For the full manual go to http://bitsplash.io/graph-and-chart

Offline Manual - This one covers only the functions within the lite edition of the asset

Table Of Contents

HOW TO STAY UPDATED

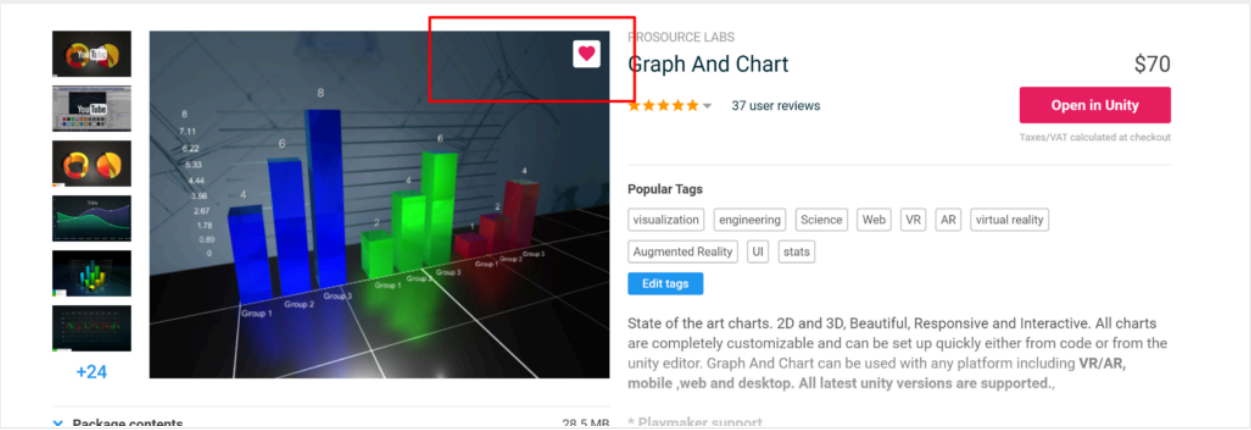
If you want to stay updated with important information and tutorials, make sure to follow us on these channels:

Follow us on GitHub. Also this repository is expected to be integrated into future extensions of Graph and Chart. It's purpose is to be the main data structure to hold arrays in Graph and Chart.

Follow us on Youtube it contains tutorial videos that you can watch.

Sign in to our newsletter in http://bitsplash.io (scroll to bottom and enter your email)

Click the heart Icon in the Asset store page you will be notified on sales and new versions



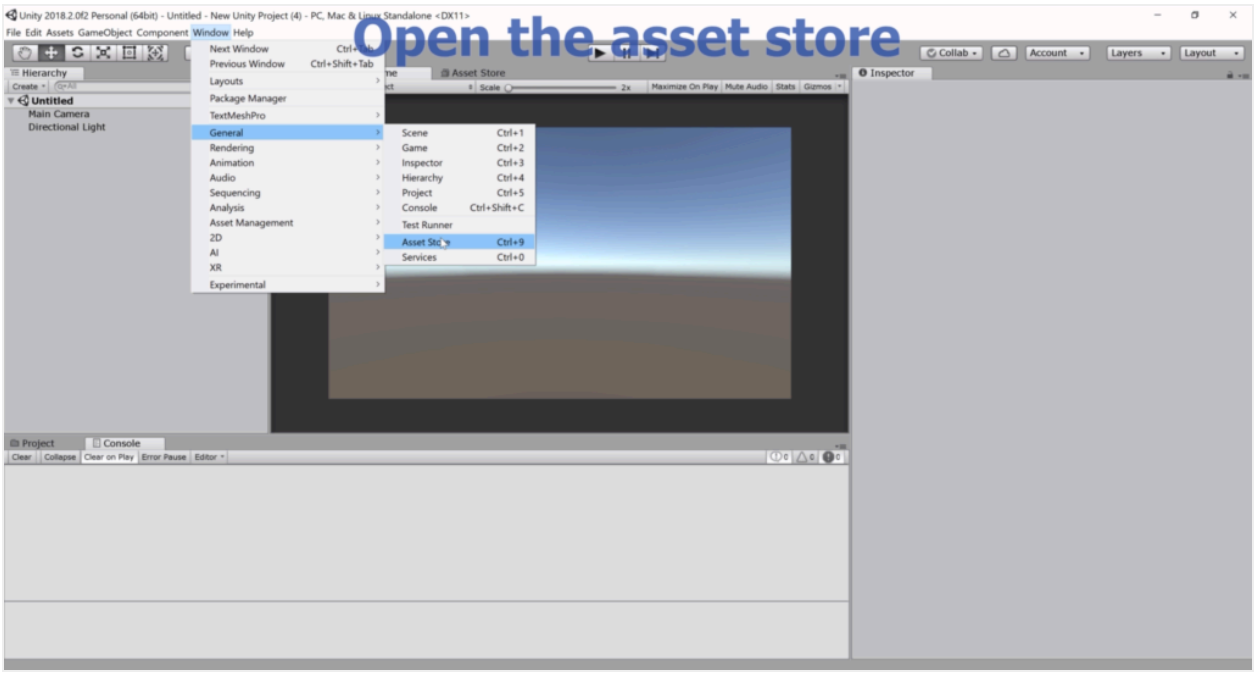
VIDEO RESOURCES

The video resources for Graph and Chart can be found in our Youtube channel . Specifically, you can view the 40 min video tutorial.

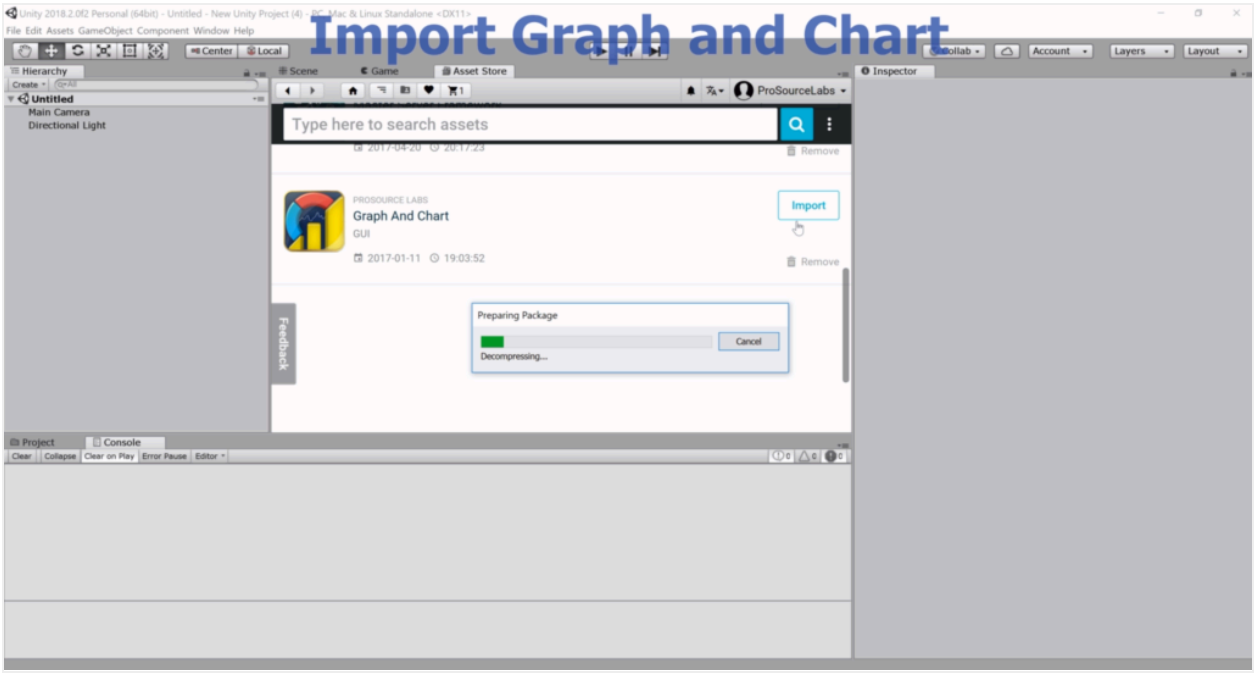
Also , some of the articles in this documentation contain video links. These video links are a part of the main tutorial that is relevant to an article. Make sure not to miss them.

IMPORTING GRAPH AND CHART

Open the Asset Store window



Import Graph And Chart



You can place the asset folder inside your project where you wish, However Make sure not to move files inside the asset folder. Deleting files from Graph and Chart may cause problems and is unnecessary , Unity will not include unused files in your final builds.

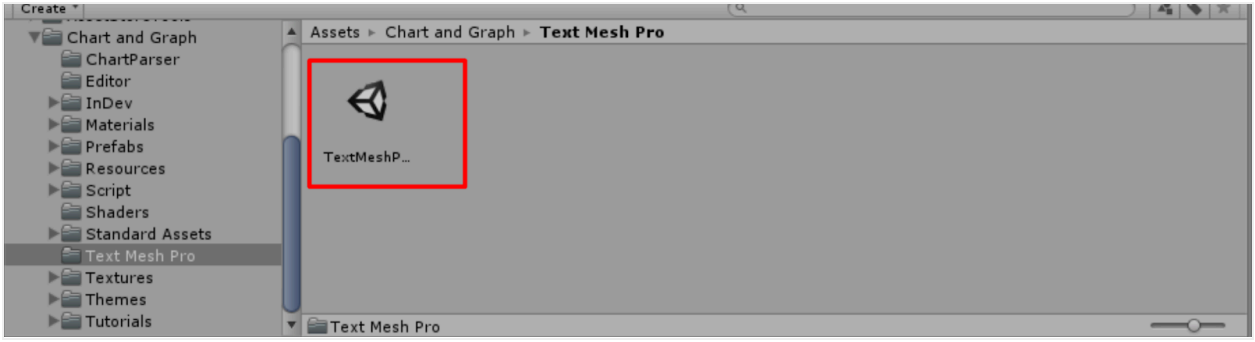
TEXTMESHPRO INSTALLATION

TEXTMESHPRO INSTALLATION

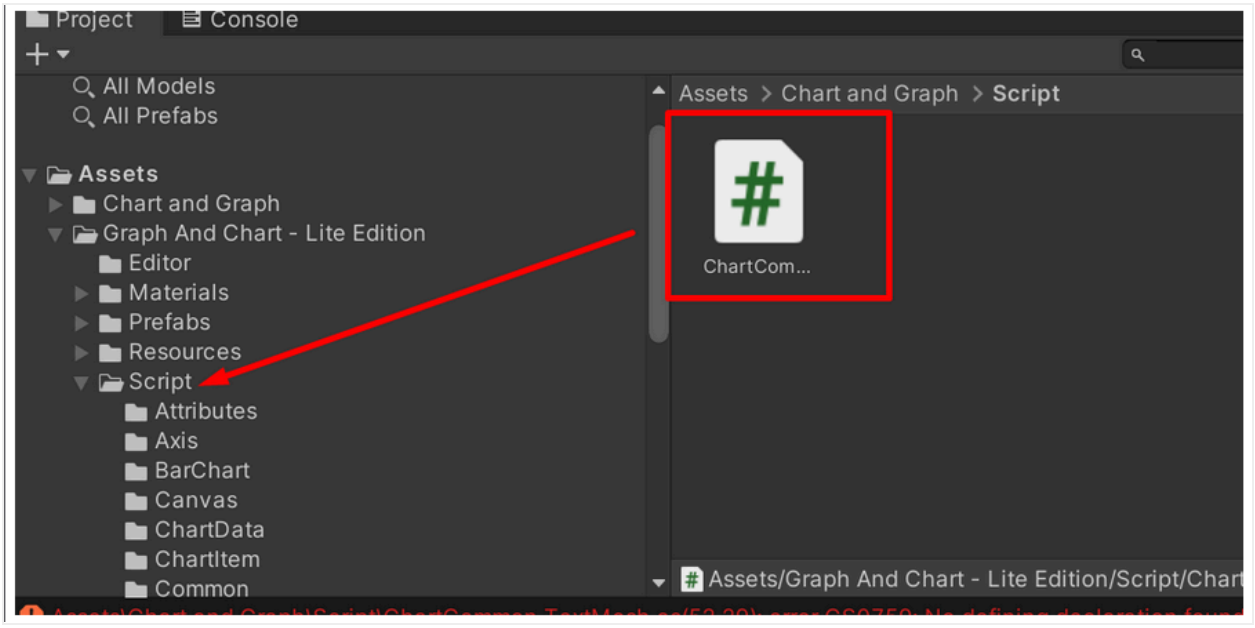
If you wish to use TextMeshPro with Graph and Chart , go through the following steps:

1. Open the project tab and go to Chart And Graph/TextMeshPro

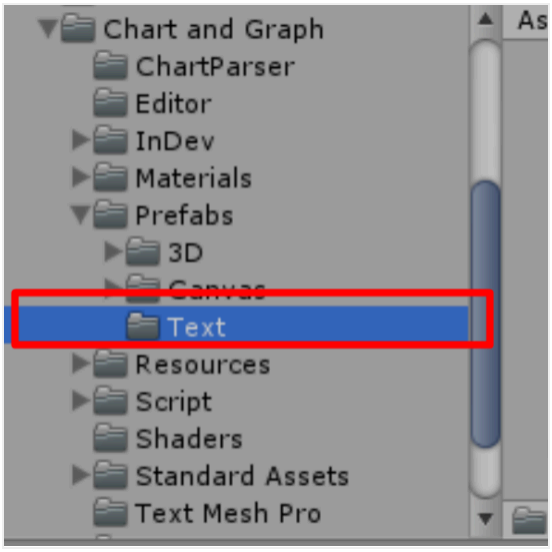
2. Unpack the TextMeshPro package



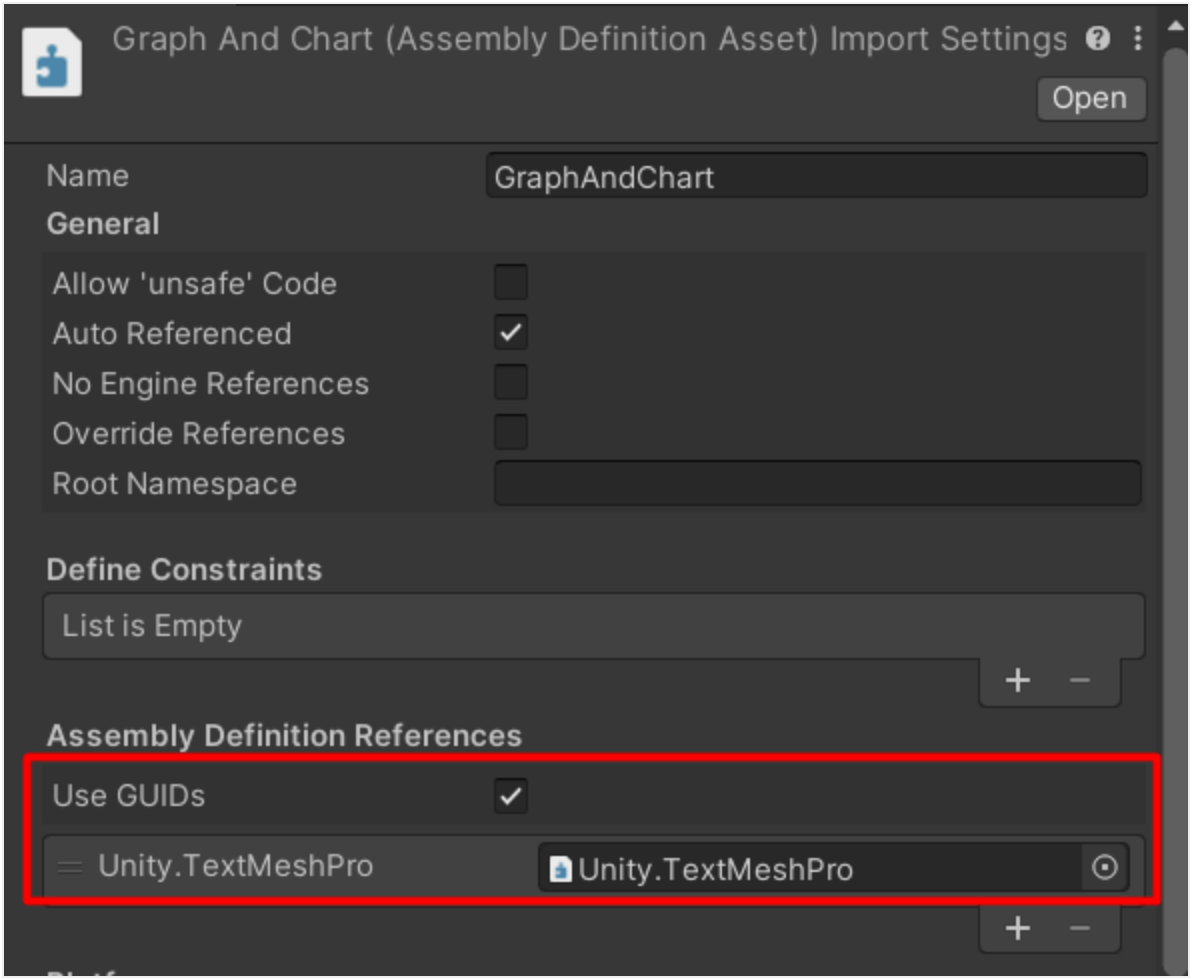
If you are using the lite or scientific version. Copy the script file to the lite or scientific version folder like in the image below:



3. The prefabs folder now contains two Text Mesh pro prefabs that can be modified duplicated and used just like any other text prefab in Graph And chart.

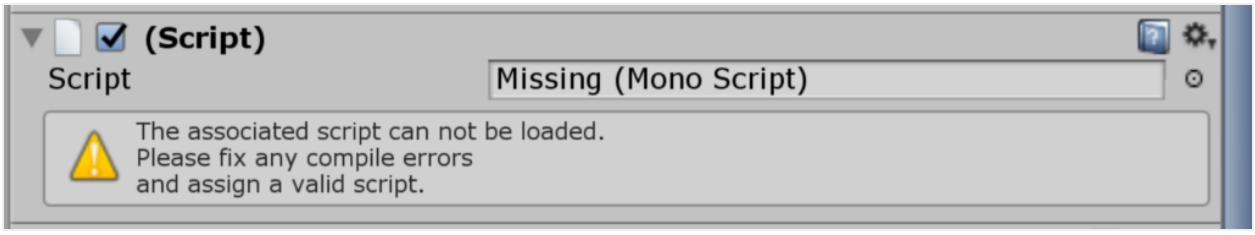


4. Locate the assembly definition file in Graph and Chart/Script, and assign a reference to the text mesh pro assembly



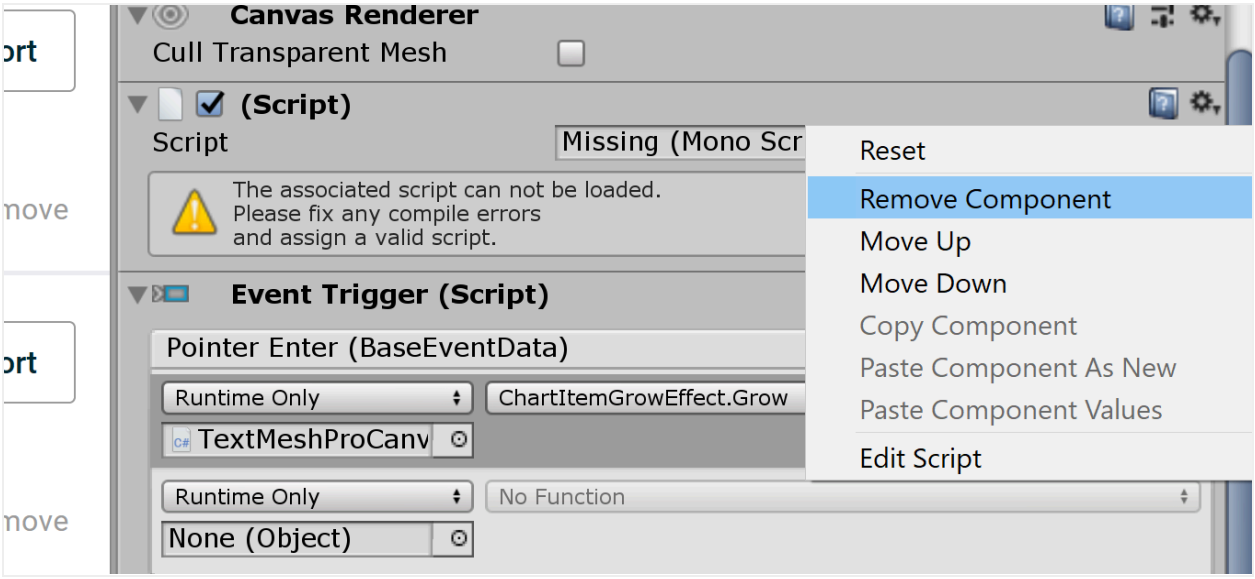
TROUBLESHOOT

if for some reason the scripting reference has not been resolved for the prefab's TextMesh scripts. You will see the following warning :

