



# Enterprise Architect

User Guide Series

# Dynamic Charts

Author: Sparx Systems  
Date: 2026-05-28  
Version: 17.1

CREATED WITH  ENTERPRISE  
ARCHITECT

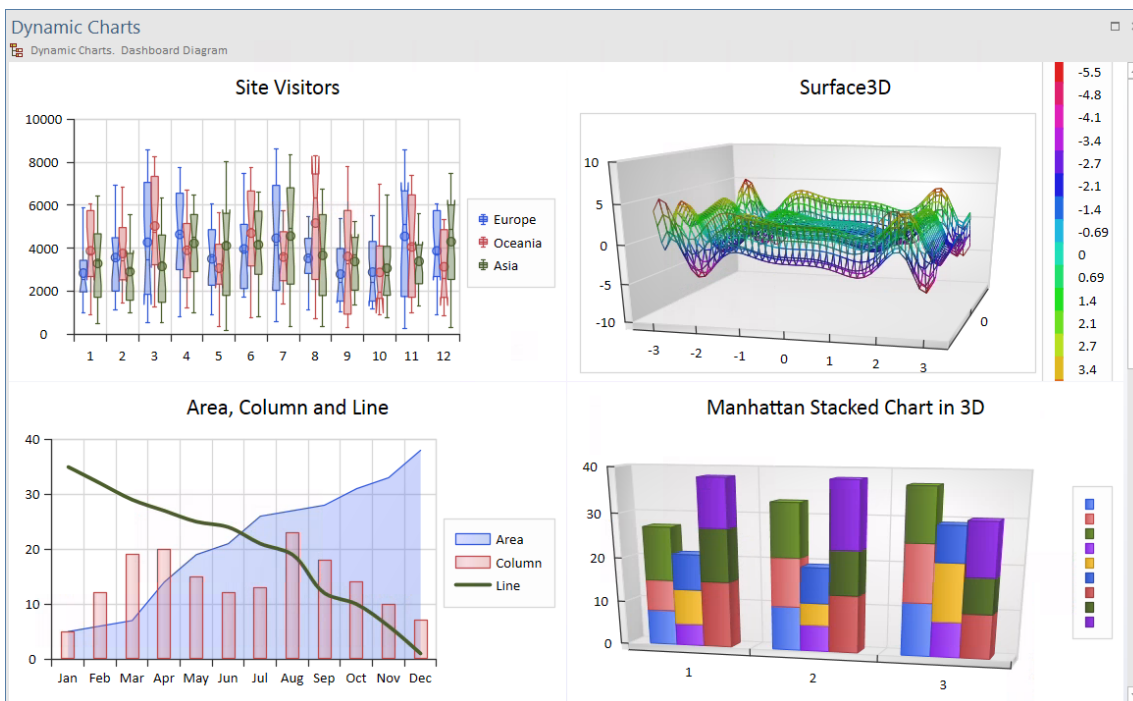
# Table of Contents

Dynamic Charts	4
Chart Definition using JSON	6
Chart Definition using Simulation	11
Chart Definition using JavaScript	17
Dynamic Chart Resources	24
The Chart API	26
Chart Class	27
Chart Enumerations	34
ChartAxisCrossType	35
ChartAxisIndex	36
ChartAxisLabelType	37
ChartAxisTickMarkType	38
ChartAxisType	39
ChartBarShape	41
ChartCategory	42
ChartColorMode	45
ChartCurveType	46
ChartDashStyle	47
ChartFrameStyle	48
ChartGradientType	49
ChartMarkerShape	51
ChartStockSeriesType	52
ChartType	53
ChartWallOptions	54

ChartAxisIndex Class .....	56
ChartDataValue Class .....	60
ChartDiagram3D Class .....	62
ChartFormatSeries Class .....	63
ChartSeries Class .....	65

# Dynamic Charts

Enterprise Architect features DynamicChart Artifacts, which are Chart elements that can be styled and rendered dynamically when the diagram opens. Dynamic Charts rely completely on code to define their series, style and content and are managed entirely through the Automation Interface by clients - typically Plug-ins and scripts. The Chart interfaces provide clients with the means to dynamically describe and populate a Chart when the diagram is viewed. The API is wide ranging and flexible, allowing you to illustrate many different scenarios graphically at runtime. The feature allows you to define any type of Chart you require using JavaScript either as code or JSON.



A good reference for the Chart interfaces and their use is the Package 'Reporting > Charts > Dynamic Charts' in the Enterprise Architect Example Model. This Package contains examples of many Charts, each described dynamically by

JavaScript clients. Each Chart example presents the two methods of rendering - a JavaScript coding method and a JSON datasource method.

Dynamic Charts are particularly useful for representing the results of Simulations, allowing you to:

- Save the results of your Simulation as visual Chart elements
- Easily include Charts populated by Simulation results in your reports
- Share user-friendly Simulation results with stakeholders without requiring any additional Simulation tools


Dynamic Charts are available in the Corporate, Unified and Ultimate editions of Enterprise Architect.

## Chart Definition using JSON

Rather than code a Dynamic Chart yourself, you might want to provide a simple description of the Chart instead.

Dynamic Charts can be designed and entirely defined by a single datasource; JSON is currently the preferred datasource format, but XML and others will be available in the future.

You define the Chart by providing a simple JSON data structure that adheres to the DynamicChart schema. The schema is available in the Schema Composer. It is also easily viewable in the Dynamic Charts Package of the Enterprise Architect Example Model.

(To view the DynamicChart Schema, select Develop > Schema Modeling > Schema Composer > Open Schema Composer, click on the  button in the 'Profile' field, and select DynamicChartSchema.)

## Datasources - JSON

To render a Chart using a JSON data structure, first select the DynamicChart Artifact element, then open the 'internal code' editor. For a selection in the Browser, right-click and choose 'Features > Edit Internal Code' or, for a selection on a diagram, right-click and choose 'Edit Chart Script'. These will open the editor for you to edit the Chart script. Create a JSON variable that defines the Chart to render, then compose your ConstructChart function.

The ConstructChart function takes as its single argument the

identity (a GUID string) of the Chart element being displayed on the opening diagram. You then call the built-in function `ConstructChartFromJSON`, passing the GUID as the first parameter and the JSON structure as the second argument, as illustrated in this example:

## Example Datasource in JSON

```
var barChart2DJSON =  
{  
  "Category" : "BarSmart",  
  "Type" : "Simple",  
  "Title" : "Vehicle Expenses",  
  "Series" :  
  [  
    {  
      "Label" : "Fuel",  
      "Data":  
      {  
        "Type" : "Column",  
        "Points" :  
        [  
          { "Category": "Jan", "Y": 1.0 },  
          { "Category": "Feb", "Y": 3.0 },  
          { "Category": "Mar", "Y": 7.0 },  
          { "Category": "Apr", "Y": 8.0 },
```

```
        { "Category": "May", "Y": 10.0 },
        { "Category": "Jun", "Y": 15.0 }
    ]
}
},
{
    "Label" : "Taxes",
    "Data":
    {
        "Type" : "Normal",
        "Points" :
        [
            { "Y":10.0 },
            { "Y":12.0 },
            { "Y":16.0 },
            { "Y":17.0 },
            { "Y":10.0 },
            { "Y":12.0 }
        ]
    }
},
{
    "Label" : "Maintenance",
    "Data":
    {
```

```
"Type" : "Normal",
"Points" :
[
  { "Y":5.0 },
  { "Y":2.0 },
  { "Y":6.0 },
  { "Y":7.0 },
  { "Y":1.0 },
  { "Y":2.0 }
]
},
{
  "Label" : "Other",
  "Data":
  {
    "Type" : "Normal",
    "Points" :
    [
      { "Y":2.5 },
      { "Y":2.5 },
      { "Y":2.5 },
      { "Y":2.5 },
      { "Y":2.5 },
      { "Y":2.5 }
    ]
  }
}
```

```
        ]
      }
    }
  ]
};
```

```
function ConstructChart(chartGuid)
{
    ConstructChartFromJSON(chartGuid,
barChart2DJSON);
}
```

