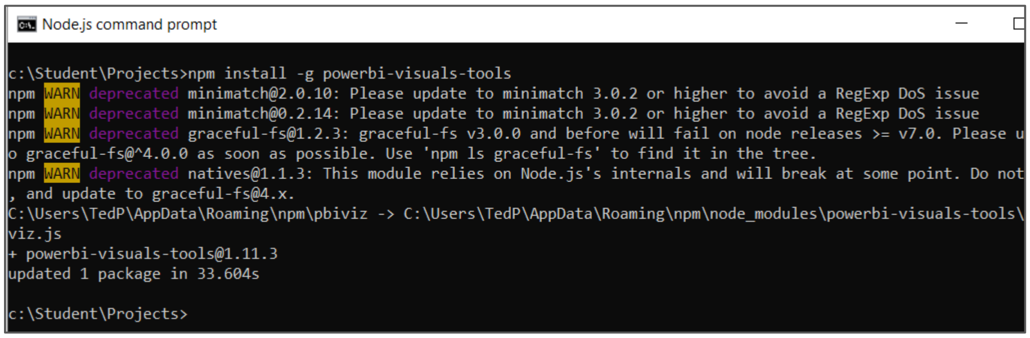


```
Your environment has been set up for using Node.js 6.9.4 (x64) and npm.

C:\Users\Student>
```

   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.

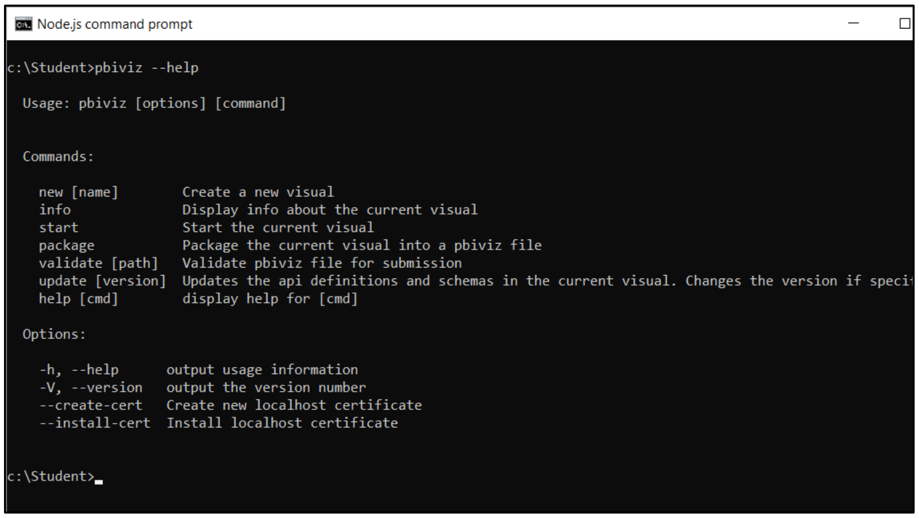


```
c:\Student\Projects>npm install -g powerbi-visuals-tools
npm WARN deprecated minimatch@2.0.10: Please update to minimatch 3.0.2 or higher to avoid a RegExp DoS issue
npm WARN deprecated minimatch@0.2.14: Please update to minimatch 3.0.2 or higher to avoid a RegExp DoS issue
npm WARN deprecated graceful-fs@1.2.3: graceful-fs v3.0.0 and before will fail on node releases >= v7.0. Please u
o graceful-fs@^4.0.0 as soon as possible. Use 'npm ls graceful-fs' to find it in the tree.
npm WARN deprecated natives@1.1.3: This module relies on Node.js's internals and will break at some point. Do not
, and update to graceful-fs@4.x.
C:\Users\TedP\AppData\Roaming\npm\pbiviz -> C:\Users\TedP\AppData\Roaming\npm\node_modules\powerbi-visuals-tools\
viz.js
+ powerbi-visuals-tools@1.11.3
updated 1 package in 33.604s

c:\Student\Projects>
```

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 **pbiviz** utility.

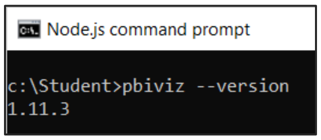


```
c:\Student>pbiviz --help

  Usage: pbiviz [options] [command]


  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
    validate [path]   Validate pbiviz file for submission
    update [version]  Updates the api definitions and schemas in the current visual. Changes the version if speci
    help [cmd]        display help for [cmd]

  Options:

    -h, --help      output usage information
    -V, --version   output the version number
    --create-cert   Create new localhost certificate
    --install-cert  Install localhost certificate


c:\Student>
```

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

```
pbiviz --version
```

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

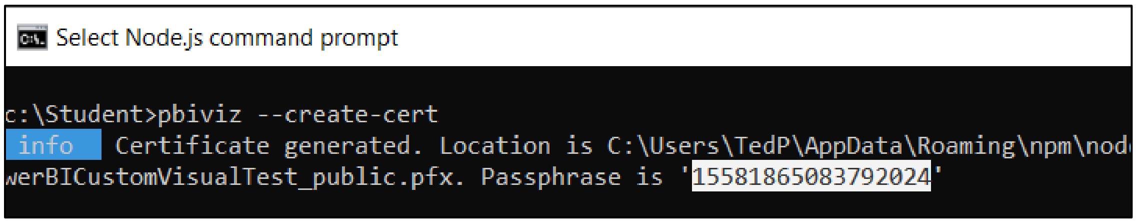


```
c:\Student>pbiviz --version
1.11.3
```

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.

```
pbiviz -create-cert
```

   b) The command should generate a new certificate and print out a passphrase to the console window.



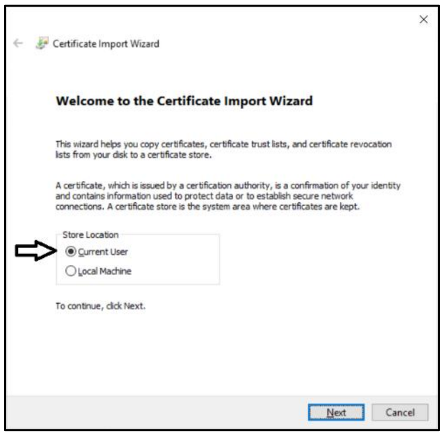
```
c:\Student>pbiviz --create-cert
 info   Certificate generated. Location is C:\Users\TedP\AppData\Roaming\npm\nod
werBICustomVisualTest_public.pfx. Passphrase is '15581865083792024'
```