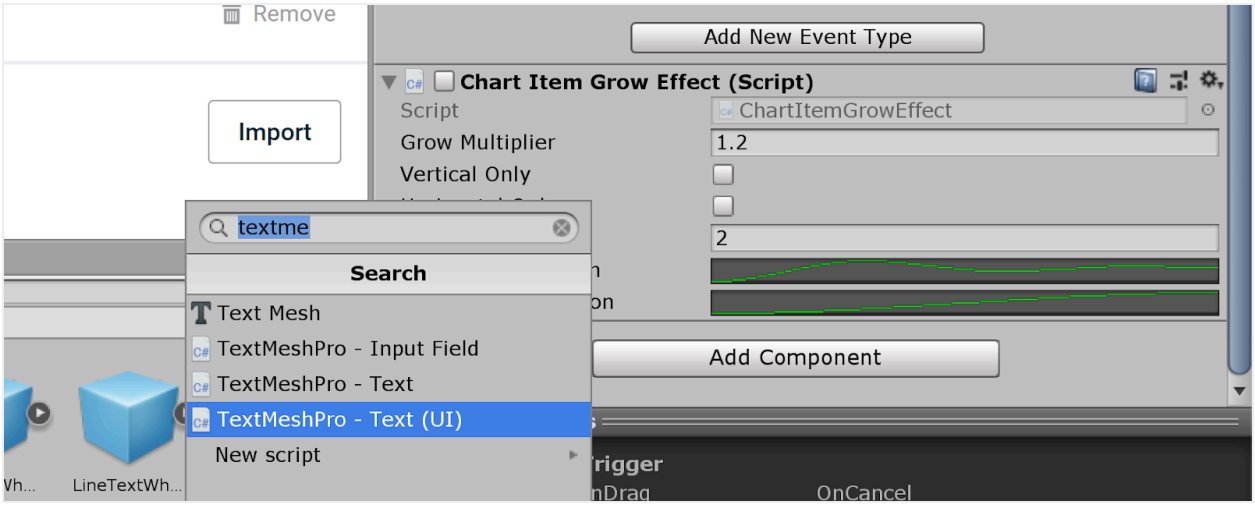


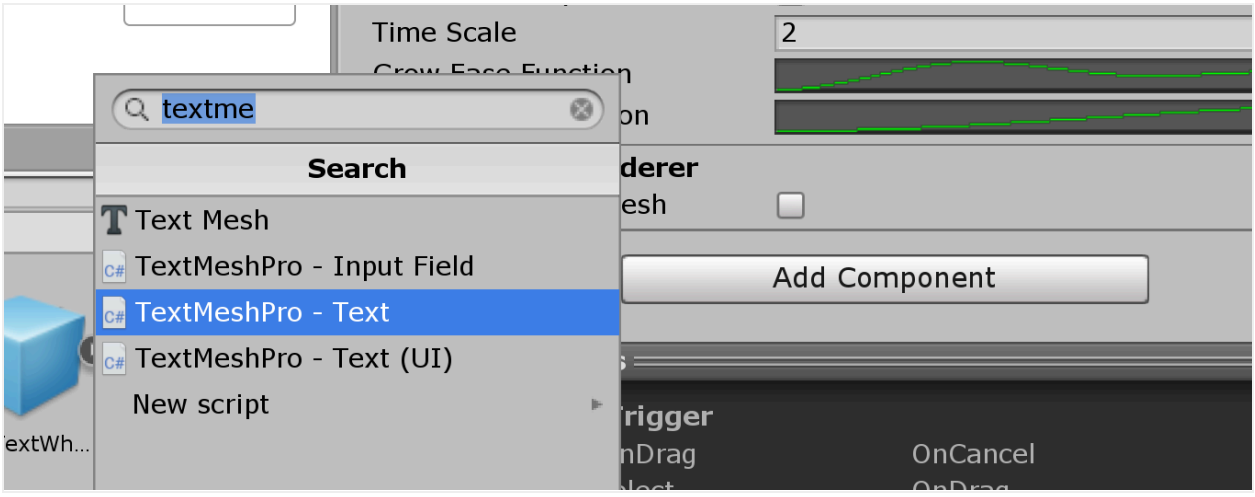
You can resolve this issue by going through these steps:

1. Remove all missing scripts from the prefabs



2. Add a new TextMeshPro-Text (UI) component to the prefab named TextMeshProCanvas





3. Add a new TextMeshPro-Text component to the prefab named TextMeshProWorldSpace

4. configure the text mesh pro components as you wish. Make sure they are justified:

FOLDERS OF INTEREST

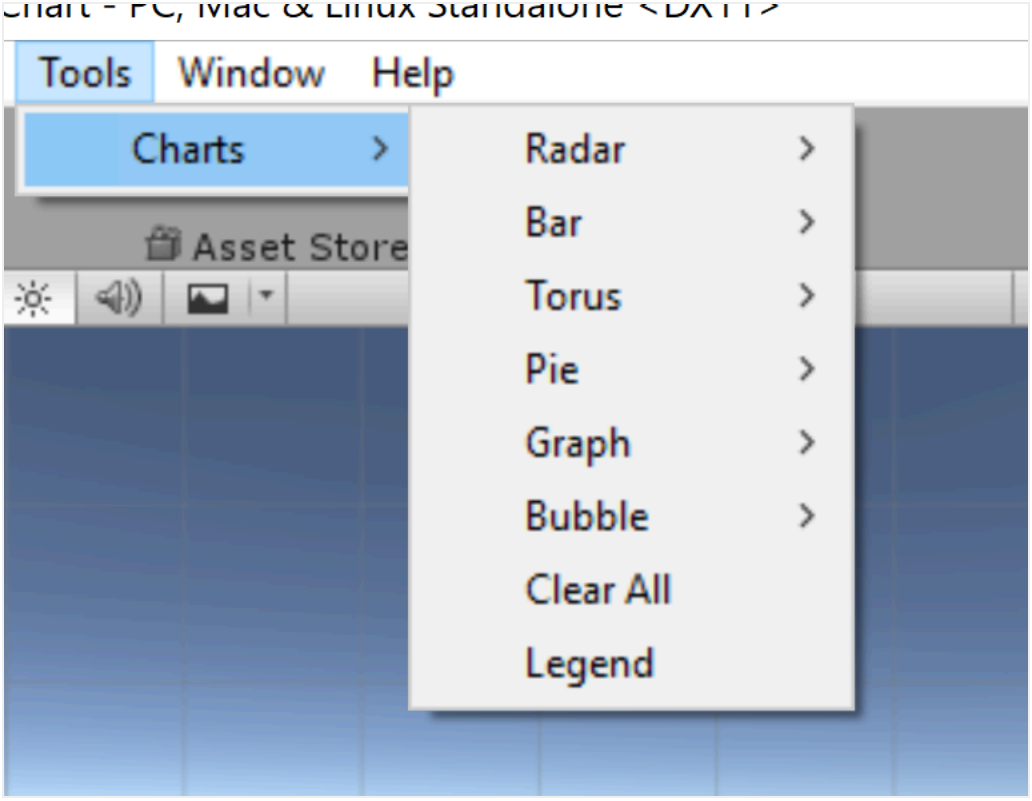
These are some of the more important folders inside the Chart And Graph folder.

- **Chart And Graph / Themes :** various sample scenes from the demo project and some that are not included in the demo project. The themes are divided into 2D and 3D. Most themes have a few presets that you can explore
- **Chart And Graph/Tutorials** : These are the product scenes of the tutorials in the Quick Start section and other sections of this documentation
- **Chart And Graph/Prefabs:** prefabs that can be assigned to the charts and labels , be sure to explore this part of the library. The content of this folder is covered thoroughly in the Prefabs section of this documentation

- **Chart And Graph/ Materials:** useful materials for canvas charts , this include line styles and point styles for the axis and graph chart

THE TOOLS MENU

After importing Graph and Chart , you should be able to see it in the Tools menu.

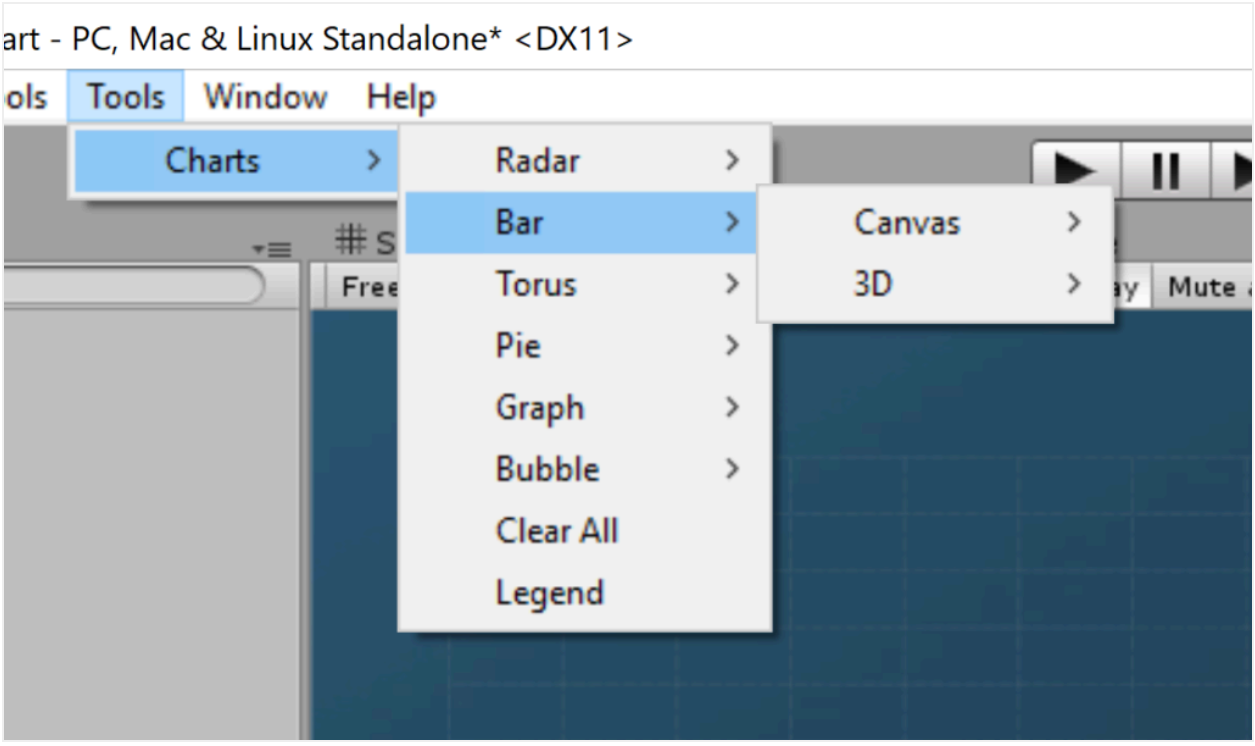


You can use this menu to add charts into the open scene with a click of a button. Each chart has a few options.

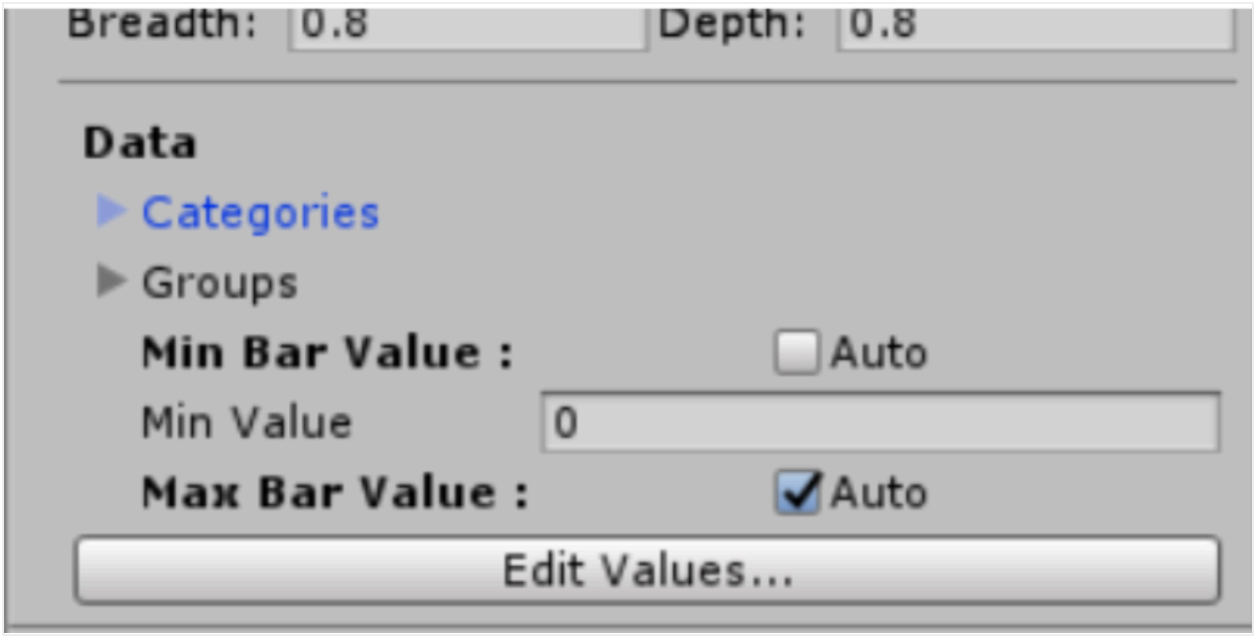
BAR CHART QUICK START

The resulting scene and script from this tutorial can be found at **Assets\Chart and Graph\Demos\Tutorials\Bar**

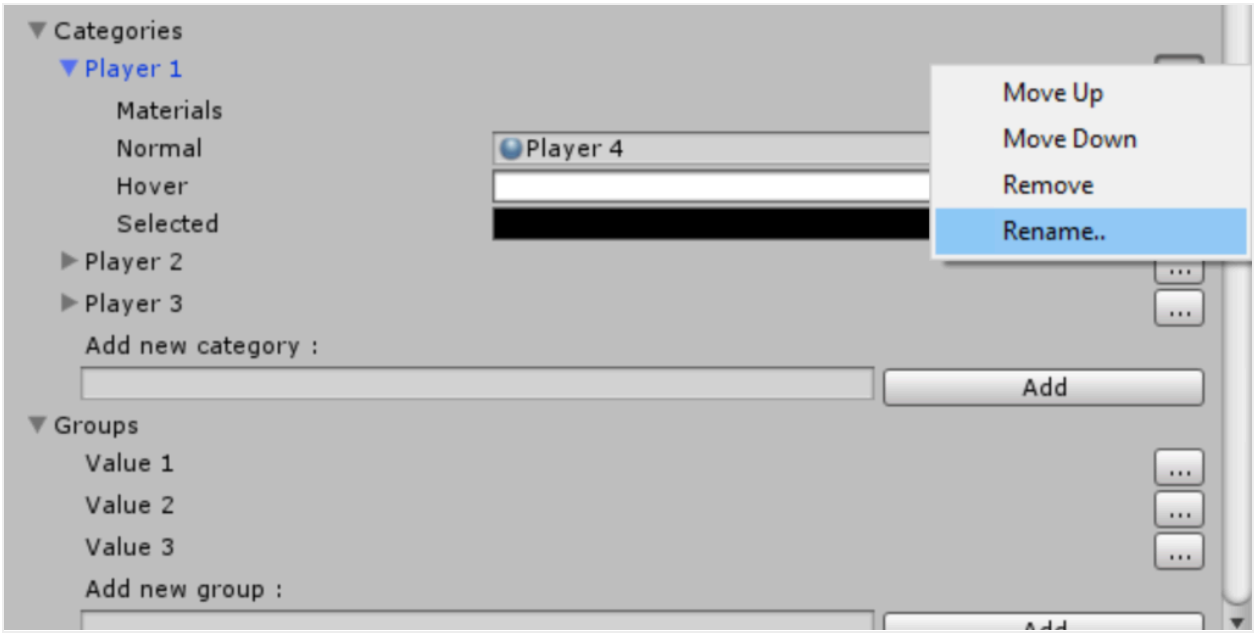
Go to Tools->Charts->Bar and select the type of bar chart you would like to create. A new chart object should appear in the scene (if a canvas chart is selected it should appear under a canvas).



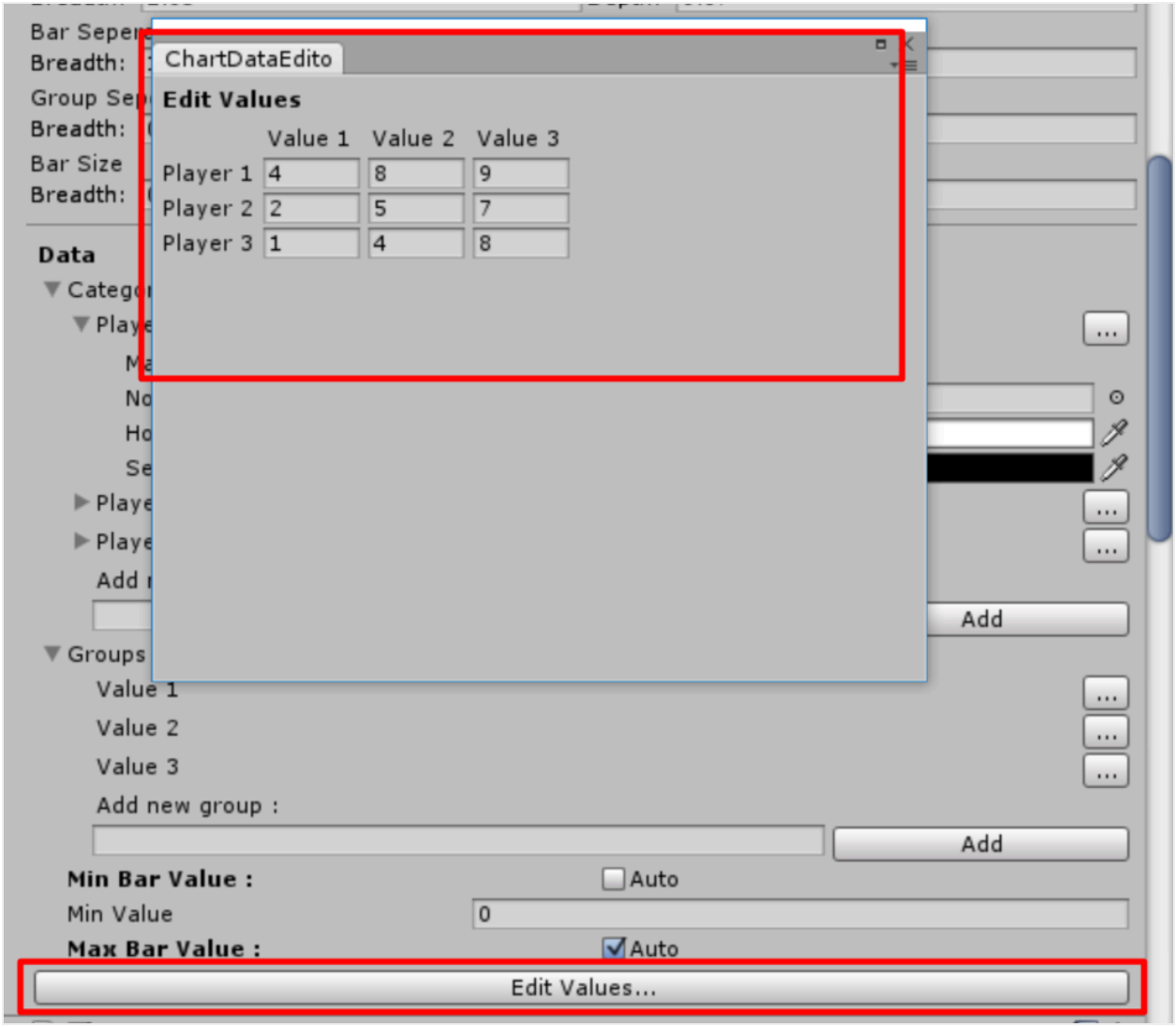
Select the bar object and view it in the inspector. Notice the data section of the bar chart inspector.