## Further Examples

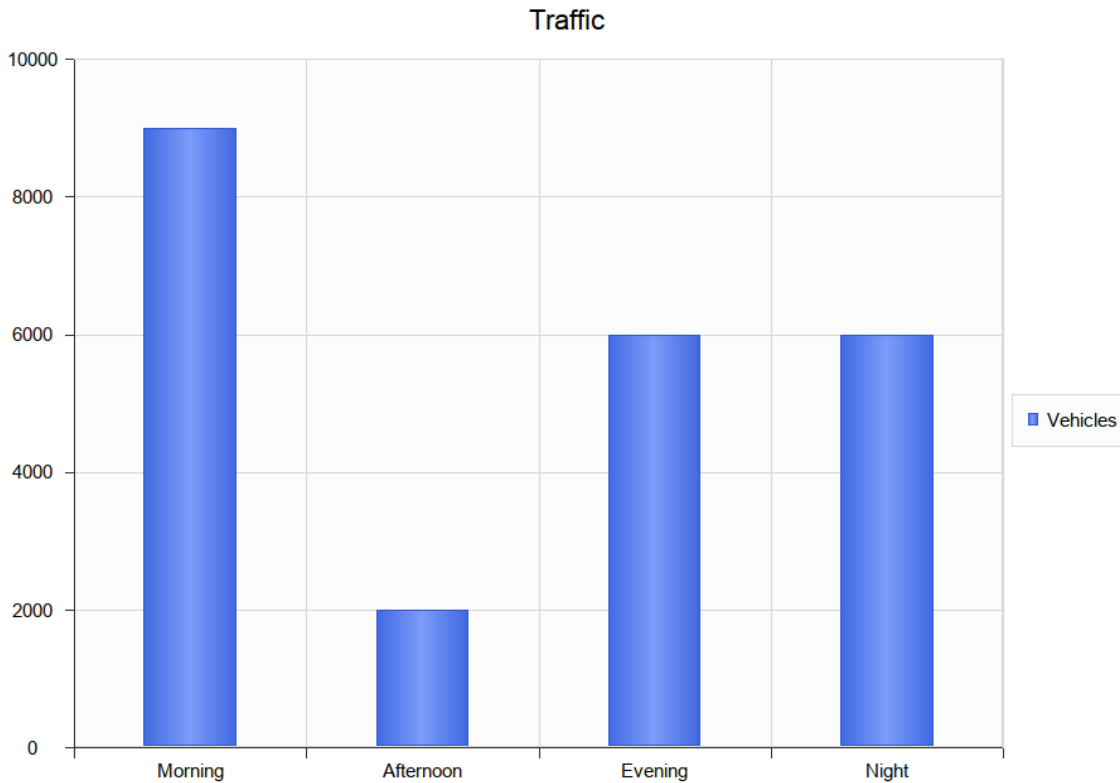
Further JSON examples are provided in the Example Model (see the Package 'Reporting > Charts > Dynamic Charts').

Each Chart example provides a Dashboard diagram and DynamicChart element. Select an element from any of these examples and press 'Alt+7' to view the behavior behind the Chart. Looking at the variety of Chart examples is the best way to get to know how to use JSON to produce the types of Chart you might be interested in.

## Chart Definition using Simulation

Simulations are a fantastic tool for observing behavior. At any point in the Simulation it is easy to tell where we are and the state we are in. As we step through a Simulation this information is typically discarded. In a Simulation that shows us, say, traffic numbers over a 24 hour period, we might easily observe the number of vehicles traveling through a tunnel during various hours in the morning and evening. It might be useful to retain this information after the Simulation completes, and to use it to provide something meaningful. The DynamicChart feature in Simulation allows us to do just that. Taking the example above, we could record the amount of traffic during each step of the Simulation and use it to produce a Chart that would clearly show how much traffic went through the tunnel during the 24 hour period of the Simulation. Charts can in effect display for us either a timeline of a Simulation or the sum effect of one.

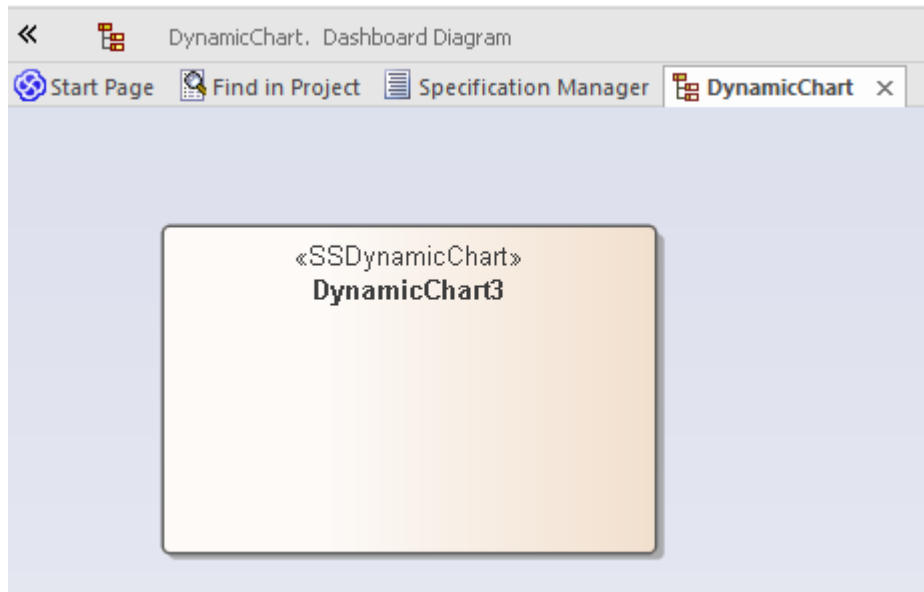
## Producing Custom Charts in Simulation



You can fashion all sorts of Charts from any Simulation. Each time a Simulation is run, any DynamicChart elements referenced (by name) by the Simulation are updated. The Simulation will search for any named Chart in the same Package as the model.