   c) Copy the passphrase to the windows clipboard. You will need to paste this passphrase value into a dialog in a later step.
   d) In the Node.js command prompt, type and execute the following command to begin the certificate installation process.

```
pbiviz --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 through pages to complete the process of installing the self-signed SSL certificate for **localhost**.

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



Welcome to the Certificate Import Wizard

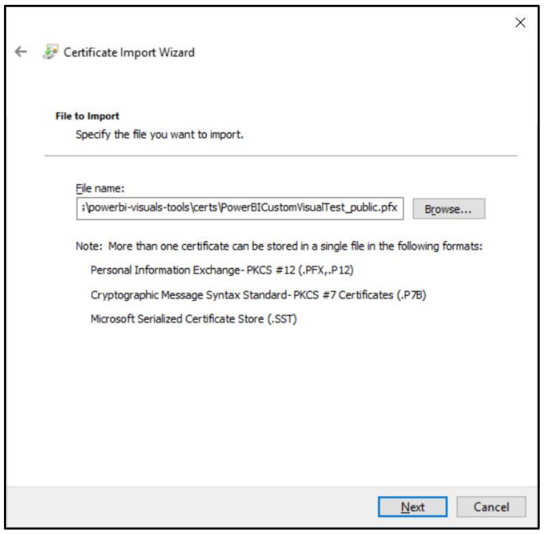
This wizard helps you copy certificates, certificate trust lists, and certificate revocation lists from your disk to a certificate store.

A certificate, which is issued by a certification authority, is a confirmation of your identity and contains information used to protect data or to establish secure network connections. A certificate store is the system area where certificates are kept.

Store Location
(●) Current User
( ) Local Machine

To continue, click Next.

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

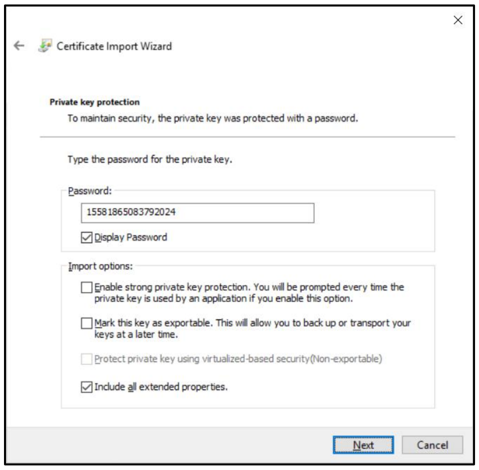


File to Import
Specify the file you want to import.

File name:
:\powerbi-visuals-tools\certs\PowerBICustomVisualTest_public.pfx

Note: More than one certificate can be stored in a single file in the following formats:
Personal Information Exchange- PKCS #12 (.PFX,.P12)
Cryptographic Message Syntax Standard- PKCS #7 Certificates (.P7B)
Microsoft Serialized Certificate Store (.SST)

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



Private key protection
To maintain security, the private key was protected with a password.

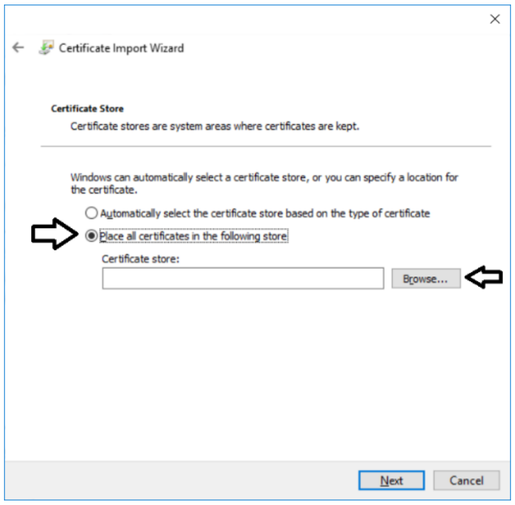
Type the password for the private key.

Password:
15581865083792024
[✓] Display Password

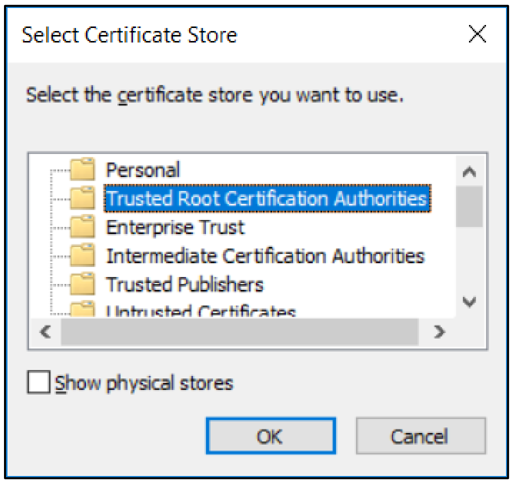
Import options:
[ ] Enable strong private key protection. You will be prompted every time the private key is used by an application if you enable this option.
[ ] Mark this key as exportable. This will allow you to back up or transport your keys at a later time.
[ ] Protect private key using virtualized-based security(Non-exportable)
[✓] Include all extended properties.

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.

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

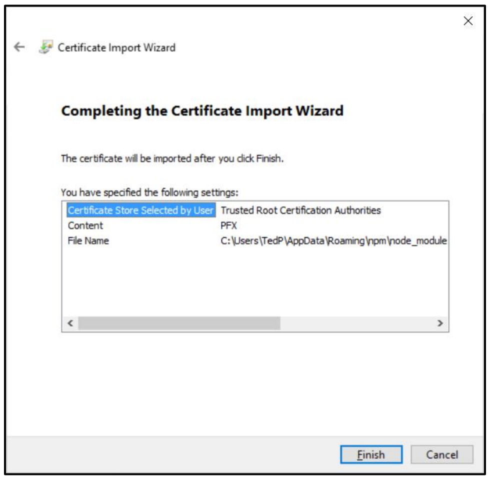

Certificate Import Wizard
Certificate Store
Certificate stores are system areas where certificates are kept.
Windows can automatically select a certificate store, or you can specify a location for the certificate.
Automatically select the certificate store based on the type of certificate
Place all certificates in the following store
Certificate store:
Browse...
Next
Cancel

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


Select Certificate Store
Select the certificate store you want to use.
Personal
Trusted Root Certification Authorities
Enterprise Trust
Intermediate Certification Authorities
Trusted Publishers
Untrusted Certificates
Show physical stores
OK
Cancel

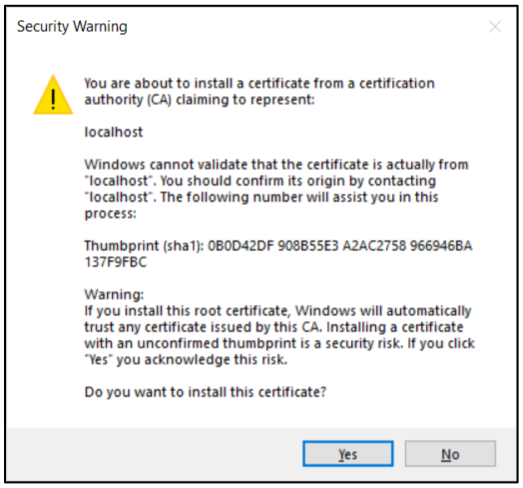
j) On the **Certificate Store** page, click the **Next** button to continue.

k) On the **Completing the Certification Import Wizard** page, click **Finish**.


Certificate Import Wizard
Completing the Certificate Import Wizard
The certificate will be imported after you click Finish.
You have specified the following settings:
Finish
Cancel

Certificate Store Selected by User	Trusted Root Certification Authorities
Content	PFX
File Name	C:\Users\TedP\AppData\Roaming\npm\node_module

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



You are about to install a certificate from a certification authority (CA) claiming to represent:

localhost

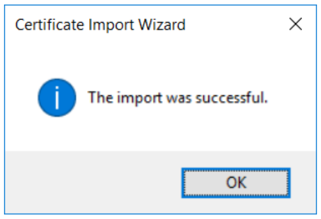
Windows cannot validate that the certificate is actually from "localhost". You should confirm its origin by contacting "localhost". The following number will assist you in this process:

Thumbprint (sha1): 0B0D42DF 908B55E3 A2AC2758 966946BA 137F9FBC

Warning:
If you install this root certificate, Windows will automatically trust any certificate issued by this CA. Installing a certificate with an unconfirmed thumbprint is a security risk. If you click "Yes" you acknowledge this risk.

Do you want to install this certificate?

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



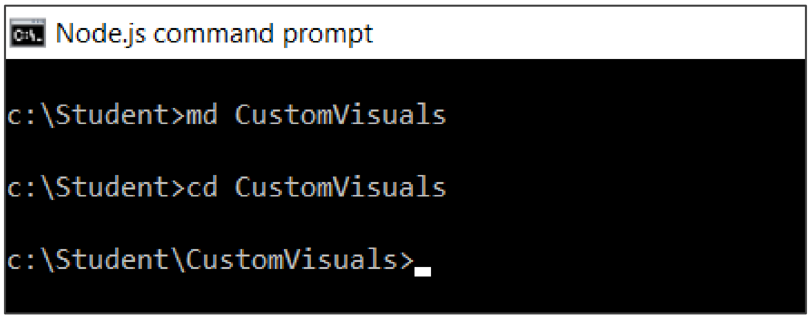
The import was successful.

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.

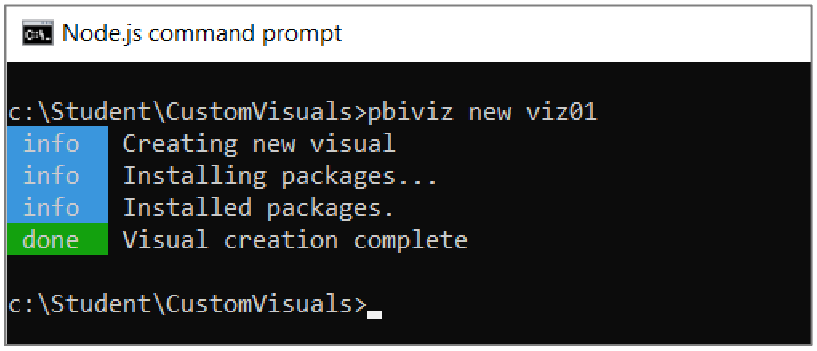


```
c:\Student>md CustomVisuals

c:\Student>cd CustomVisuals

c:\Student\CustomVisuals>_
```

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

```
pbiviz new viz01
```

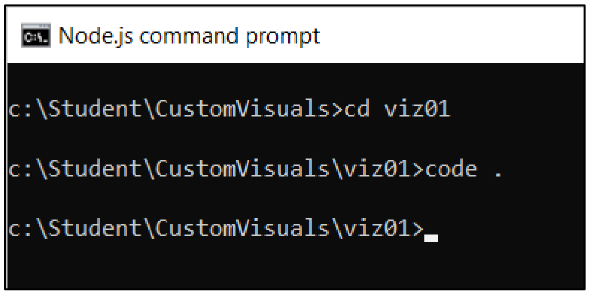
f) You should see output in the console window that the visual has been created successfully.



Node.js command prompt