Expand both Categories and groups. You can add groups and categories by typing a name and clicking the add button. You can remove and rename groups and categories by clicking the “…” button near each one. For the sake of demonstration , We will assume that “category *” is renamed to “player *” and “group *” is renamed to “value *” . After creating the categories the data section should look like this :



To edit the chart values from the editor click the “Edit Values” button



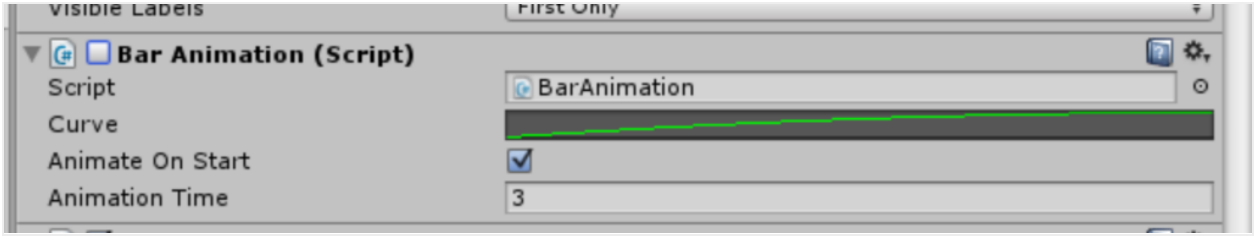
To update the bar chart from your script , create a new monobehaviour named BarChartFeed and add the following code to your script :

```
void Start () {

        BarChart barChart = GetComponent<BarChart>();

        if (barChart != null)
```

```
        {

            barChart.DataSource.SetValue("Player 1", "Value 1", Random.value * 20);

            barChart.DataSource.SlideValue("Player 2","Value 1", Random.value * 20,
40f);

        }

    }
```

NOTICE: calling SetValue and SlideValue must be done with your own group and category names. Notice spaces and case letters

make sure there is no BarAnimation component attached to the bar chart.

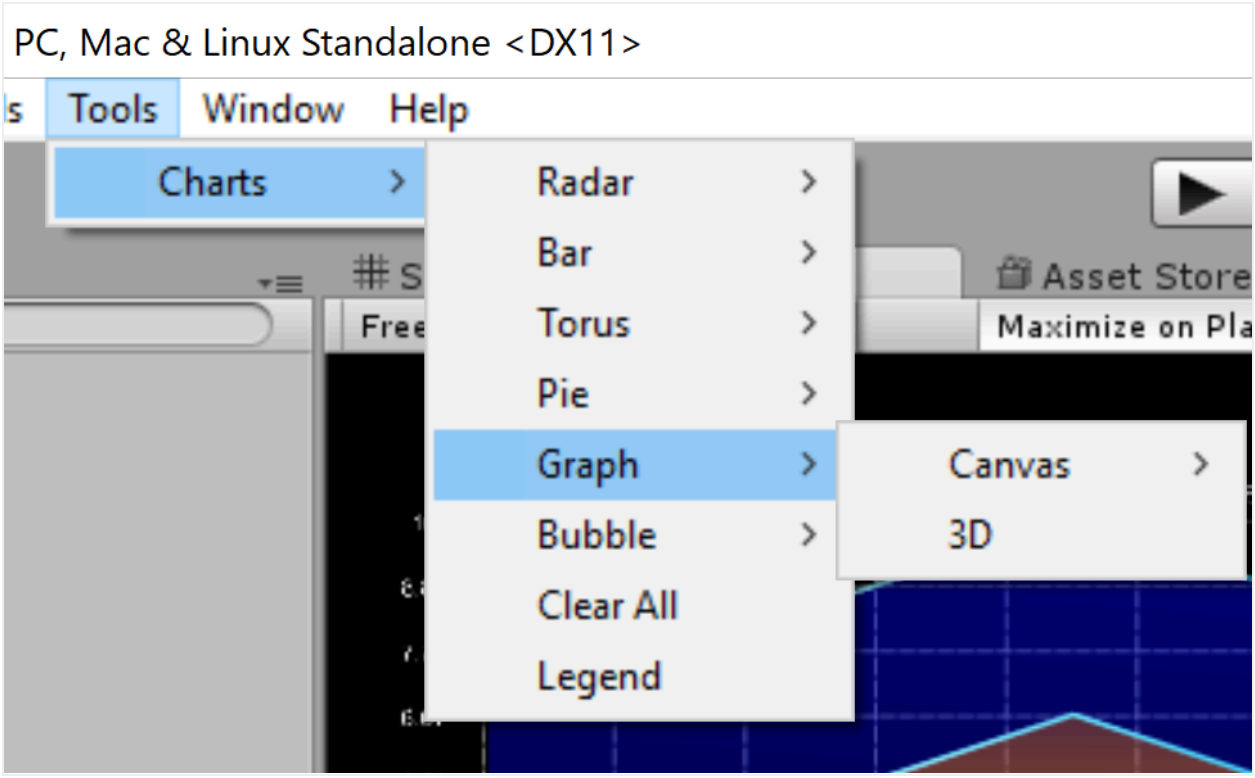


Add a BarFeedChart component to the bar chart object and click play

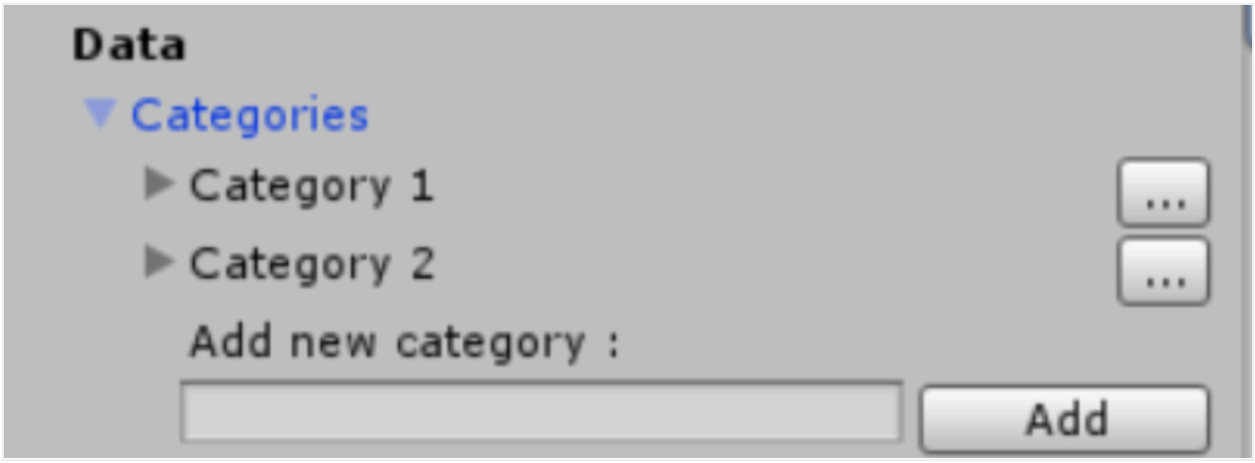
GRAPH CHART QUICK START

The resulting scene and script from this tutorial can be found at **Assets\Chart and Graph\Demos\Tutorials\Graph**

Add a canvas to the scene. Go to Tools->Charts->Graph and select the type of graph you would like to create.



Select the graph object and view it in the inspector. Notice the data section of the chart inspector.



Expand the Categories foldout. You can add categories by typing a name and clicking the add button. You can remove and rename categories by clicking the "…" button near each one. For the sake of demonstration , We will assume that "category *" is renamed to "player *". After creating the categories the data section should look like this :

Graph data can only be changed through source code. The graph appearance can be customized in the editor. To update the graph chart data from your script , create a new monobehaviour named GraphChartFeed and add the following code to your script :

```
void Start ()

    {

        GraphChart graph = GetComponent<GraphChart>();
```

```
        if (graph != null)

        {

            graph.DataSource.StartBatch();  // start a new update batch

            graph.DataSource.ClearCategory("Player 1");  // clear the categories we have created in the inspector

            graph.DataSource.ClearCategory("Player 2");

            for (int i = 0; i < 30; i++)

            {

//add 30 random points , each with a category and an x,y value

                graph.DataSource.AddPointToCategory("Player 1",Random.value*10f,Random.value*10f);

                graph.DataSource.AddPointToCategory("Player 2", Random.value * 10f, Random.value * 10f);

            }

            graph.DataSource.EndBatch(); // end the update batch . this call will render the graph
```

```
        }
```

```
    }
```

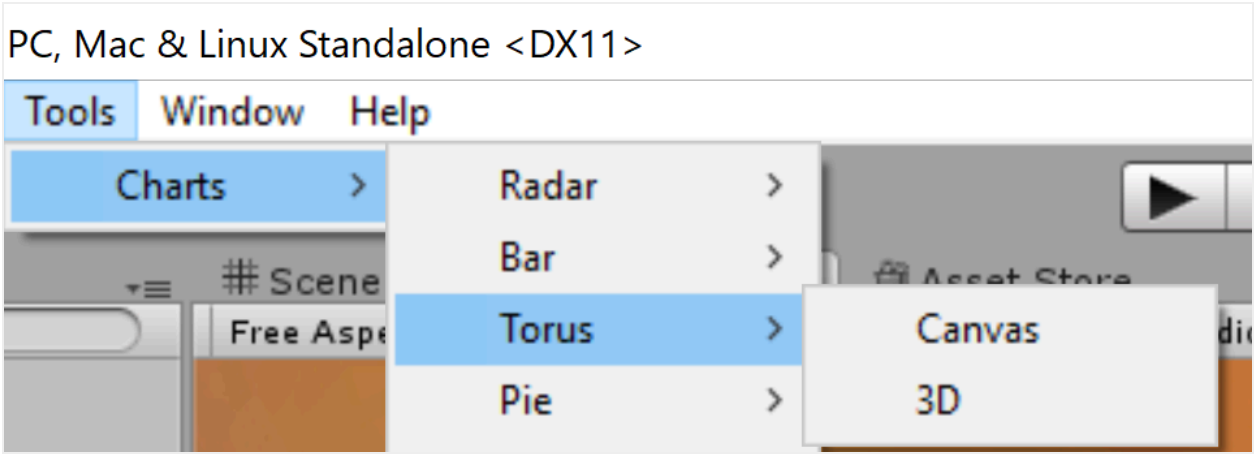
NOTICE: calling set value must be done with your own category names. Notice spaces and case letters

Add a GraphFeedChart component to the graph chart object and click play.

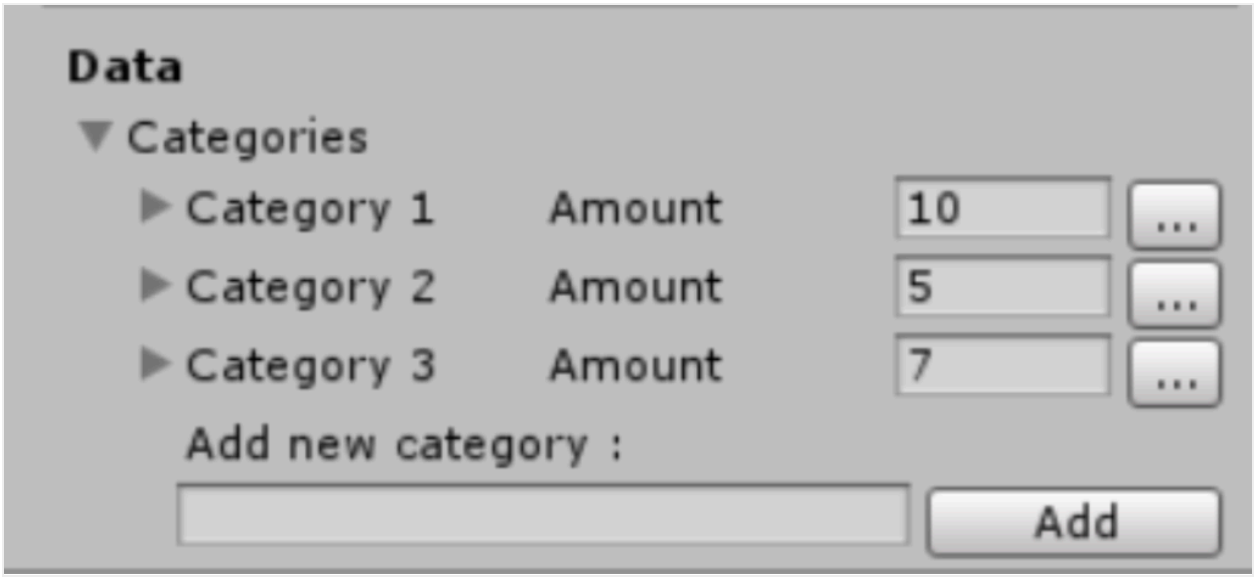
PIE/TORUS CHART QUICK START

The resulting scene and script from this tutorial can be found at **Assets\Chart and Graph\Demos\Tutorials\Pie**

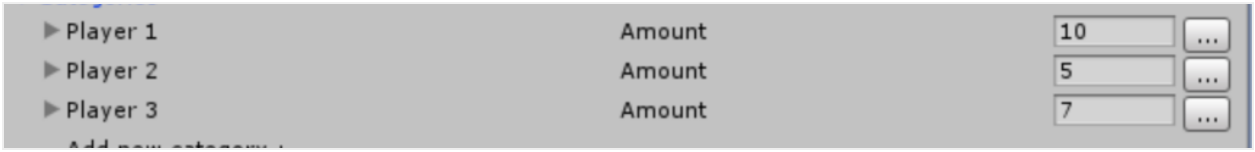
Go to Tools->Charts->Pie Or Tools->Charts->Torus and select the type of chart you would like to create. A new chart object should appear in the scene (if a canvas chart is selected it should appear under a canvas).



Select the pie or torus object and view it in the inspector. Notice the data section of the chart inspector.



Expand the Categories foldout. You can add categories by typing a name and clicking the add button. You can remove and rename categories by clicking the “…” button near each one. For the sake of demonstration , We will assume that “category *” is renamed to “player *”. the data section of your inspector should now look like this :



You can change the amount of each pie slice by editing it's input box. To update the pie chart from your script , create a new monobehaviour named PieChartFeed and add the following code to it :

```
void Start ()

 {

        PieChart pie = GetComponent<PieChart>();

        if(pie != null)

        {

            pie.DataSource.SlideValue("Player 1", 50, 10f);

            pie.DataSource.SetValue("Player 2", Random.value * 10);

        }

}
```

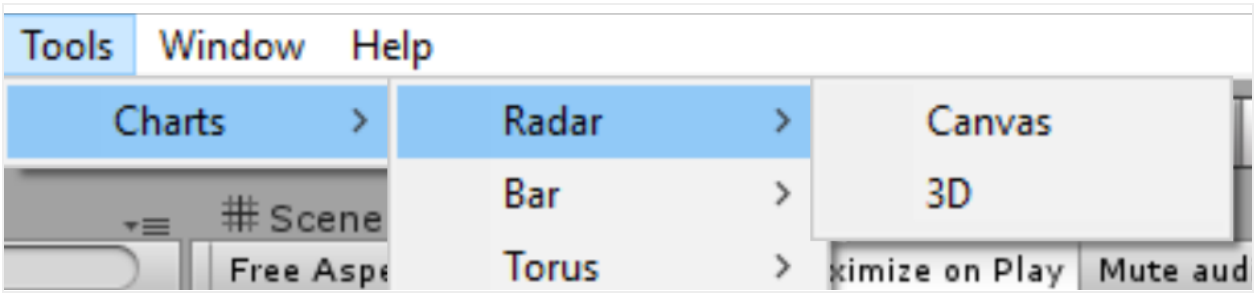
NOTICE: calling set value must be done with your own category names. Notice spaces and case letters

Add a PieFeedChart component to the pie chart object and click play.

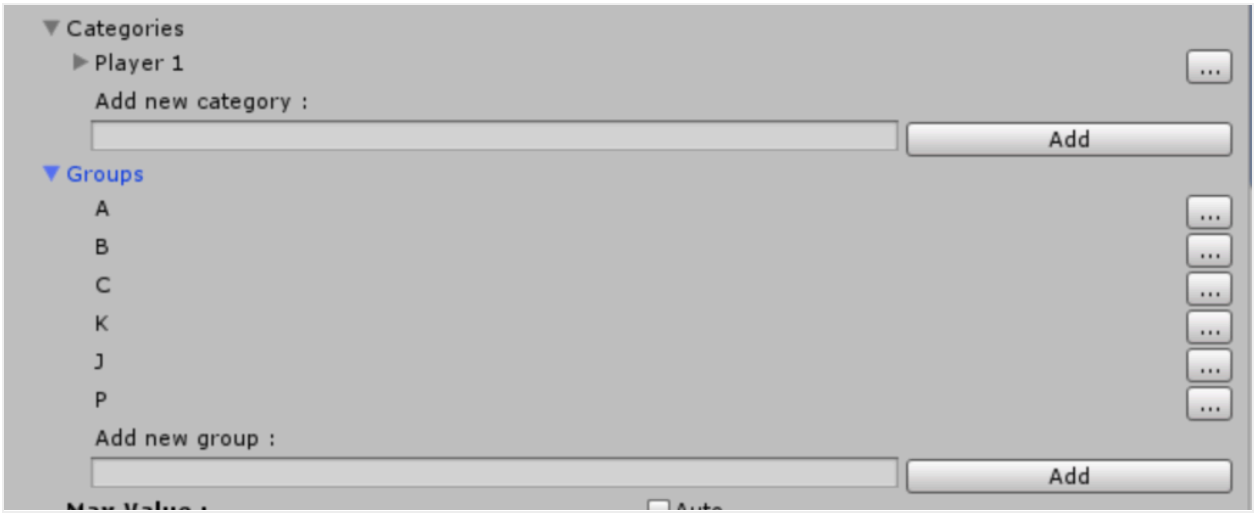
RADAR CHART QUICK START

The resulting scene and script from this tutorial can be found at **Assets\Chart and Graph\Demos\Tutorials\Radar**

Go to Tools->Charts->Radar and select the type of radar chart you would like to create. A new chart object should appear in the scene (if a canvas chart is selected it should appear under a canvas).

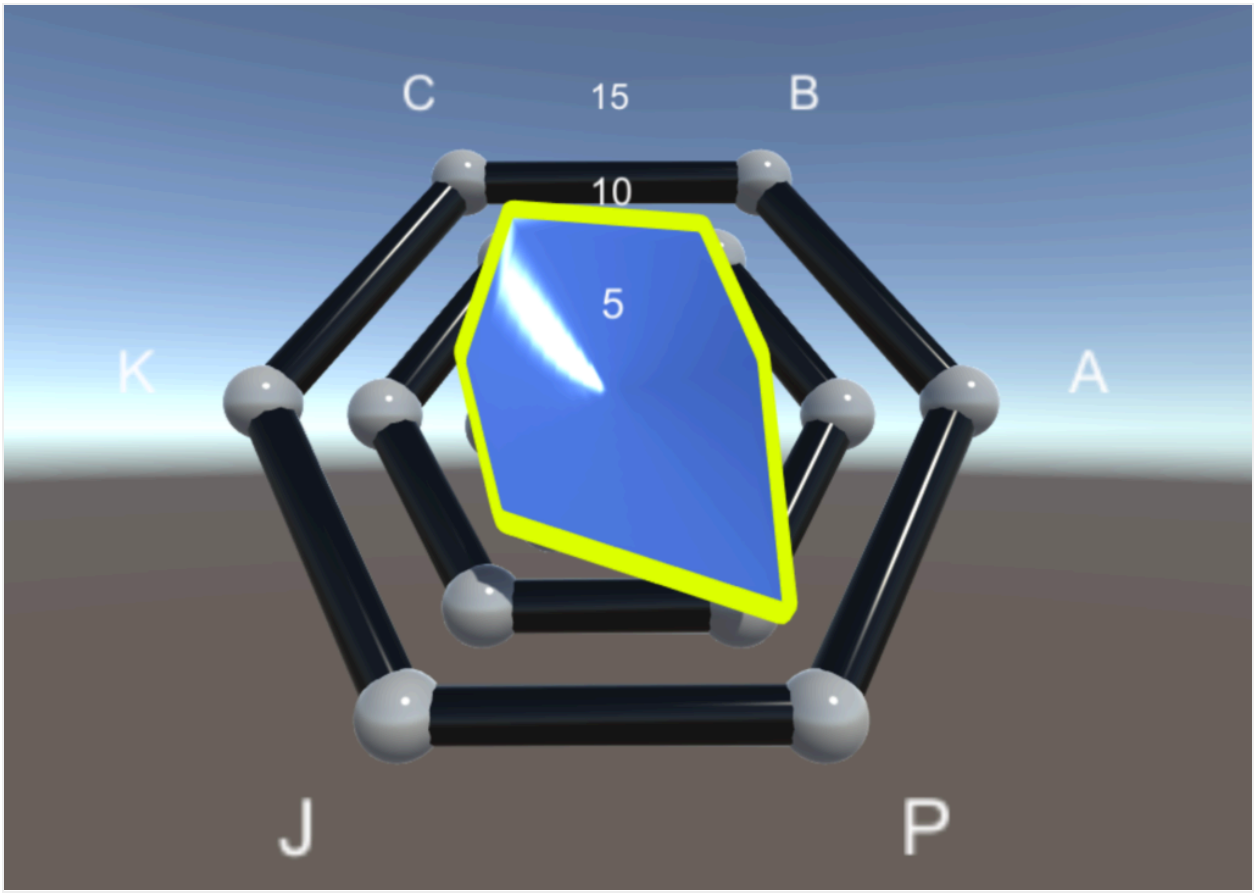


Select the radar object and view it in the inspector. Notice the data section of the radar chart inspector.