Follow this simple process:

1. Create a DynamicChart element for the Simulation.



2. In the initial step of the Simulation, use JavaScript to define a variable to hold the vehicle numbers.

```
//  
// the traffic variable will hold the traffic numbers as  
// Simulation proceeds and is initially zero  
// each element of the array represents a period of the  
// day, morning, afternoon, evening and night.  
//
```

```
var traffic = [0,0,0,0];
```

3. Next write the JavaScript that describes, in JSON format, the Chart to produce.

```
//  
// The JSON instance describing the chart to produce.  
// (complies with the EA DynamicChart Schema)  
//
```

```
var chartData =  
{  
  "Category" : "Column",  
  "Type" : "Simple",  
  "Title" : "Traffic",
```

```

"Series" :
[
  { "Label" : "Vehicles",
    "Data" :
    {
      "Type" : "Column",
      "Points" :
      [
        { "Category": "Morning", "Y" : 0 },//
The Y values of the axis are initially zero
        { "Category": "Afternoon", "Y" : 0 },
// they will be filled in at end of Simulation
        { "Category": "Evening", "Y" : 0 },
        { "Category": "Night", "Y" : 0 }
      ]
    }
  }
];

```

4. At various transitions in the Simulation update the traffic numbers.

```

//
// 2000 vehicles went through the tunnel in the afternoon
(element 1)
//
traffic[1] += 2000;

```

5. At the end of the Simulation, use the data captured during the run to fill the series.

```

// fill points in series with the number of vehicles for each

```

part of the day

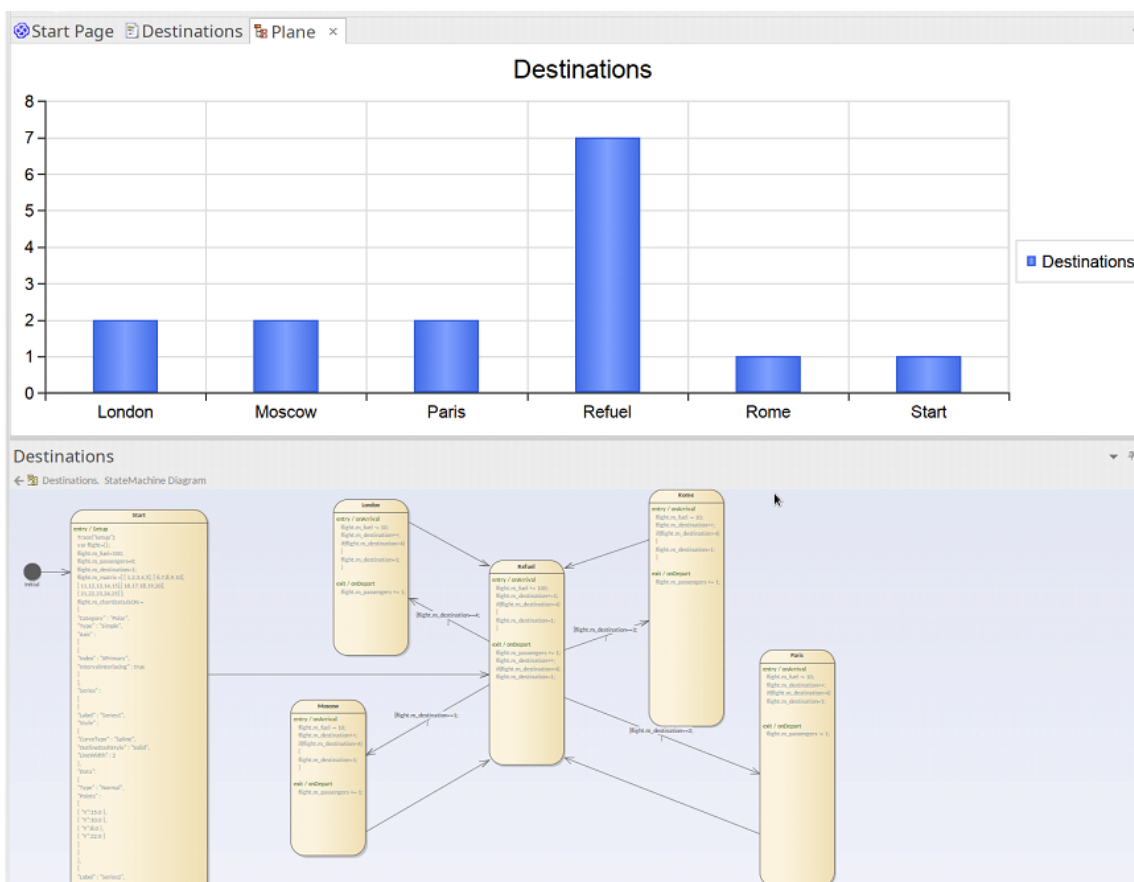
```
var dataPoints = chartData.Series[0].Data.Points;
for(var dp = 0; dp < traffic.length; dp++)
{
    dataPoints[dp].Y = traffic[dp];
}
}
```

6. Update the model.

// Call the EA function to populate the DynamicChart element named 'Vehicles' with this data.

```
sim.GenerateChart( "Vehicles",
JSON.stringify(chartData));
```

## Default Chart Produced by Simulation



In addition to Charts you specifically produce, a summary

Chart can be produced automatically by a simulation. All that is required is adding a DynamicChart Artifact to the Package and giving it the same name as the StateMachine. The default Chart summarizes the State transitions taken during execution of a simulation. If a default Chart is found when the simulation completes, that Chart's data is updated, and displayed automatically.

To add a default Chart to your StateMachine simulation follow these steps:

1. Locate the Package containing the StateMachine on which to perform the simulation.
2. Create a Dashboard diagram as a child of that Package.
3. Add a DynamicChart Artifact to the Dashboard diagram, and give it the same name as the StateMachine.

When the simulation ends, simply open the Dashboard diagram to reveal the summary.