```
c:\Student\CustomVisuals>pbiviz new viz01
 info   Creating new visual
 info   Installing packages...
 info   Installed packages.
 done   Visual creation complete

c:\Student\CustomVisuals>_
```

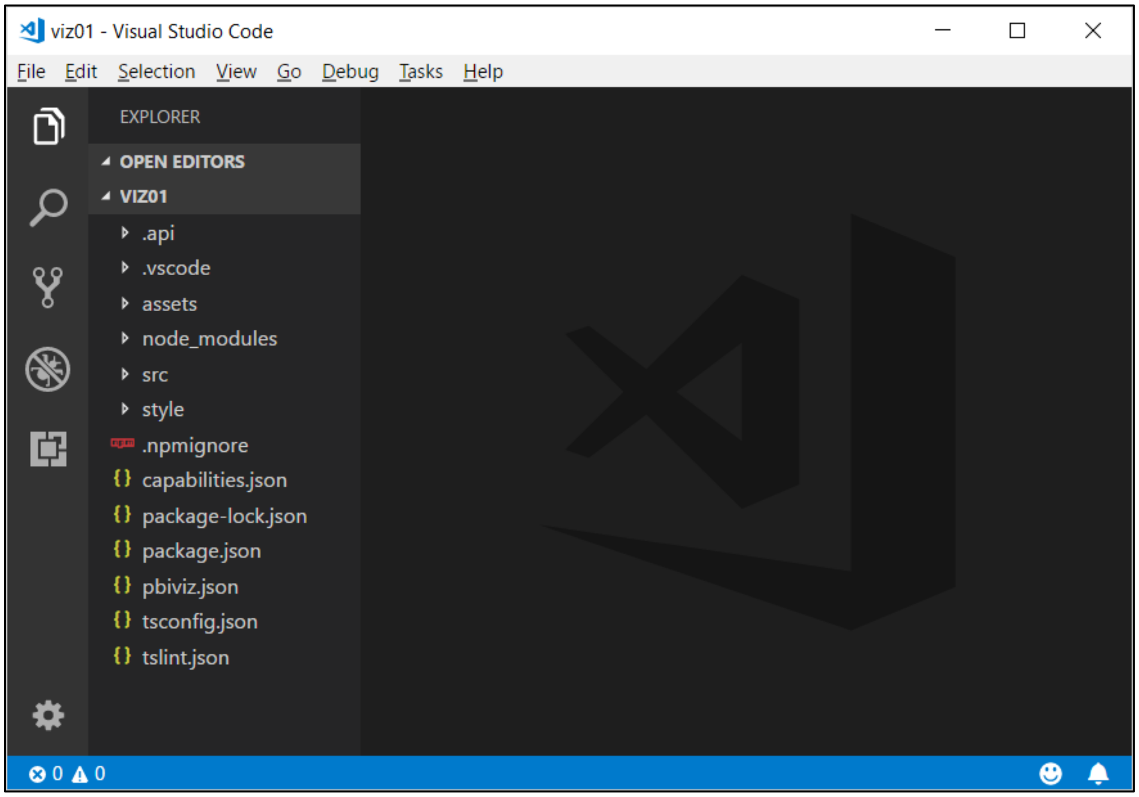
2. Open project using Visual Studio Code.
   a) In the Node.js command prompt, type and execute **cd viz01** to make the folder for this project the current directory
   b) In the Node.js command prompt, type and execute **code .** to open the project with Visual Studio Code.



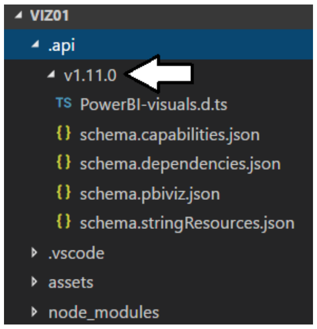
Node.js command prompt

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

c:\Student\CustomVisuals\viz01>code .

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

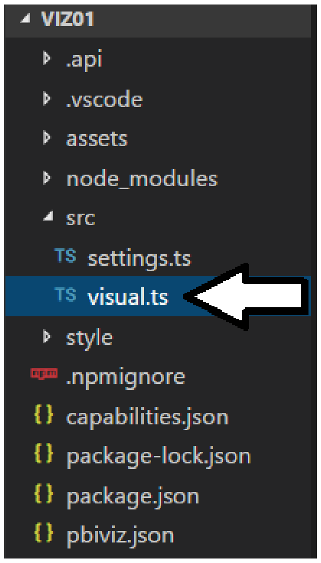
   c) Visual Studio Code should start and open the root folder of the **viz01** project.



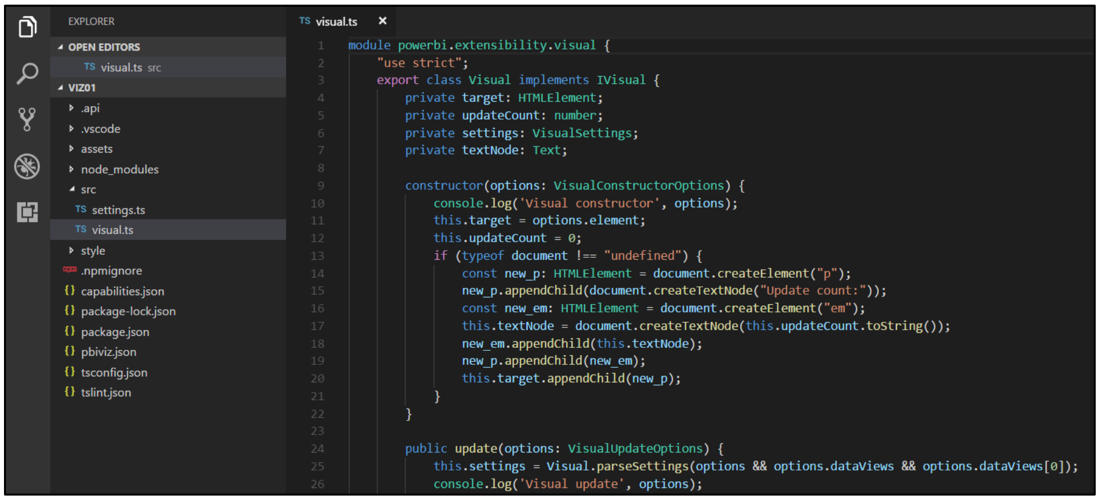
3. Inspect the API version of the project.
   a) Expand the **.api** node. You should be able to determine what version of the API has been used in your new project.



   b) Collapse the **.api** node.
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**.



```
module powerbi.extensibility.visual {
    "use strict";
    export class Visual implements IVisual {
        private target: HTMLElement;
        private updateCount: number;
        private settings: VisualSettings;
        private textNode: Text;

        constructor(options: VisualConstructorOptions) {
            console.log('Visual constructor', options);
            this.target = options.element;
            this.updateCount = 0;
            if (typeof document !== "undefined") {
                const new_p: HTMLElement = document.createElement("p");
                new_p.appendChild(document.createTextNode("Update count:"));
                const new_em: HTMLElement = document.createElement("em");
                this.textNode = document.createTextNode(this.updateCount.toString());
                new_em.appendChild(this.textNode);
                new_p.appendChild(new_em);
                this.target.appendChild(new_p);
            }
        }