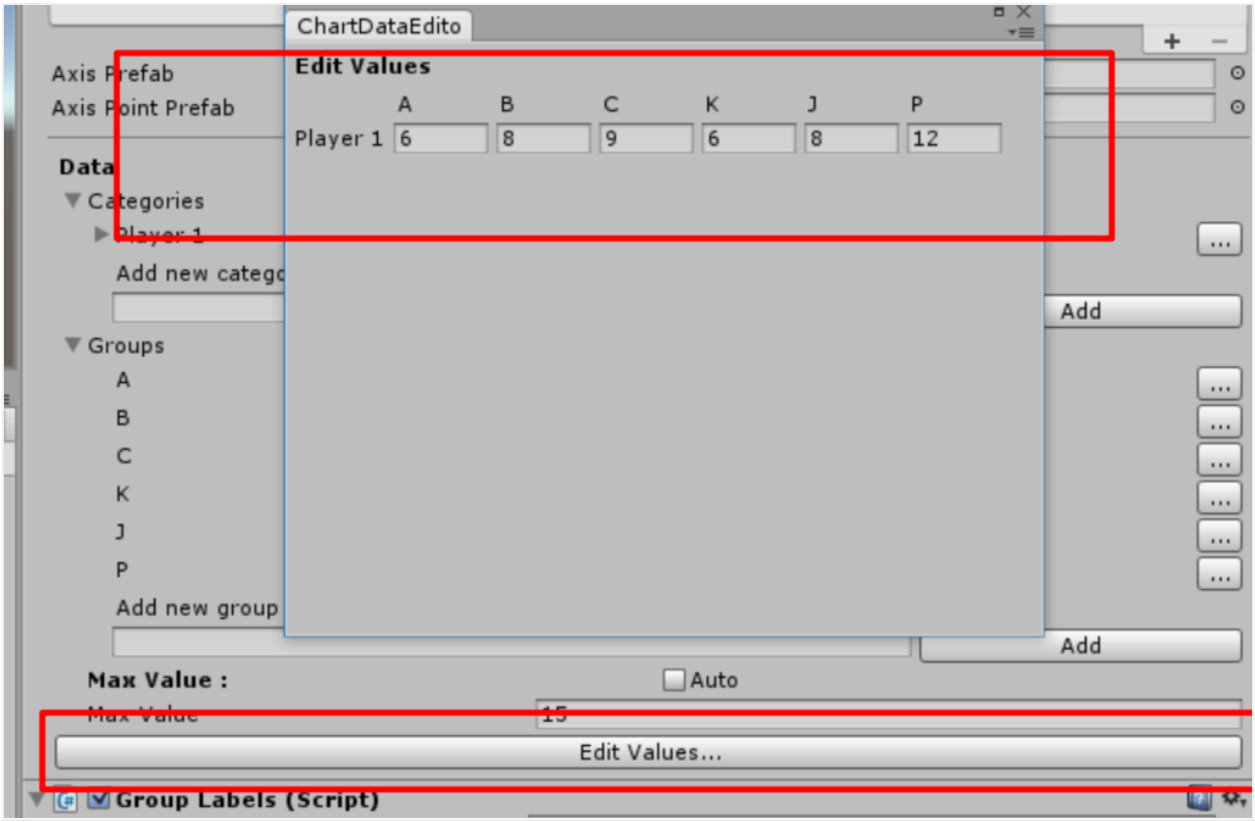
Expand both Categories and groups. You can add groups and categories by typing a name and clicking the add button. You can remove and rename groups and categories by clicking the “…” button near each one. For the sake of demonstration , We will assume you have created the category “Player 1” and the groups A,B,C,K,J,P . After creating the categories and group the data section should look like the image above.

In the radar , the groups make the edges of the axis and the categories make the data parts on the axis. So the configuration in the image above will result in a radar chart looking like this :



the groups are the edges of the chart and the category is the blue part

You can edit the category values manually by click Edit Values:



To update the radar chart from your script , create a new monobehaviour named RadarChartFeed and add the following code to it :

```
   void Start ()
    {
        var radar = GetComponent<RadarChart>();
        if (radar != null)
        {
            radar.DataSource.SetValue("Player 1", "A", 10);
        }
    }
```

NOTICE: calling SetValue must be done with your own group and category names. Notice spaces and case letters

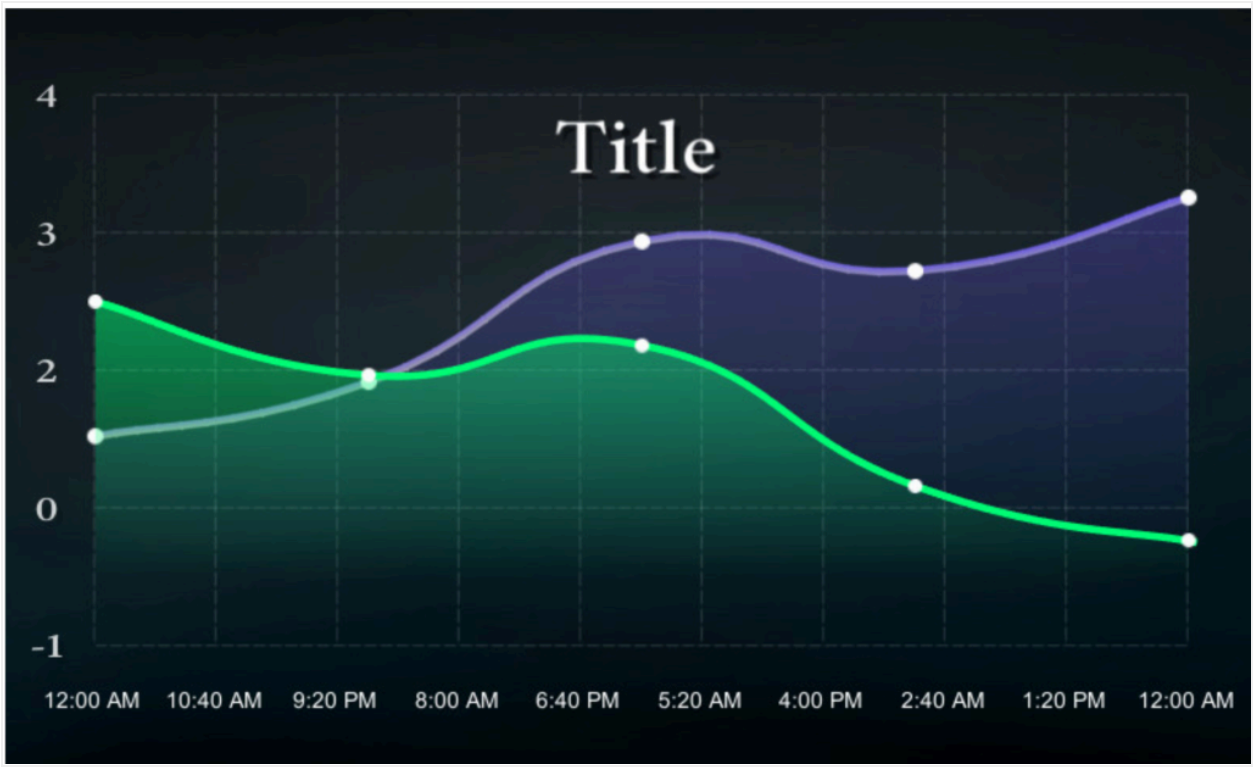
Add a RadarChartFeed component to the radar chart object and click play.

CHART CATEGORIES AND GROUPS

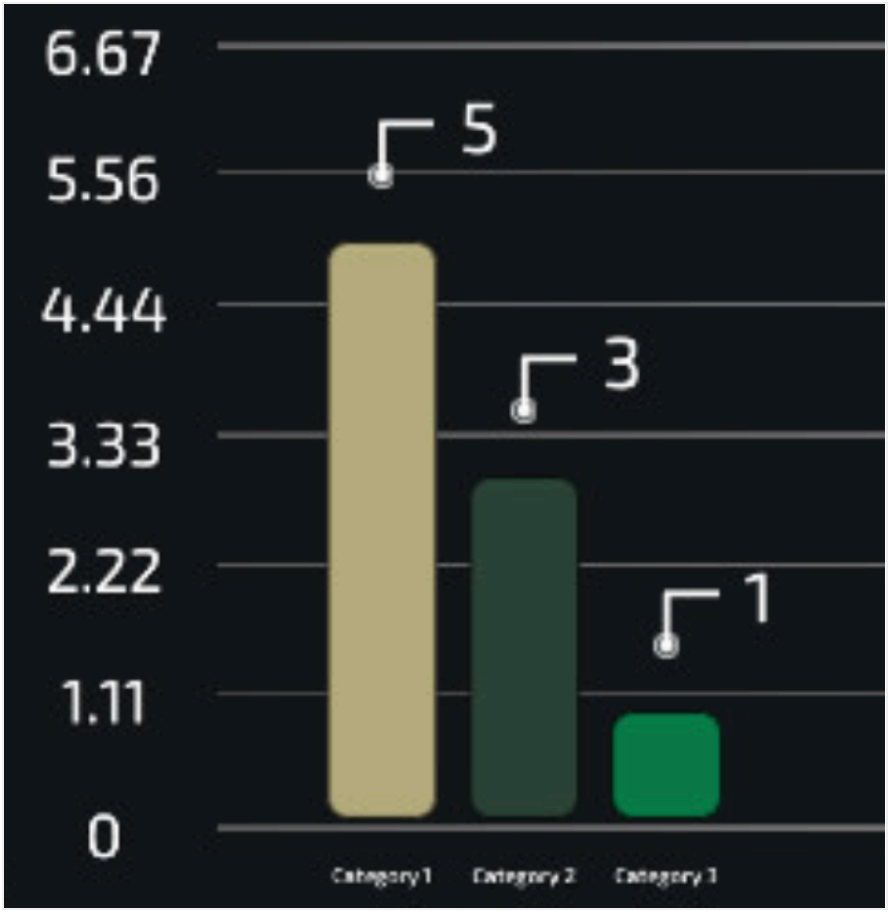
All charts in Graph and Chart are divided in a way that allows you to easily control their data. The division of all charts is either to categories or to both categories and groups. The goal of this article is to give you a basic understanding of the concept of categories and groups , you can learn more about their use by learning how to use the different charts of Graph and Chart.

WHAT ARE CATEGORIES

Categories define a part of the chart that can be customized. This can be for example a data series in a graph chart :



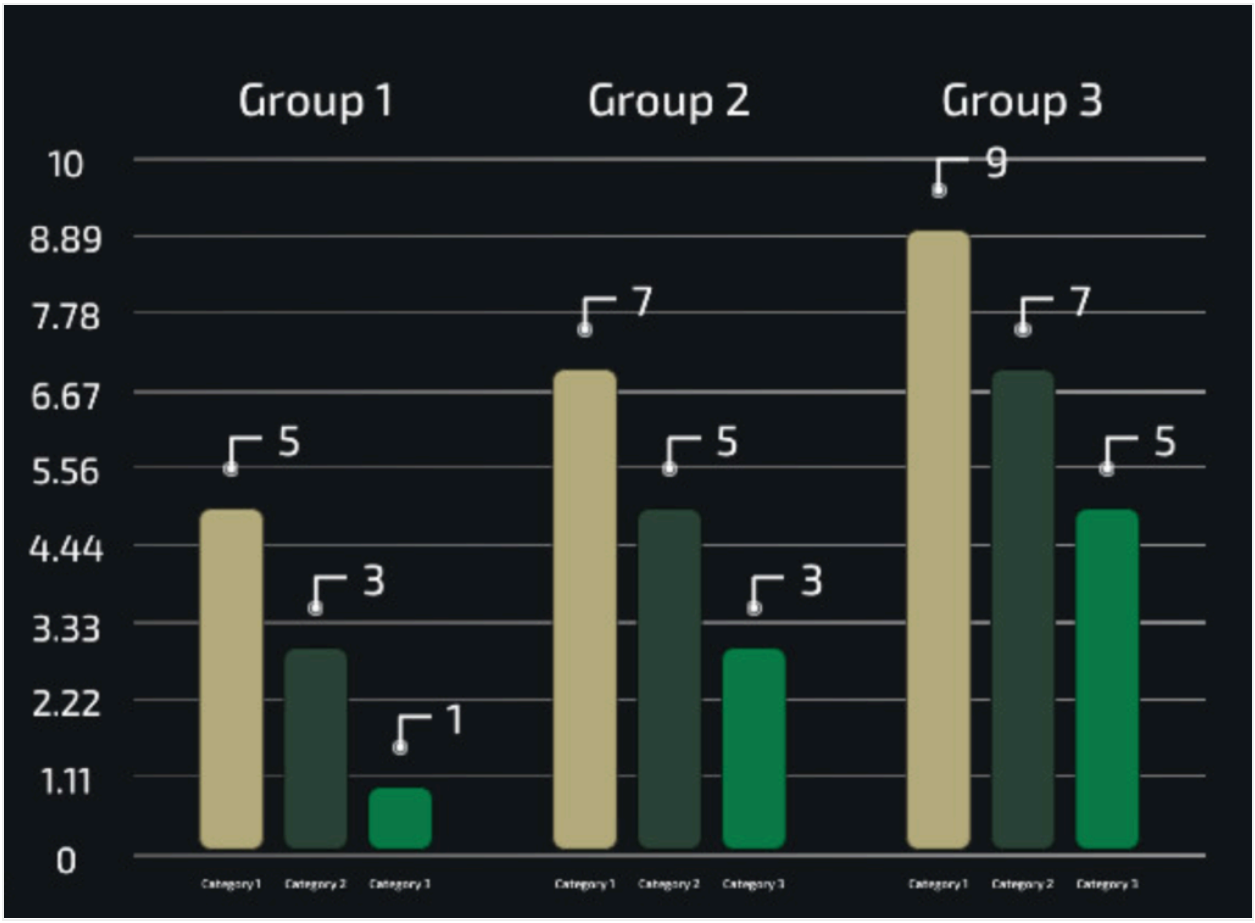
What you can see in the above image is a graph chart with two categories . one is purple and the other is green. Each of these has it’s own data. Here is another example with a bar chart :



Here there are 3 categories , you can see that each of the bars has it’s own unique visual customization and material.

WHAT ARE GROUPS

Groups are used in the bar chart and in the radar chart. Their goal is to group a few categories together. if we consider the bar chart example above , we can add 2 more groups and have this chart :



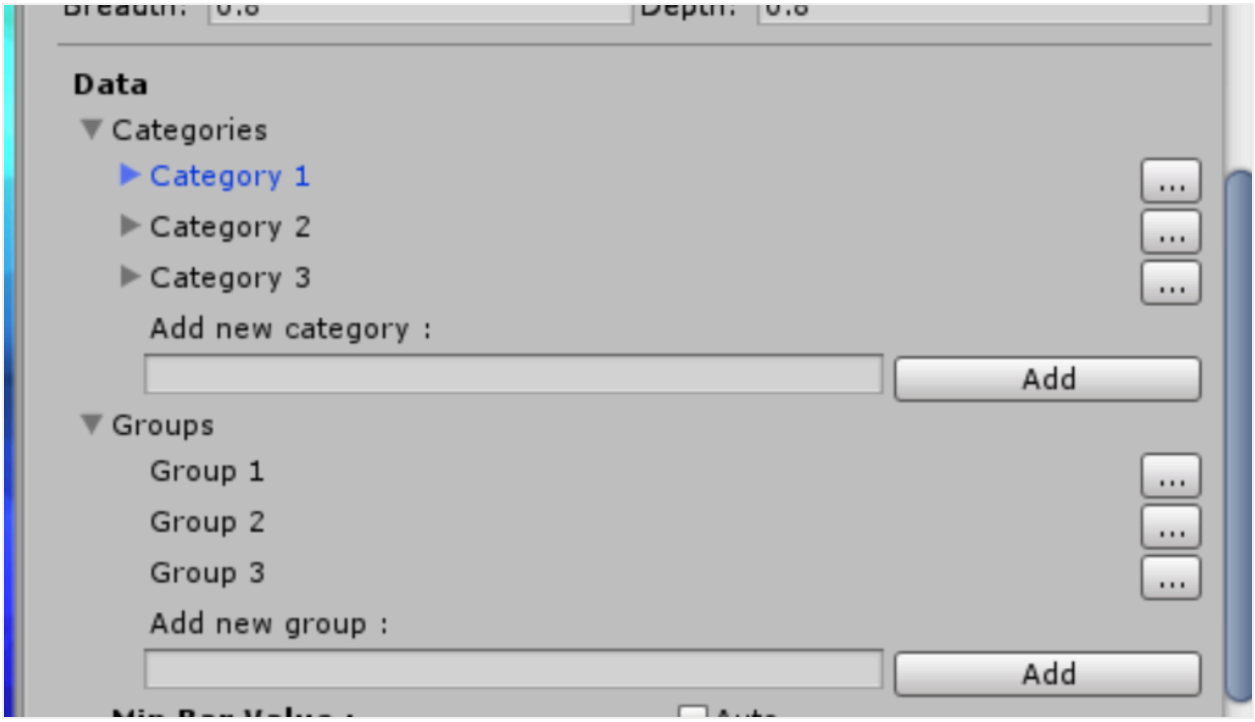
So in here , each new groups adds 3 bars. This allows you to easily add more data and to order it in a way that makes sense

NAMING CATEGORIES AND GROUPS

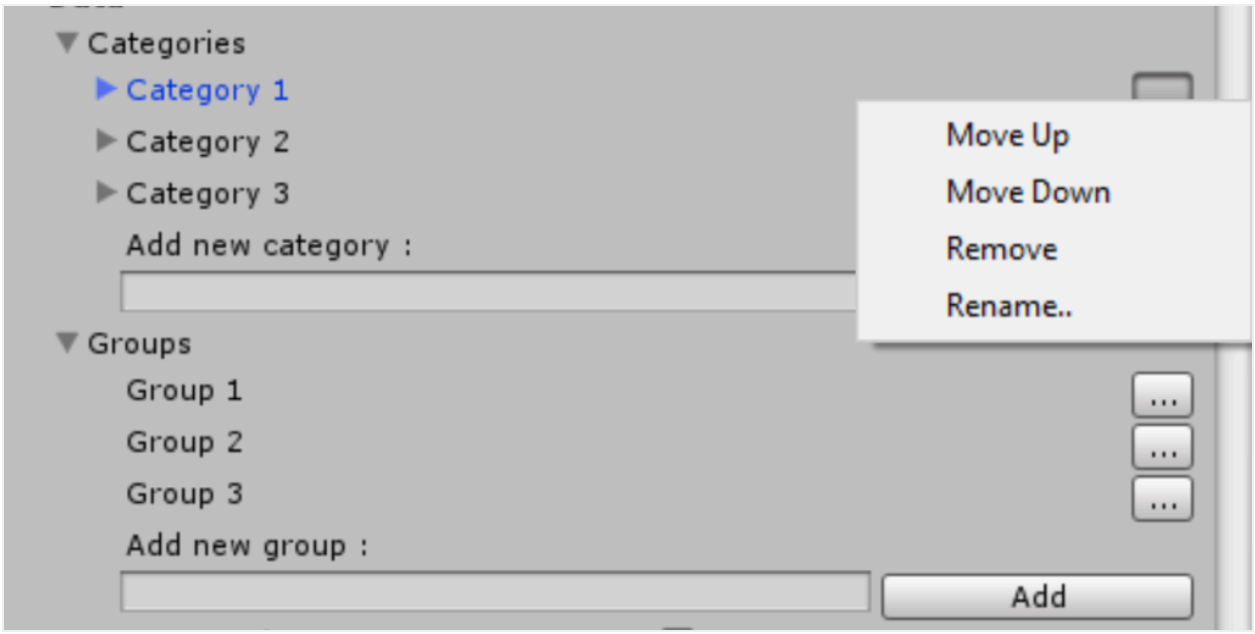
Each category and each group has a unique name that identifies it within a chart. You can use this name to load data into the category or group from script for example. The names of categories and groups are case sensitive and are sensitive to spaces as well. Also , you cannot have two categories of the same name within a chart , and you cannot have two groups with the same name as well.