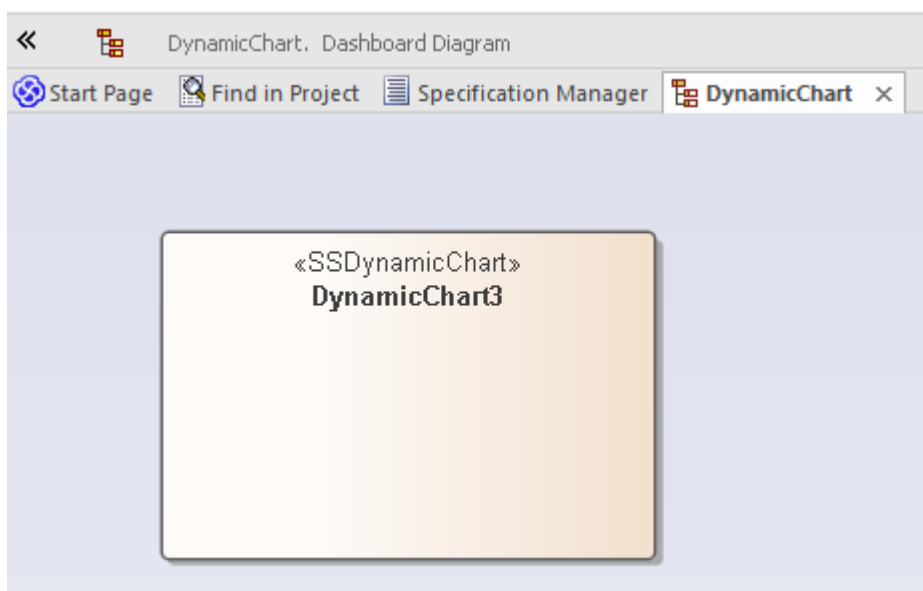
# Chart Definition using JavaScript

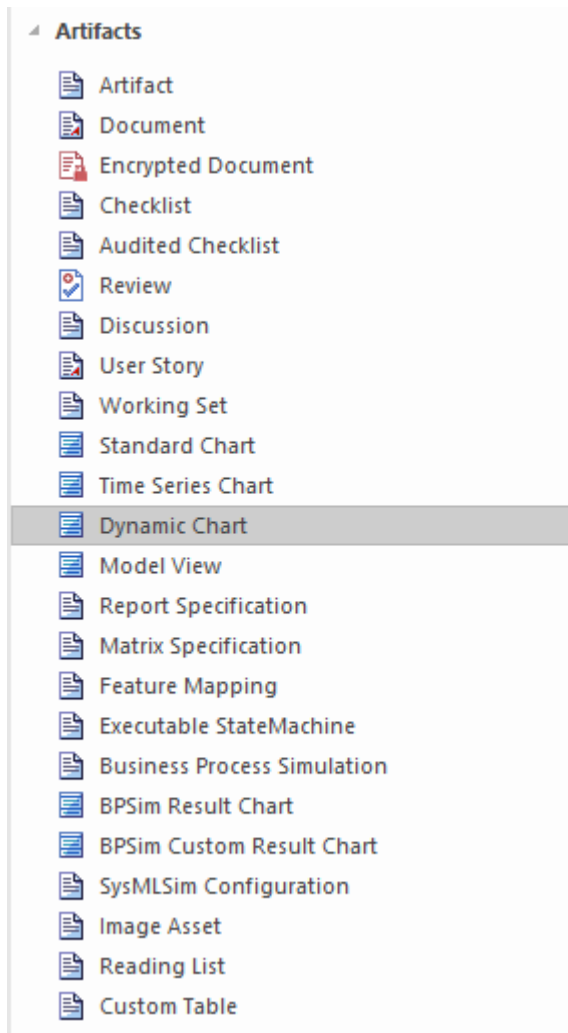
In this topic we discuss coding a DynamicChart Artifact using just JavaScript and the Chart automation interface.

## Define a Chart through JavaScript

The first thing you do is create a Dashboard diagram in the appropriate Package. Right-click on the Package and select the 'New Diagram' option. This opens the 'Diagram Builder' tab page of the 'Model Builder' dialog.

On the 'Diagram Builder' tab page, click on the ☰ icon and select the perspective 'Publishing > Dynamic Charts'. In the 'Select From' panel, select 'Dynamic Charts' and in the 'Diagram Types' panel select 'DynamicChart'. Click on 'Create Diagram' to create the new diagram. When the empty diagram displays, drag the 'Dynamic Chart' icon onto it from the 'Charts' page of the Toolbox.





Now you write the JavaScript to style and render the Chart, starting with the `ConstructChart` function that will be invoked automatically whenever the diagram containing the `DynamicChart` Artifact is opened for viewing. The GUID of the element is passed to `ConstructChart` as a parameter.

Within this function, it is entirely up to you what type of Chart to display, the style of the Chart, the number of series it contains, and the data points that make up the series.

Using the Chart Package from the Automation Interface, it is possible to display almost any Chart you require.

In this example you will create a Grouped Column Chart that shows vehicle expenses over a few months. Each group will represent a month and will be broken down into the

different expenses incurred during that month.

To begin, click on the Artifact and press Alt+7, or click on the 'Edit Chart Script' context menu option; each method displays the Code Editor window. The code to use is provided here, followed by the Chart it will produce when the diagram is opened.

Importantly, note:

- The **!INC Local Scripts.ChartAutomation** statement; all Chart scripts must include this statement
- The ConstructChart function (7th line in)

## Code

```
!INC Local Scripts.ChartAutomation
```

```
var monthNames = [ "Jan", "Feb", "Mar", "Apr", "May",  
"Jun", "Jul", "Aug", "Sep", "Oct", "Nov", "Dec" ];
```

```
function Rand(min, max) {  
  min = cephes.ceil(min);  
  max = cephes.floor(max);  
  return cephes.floor(cephes.drand() * (max - min)) + min; }
```

```
function ConstructChart( guid )  
{
```

```
var chart as EA.Chart; // The
script first of all
var element = GetElementByGuid(guid); //
declares the automation
var series1 as EA.ChartSeries; // objects
it will use
var series2 as EA.ChartSeries;
var series3 as EA.ChartSeries;
var series4 as EA.ChartSeries;

chart = element.GetChart();

var chartCategory = ChartCategory.Column();
var chartType = ChartType.SIMPLE();
chart.SetChartType( chartCategory, chartType, false,
true);

chart.Title = "Vehicle Expenses";
series1 = chart.CreateSeries("Fuel"); // The
script then obtains the Chart object and creates the
series2 = chart.CreateSeries("Taxes"); // series.
A chart is composed of a number of series, and
series3 = chart.CreateSeries("Maintenance"); // in
this example each series will represent a type of expense.
series4 = chart.CreateSeries("Other");
```

series1.AddDataPoint3( monthNames[0], 14); // A series is composed of a number of datapoints and, here, the

series1.AddDataPoint3( monthNames[1], 4); // script adds the values for each of the points to each series.

```
series1.AddDataPoint3( monthNames[2], 3);
```

```
series1.AddDataPoint3( monthNames[3], 2);
```

```
series1.AddDataPoint3( monthNames[4], 1);
```

```
series2.AddDataPoint(10);
```

```
series2.AddDataPoint(12);
```

```
series2.AddDataPoint(15);
```

```
series2.AddDataPoint(17);
```

```
series2.AddDataPoint(12);
```

```
series3.AddDataPoint(5);
```

```
series3.AddDataPoint(7);
```

```
series3.AddDataPoint(11);
```

```
series3.AddDataPoint(14);
```

```
series3.AddDataPoint(19);
```

```
series4.AddDataPoint(2);
```

```
series4.AddDataPoint(3);
```