        public update(options: VisualUpdateOptions) {
            this.settings = Visual.parseSettings(options && options.dataViews && options.dataViews[0]);
            console.log('Visual update', options);
```

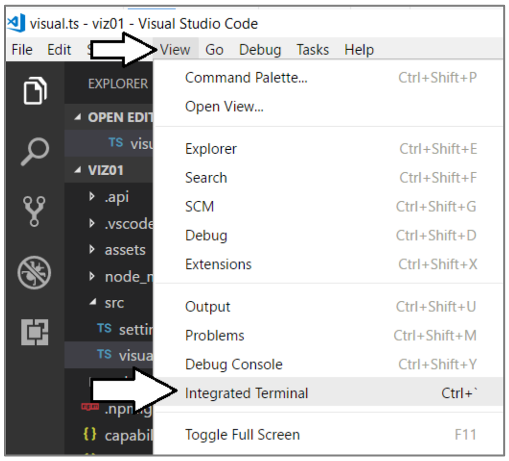
c) Delete all the existing code from **visual.ts** and replace it with the following code.

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

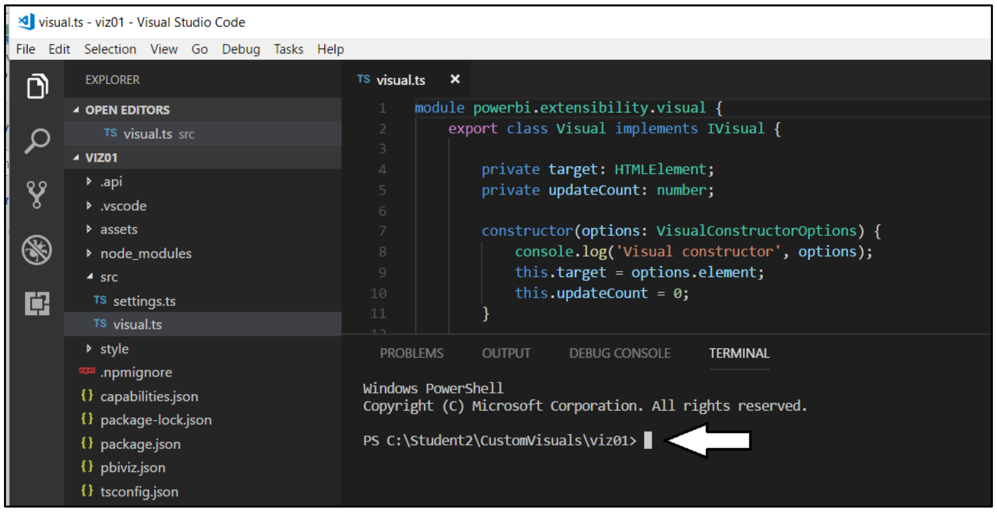
If it's easier, you can copy and paste this code from **C:\Student\Modules\05_PBIVIZ\Lab\Snippets\Exercise2-Visual-Starter.ts.txt**.

5. Open the **Integrated Terminal** so you can execute **pbiviz** 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 comand line prompt within Visual Studio Code.



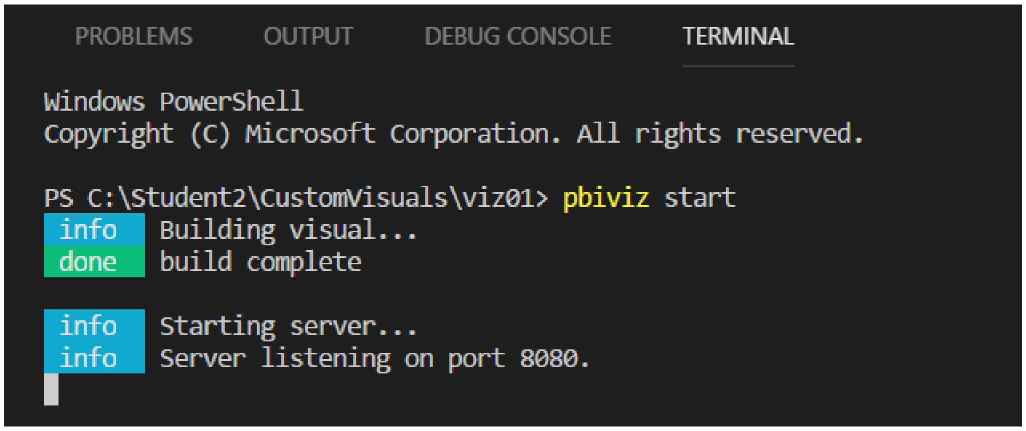
```
module powerbi.extensibility.visual {
    export class Visual implements IVisual {

        private target: HTMLElement;
        private updateCount: number;

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

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

PS C:\Student2\CustomVisuals\viz01>
```

6. Use the **start** command of the **pbiviz** 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.

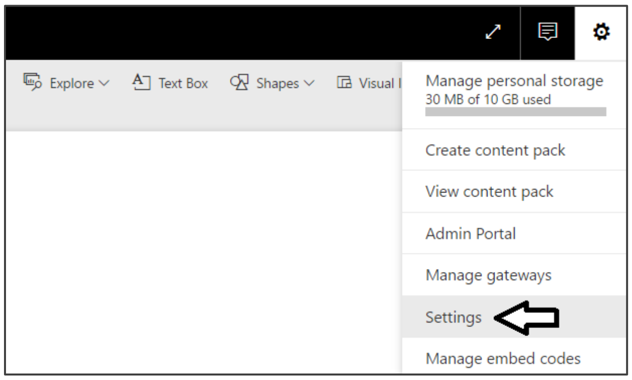
```
pbiviz start
```

   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**.

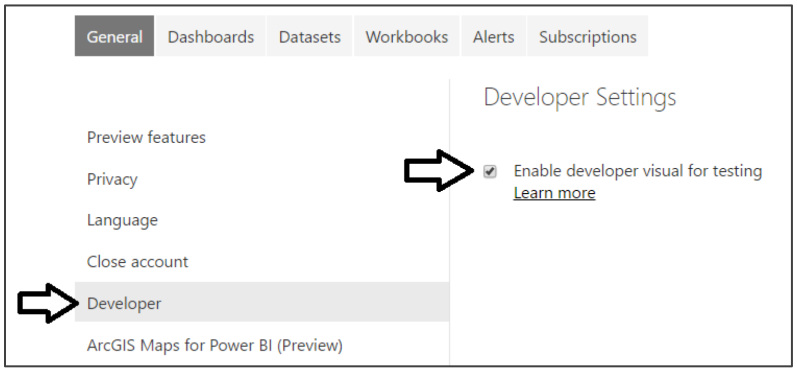


```
Windows PowerShell
Copyright (C) Microsoft Corporation. All rights reserved.

PS C:\Student2\CustomVisuals\viz01> pbiviz start
 info  Building visual...
 done  build complete

 info  Starting server...
 info  Server listening on port 8080.
```

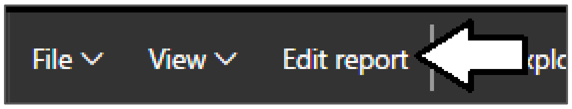
7. Test visual by loading it into the Power BI environment.
   a) Log into our personal workspace at https://app.powerbi.com.
   b) Once you have logged into your personal workspace, drop down the Power BI Setting menu (*it's the one with the gear icon*) and then select the **Settings** menu command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



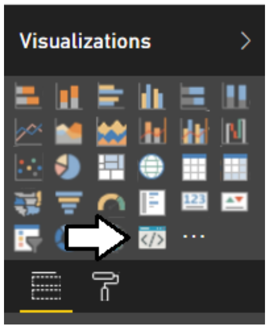
   c) Select **Developer** on the left and then select the **Enable developer visual for testing** checkbox.



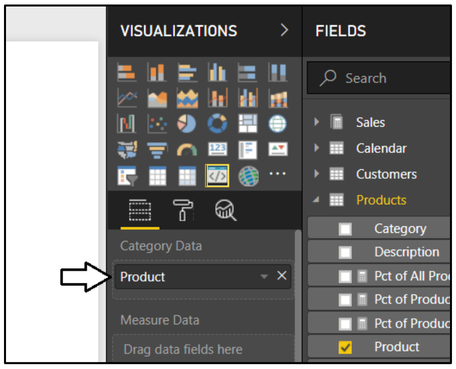
d) Now navigate to any report in your workspace. It doesn't matter what report you use as long as you have a report.
e) Move the report into edit mode by clicking the **Edit report** button.



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.

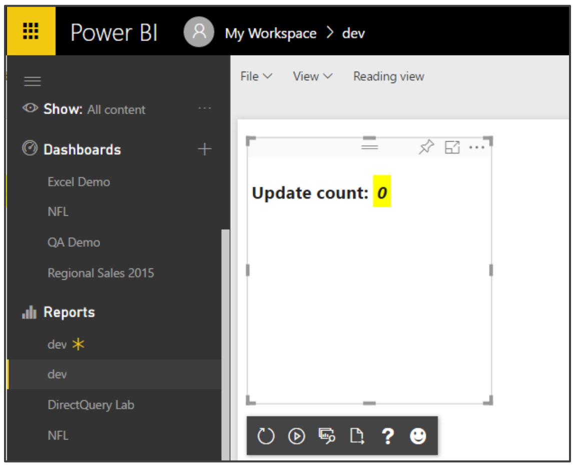


h) Select the developer visual and then add a field into the **Category Data** well in the **Fields** pane.



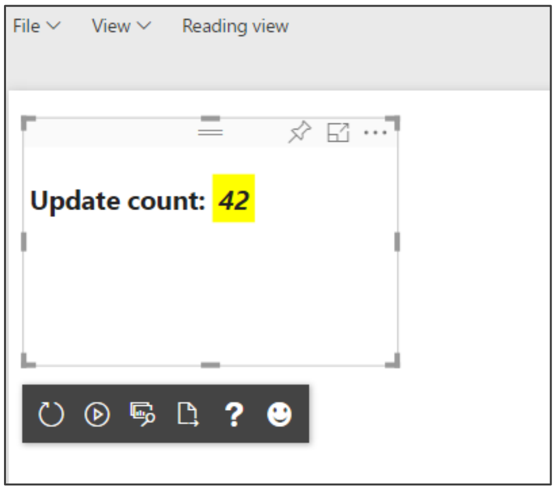
It doesn't really matter which field you add. However, you must add at least one field for the visual rendering to occur properly.

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



Update count: 0

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.



Update count: *42*

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

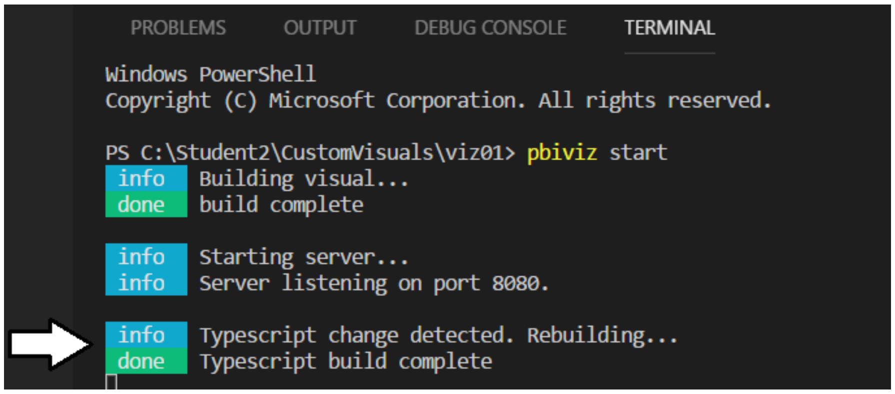
d) Replace the code in the **update** method by copying and pasting the following code.