CATEGORIES AND GROUPS IN THE INSPECTOR WINDOW

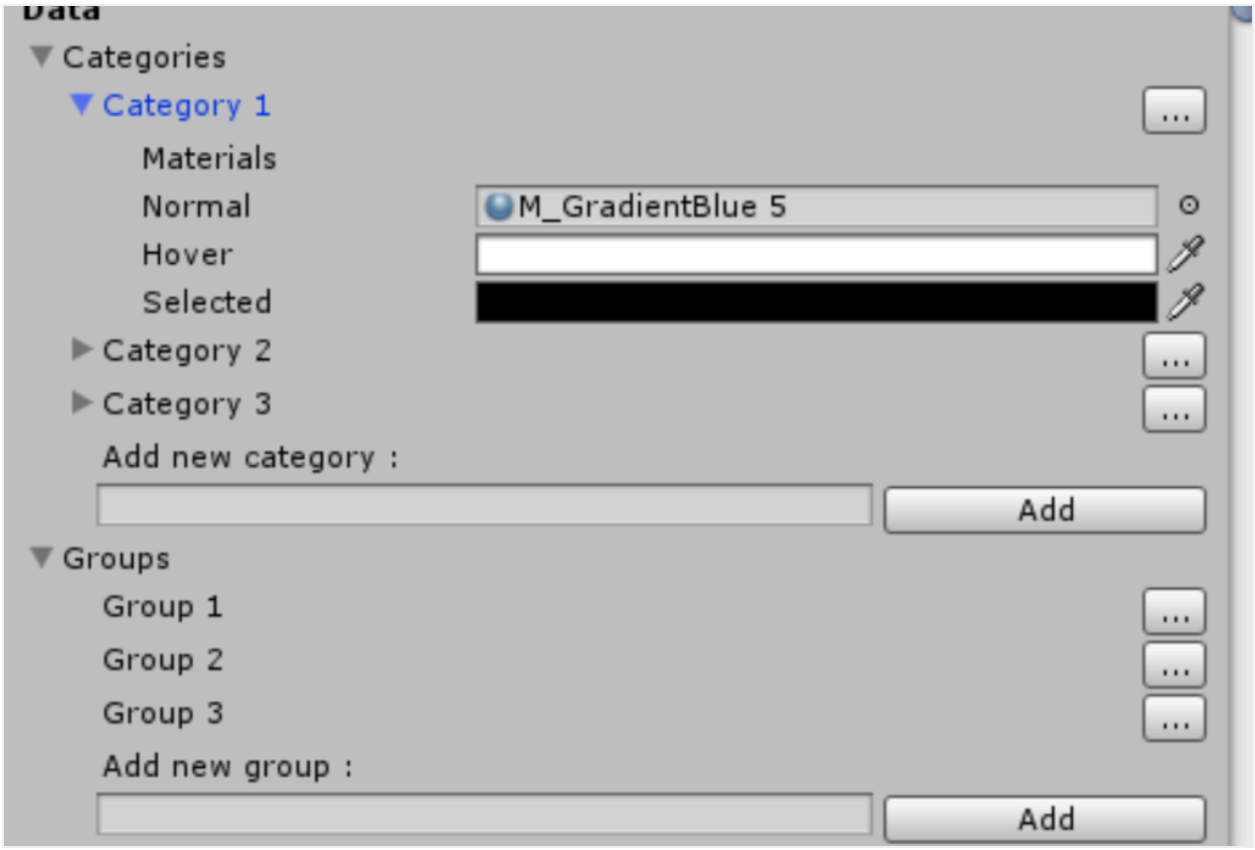
The categories and groups of charts can be set from both script and inspector. To find them in the inspector, go to the chart component and scroll down to the **Data** section



You can type in a new name and add a category or a group. you can also edit the current categories or groups by click the button to the right of them :

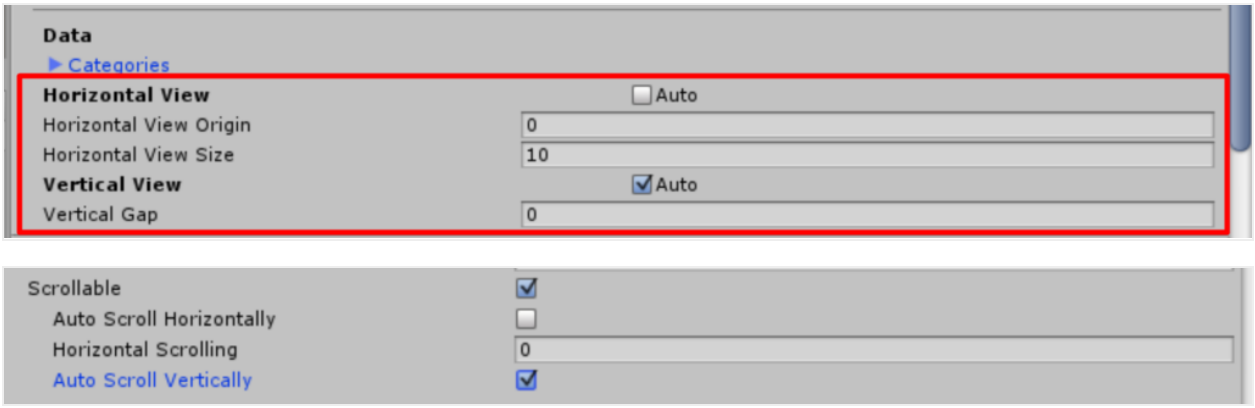


Categories can also be expanded, the settings of categories change according the chart they belong to :



VIEW PORTION BASICS

The view portion can be set from both inspector and code. if you will go to the inspector of a graph or a candle chart you will find these settings:



You can use these settings to control the view window of the chart. If you wish the view portion to fit all the data you can check the **Auto**checkbox.

If you disabled automatic view for an axis. You can set it's properties:

- **Scrollable** check this in order to clip the parts of the chart that are out of bounds
- **Horizontal/Vertical Scrolling** sets the amount of scrolling from the axis origin. use this to navigate inside the chart view.
- **Auto** if this value is true , the other values are ignored and the view is set to fit all the data inside
- **Auto Scroll** makes it so that the view scrolls to last point that was added to the chart. It makes sense to disable **Auto** when using this
- **View Origin** is the beginning of the axis. by default it is 0. If you set it to 1 , this means that the main axis will be place on point 1
- **View Size** is the size of the view portion. so if the graph start at 0 and you set the view size to 10 , all the points between 0 and 10 are visible

SETTING DATES IN THE VIEW PORTION

Sometimes we would like to use dates or time spans instead of numbers. When you need to do so , simply use the time in seconds for that portion. For example : To make a view size of 1 hour , you can set in the value 3600 which is the amount of seconds in an hour.

If you want to set the Scrolling of the chart to 1/1/2019. you can do so by setting it to

1546300800 , This is the unix time stamp of this date.

SETTING THE VIEW PORTION IN CODE

All the setting described in the previous parts of the document can be set using code as well. It can be done like so :

```
graph.DataSource.AutomaticHorizontalView = false;

graph.DataSource.HorizontalViewOrigin = 0;

graph.DataSource.HorizontalViewSize = 1000;

graph.HorizontalScrolling = 200;

graph.AutoScrollHorizontally = false;
```

If you wish to use dates and times in order to set the view , it can be done using the utility methods in the class ChartDateUtillity:

graph.DataSource.HorizontalViewSize = ChartDateUtility.TimeSpanToValue(TimeSpan.FromHours(5));

graph.HorizontalScrolling = ChartDateUtility.DateToValue(new DateTime(2019, 1, 1));

HANDLING CLICK AND HOVER EVENTS

We would sometimes want to have our own custom code that handles events such as user click or hover. Graph and Chart allows you to easily add your own code using Unity Events . This is a simple tutorial that will show you how it's done. This tutorial is based on the script in Script\Utils\InfoBox . You can see it work in some scenes in Graph and Chart, for example in Themes\2d\Bar\Preset2

CREATING A HANDLER FOR EVENTS

before we hook up an event , we need to create the code that handles it. In this example we will use the bar chart , but it applies to all charts , you can handle a bar chart event in the following way:

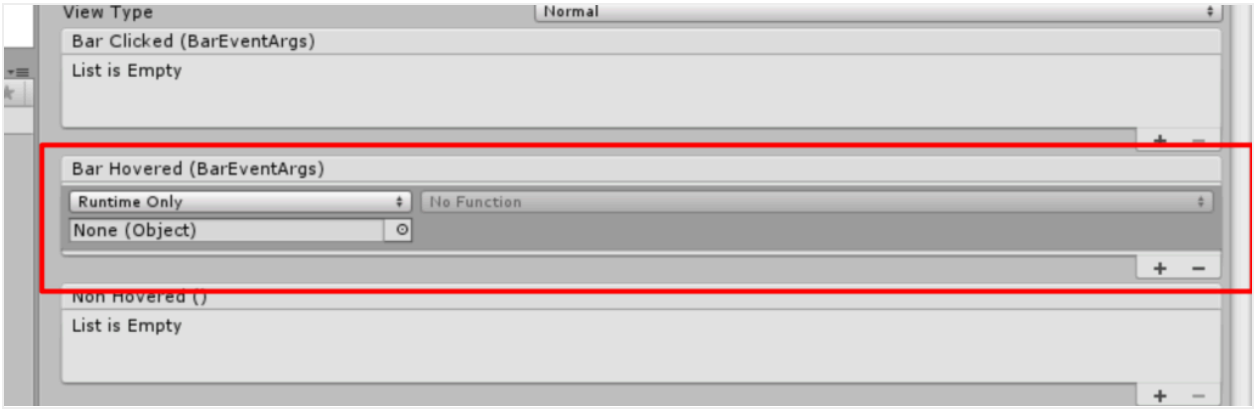
```
   public void BarHovered(BarChart.BarEventArgs args)

    {

        infoText.text = string.Format("({0},{1}) : {2}", args.Category, args.Group, args.Value);

    }
```

Notice that the method you are using has to have void as a return type , and it has to receive only one argument of type BarChart.BarEventArgs . The method also has to be **public** for it to show in the inspector. From within this method you can make use of the data in "args" , args contains information relevant to the specific event that has just happened.

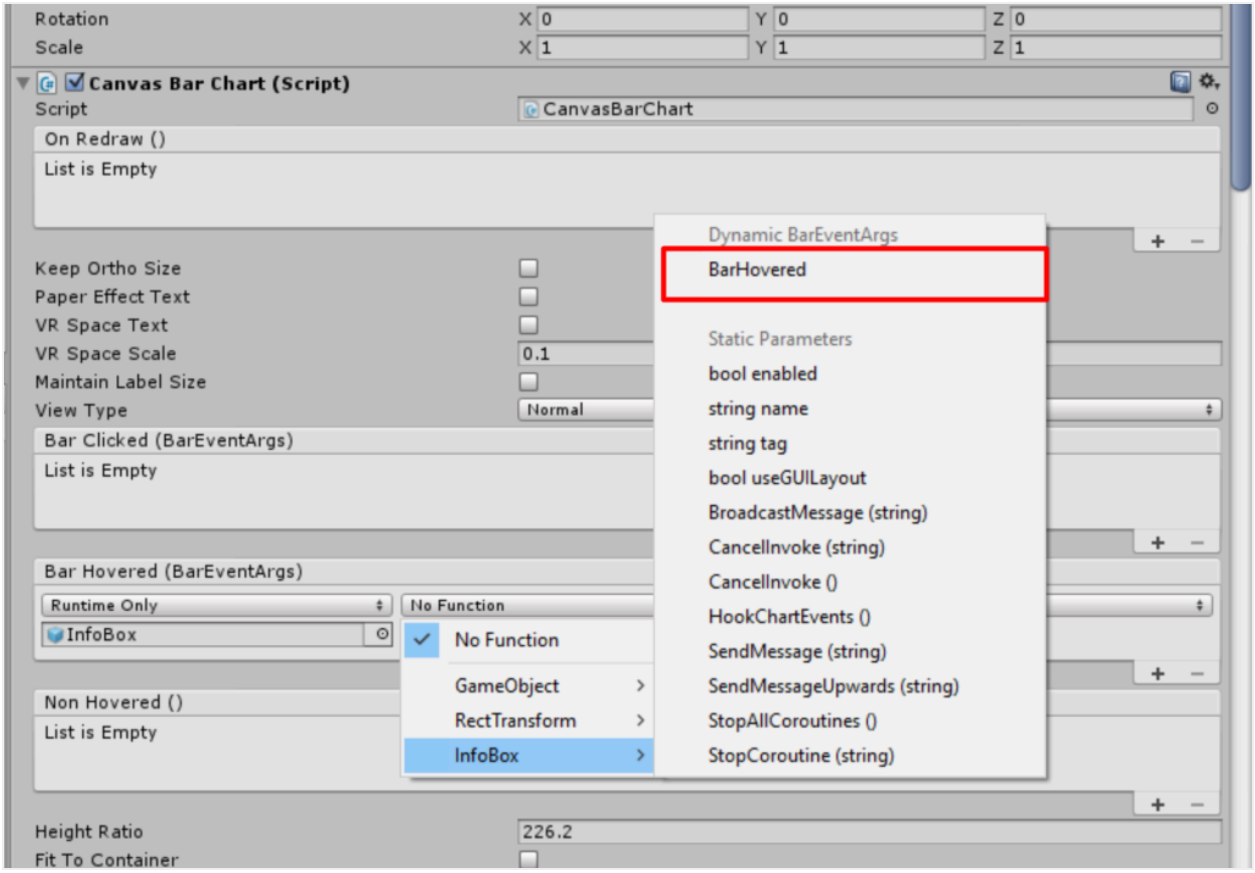
HOOKING THE EVENT

Once we have an event handler method , we need to hook it to the a chart. This way it will be called every time the event occurs. You can hook an event either from the inspector or from code:

Note: the info box events are not meant to be hooked using the inspector , this is a demonstration of how you can set up events from other scripts. Later we will see how the info box hooks events from code.



From the inspector , you can locate the event on the chart object , then drag and drop the handler object :



Now with some use cases we would like to hook the event from code. This can be done like this :

```
    bar.BarHovered.AddListener(BarHovered);
```

once this method is called you will have your method called each time the event happens.

CHART LEAVE EVENT

After the user is done hovering over the chart , we would like to clear the text box we have filled in the method BarHovered. for this purpose there is an important event called NonHovered. This event is called once there is not longer an interaction with the chart. here is an example of how you can set it :

```
   bar.NonHovered.AddListener(NonHovered); // call this method to hook the event
listener
```

```
...
```

```
    void NonHovered()
```

```
    {
```

```
        infoText.text = ""; // clear the textbox
```

```
    }
```

HOW TO MAKE DIFFERENT INTERACTIONS FOR DIFFERENT CHART OBJECTS

A lot of times we would like to make different interactions for different chart items . For example, we would like to load a different UI element whenever a bar is clicked. You can distinguish between bars using the parameters in the event args. Each bar is defined by it’s Group and Category. So we can load UI based on the Group and Category provided in the event args. For graph chart , you can use the category and the index provided in the event args.

EVENT ARGS REFERENCE

in this section we will go through the event args classes and their properties

1. BarEventArgs

```
        public Vector3 TopPosition { get; private set;}
        public double Value { get; private set; }
        public string Category { get; private set; }
        public string Group { get; private set; }
    }
```

- **TopPosition** is the position of the top part of the bar item. The position is in world space or in canvas coordinates depending on the type of chart
- **Value** is the value or amount set for the bar item.
- **Category** is the category to which the bar belongs to
- **Group** is the group to which the bar belongs to
-

2. GraphEventArgs

```
        public float Magnitude { get; private set; }
        public int Index { get; private set; }
        public string XString { get; private set; }
        public string YString { get; private set; }
        public Vector3 Position { get; private set; }
        public DoubleVector2 Value { get; private set; }
        public string Category { get; private set; }
        public string Group { get; private set; }
    }
```

- **Magnitude** is the size of the point for a bubble chart or a size based graph
- **Index** is the index of the graph point that recived the event. This index is into the data array of the graph category
- **XString/YString** are human readable strings that portray the x and y values according to the graph chart's settings

- **Position** is the position where the event took place. It is either in world space coordinates or canvas coordiantes depending on the type of graph chart.
- **Value** is the value of the point that is interacted with
- **Category** is the category of the point that is interacted with
- **Group** is not used and is reserved for future use

3. PieEventArgs

```
    {
        public double Value { get; private set; }
        public string Category { get; private set; }
    }
```

- **Value** is the value of the pie item that was interacted with.
- **Category** is the category of the pie item that was interacted with.

4.RadarEventArgs

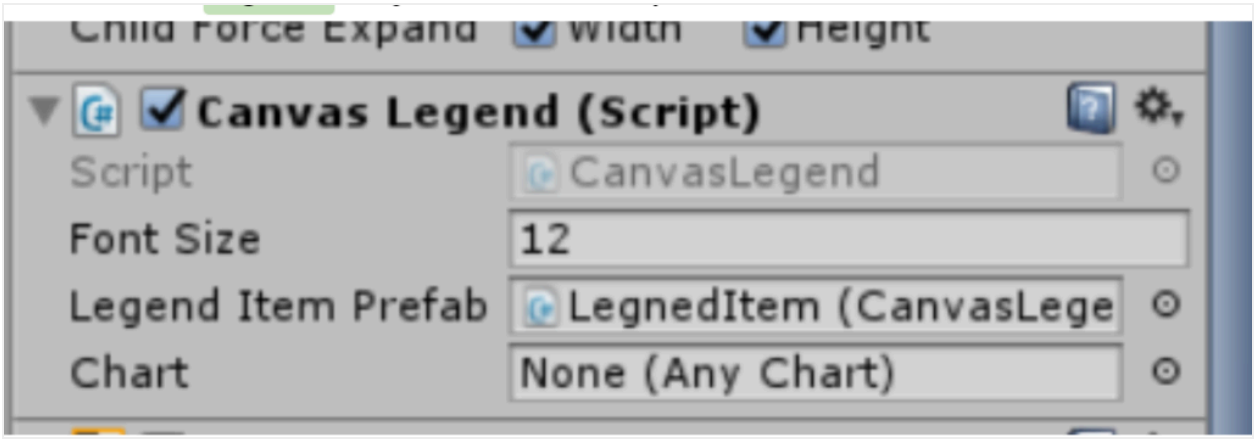
```
        public int Index { get; private set; }
        public string Category { get; private set; }
        public string Group { get; private set; }
        public double Value { get; private set; }
        public Vector3 Position { get; private set; }
    }
```

- **Index** is the index of the item that was interacted with
- **Category** is the category of the item that was interacted with
- **Group** is the group of the item that was interacted with
- **Value** is the value of the item that was interacted with
- **Position** is the position where the interaction took place. This is the positions in world space coordinates or in canvas coordinates depending on the type of radar chart being used

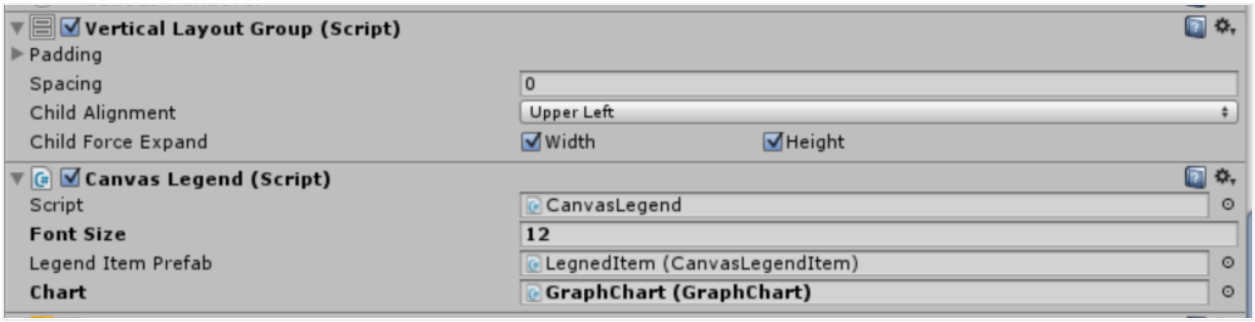
LEGENDS

Legends can be created easily for any chart. add a legend to your scene by doing the following:

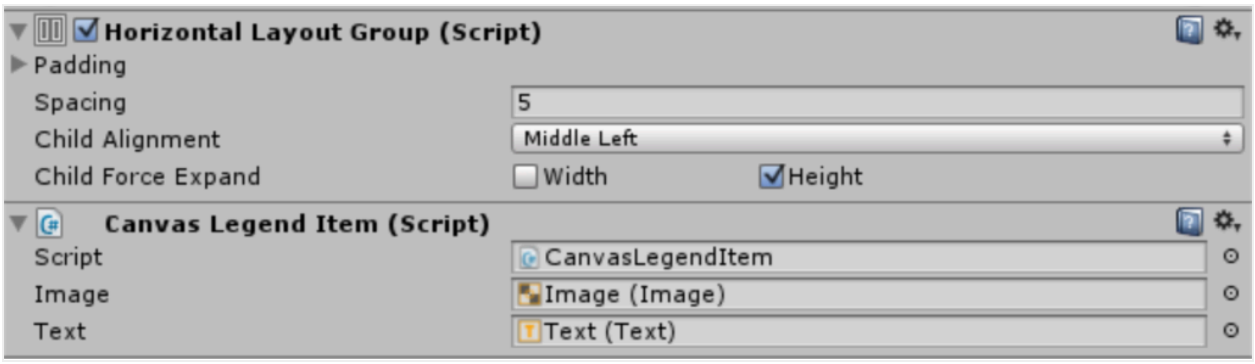
1. Make Sure there is a canvas in the scene
2. Go the Tool menu and select Chart->Legned . A new legend object will be added to the scene
3. Select the legend object in the scene view
4. Drag a chart game object into the chart field
5.