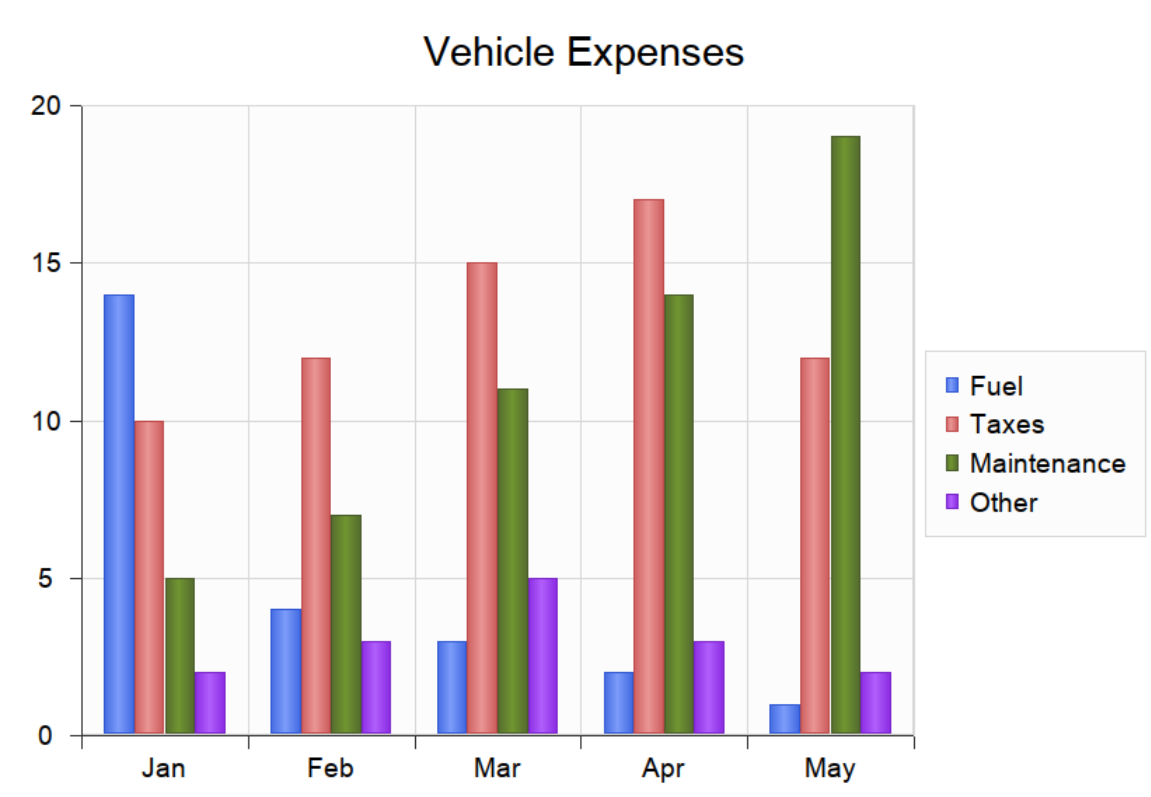
```
series4.AddDataPoint(5);
```

```
series4.AddDataPoint(3);
```

```
series4.AddDataPoint(2);  
  
series1.SetGroupID(0);  
series1.SetGroupID(0);  
series3.SetGroupID(1);  
series4.SetGroupID(1);  
  
chart.Redraw();  
}
```

## Output

This is the Chart produced by the code.



## Debugging a Dynamic Chart

After creating Dynamic Chart JavaScript, you can debug it as for any other code. Right-click on the Dynamic Chart in a diagram and select the 'Debug Chart Script' option. The script displays in the Debug View.

## Further Examples

Further coding examples are provided in the Example Model (see the Package 'Reporting > Charts > Dynamic Charts'). Each Chart example provides a Dashboard diagram and a DynamicChart element. Select any of these elements and press Alt+7 to view the behavior code behind the Chart. Referring to these examples is the best way to understand how to code each type of Chart.

JavaScript is the primary language for coding Dynamic Charts using automation. However, it is certainly feasible for a third-party automation client to be involved in that process, the JavaScript host delegating tasks to the automation clients in languages such as C# and C++, which might be able to obtain data from outside the model in ways not available to scripts.

# Dynamic Chart Resources

A number of resources are available to you to aid in constructing Charts dynamically in Enterprise Architect, as described in the *Resources* table.

## Resources

Resource	Description
JavaScript Library	A library of JavaScript functions, with examples of coding Charts using the Automation Interface or using JSON as a data source for Charts. This library can be found in the 'Local Scripts' group of the Scripting control in Enterprise Architect.
Example Model	The Enterprise Architect Example Model contains a number of Chart examples, demonstrating the various types of Chart that can be created and styled dynamically when they are displayed. These are in the Package 'Reporting > Charts > Dynamic Charts'. Examples include 3D Surface and Wireframe Charts, Stock Charts with series showing price and volume fluctuations, Box Plot Charts showing

	<p>visitor numbers, Manhattan Stack Charts, and many others.</p> <p>Each example is provided in two versions. The first version uses JavaScript code to style and fill the Chart. The second uses a JSON data source to describe and fill the Chart.</p>
<b>Chart Schemas</b>	<p>The Example Model contains a reference model for Dynamic Charts, including Class diagrams, a Schema Composer profile, and the JSON and XML schemas generated using the profile, as Artifacts that you can quickly view by double-clicking on them.</p>

# The Chart API

The Chart interface is the API object that provides methods for dynamically creating Charts. It can be used to construct any of the supported Chart types.

A Chart interface is obtained using the GetChart method on a DynamicChart element. A DynamicChart element can be created from the 'Charts' page of the Diagram Toolbox, and is typically used on a Dashboard diagram.

## Chart Class

The Chart Class is the primary interface for Chart elements; it is used to create a series, add datapoints to a series and configure the chart appearance.

## Chart Attributes

Attribute	Description
Title	String Notes: Read/Write The title of the chart.
Category	ChartCategory Notes: Read only The chart category; provided in the SetChartType method.
Type	ChartType Notes: Read only The chart type; provided in the SetChartType method.

## Chart Methods

Method	Description
AddChartData aYZZ(double Y, double X, double Z, long seriesIndex)	long Adds a datapoint to an existing series. Parameters: <ul style="list-style-type: none"> <li>• Y: double, the primary Y axis value</li> <li>• X: double, the primary X axis value</li> <li>• Z: double, the primary Z axis value</li> <li>• seriesIndex: long, the index of the series (returned by the CreateSeries methods)</li> </ul>
AddChartData aYY1(string category, double Y, double Y1, long seriesIndex)	long Adds a datapoint to an existing series. Parameters: <ul style="list-style-type: none"> <li>• category: string - the x axis group, column or label</li> <li>• Y: double, the primary Y axis value</li> <li>• Y1: double, the secondary Y axis value</li> <li>• seriesIndex: long, the index of the series (returned by the CreateSeries and CreateSeriesEx methods)</li> </ul>
CreateSeries( string name)	LDISPATCH Adds a new series to the chart. Returns an interface that can be used to add data to

	<p>the series and configure its appearance.</p> <p>Parameters:</p> <ul style="list-style-type: none"> <li>• name: string, the displayed name of the series</li> </ul>
<p>CreateSeries Ex(string name, long color, ChartType type, ChartCategory category)</p>	<p>LDISPATCH</p> <p>Creates a series of a particular chart category and type and returns an IChartSeries interface. This allows charts to form multiple series in various ways. The CombinedCharts in the EAExample Model are an example of this, displaying three series for the Area, Column and Line categories respectively.</p> <p>Parameters:</p> <ul style="list-style-type: none"> <li>• name: string, the name of the series</li> <li>• color: long, RGB color value,-1 for default</li> <li>• type: ChartType, one of the ChartType enumerations</li> <li>• category: ChartCategory, one of the ChartCategory enumerations</li> </ul>
<p>EnableResize Axes(boolean bEnable)</p>	<p>void</p> <p>Grants or denies the ability to resize axes.</p> <p>Parameters:</p> <ul style="list-style-type: none"> <li>• bEnable: boolean</li> </ul>