```
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>`;
}
```

You can copy and paste this code from **C:\Student\Modules\05_PBIVIZ\Lab\Snippets\Exercise2-Replace-Update-Function.ts.txt**.

e) After you have updated the **update** method, save your changes to the file **visual.ts**.

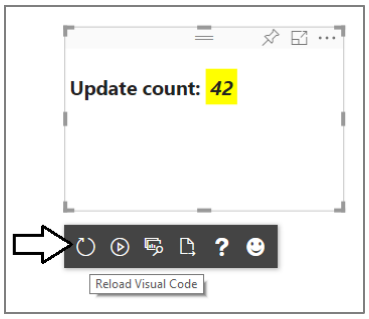
f) When you save **visual.ts**, you will notice that the Node.js command prompt will run to recompile the code for your visual.



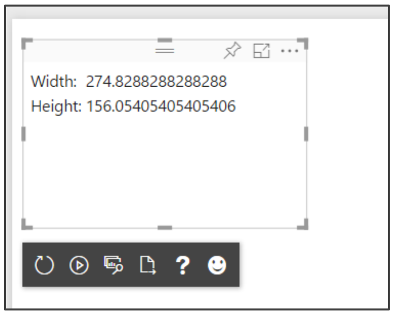
```
Windows PowerShell
Copyright (C) Microsoft Corporation. All rights reserved.

PS C:\Student2\CustomVisuals\viz01> pbiviz 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 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.

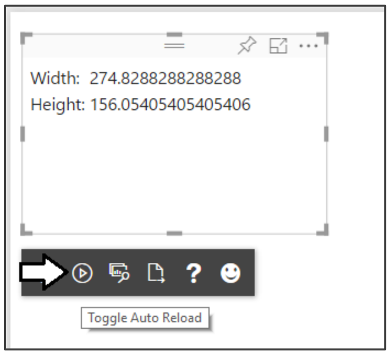


i) You should now see that the visual output as shown in the following screenshot which shows the visual width and height.



Width: 274.8288288288288
Height: 156.05405405405406

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



Width: 274.8288288288288
Height: 156.05405405405406

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

a) Return to Visual Studio Code.

b) Move down to the **update** method.

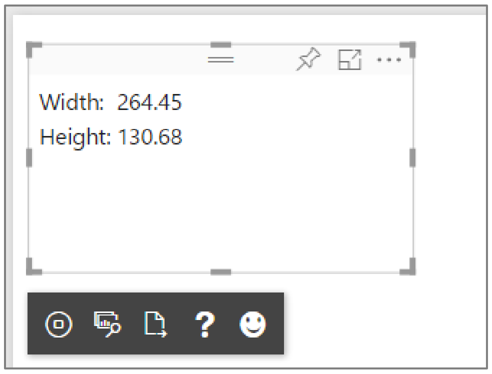
c) Modify the code that retrieve the visual width and height by adding the **toFixed(2)** method to configure the output to only show two digits of accuracy after the decimal.

```
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>`;
}
```

You can copy & paste this from **C:\Student\Modules\05_PBIVIZ\Lab\Snippets\Exercise2-Replace-Update-Function-Part2.ts.txt**.

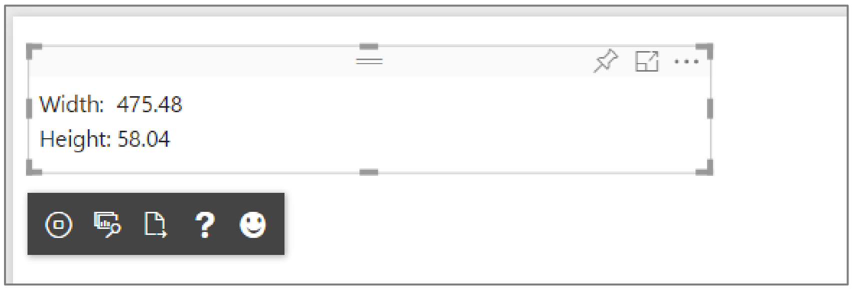
d) Save your changes to **visual.ts**.

e) Return to the browser and you should see the effect of your changes in that the values now show only two points of precision.



Width: 264.45
Height: 130.68

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



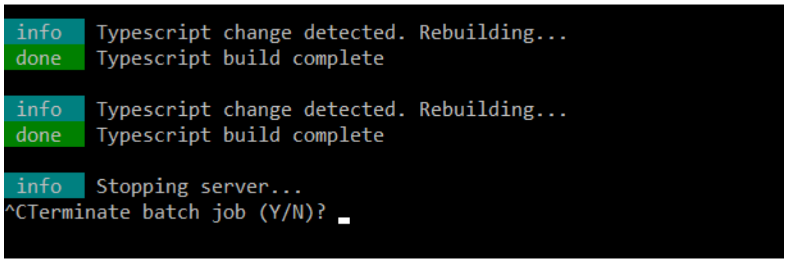
Width: 475.48
Height: 58.04

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

10. Stop visual debugging session.

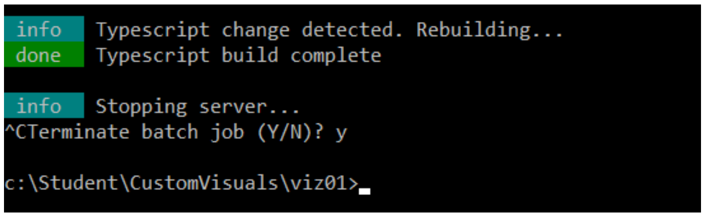
a) Return to Node.js command prompt.

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)?
```

c) 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>
```

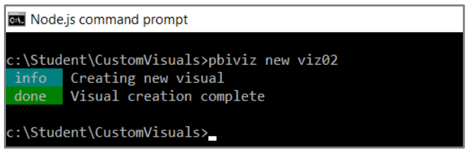
Now you have created and tested a simple custom visual project. Next you will create another custom visual project that uses jQuery.