STRUCTURE OF THE LEGEND OBJECT



The legend is composed with unity layout groups. You can create your own legend prefab and use other layout groups or options. Also you can change **Legend Item Prefab** and use your own prefab instead. When creating a new legend item prefab it is important to add a **CanvasLegendItem**component to it. It is also important to assign it's image an text fields :

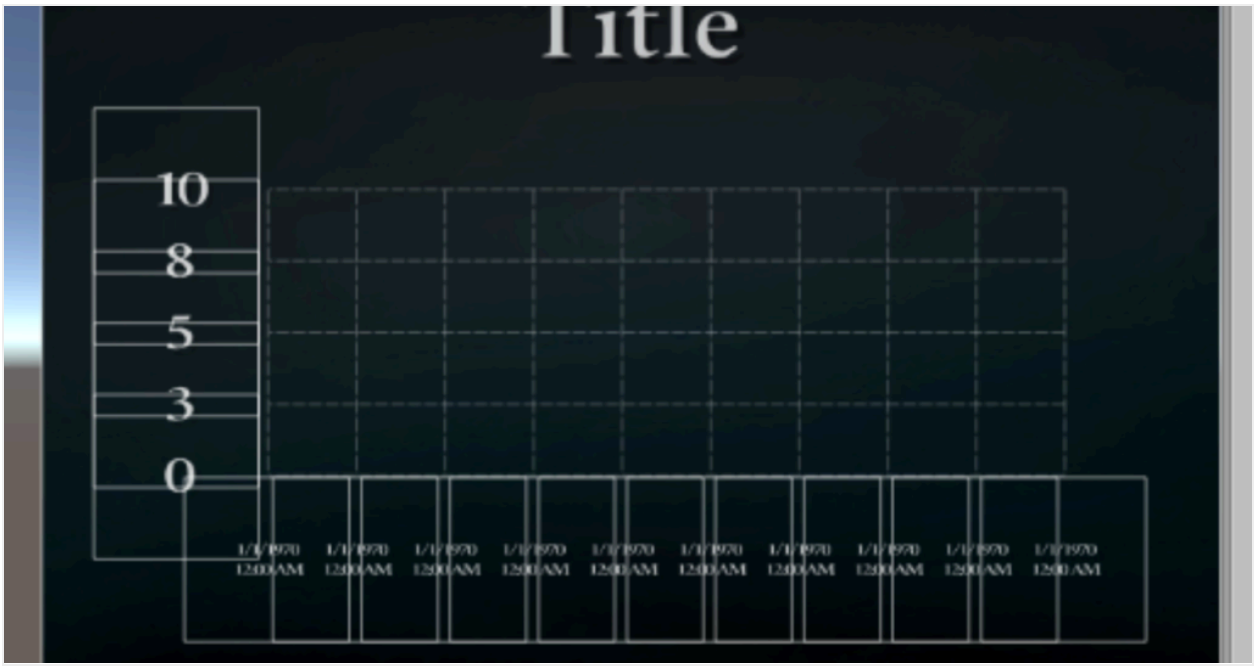


AXIS – QUICK START

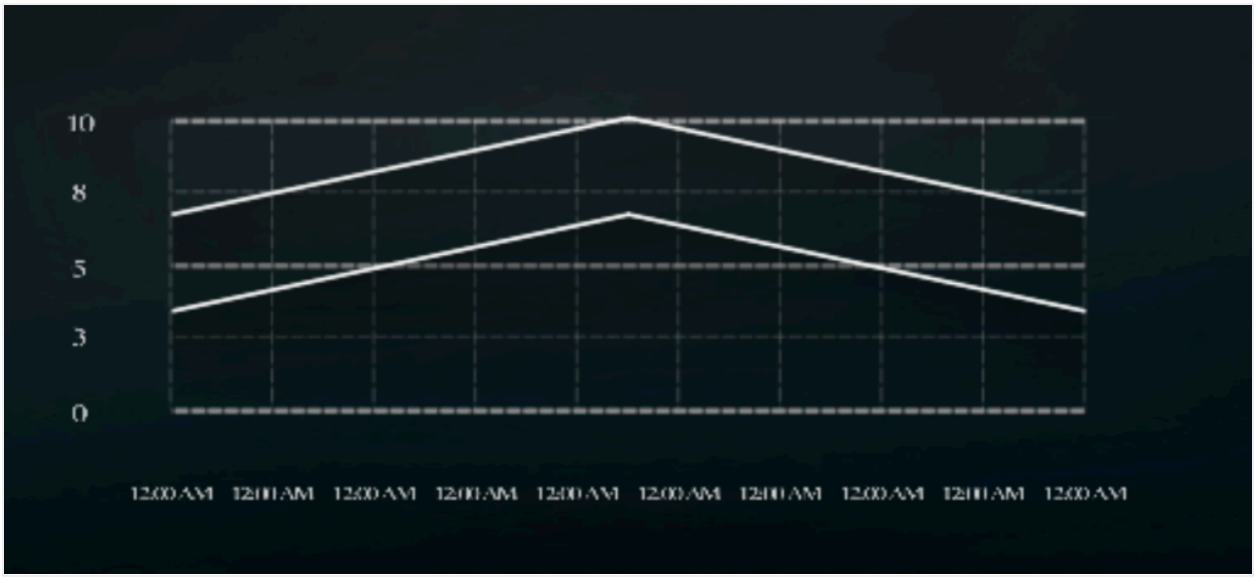
You can add axis to a chart by adding an axis component to the charts gameobject :

The available axis type are Vertical Axis and Horizontal Axis , Notice that some charts would not make since with some axis , for example the pie chart. So the axis can only be added to a chart that supports axis.

Once the axis is added , you will see them on the chart immediately :

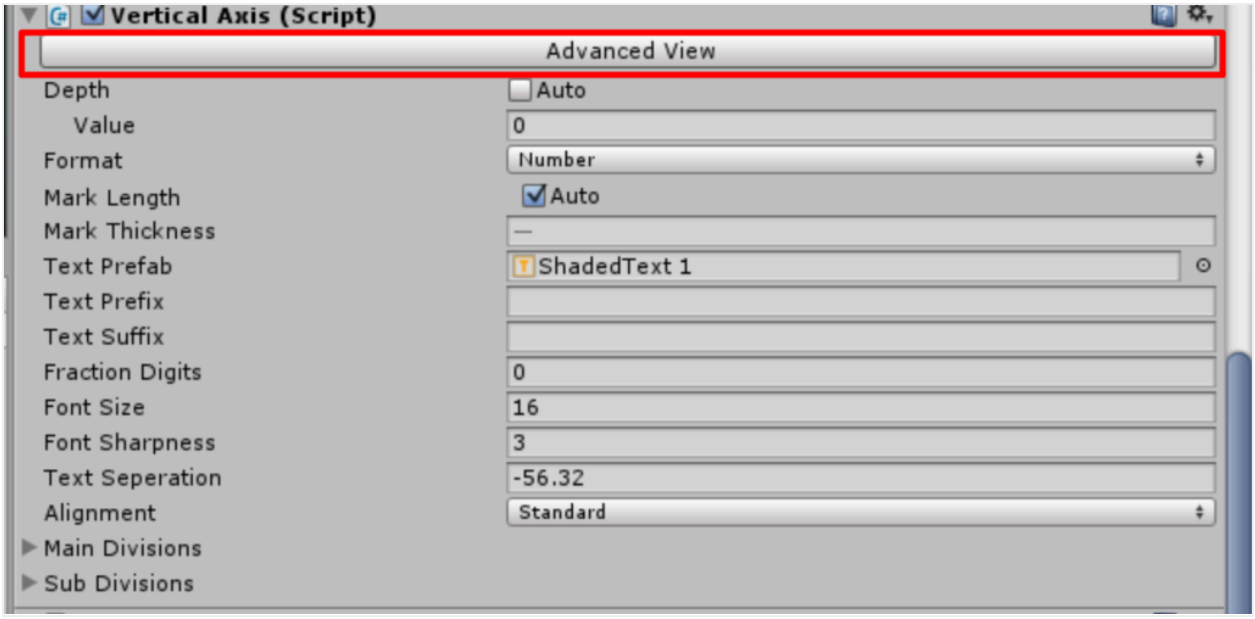


The lines that make the axis are called divisions in Graph and Chart. Axis can be used with main division and sub divisions , so you can create neat axis designs like this :



notice how the main divisions are thick then the sub divisions

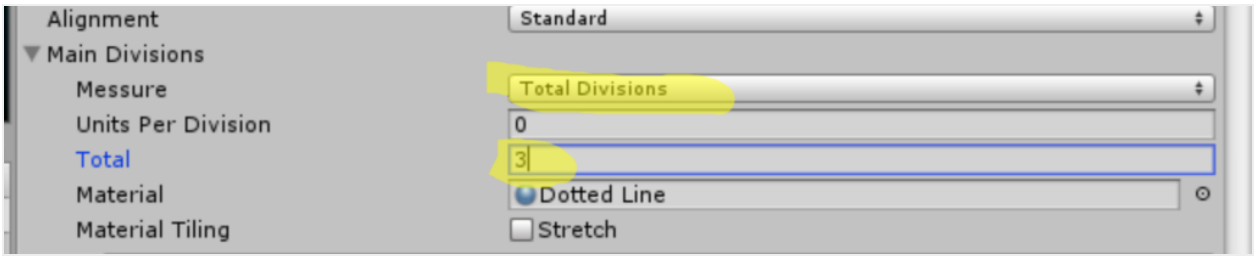
You can use simple view if you do not wish to have different style for each type of division , or to click on advanced view if you want full control of the design :



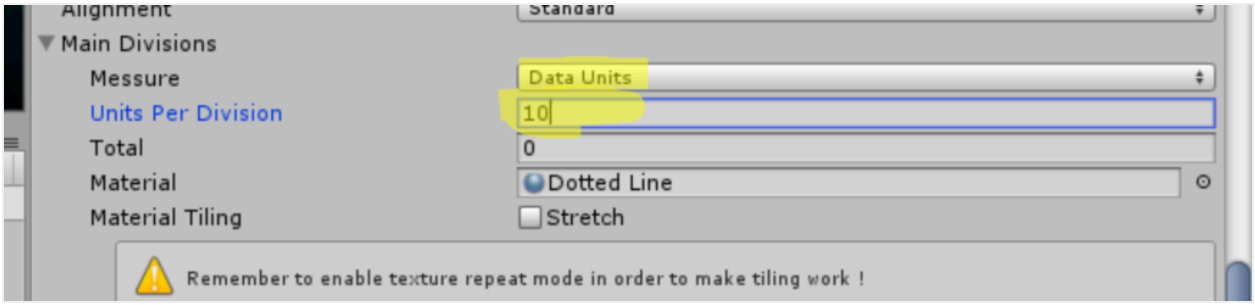
SETTING UP MAIN DIVISIONS

There are two options to setting up main divisions. You can set them so they have a fixed data unit , or as the amount of total division.

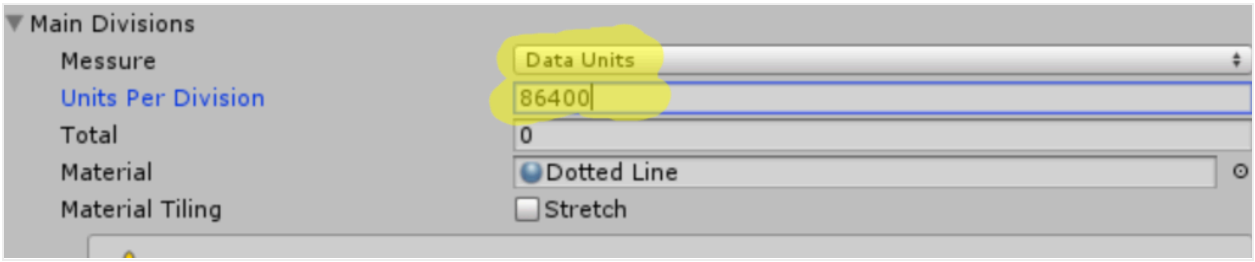
Example: If you would like to have exactly 3 main divisions , you can set **Total** to 3 and **Messure** to **Total Divisions**



Example 2: if you would like to have a main division each 10 units , you can set **Units Per Division** to 10 and select **Data Units**

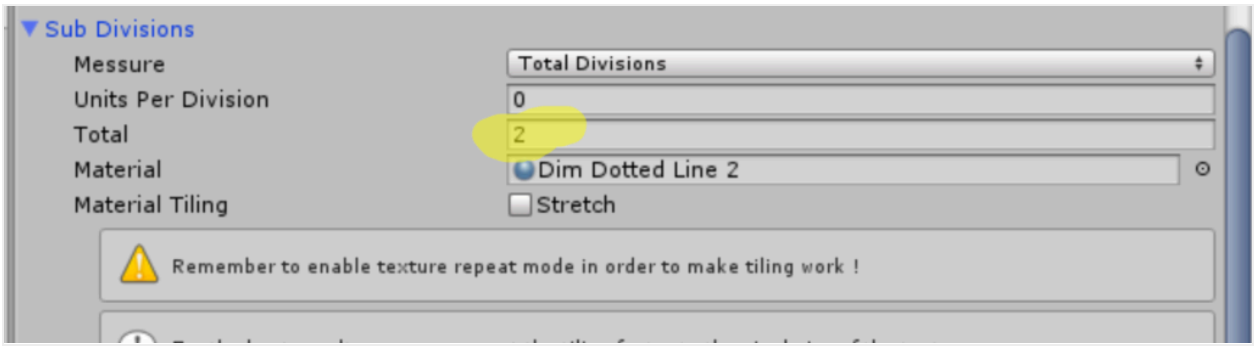


Example 3: lets say we are using date format for our chart and we would like to have a main division each 1 day. Then you can write the number of seconds in a day



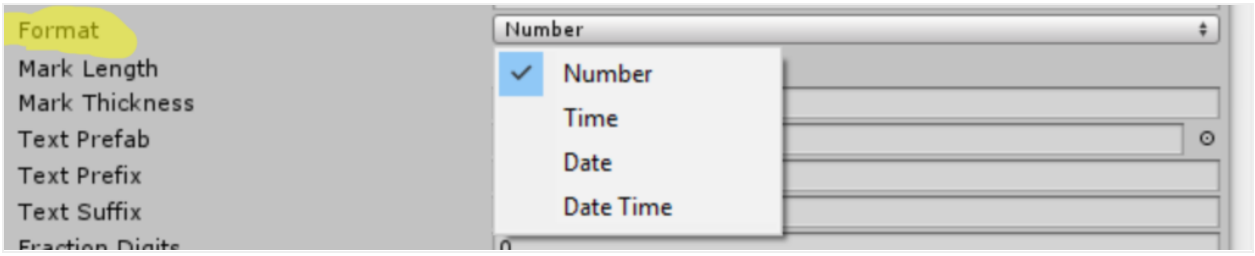
SETTING UP SUB DIVISIONS

sub divisions are created for each main division. So if we have 2 main divisions and 3 sub divisions we have a total of 6 divisions. You can also set them to 0 and have only main divisions. setting up sub divisions can be done like so :

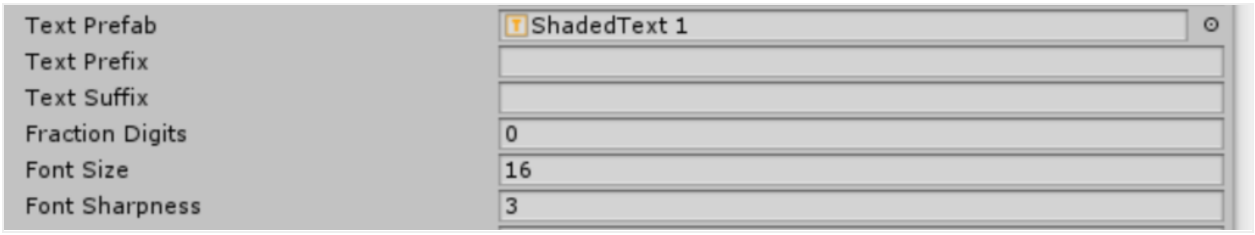


SETTING UP TEXT LABELS

The text labels can be of dates, times and numbers . you can choose between them by setting the format property of the axis:



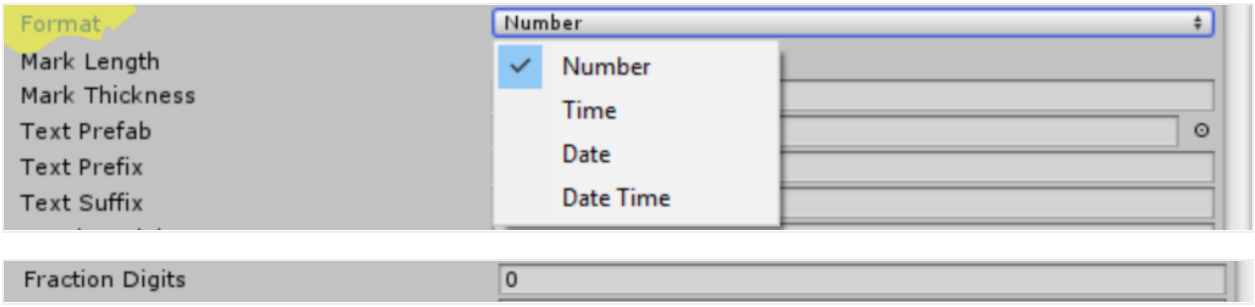
both main and sub divisions can have text label showing the value their positioned in. You can configure these text labels in the inspector:



Also notice that you can add a text prefix or suffix. This can be used to make the values more understandable to the end user. For example you can set the Suffix to “USD” , “$” ,”Km\H” etc.

DATE, TIME AND FORMAT

In many cases we would like to show dates or times in the chart axis. This can be done by setting the axis format using the Inspector :



- **Date** means only a date is shown , for example 1/1/2019
- **Time** means only the time is shown , for example 1:00 AM
- **Date Time** shows both date and time , for example 1/1/2019 1:00 AM
- **Number** shows a plain number. you can control how many fraction digits are shown by setting **Fraction Digits**

PASSING DATES INTO METHODS AND THE VIEW PORTION

Under the hood , Graph and Chart uses only doubles to specify values. When a date is passed to graph and chart , it is converted to a double. Most methods of graph and chart contain overloads that allow you to use dates without thinking about converting them. You do have to know how to use dates and times in C# . Check out the following useful links:

- Date Time Structure
- Time Span Structure

When writing dates and times in the inspector , you can specify their value in seconds.