<p>GetChartAxis (ChartAxisType type)</p>	<p>LDISPATCH</p> <p>Returns an IChartAxis interface for the specified axis.</p> <p>Parameters:</p> <ul style="list-style-type: none"> <li>• type: ChartAxisType, one of the ChartAxisType enumerations</li> </ul>
<p>GetDiagram3D()</p>	<p>LDISPATCH</p> <p>Returns an IChartDiagram interface that can be used to specify the rendering engine; Software or OpenGL.</p>
<p>GetSeries(long index)</p>	<p>LDISPATCH</p> <p>Returns an IChartSeries interface for the given index. Indices for series begin at zero.</p> <p>Parameters:</p> <ul style="list-style-type: none"> <li>• index: long</li> </ul>
<p>GetSeriesCount()</p>	<p>long</p> <p>Returns the number of series represented by the chart.</p>
<p>Redraw()</p>	<p>void</p> <p>Redraws the chart.</p>
<p>SetChartType (ChartType</p>	<p>void</p>

<p>type, ChartCategory category, boolean bRedraw, boolean bResizeAxis)</p>	<p>This is typically the first call made on the Chart interface. It defines the style and appearance of the Chart when it is rendered.</p> <p>Parameters:</p> <ul style="list-style-type: none"> <li>• type: ChartType</li> <li>• category: ChartCategory</li> <li>• bRedraw: Boolean</li> <li>• bResizeAxis: Boolean</li> </ul>
<p>SetCurveType (ChartCurve Type type)</p>	<p>void</p> <p>Sets the interpolation method of drawing the curve.</p> <p>Parameters:</p> <ul style="list-style-type: none"> <li>• type: ChartCurveType, one of the ChartCurveType enumerations, such as Line, Spline or SplineHermite.</li> </ul>
<p>SetSeriesShadow (boolean bShow)</p>	<p>void</p> <p>Displays or hides shadows on series.</p> <p>Parameters:</p> <ul style="list-style-type: none"> <li>• bShow: Boolean</li> </ul>
<p>SetThemeOpacity (long percentage)</p>	<p>void</p> <p>Sets the opacity of the chart.</p> <p>Parameters:</p> <ul style="list-style-type: none"> <li>• percentage: long, chart transparency as</li> </ul>

	a percentage
ShowAxis(long index)	<p>void</p> <p>Shows the axis for the given index.</p> <p>Parameters:</p> <ul style="list-style-type: none"> <li>• index: long, one of the ChartAxisType enumerations</li> </ul>
ShowDataLabels(boolean show, boolean border, boolean dropLineTomarker)	<p>void</p> <p>Shows or hides data labels on the chart.</p> <p>Parameters:</p> <ul style="list-style-type: none"> <li>• show: Boolean, show or hide labels</li> <li>• border: Boolean, show or hide border on labels</li> <li>• dropLineTomarker: Boolean, changes position of label with respect to line</li> </ul>
ShowDataMarkers(boolean show, long size, ChartmarkerShape shape)	<p>void</p> <p>Shows or hides data markers on the chart. Also allows setting the appearance of markers.</p> <p>Parameters:</p> <ul style="list-style-type: none"> <li>• show: Boolean, show or hide markers</li> <li>• size: long, size of markers in pixels</li> <li>• shape: ChartmarkerShape, one of the ChartmarkerShape enumerations</li> </ul>



# Chart Enumerations

These enumerations, used specifically by methods in the Chart interface, are described in the topics of this section. Click on the enumeration name in the list to the left of this text.

# ChartAxisCrossType

## Enum Values

Enum	Value
Auto	value: 0
MaximumAxisValue	value: 1
MinimumAxisValue	value: 2
AxisValue	value : 3
Ignore	value: 4
FixedDefaultPos	value: 5

# ChartAxisIndex

## Enum Values

Enum	Value
Unknown	value: -1
X	value: 0
Y	value: 1
Z	value: 2

# ChartAxisLabelType

## Enum Values

Enum	Value
NoLabels	value: 0
NextToAxis	value: 1
High	value: 2
Low	value: 3

# ChartAxisTickMarkType

## Enum Values

Enum	Value
NoTicks	value: 0
Inside	value: 1
Outside	value: 2
Cross	value: 3

# ChartAxisType

A set of constants that refer to the various axes used in charts.

## Enum Values

Enum	Value
CHART_Y_PRIMARY_AXIS	value: 0
CHART_Y_SECONDARY_AXIS	value: 1
CHART_X_PRIMARY_AXIS	value: 2
CHART_X_SECONDARY_AXIS	value: 3
CHART_Z_PRIMARY_AXIS	value: 4

CHART_Z_S ECONDARY _AXIS	value: 5
CHART_Y_ POLAR_AXI S	value: 6
CHART_X_ POLAR_AXI S	value: 7
CHART_A_ TERNARY_ AXIS	value: 8
CHART_B_ TERNARY_ AXIS	value: 9
CHART_C_ TERNARY_ AXIS	value: 10

# ChartBarShape

## Enum Values

Enum	Value
Box	value: 0
Pyramid	value: 1
PyramidPartial	value: 2

# ChartCategory

## Enum Values

Enum	Value
chartDefault	value: 0
chartLine	value: 1
chartPie	value: 2
chartPie3D	value: 3
chartPyramid	value: 4
chartPyramid 3D	value: 5
chartFunnel	value: 6
chartFunnel3 D	value: 7
chartColumn	value: 8
chartBar	value: 9

chartHistogram	value: 10
chartArea	value: 11
chartStock	value: 12
chartBubble	value: 13
chartLongData	value: 14
chartHistoricalLine	value: 15
chartPolar	value: 16
chartDoughnut	value: 17
chartDoughnut3D	value: 18
chartTorus3D	value: 19
chartTernary	value: 20
chartColumn	

3D	value: 21
chartBar3D	value: 22
chartLine3D	value: 23
chartArea3D	value: 24
chartSurface3D	value: 25
chartDoughnutNest	value: 26
chartBoxPlot	value: 27
chartBarSmart	value: 28
chartBar3DSmart	value: 29

# ChartColorMode

## Enum Values

Enum	Values
Single	value: 0
Multiple	value: 1
Palette	value: 2
Custom	value: 3
Series	value: 4

# ChartCurveType

## Enum Values

Enum	Value
NoLine	value: 0
Line	value: 1
Spline	value: 2
SplineHermit e	value: 3
Step	value: 4
ReversedStep	value: 5

# ChartDashStyle

## Enum Values

Enum	Value
Solid	value: 0
Dash	value: 1
Dot	value: 2
DashDot	value: 3
DashDotDot	value: 4
Custom	value: 5

# ChartFrameStyle

## Enum Values

Enum	Value
None	value: 0
Mesh	value: 1
Contour	value: 2
ContourMesh	value: 3

# ChartGradientType

## Enum Values

Enum	Value
None	value: 0
Horizontal	value: 1
Vertical	value: 2
DiagonalLeft	value: 3
DiagonalRight	value: 4
CenterHorizontal	value: 5
CenterVertical	value: 6
RadialTop	value: 7
RadialCenter	value: 8

RadialBottom	value: 9
RadialLeft	value: 10
RadialRight	value: 11
RadialTopLeft	value: 12
RadialTopRight	value: 13
RadialBottomLeft	value: 14
RadialBottomRight	value: 15
Bevel	value: 16
PipeVertical	value: 17
PipeHorizontal	value : 18