Exercise 3: Create a Custom Visual using jQuery

In this exercise, you will create a new visual named **viz02** which uses the jQuery library 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 ..
```

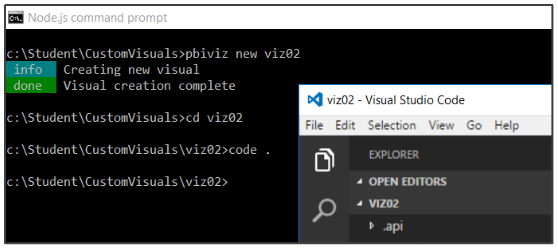


```
c:\Student\CustomVisuals>pbiviz new viz02
 info   Creating new visual
 done   Visual creation complete

c:\Student\CustomVisuals>
```

   c) Type and execute the following 3 commands to create a new visual project named **viz02** and open it in Visual Studio Code.

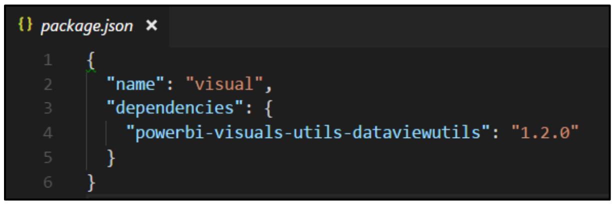
```
pbiviz new viz02
cd viz02
Code .
```

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



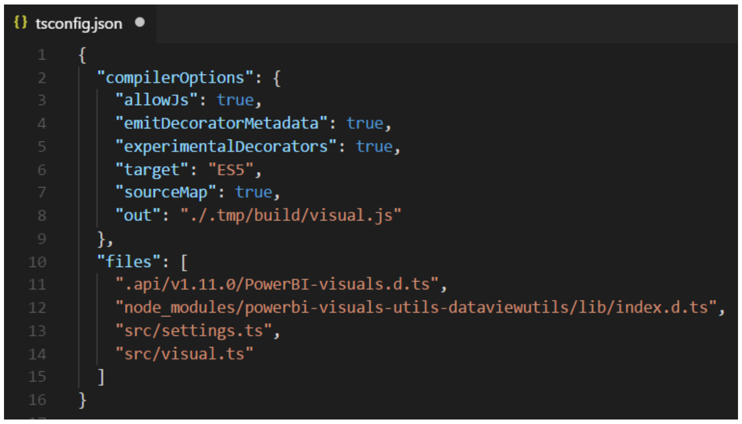
```
c:\Student\CustomVisuals>pbiviz new viz02
 info   Creating new visual
 done   Visual creation complete

c:\Student\CustomVisuals>cd viz02

c:\Student\CustomVisuals\viz02>code .

c:\Student\CustomVisuals\viz02>
```

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



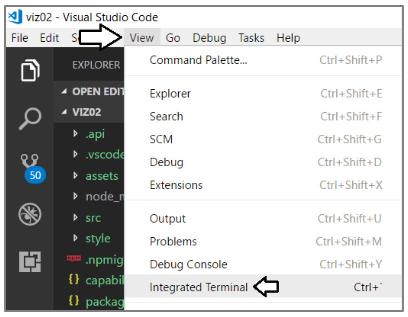
```
{
  "name": "visual",
  "dependencies": {
    "powerbi-visuals-utils-dataviewutils": "1.2.0"
  }
}
```

   f) Open the **tsconfig.json** file and inspect its contents. Take note of the files listed in the **files** section.

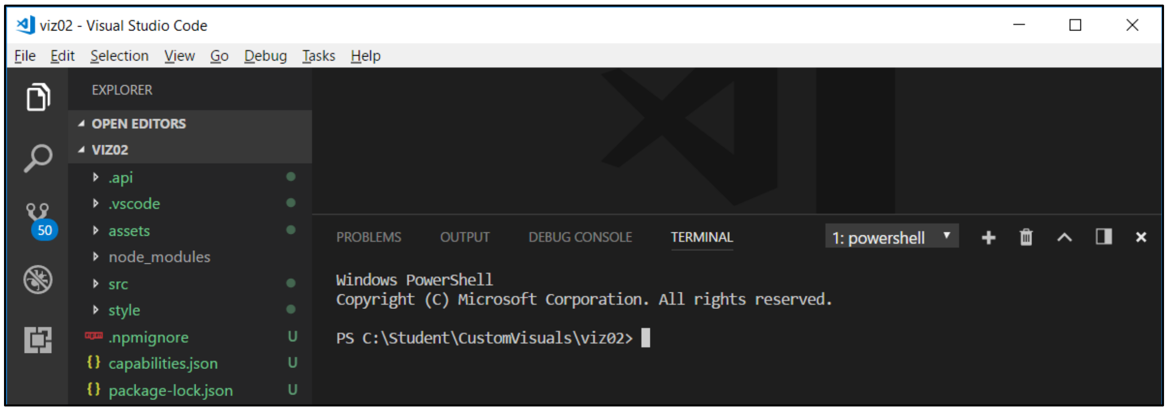


```
{
  "compilerOptions": {
    "allowJs": true,
    "emitDecoratorMetadata": true,
    "experimentalDecorators": true,
    "target": "ES5",
    "sourceMap": true,
    "out": "./.tmp/build/visual.js"
  },
  "files": [
    ".api/v1.11.0/PowerBI-visuals.d.ts",
    "node_modules/powerbi-visuals-utils-dataviewutils/lib/index.d.ts",
    "src/settings.ts",
    "src/visual.ts"
  ]
}
```

2. Open the Integrated Terminal so you can execute command-line instructions from within Visual Studio Code.
   a) Inside Visual Studio Code, select the **View > Integrated Terminal** menu command.


viz02 - Visual Studio Code
File Edit S View Go Debug Tasks Help
Command Palette... Ctrl+Shift+P
Explorer Ctrl+Shift+E
Search Ctrl+Shift+F
SCM Ctrl+Shift+G
Debug Ctrl+Shift+D
Extensions Ctrl+Shift+X
Output Ctrl+Shift+U
Problems Ctrl+Shift+M
Debug Console Ctrl+Shift+Y
Integrated Terminal Ctrl+`

   b) You should see the **Integrated Terminal** window with a command prompt inside Visual Studio Code.

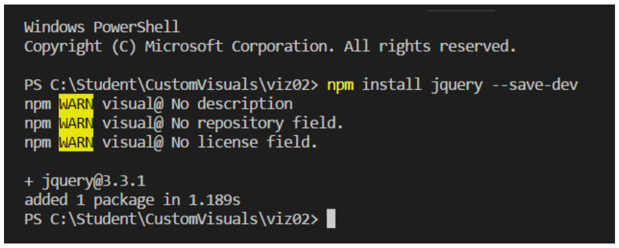


```
Windows PowerShell
Copyright (C) Microsoft Corporation. All rights reserved.

PS C:\Student\CustomVisuals\viz02>
```

3. Add support for the jQuery library into the **viz02** project.
   a) Return to the Node.js command prompt in the Integrated Terminal.
   b) Type and execute the following command.

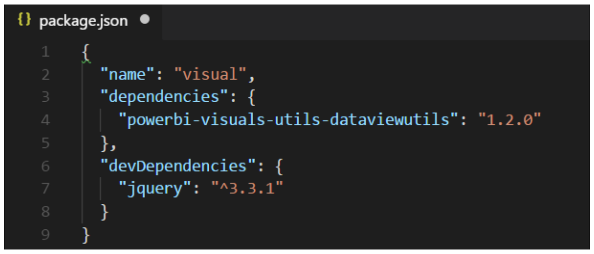
```
npm install jquery --save-dev
```

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



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

PS C:\Student\CustomVisuals\viz02> npm install jquery --save-dev
npm WARN visual@ No description
npm WARN visual@ No repository field.
npm WARN visual@ No license field.

+ jquery@3.3.1
added 1 package in 1.189s
PS C:\Student\CustomVisuals\viz02>
```

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

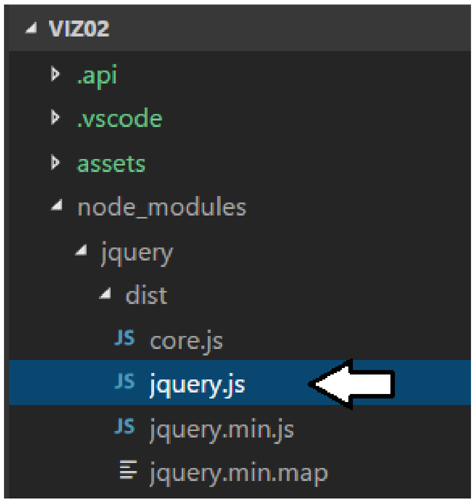


```
{
  "name": "visual",
  "dependencies": {
    "powerbi-visuals-utils-dataviewutils": "1.2.0"
  },
  "devDependencies": {
    "jquery": "^3.3.1"
  }
}
```