- When setting a date , simple use it's Unix TimeStamp so for example the date 1/1/2019 can be set by typing
  1546300800
- When setting a time span , simply write it in seconds. For example an hour is 3600

if you need to convert numbers and dates from within the code, You can use the class **ChartDateUtillity**

```
graph.DataSource.HorizontalViewSize =
ChartDateUtility.TimeSpanToValue(TimeSpan.FromHours(5));

graph.HorizontalScrolling =  ChartDateUtility.DateToValue(new DateTime(2019, 1, 1));

DateTime d = ChartDateUtility.ValueToDate(g.HorizontalScrolling);
```

FORMATING DATES AND TIMES

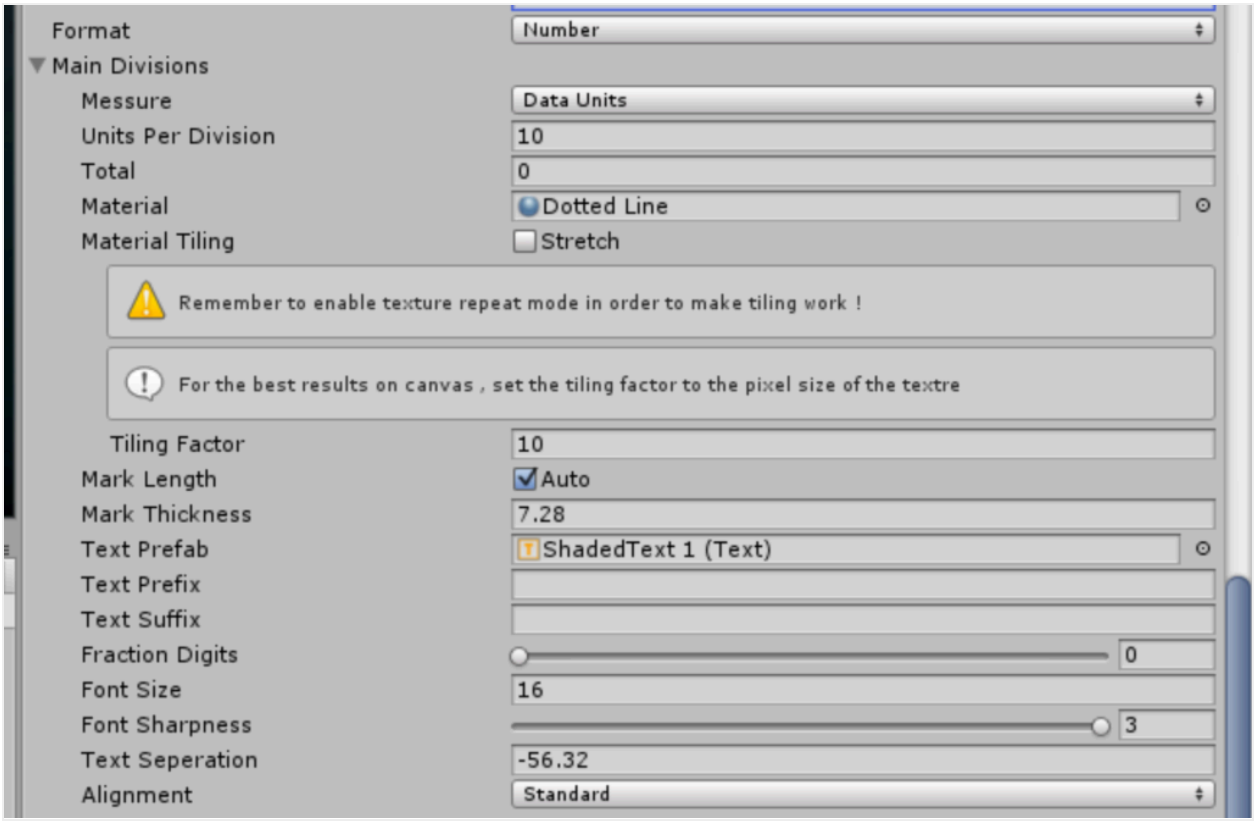
Graph and Charts uses the default formatting of the end user's system to display dates. If you wish to change this behviour you can override it in ChartDateUtillity.cs

```
public static string DateToTimeString(DateTime dateTime)
{
    return dateTime.ToString("t");
}

public static string DateToDateString(DateTime dateTime)
{
    return dateTime.ToString("d");
}
public static string DateToDateTimeString(DateTime dateTime)
{
    return string.Format("{0}{1}{2}", dateTime.ToString("d"), Environment.NewLine, dateTime.ToString("t"));
}
public static DateTime ValueToDate(double value)
```

you can change these methods to use your own formatting , [learn more about date formats in c#]

AXIS SETTINGS

This article covers all the settings of the axis components. it is recommended to read [Axis – Quick Start] before reading this article.



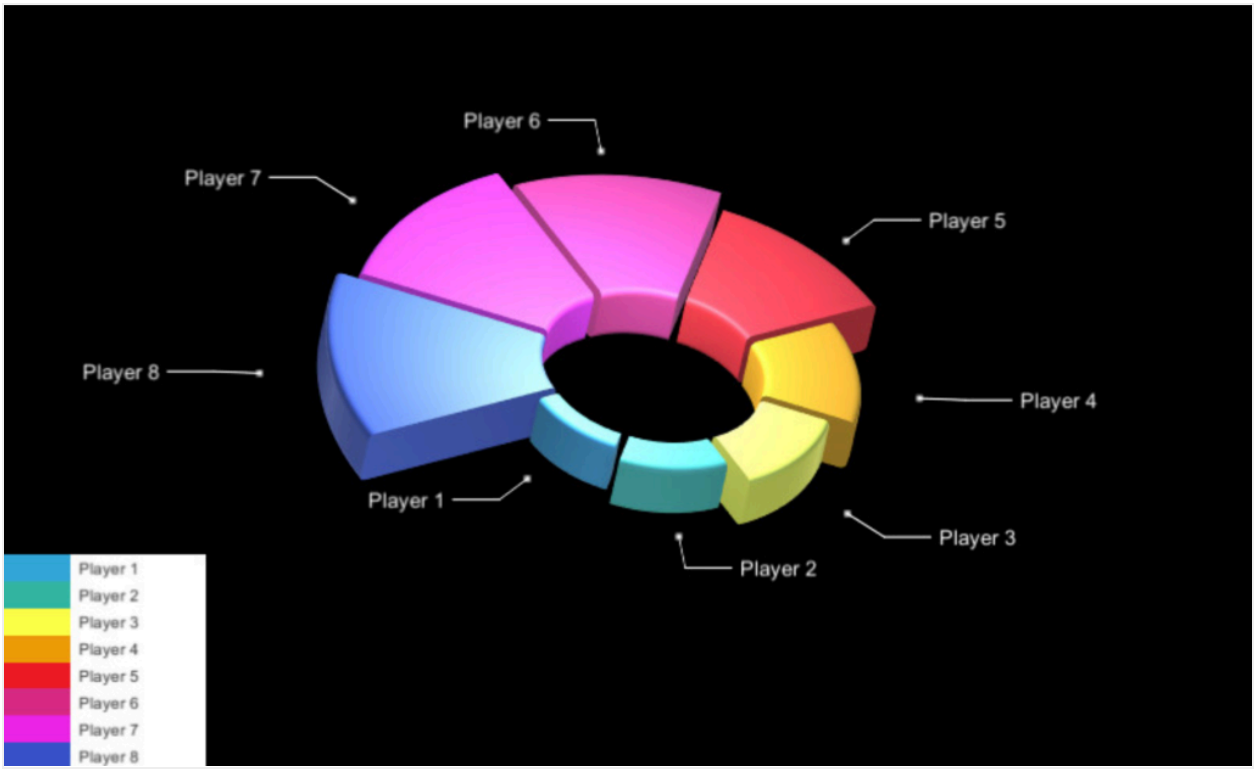
- **Format** select between date,date time , time and number. the selection will affect to format of the item labels of the chart. see Date, Time and Format
- **Measure** is the measure by which divisions are created. **Total divisions** divides the axis into an exact amount of divisions**, Data Units** Allows you to specify a unit amount for each division**.** See Axis – Quick Start for more information about measures.
- **Units Per Division/Total** are the parameter used for the **Measure** method specified See Axis – Quick Start for more information about measures.
- **Material** is the material used for line drawing
- **Material Tiling** defines how the material is tiled along the axis line. You can use material tiling to create effects such as dotted lines , or line patterns. See How to use material tilling
- **Mark Length** sets the length of the axis lines. The default is that the line is drawn across all the chart. If you wish to specify another length , disable the **Auto** check box
- **Mark Thickness** sets the thickness the axis lines.

- **Text Prefab** sets the prefab used for creating the axis labels. If no prefab is set , the default one is used. The text prefabs are located in Prefabs/Text
- **Text Prefix/Suffix** sets a prefix and a suffix for the text displayed in the labels. This is useful when you want to help the user make sense of the axis data. You can for example set it to “USD” ,”$”, “KM/H” .
- **Fraction Digits** sets the amount of allowed fraction digits for numbers in the chart. all values are rounded to fit into this number. If you want to display integers only you can set it to 0
- **Font Size** is the font size of the text labels
- **Font Sharpness** tweaks the sharpness of the font of the item label.
- **Text Separation** is the distance of the axis labels from their base position. The higher this value , the further the labels are for the axis.
- **Alignment** sets the base position of the label.

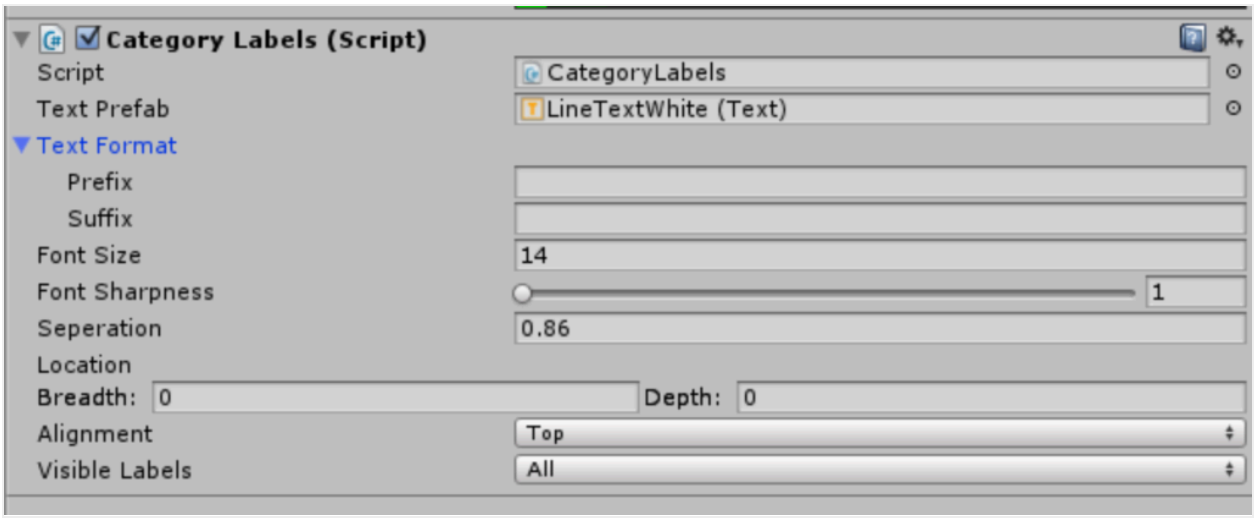
All of the these properties can be changed from script as well. see Obtaining a chart object for scripting

CATEGORY LABELS

On some use cases , we would like to make it clear to the user which part of the chart corresponds with each category. for example in the pie chart :

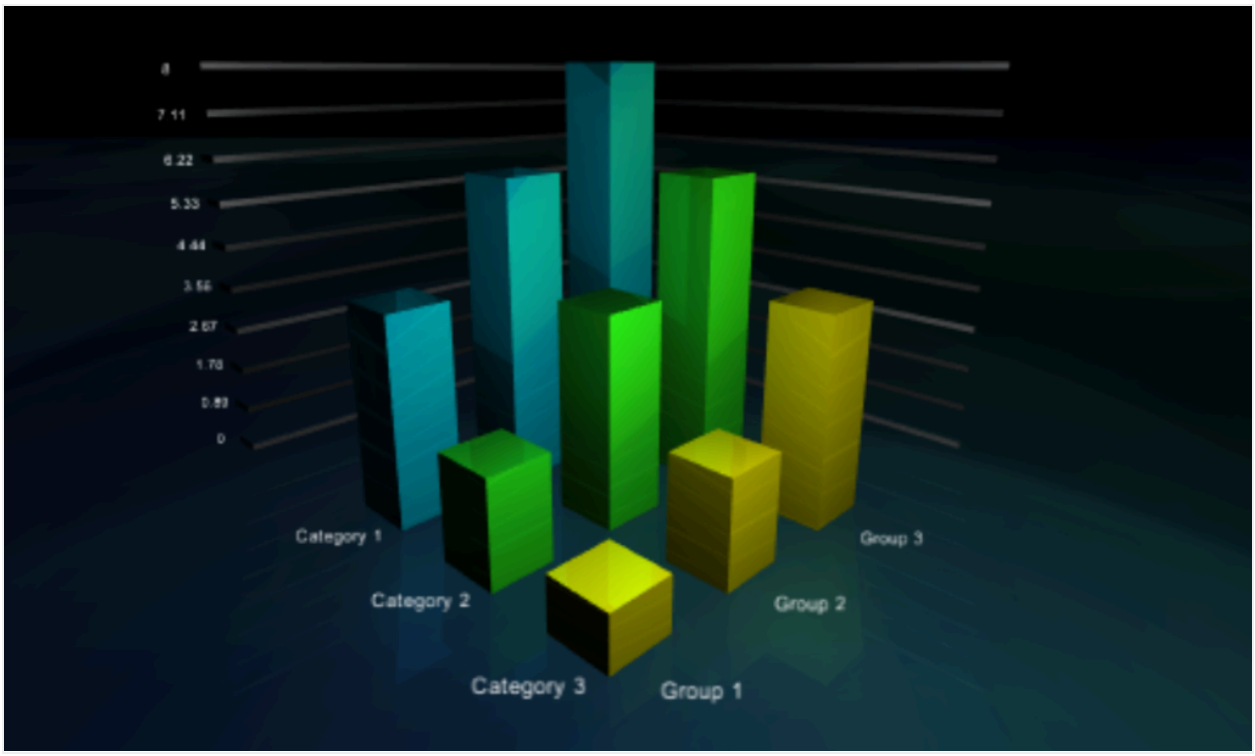


You can do that by adding a Category labels component into your chart's game object. This article is a reference to the properties of category labels.



- **Text Prefab** sets the text prefabs that is used with the category labels. You can find many text prefabs under the folder **Prefabs/Text .** some text prefabs contain additional graphics , such as the lines in the image above

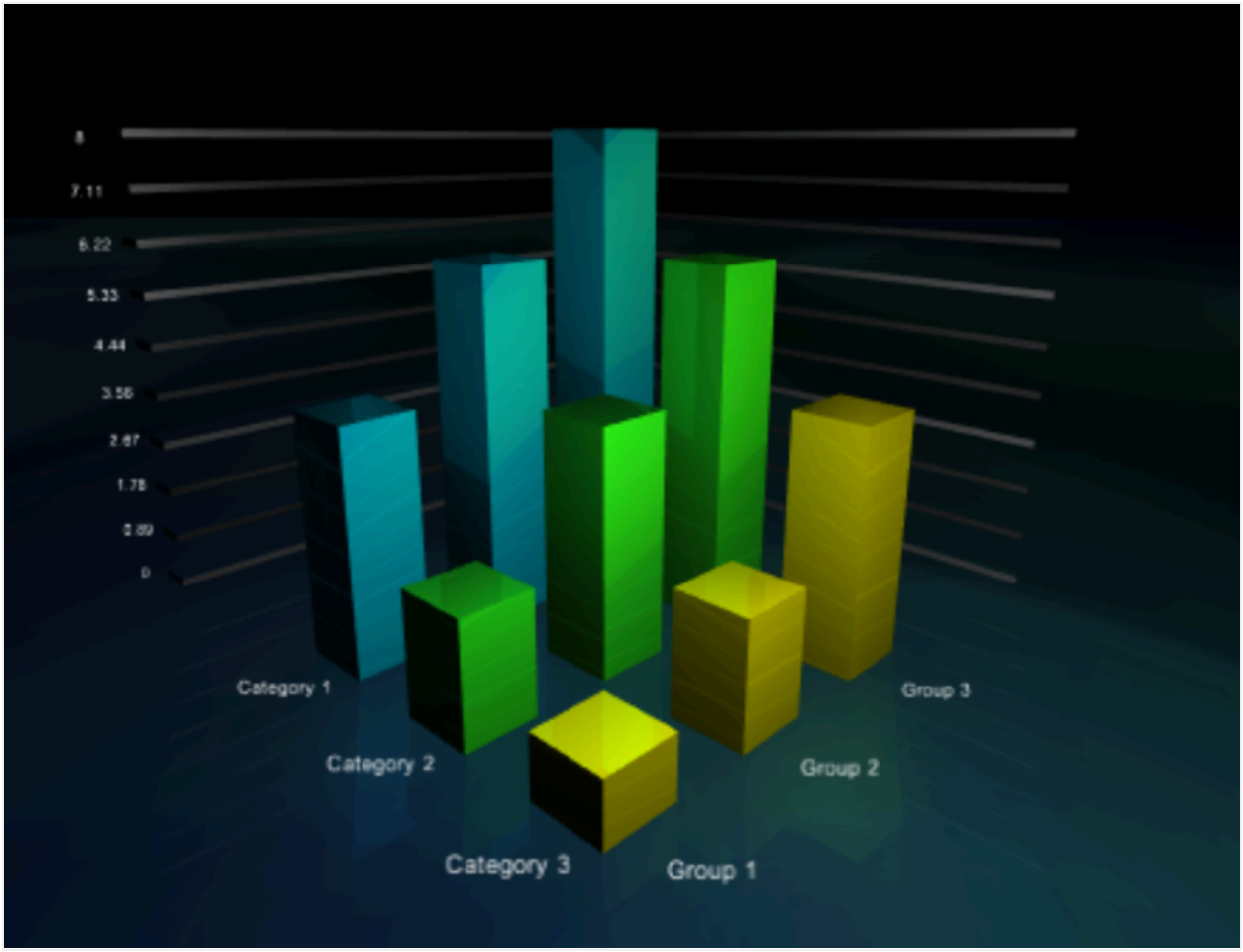
- **TextFormat Prefix/Suffix** allows you to set a suffix and prefix to the category name. This can help the user understand more about the chart. You can set this to a string such as “Category:”
- **Font size** is the size of the font used for the category labels.
- **Font sharpness** allows you to tweak the sharpness of the category labels.
- **Separation** is the separation of the category labels from their base position. the larger this value is , the further the label is from the chart.
- **Location** completes the **Separation** property by allowing you to set separation on all coordinate axis.
- **Alignment** selects the base position of the labels. **Top** means the labels are at the edge the chart item , **Base** means they are at the base.
- **Visible Labels** controls which labels are visible. On some use cases we would not like to show all of them. **All** means all labels are shown, **First Only** means only the first category label for each group is shown. This would look like this :
-



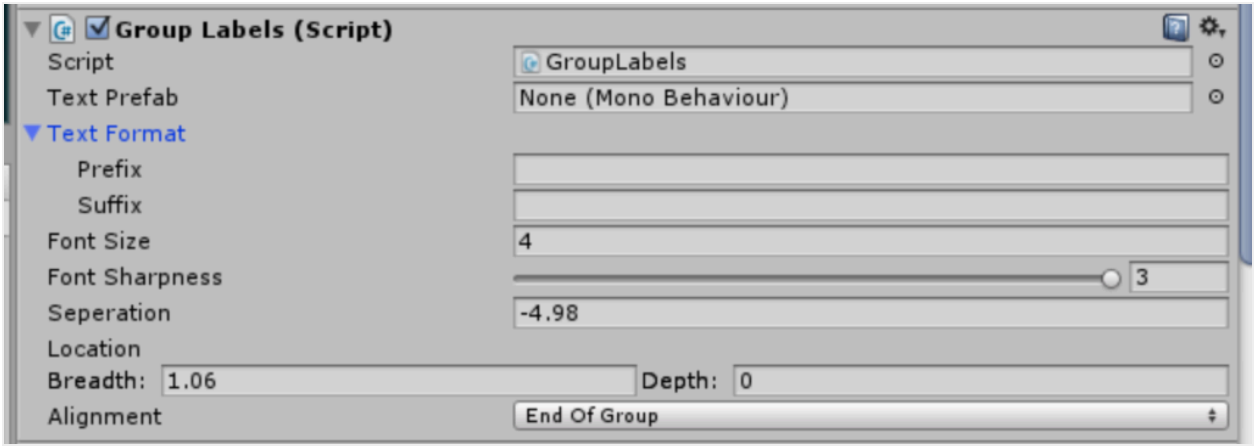
The labels are only visible for the first bar in each group. If we would have chosen **All** we would have a category label on each individual bar

GROUP LABELS

On some use cases , we would like to make it clear to the user which part of the chart corresponds with each group. for example in the bar chart :



You can do that by adding a Group labels component into your chart’s game object. This article is a reference to the properties of group labels.