# ChartMarkerShape

## Enum Values

Enum	Value
Circle	value: 0
Triangle	value: 1
Rectangle	value: 2
Rhombus	value: 3

# ChartStockSeriesType

## Enum Values

Enum	Value
Bar	value: 0
Candle	value: 1
LineOpen	value: 2
LineHigh	value: 3
LineLow	value: 4
LineClose	value: 5
LineCustom	value: 6

# ChartType

## Enum Values

Enum	Value
chartTypeDE FAULT	value: 0
chartTypeSI MPLE	value: 1
chartTypeST ACKED	value: 2
chartType100 STACKED	value: 3
chartTypeRA NGE	value: 4

# ChartWallOptions

## Enum Values

Enum	Value
None	value: 0, 0x0000
FillLeftWall	value: 1, 0x0001
OutlineLeftWall	value: 2, 0x0002
FillRightWall	value: 4, 0x0004
OutlineRightWall	value: 8, 0x0008
FillFloor	value: 16, 0x0010
OutlineFloor	value: 32, 0x0020
DrawAll	value: 65535, 0xFFFF
DrawLeftWall	FillLeftWall   OutlineLeftWall
1	

DrawRightWall	FillRightWall   OutlineRightWall
DrawFloor	FillFloor   OutlineFloor
DrawAllWalls	DrawLeftWall   DrawRightWall
OutlineAllWalls	OutlineLeftWall   OutlineRightWall
OutlineAll	OutlineAllWalls   OutlineFloor
FillAllWalls	FillLeftWall   FillRightWall
FillAll	FillAllWalls   FillFloor
Default	OutlineAll

# ChartAxisIndex Class

## ChartAxisIndex Attributes

Attribute	Description
Visible	Boolean Shows or hides the axis.

## ChartAxisIndex Methods

Method	Description
EnableMajorUnitIntervalInterlacing(booleantinterlace)	void Turns interlacing on or off.
GetGuid()	string Returns the guid of the axis. Uniquely identifies an axis.
GetLabel()	string Returns the value of the label of the axis.

<p><b>SetAxisName</b> (string label, boolean showonaxis)</p>	<p>void</p> <p>Sets the label for the axis and whether it should be displayed on the chart.</p> <p>Parameters:</p> <ul style="list-style-type: none"> <li>• label: string, the text for the label</li> <li>• showonaxis: Boolean, a true value indicates that the label is displayed</li> </ul>
<p><b>SetCrossType</b> (long type)</p>	<p>void</p> <p>Provides a directive or hint for use when calculating the position of labels on an axis.</p> <p>Parameters:</p> <ul style="list-style-type: none"> <li>• type: long, one of the <code>ChartAxisCrossType</code> enumerations</li> </ul>
<p><b>SetDataFormat</b> (string format, boolean formatAsDate)</p>	<p>void</p> <p>Sets the format string for the conversion of values to strings (e.g. "%0.4f"). If the datapoints represent datetime values, the <code>formatAsDate</code> argument should be true, and the format string set appropriately (e.g. "%H:%M")</p> <p>Parameters:</p> <ul style="list-style-type: none"> <li>• format: string, the format to use when converting datapoint values to string</li> <li>• formatAsDate: Boolean, a true value</li> </ul>

	indicates the datapoint represent a datetime
SetDisplayUnits(double units)	<p>void</p> <p>Sets the display units on the axis. Basically, the datapoint values are divided by this figure to give a major unit value. For example, if the datapoint contains meter values, a value of 1000 would result in kilometers being used as the major unit on the axis.</p> <p>Parameters:</p> <ul style="list-style-type: none"> <li>• units: double, the value of a single unit on the axis</li> </ul>
SetFixedDisplayRange(double fmin, double fmax)	<p>void</p> <p>Sets a fixed range for the axis.</p> <p>Parameters:</p> <ul style="list-style-type: none"> <li>• fmin: double, the minimum value</li> <li>• fmax: double, the maximum value</li> </ul>
SetLabelType(long labelpos)	<p>void</p> <p>Sets the position of labels on the axis.</p> <p>Parameters:</p> <ul style="list-style-type: none"> <li>• labelpos: long, one of the ChartAxisLabelType enumerations</li> </ul>
SetTickMark	

(long tickmarkpos)	void Sets the position of tick marks on the axis. Parameters: <ul style="list-style-type: none"><li>• tickmarkpos: long, one of the ChartAxisTickMarkType enumerations</li></ul>
ShowMajorGridLines(boolea show)	void Shows or hides grid lines.

# ChartDataValue Class

The ChartDataValue class provides an interface that allows values to be obtained from points in a series.

## ChartDataValue Methods

Method	Description
GetValue()	double Returns the value associated with the datapoint.
IsEmpty()	Boolean True if no value exists for the datapoint.
SetEmpty(boolean empty)	void Sets a datapoint on a series to be empty. Parameters: <ul style="list-style-type: none"><li>• empty: Boolean, true if the datapoint is to be considered as empty, having no value</li></ul>
SetValue(double value)	void Sets the value of a datapoint. Parameters: <ul style="list-style-type: none"><li>• value: double, the value of the</li></ul>

	datapoint; setting a value makes a datapoint non-empty
--	--

# ChartDiagram3D Class

## ChartDiagram3D Methods

Method	Description
SetDrawWallOptions(long options, boolean redraw)	<p>void</p> <p>Sets the option for how walls and floors - if any - are displayed on the 3D chart. The options parameter is a bitmask of one or more values from the ChartWallOptions enum.</p> <p>Parameters:</p> <ul style="list-style-type: none"> <li>options: Long, bitmask of wall and floor display options</li> <li>redraw: Boolean, redraws the chart after the function completes</li> </ul>
SetRenderingType(long engine)	<p>void</p> <p>Parameters:</p> <ul style="list-style-type: none"> <li>engine: long, 0 for software, 1 for OpenGL</li> </ul>

# ChartFormatSeries Class

A helper class for the ChartSeries class that allows setting appearance options.

## ChartFormatSeries Methods

Method	Description
SetCurveType(ChartCurveType type)	<p>void</p> <p>Sets the graphic option for rendering lines.</p> <p>Parameters:</p> <ul style="list-style-type: none"> <li>type: long, one of the ChartCurveType enumerations</li> </ul>
SetSeriesLineWidth(long width)	<p>void</p> <p>Sets the line width in pixels.</p> <p>Parameters:</p> <ul style="list-style-type: none"> <li>width: long, a pixel value</li> </ul>
SetSeriesOutlineDashStyle(ChartDashStyle dashstyle)	<p>void</p> <p>Sets the dash style of the line on the chart/graph.</p> <p>Parameters:</p> <ul style="list-style-type: none"> <li>dashstyle: ChartDashStyle, one of the ChartDashStyle enumerations</li> </ul>



# ChartSeries Class

## ChartSeries Methods

Method	Description
<b>AddBoxPlot</b> <b>Data(double</b> <b>ave, double</b> <b>min, double</b> <b>q1, double</b> <b>q2, double</b> <b>q3, double</b> <b>max, double</b> <b>notched)</b>	<p>long</p> <p>For a chart having the BoxPlot category, adds a single datapoint to the series.</p> <p>Parameters:</p> <ul style="list-style-type: none"> <li>• ave: double, the mean value at this point</li> <li>• min: double, the minimum value at this point</li> <li>• q1: double, the first quartile value</li> <li>• q2: double, the second quartile value</li> <li>• q3: double, the third quartile value</li> <li>• max: double the maximum value at this point</li> <li>• notched: double, for a series with notched style, the notched value to express at this point</li> </ul>
<b>AddDataPoin</b> <b>t(double Y)</b>	<p>long</p> <p>Adds a datapoint to the series. Returns the index of the point, which is the</p>