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

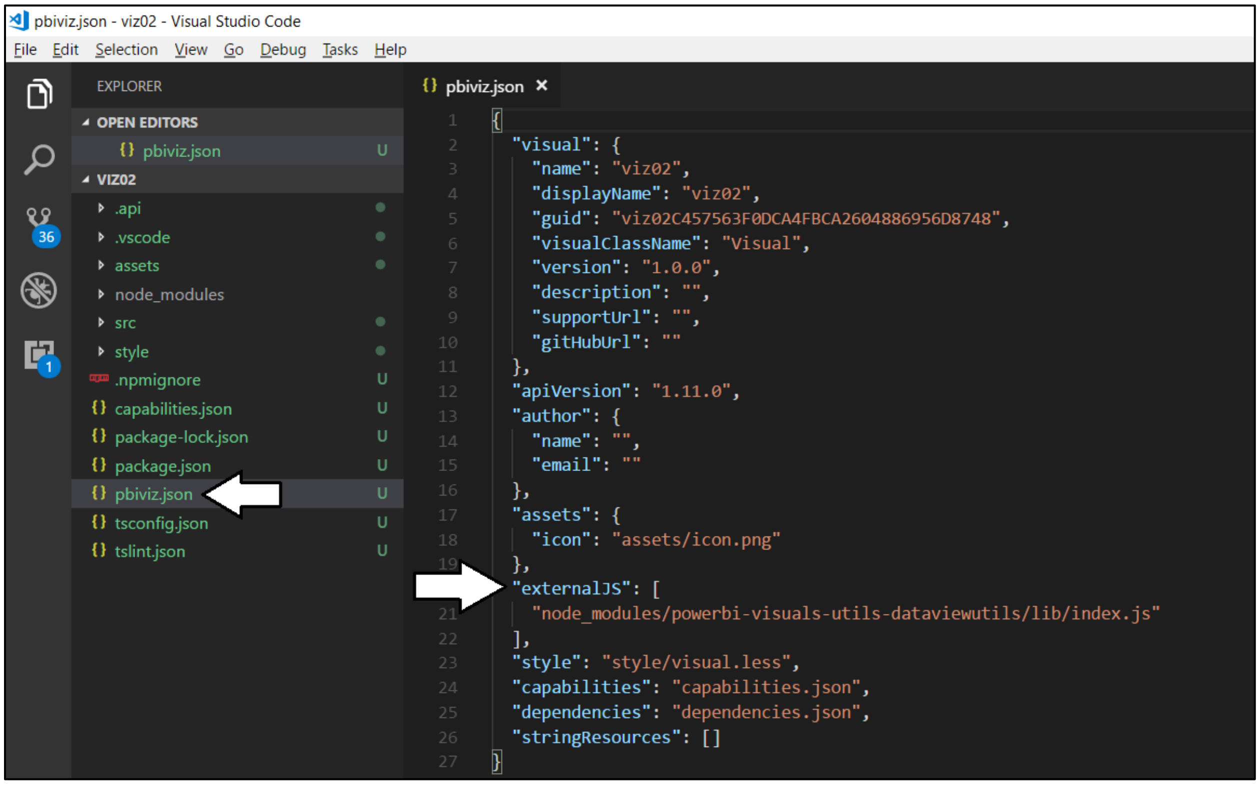
f) After you refresh the project, you should see a new folder named **node_modules**.

g) Expand the **node_modules** folder and locate the main jQuery source file located at **node_modules/jquery/dist/jquery.js**.



4. Add the external file reference for **jquery.js** into the **pbiviz.json** file.

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

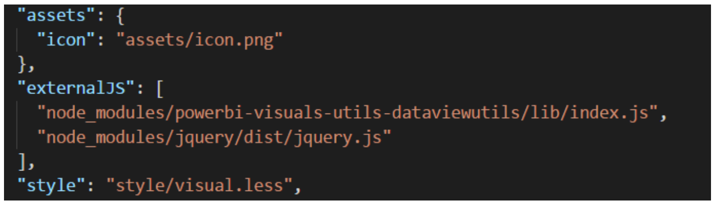


```
{
  "visual": {
    "name": "viz02",
    "displayName": "viz02",
    "guid": "viz02C457563F0DCA4FBCA2604886956D8748",
    "visualClassName": "Visual",
    "version": "1.0.0",
    "description": "",
    "supportUrl": "",
    "gitHubUrl": ""
  },
  "apiVersion": "1.11.0",
  "author": {
    "name": "",
    "email": ""
  },
  "assets": {
    "icon": "assets/icon.png"
  },
  "externalJS": [
    "node_modules/powerbi-visuals-utils-dataviewutils/lib/index.js"
  ],
  "style": "style/visual.less",
  "capabilities": "capabilities.json",
  "dependencies": "dependencies.json",
  "stringResources": []
}
```

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

```
"externalJS": [
  "node_modules/powerbi-visuals-utils-dataviewutils/lib/index.js",
  "node_modules/jquery/dist/jquery.js"
],
```

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

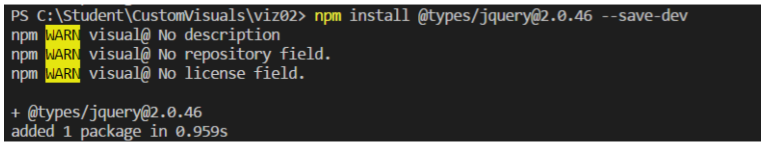

```
"assets": {
  "icon": "assets/icon.png"
},
"externalJS": [
  "node_modules/powerbi-visuals-utils-dataviewutils/lib/index.js",
  "node_modules/jquery/dist/jquery.js"
],
"style": "style/visual.less",
```

In the next step, you will add the TypeScript typed definition files for jQuery. However, you will not add the latest version because it has incompatibilities with the **pbiviz** build process. Therefore, you will install an older version which is 2.0.46.

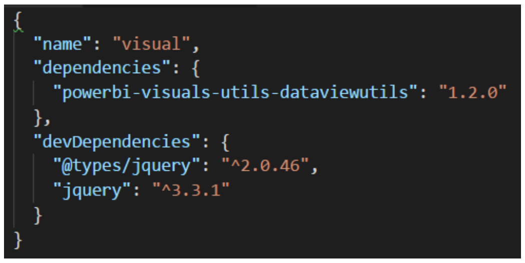
5. Add TypeScript typed definition package for the jQuery library to enabled strongly-typed programming.
   a) Return to the Node.js command prompt.
   b) Type and execute the following command to add version 2.0.46 of the the typed definition files for jQuery into your project.

```
npm install @types/jquery@2.0.46 --save-dev
```

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

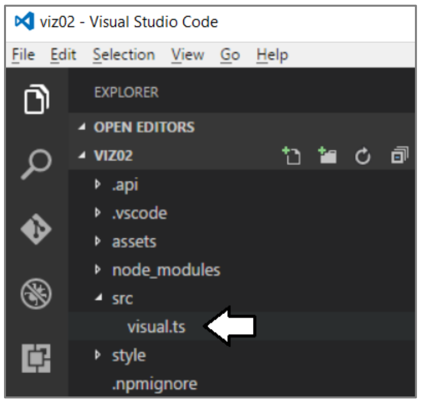

```
PS C:\Student\CustomVisuals\viz02> npm install @types/jquery@2.0.46 --save-dev
npm WARN visual@ No description
npm WARN visual@ No repository field.
npm WARN visual@ No license field.

+ @types/jquery@2.0.46
added 1 package in 0.959s
```

   d) Return to Visual Studio Code and inspect the **package.json** file. You should see the new reference to **@types/jquery**.


```
{
  "name": "visual",
  "dependencies": {
    "powerbi-visuals-utils-dataviewutils": "1.2.0"
  },
  "devDependencies": {
    "@types/jquery": "^2.0.46",
    "jquery": "^3.3.1"
  }
}
```

In early versions of TypeScript, you were required to add an entry into the **files[]** section of the **tsconfig.json** file to reference the TypeScript typed definition file for jQuery named **index.d.ts**. However, this step is no longer required because recent versions of the TypeScript compiler automatically reference type definition files that are added into the **@types** folder in the **node_modules** folder.

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 = $("<table>", {"id": "myTable"})
        .append( $("<tr>")
        .append( $("<td>").text("Width") )
        .append( $("<td>").text(options.viewport.width.toFixed(0))
        ))
        .append( $("<tr>")
        .append( $("<td>").text("Height") )
        .append( $("<td>").text(options.viewport.height.toFixed(0))
        )
    );

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

You can copy & paste the code for this visual from **C:\Student\Modules\05_PBIVIZ\Lab\Snippets\Exercise3-Visual-Starter.ts.txt**..

e) Save your changes to **visual.ts**.

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

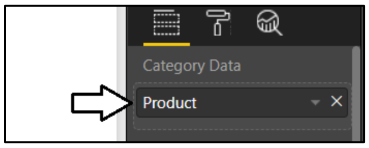
a) Return to the Node.js command prompt and run the following command to start a new debugging session.

```
pbiviz start
```

b) Move back to the browser and return to the Power BI report you used in the previous exercise.

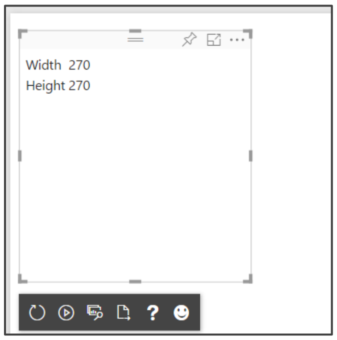
c) Make sure the report is in edit mode and add a Developer Visual to the page.