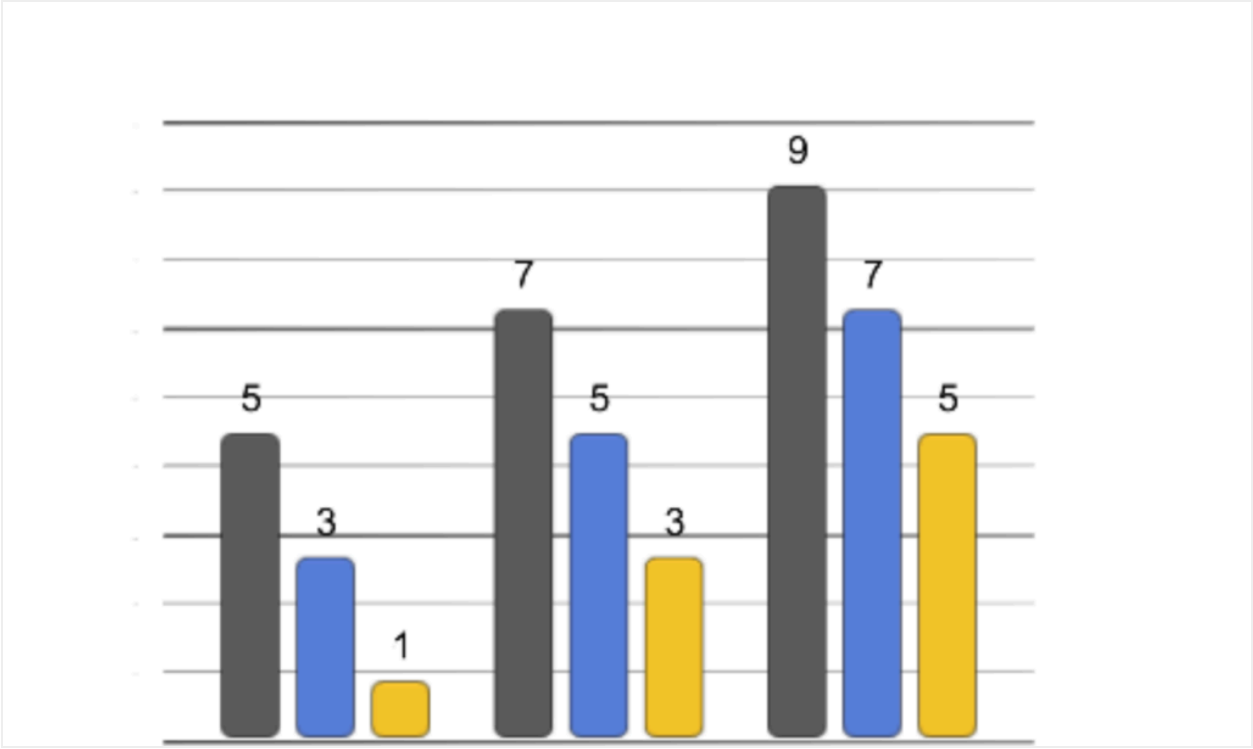


- **Text Prefab** sets the text prefabs that is used with the group labels. You can find many text prefabs under the folder **Prefabs/Text .** some text prefabs contain additional graphics.
- **TextFormat Prefix/Suffix** allows you to set a suffix and prefix to the group name. This can help the user understand more about the chart. You can set this to a string such as “Group:”
- **Font size** is the size of the font used for the group labels.
- **Font sharpness** allows you to tweak the sharpness of the group labels.
- **Separation** is the separation of the group labels from their base position. the larger this value is , the further the label is from the chart.
- **Location** completes the **Separation** property by allowing you to set separation on all coordinate axis.
- **Alignment** sets the base position of the group labels
    - **Center/End of Group/Beginning of Group** shows one label for each group. The label is positioned at the center , end or beginning
    - **Bar Top / Bar Bottom** shows a group label for each bar . The label is positioned either at the top of the bar or the bottom
    - **First Bar** shows a label on the first bar of each group.
    - **Alternating Sides** shows a label either on the first bar or last bar of each group. This can be used to make the text more readable on charts with many bars

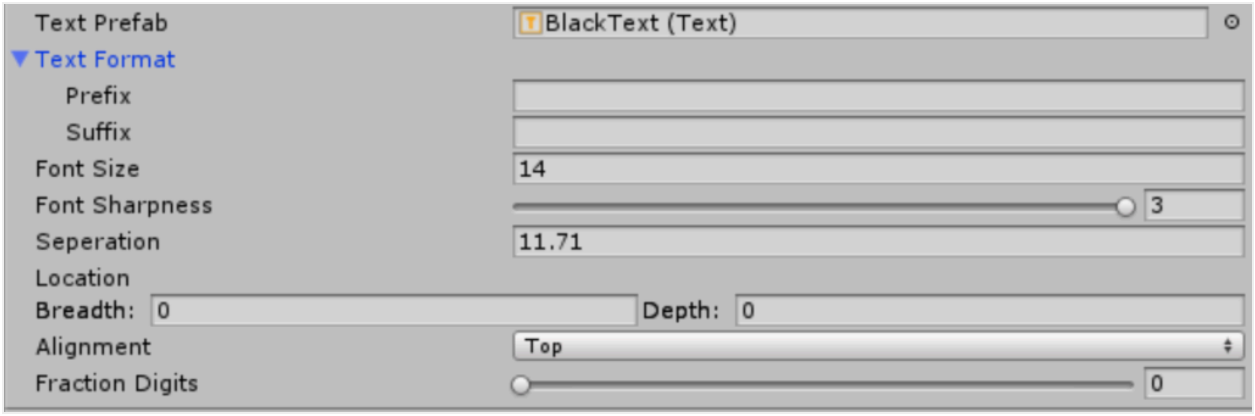
All the properties described in this article can also be changed from script Obtaining a chart object for scripting

ITEM LABELS

On some use cases , we would like to show the values of chart items. for example:



You can do that by adding an Item labels component into your chart's game object. This article is a reference to the properties of item labels.

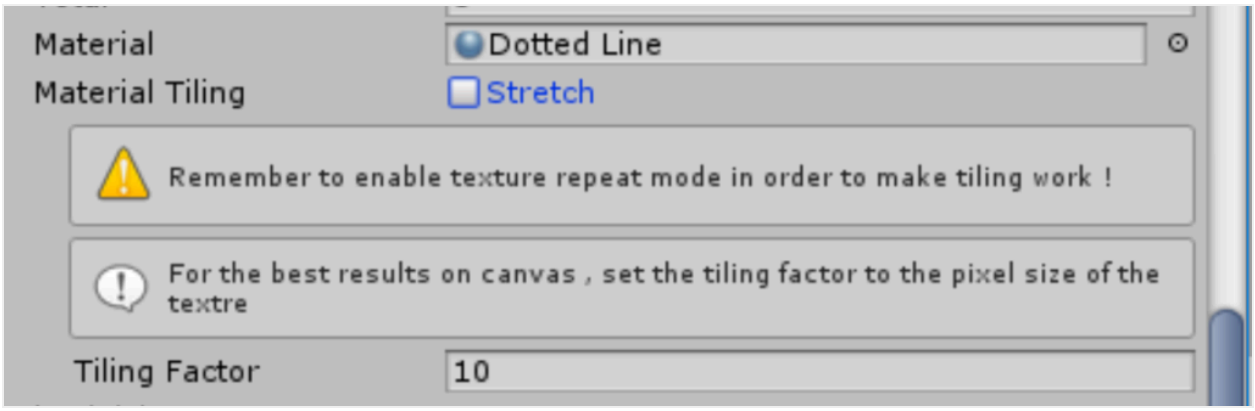


- **Text Prefab** sets the text prefabs that is used with the item labels. You can find many text prefabs under the folder **Prefabs/Text** some text prefabs contain additional graphics.
- **TextFormat Prefix/Suffix** allows you to set a suffix and prefix to the item name. This can help the user understand more about the chart. You can set this to a string such as “Item:”
- **Font size** is the size of the font used for the item labels.
- **Font sharpness** allows you to tweak the sharpness of the item labels.
- **Separation** is the separation of the item labels from their base position. the larger this value is , the further the label is from the chart.
- **Location** completes the **Separation** property by allowing you to set separation on all coordinate axis.
- **Alignment** selects the base position of the labels. **Top** means the labels are at the edge the chart item , **Base** means they are at the base.
- **Fraction digits** is the number of fraction digits shown in the label. The real number is rounded to fit the specified number , you can set this to 0 to show only integers.

All the properties described in this article can also be changed from script Obtaining a chart object for scripting

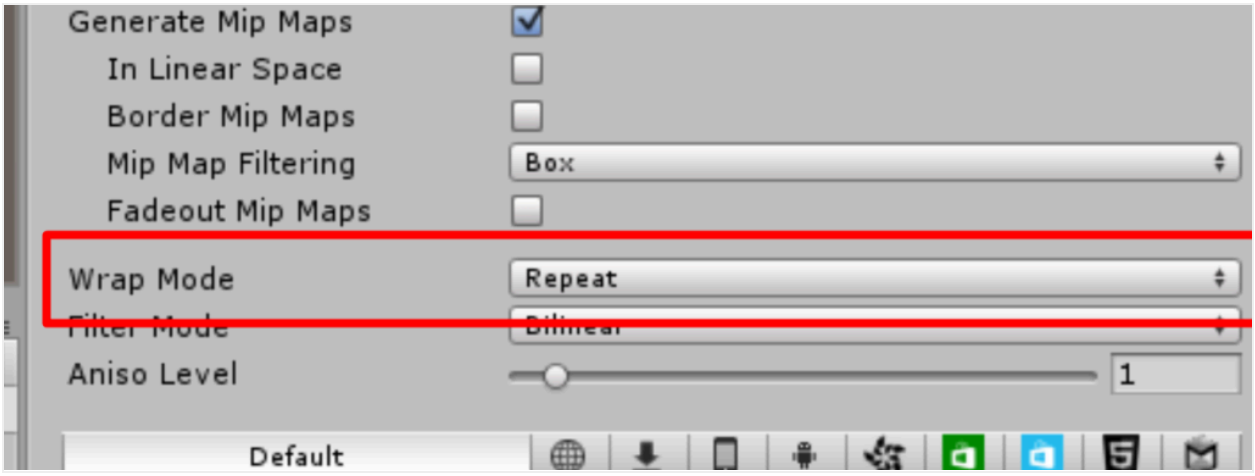
HOW TO USE MATERIAL TILLING

The Material Tiling structure is used through Graph and Chart. It allows you to fine tune the way a material tiled along a line. This proved useful for dashed lines and pattern lines. In this article we will cover the features of material tiling.



If you check the **Stretch** check box , the material will be stretched along the axis or chart line. This can be good for gradient lines , and will make the line color change along line.

If **Stretch** is not checked , the texture is tiled along the axis and chart line. It is important to set the texture wrap mode to repeat when using this option (texture that come with Graph and Chart already have this set) :



When tiling a texture , it is important to specify a tiling factor. The tiling factor is the distance span on which the texture is tiled. For canvas charts , It can be the pixel size of the texture used.

A good way to find the right tile factor by tweaking it's value and viewing the editor preview of the chart. When the texture looks like it fit well on the axis line , you'll know you have a good tiling factor.

HANDLING LARGE AMOUNTS OF GRAPH DATA

Let's say we have a data set of 250,000 points. While Graph and Chart would be capable of showing some few hundreds of points on screen on a given moment, you

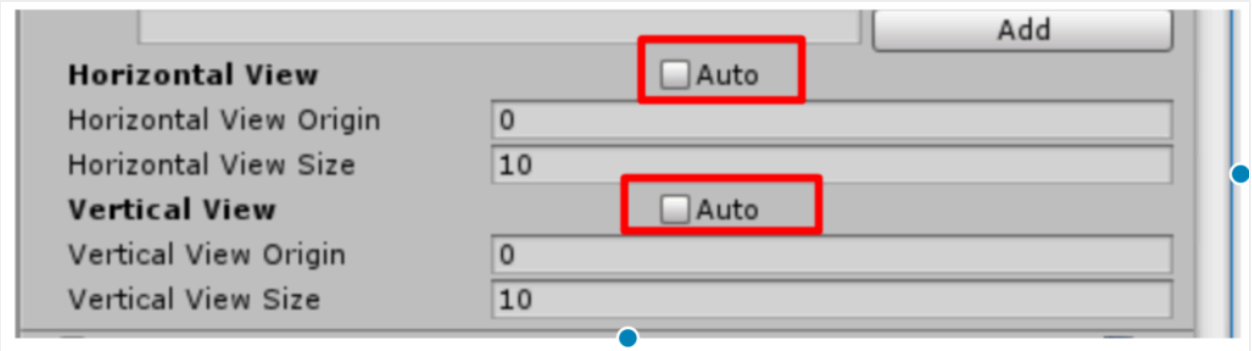
can actually create a graph that will allow your users to scroll through the entire data set. You can even add a zoom functionality that will allow you to view entire graph data. This tutorial is based on the Large Data Set example that is located in Tutorial/Large Data Graph

HOW DOES THE LARGE DATA SAMPLE WORK

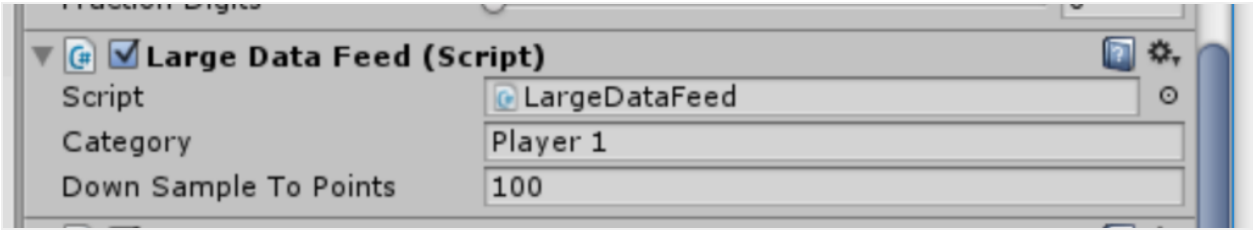
The large data sample uses a page based mechanism to load and unload graph data based on the view portion. This way you can load to the chart only 400-500 points that are currently visible , and save all the memory and performance overhead of the rest of the 100,000 points. Also , if the amount of points in a page is very large , the large data script will down sample the data so that it fits the performance requirements.

HOW TO USE THE LARGE DATA SAMPLE

remember to disable automatic view :



Add a large data feed component to your Graph Chart



- **Category** is the category to which the large data feed loads the data
- **Down Sample To Points** is the maximum amount of points allowed on screen at a given moment. If there are more points they will be down sampled to fit this value. You can set this to 0 to disable down sampling
-

If you have initial data that is to be set when the graph loads , replace the contents of SetInitialData in LargeDataFeed.cs with your own loading logic. otherwise you can empty it's contents.

```
/// <summary>
/// called with Start(). These will be used to load the data into the large data feed. You should replace this with your own loading logic.
/// </summary>
void SetInitialData()
{
    List<DoubleVector2> data = new List<DoubleVector2>(250000);
    double x = 0f;
    double y = 200f;
    for (int i = 0; i < 250000; i++)    // initialize with random data
    {
        data.Add(new DoubleVector2(x, y));
        y += UnityEngine.Random.value * 10f - 5f;
        x += UnityEngine.Random.value ;
    }
    SetData(data);
}
/// <summary>
```

Notice : The data used with large data feed must be sorted along the x axis.

At any time you want to reset the data of the Large Data Feed , you can obtain it from script and call SetData

```
List<DoubleVector2> newData = new List<DoubleVector2>();
```

```
var dataFeed = graph.GetComponent<LargeDataFeed>();
```

```
dataFeed.SetData(newData);
```

THE INTERNALS OF THE LARGE DATA SAMPLE

You can use the large data sample without ever needing to understand it's internals. however If you wish to understand more about how the sample is built , you can proceed to read the following section. It will help you better understand how to create similar implementations using graph and chart.

The concept if very basic when you come to think about it. We have the update method that checks if the chart is out of the boundaries of the currently set page.

```
{
    //check the scrolling position of the graph. if we are past the view size , load a new page
    double pageStartThreshold = currentPagePosition - mCurrentPageSizeFactor;
    double pageEndThreshold = currentPagePosition + mCurrentPageSizeFactor- graph.DataSource.HorizontalViewSize;

    if (graph.HorizontalScrolling < pageStartThreshold || graph.HorizontalScrolling > pageEndThreshold || currentZoom >= graph.DataSource.HorizontalViewSize * 2f)
    {
        currentZoom = graph.DataSource.HorizontalViewSize;
        mCurrentPageSizeFactor = PageSizeFactor * 0.9f;
        LoadPage(graph.HorizontalScrolling);
    }
}
```

If we are out of the currently set page, LoadPage is called:

```
void LoadPage(double pagePosition)
{
    if (graph != null)
    {

        Debug.Log("Loading page :" + pagePosition);
        graph.DataSource.StartBatch(); // call start batch
        graph.DataSource.HorizontalViewOrigin = 0;
        int start, end;
        findPointsForPage(pagePosition, out start, out end); // get the page edges
        graph.DataSource.ClearCategory(Category); // clear the cateogry

        if (DownSampleToPoints <= 0)
            LoadWithoutDownSampling(start, end);
        else
            LoadWithDownSampling(start, end);
        graph.DataSource.EndBatch();
        graph.HorizontalScrolling = pagePosition;
    }
    currentPagePosition = pagePosition;
}
```

It will clear and populate the chart with only needed data. In a similar way to the [Graph Tutorial]

Load Page obtains the stand and end points in the current window using findPointsPerPage , which takes advantage of binary search to quickly find the start and end points of the page.

```
void findPointsForPage(double position, out int start, out int end) // given a page position , find the right most and left most indices in the data for that page.
{
    int index = FindClosestIndex(position); // use binary search to find the closest position to the current scroll point
    int i = index;
    double endPosition = position + PageSizeFactor;
    double startPosition = position - PageSizeFactor;

    //starting from the current index , we find the page boundries
    for (start = index; start > 0; start--)
    {
        if (mData[start].x < startPosition) // take the first point that is out of the page. so the graph doesn't break at the edge
            break;
    }

    for (end = index; end < mData.Count; end++)
    {
        if (mData[end].x > endPosition) // take the first point that is out of the page
            break;
    }

}
```

finally if the total amount of points is too high , the data is down sampled by averaging a set of points into one point:

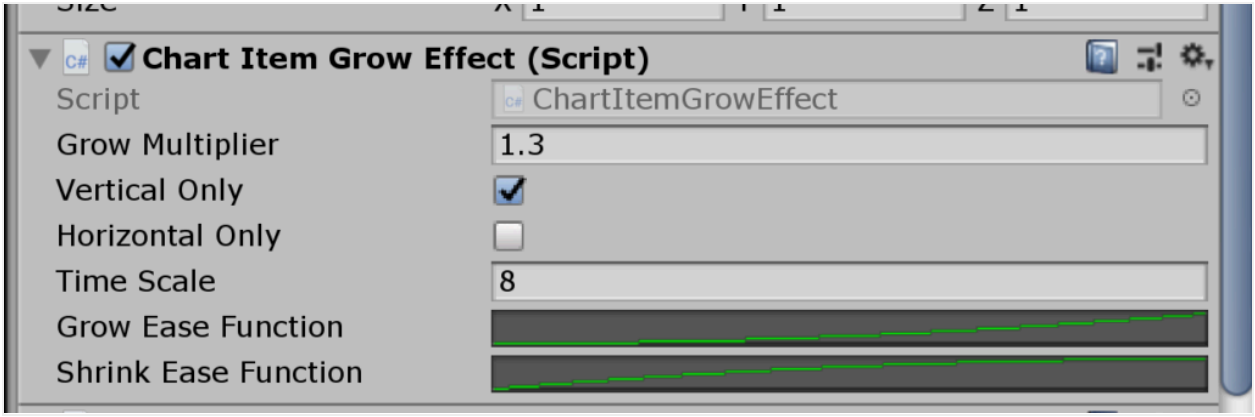
```
void LoadWithDownSampling(int start,int end)
{
    int total = end - start;

    if (DownSampleToPoints >= total)
    {
        LoadWithoutDownSampling(start, end);
        return;
    }

    double sampleCount = ((double)total) / (double)DownSampleToPoints;
   // graph.DataSource.AddPointToCategory(Category, mData[start].x, mData[start].y);
    for (int i=0; i<DownSampleToPoints; i++)
    {
        int fractionStart = start + (int)(i * sampleCount); // the first point with a fraction
        int fractionEnd = start + (int)((i+1) * sampleCount); // the first point with a fraction
        fractionEnd = Math.Min(fractionEnd, mData.Count - 1);
        double x = 0, y = 0;
        double divide = 0.0;
        for (int j= fractionStart; j<fractionEnd; j++)  // avarge the poins
        {
            x += mData[j].x;
            y += mData[j].y;
            divide++;
        }
        if (divide > 0.0)
        {
            x /= divide;
            y /= divide;
            graph.DataSource.AddPointToCategory(Category, x, y);
        }
        else
            Debug.Log("error");
    }
    int last = end - 1;
//   graph.DataSource.AddPointToCategory(Category, mData[last].x, mData[last].y);
}
```

HOVER PREFABS

Some charts , like the 2D graph chart , are designed with the intention of handling large amounts of data and streaming data. For this reason , they do not use prefabs in the conventional way. In these charts , the prefabs are placed on top of the chart to create customized interactivity. These prefabs are called hover prefabs , they are used in the 2D graph, 2D radar and 2D candle charts.

THE LIFE TIME OF A HOVER PREFAB

Hover prefabs are created when the user hovers over a chart item. For example a line or a point. The hover prefabs are either created as a new object , or from a pooled hover object. The hover prefab is alive as long as it's animation effect is showing. for example a common effect is :



Therefore it is important to have an animation effect attached to the hover prefab. Otherwise it will be created and placed in the scene indefinitely. Once the animation ends , the hover object is either pooled or destroyed.

HOW CAN HOVER PREFABS BE CUSTOMIZED

because of the short lifetime of hover prefabs , they are less customization then the other prefabs in graph and chart. However any script that is attached to them will be active during their life time. So you do have the option to add your own scripts to hover prefabs.

WHERE CAN I FIND READY MADE HOVER PREFABS ?

These can be found at **Prefabs/Canvas/Hover** you can see an overview of the available hover prefabs in How to use prefabs with Graph and Chart