	<p>number of points -1.</p> <p>Parameters:</p> <ul style="list-style-type: none"> <li>• Y: double, the Y axis value</li> </ul>
<p>AddDataPoint2(double Y, double X)</p>	<p>long</p> <p>Adds a datapoint to the series. Returns the index of the point, which is the number of points -1.</p> <p>Parameters:</p> <ul style="list-style-type: none"> <li>• Y: double, the Y axis value</li> <li>• X: double, the X axis value</li> </ul>
<p>AddDataPoint3(string category, double Y)</p>	<p>long</p> <p>Adds a Y axis value for a given category on the X axis.</p> <p>Parameters:</p> <ul style="list-style-type: none"> <li>• category: string, the category or column name</li> <li>• Y: double, the value</li> </ul>
<p>AddStockData(double open, double high, double low, double closing, VARIANT timestamp)</p>	<p>void</p> <p>Adds data to a series for a chart of the Stock category.</p> <p>Parameters:</p> <ul style="list-style-type: none"> <li>• open: double, opening value</li> <li>• high: double, high value</li> <li>• low: double, low value</li> </ul>

	<ul style="list-style-type: none"> <li>• closing: double, closing value</li> <li>• timestamp: {datetime, double utcsecs} either VARIANT date value or double, in which case the value is interpreted as the number of seconds since midnight on January 1st, 1970, UTC time</li> </ul>
AddSurfaceColors(VARIANT colors)	<p>void</p> <p>Adds one or more colors to the series.</p> <p>Parameters:</p> <ul style="list-style-type: none"> <li>• colors: long, or long[], a single RGB color or an array of RGB color values</li> </ul>
CloseShape(boolean close, boolean fill)	<p>void</p> <p>Connects the first and last datapoints and fills the shape if 'fill' is true.</p> <p>Parameters</p> <ul style="list-style-type: none"> <li>• close: Boolean, if true closes the series</li> <li>• fill: Boolean, fills the shape</li> </ul>
GetDataPointCount()	<p>long</p> <p>Returns the number of datapoints in the series.</p>
GetDataPointValue(long index)	<p>LDISPATCH</p> <p>Returns a ChartDataValue interface for the datapoint with the given index.</p>

	<p>Parameters:</p> <ul style="list-style-type: none"> <li>• index: long, the index of the datapoint (typically returned by <code>AddDataPoint</code> functions; a value in the range 0 to n-1, where n is the number of points returned by the <code>GetDataPointCount</code> function)</li> </ul>
<code>GetSeriesFormat()</code>	<p>LDISPATCH</p> <p>Returns a <code>ChartFormatSeries</code> interface that allows the chart appearance to be changed.</p>
<code>SetBarShape(long barshape)</code>	<p>void</p> <p>Sets the shape for Bar charts, 0 for Box, 1 for Pyramid, 2 for PyramidPartial.</p> <p>Parameters:</p> <ul style="list-style-type: none"> <li>• barshape: <code>ChartBarShape</code>, one of the <code>ChartBarShape</code> enumerations</li> </ul>
<code>SetColorMapCount(long count)</code>	<p>void</p> <p>Sets the number of colors used when rendering the series. Typical values are 4, 8, 16 and 32</p> <p>Parameters:</p> <ul style="list-style-type: none"> <li>• count: long, the number of colors to use</li> </ul>
<code>SetColorMod</code>	<p>void</p>

e(ChartColor Mode mode)	<p>For 3D charts, sets the interpolation method for filling shapes. Single, for example, would result in the 3D object being filled by varying the color slightly. The level of variation will depend on the number of colors used by the chart (see <i>SetColorMapCount</i>).</p> <p>Parameters:</p> <ul style="list-style-type: none"> <li>• mode: ChartColorMode</li> </ul>
SetDrawFlat(boolean flat)	<p>void</p> <p>Draws the shape flattened when set to true.</p> <p>Parameters:</p> <ul style="list-style-type: none"> <li>• flat: Boolean, draw flat</li> </ul>
SetFrameColor(long color)	<p>void</p> <p>Sets the color of the frame for 3D objects.</p> <p>Parameters:</p> <ul style="list-style-type: none"> <li>• color: long, the RGB color value for coloring the frame</li> </ul>
SetFrameStyle(ChartFrameStyle style)	<p>void</p> <p>Sets the frame style for the chart - none, mesh, contour or both.</p> <p>Parameters:</p> <ul style="list-style-type: none"> <li>• style: ChartFrameStyle, one of the</li> </ul>

	ChartFrameStyle enumerations
SetGradientType(long type)	<p>void</p> <p>Sets the gradient type to use.</p> <p>Parameters:</p> <ul style="list-style-type: none"> <li>• type: long, one of the ChartGradientType enumerations</li> </ul>
SetGroupID(long id)	<p>void</p> <p>Groups series on a stacked chart having the same id. Must be a non-negative number.</p> <p>Parameters:</p> <ul style="list-style-type: none"> <li>• id: long, a non-negative number used to group the series on a chart</li> </ul>
SetLevelRangeMode(long mode)	<p>void</p> <p>Sets the mode for ranges in series.</p> <ul style="list-style-type: none"> <li>• 0 - Minimum and maximum for Series</li> <li>• 1 - Minimum and maximum for Y axis</li> <li>• 2 - Custom</li> </ul> <p>Parameters:</p> <ul style="list-style-type: none"> <li>• mode: long, either 0 or 1 supported</li> </ul>
SetRelatedAxis(string axis, long index)	<p>void</p> <p>Sets the related axis for a series. The related axis is created using the Split</p>

	<p>function of the ChartAxis interface. The axis is first created using Split, then a new series is created, and this function called on it to one of its axes. The axis is specified by the index parameter; the value is one of the ChartAxisIndex enumerations (0 for X, 1 for Y or 2 for Z)</p> <p>Parameters:</p> <ul style="list-style-type: none"> <li>• axis: string, the guid of the axis returned by a ChartAxis.Split method call; returned by the ChartAxis.GetGuid method</li> <li>• index: long, one of the ChartAxisIndex enumerations</li> </ul>
<p>SetStockSeriesType(ChartStockSeriesType type)</p>	<p>void</p> <p>For Stock charts, sets the graphic used to render the series.</p> <p>Parameters:</p> <ul style="list-style-type: none"> <li>• type: ChartStockSeriesType, one of the ChartStockSeriesType enumerations</li> </ul>
<p>SetWireFrame(boolean wired)</p>	<p>void</p> <p>Sets the wireframe option on or off. When set to true, the chart is no longer rendered as a solid object but is instead rendered as a frame composed of wires.</p> <p>Parameters:</p>

- |  |   |
|--|---|
|  | <ul style="list-style-type: none"><li>• <b>wired: Boolean, displays as a wireframe object if true</b></li></ul> |
|--|---|