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

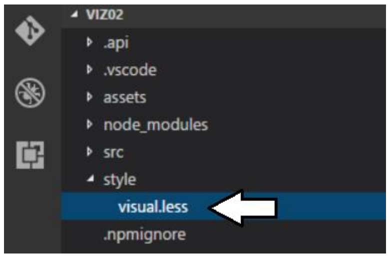


It doesn't really matter which field you add. However, you must add at least one field for the visual rendering to occur properly.

e) The developer visual should now match the following screenshot.


Width 270
Height 270

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**.



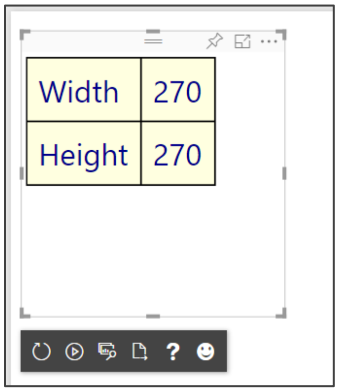
   c) Replace the contents of **visual.less** with the following CSS code.

```
#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.



Width	270
Height	270

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.

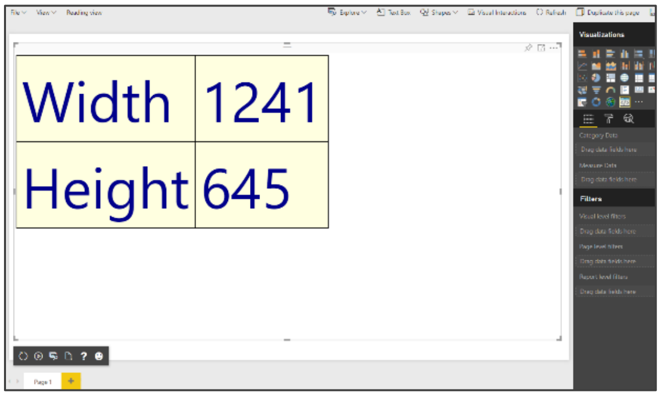
```
this.container.empty().append(table);
```

   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);
```

You can copy & paste the code for this visual from **C:\Student\Modules\05_PBIVIZ\Lab\Snippets\Exercise3-Font-Scaling.ts.txt**

   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.

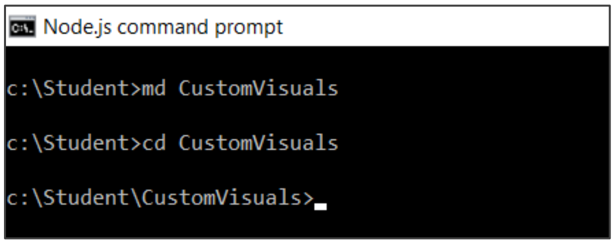


Exercise 4: Create a Custom Visual using the D3 Library

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 ..
```

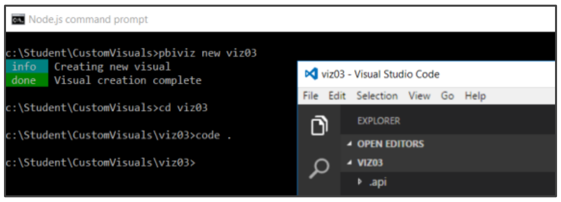

Node.js command prompt

```
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.

```
pbiviz new viz03
cd viz03
code .
```

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



```
c:\Student\CustomVisuals>pbiviz new viz03
 info  Creating new visual
 done  Visual creation complete

c:\Student\CustomVisuals>cd viz03

c:\Student\CustomVisuals\viz03>code .

c:\Student\CustomVisuals\viz03>
```

2. Open the Integrated Terminal so you can execute command-line instructions from within Visual Studio Code.
   a) Inside Visual Studio Code, select the **View > Integrated Terminal** menu command.
   b) You should see the **Integrated Terminal** window with a command prompt inside Visual Studio Code.

Power BI is not compatible with the latest version of D3 which is version 4. Therefore, you will install the latest build of D3 version 3.

3. Add the NPM package for version 3 of the D3 library.
   a) In the command prompt inside the Integrated Terminal, type and execute the following command.

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

Note that the **install** command of the **npm** utility will interpret **d3@3** as a command to install the latest build of D3 version 3.

   b) Wait for the **npm 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-dev
```

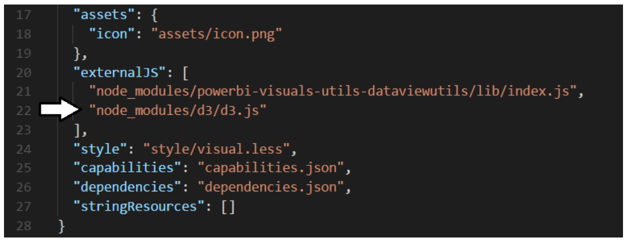
   d) Wait for the **npm install** command to complete.
   e) Return to Visual Studio Code.
   f) Open the **package.json** file to see the new dependency on the D3 library and the D3 typed definition files.

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

4. Add the external file reference for **d3.js** into the **pbiviz.json** file.
   a) Open the **pbiviz.json** file and locate the **externalJS** setting.
   b) Update the array for the **externalJS** setting by adding the path to **d3.js**.

```
"externalJS": [
  "node_modules/powerbi-visuals-utils-dataviewutils/lib/index.js",
  "node_modules/d3/d3.js"
],
```

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



```
  "assets": {
    "icon": "assets/icon.png"
  },
  "externalJS": [
    "node_modules/powerbi-visuals-utils-dataviewutils/lib/index.js",
    "node_modules/d3/d3.js"
  ],
  "style": "style/visual.less",
  "capabilities": "capabilities.json",
  "dependencies": "dependencies.json",
  "stringResources": []
}
```

   d) Save your changes and close **pbiviz.json**.

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

5. 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.

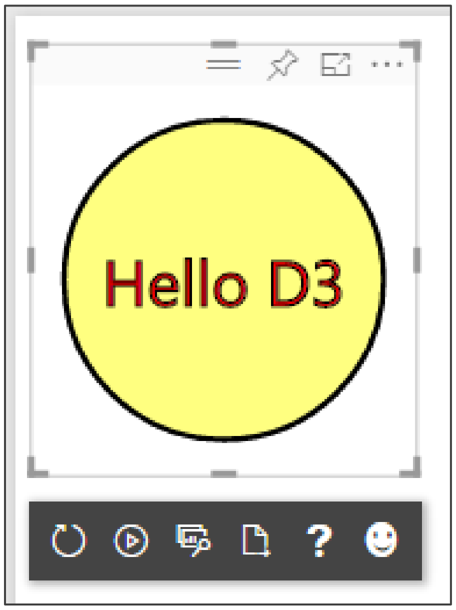
```
public update(options: VisualUpdateOptions) {

    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);
}
```

You can copy & paste all this code from **C:\Student\Modules\05_PBIVIZ\Lab\Snippets\Exercise4-Visual-Starter.ts.txt**

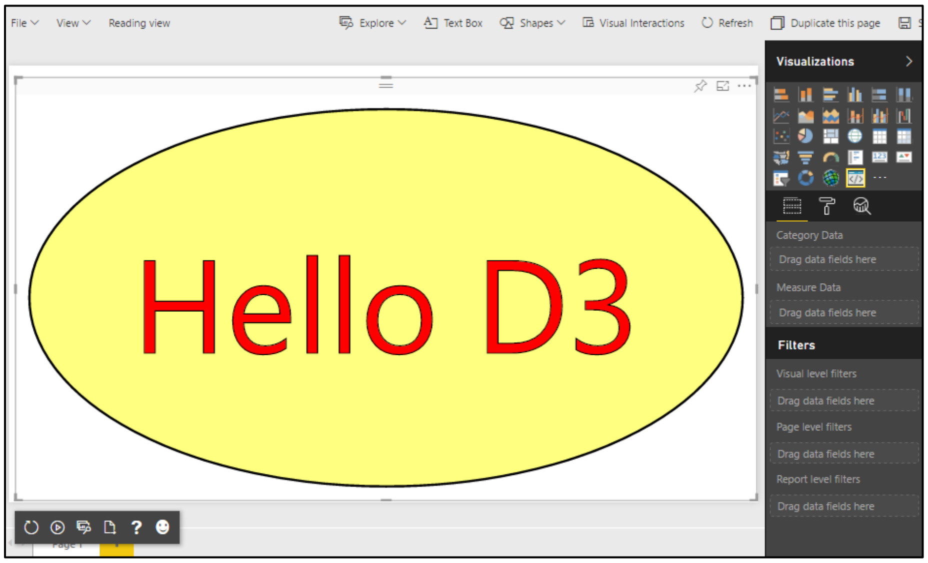
6. Test out your new visual on a Power BI report.
   a) Return to the Node.js command prompt and run the following command to start a new debugging session.

   ```
   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 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 to scales to various sizes.



Congratulations. You have now completed this lab.