

Mapping

Table of Contents

1.	Mapping API	1
2.	Mapping Data and Layers	2-3
3.	Map Styling	4-5
4.	Mapping Rendering	6
5.	Map Interactivity	7

1 Mapping API

Mapping API

The supporting mapping API contains a number of classes and can be found under the `dotnetCHARTING.Mapping` namespace. The chart object contains the property: '**Chart.Mapping**' which exposes all the chart mapping properties necessary to create maps.

The map layers and shapes are laid out in a manner similar to the chart series and element objects. This table demonstrates the relationships.

Mapping	Charting
MapLayerCollection	SeriesCollection
MapLayer	Series
Shape	Element

 See Sample Mapping/IteratingShapes.aspx.

Map Layer

The `MapLayer` object represents a shape file or ecw file and includes data retrieved from those files. Any number of layers can be added to a chart at a time. The object also exposes a number of properties to allow styling manipulation for shapes within a shape file.

The `MapLayer` provides a `DefaultShape` property which holds the default settings that will be applied to all shapes within the map layer.

2 Mapping Data and Layers

MapDataEngine object

The **MapDataEngine** class exposes methods that populate MapLayer objects with data and map information retrieved from shape or ecw files. The following code snippet demonstrates simple usage of this engine loading the states.shp shapefile into a map layer:

```
[C#]
MapLayer myLayer = MapDataEngine.LoadLayer("states.shp");

[Visual Basic]
MapLayer myLayer = MapDataEngine.LoadLayer("states.shp")
```

The myLayer object now contains visual and statistical data about each US state. These states are represented as **Shape** objects. Each shape has a number of attributes that contain the statistical data. This data is stored as key value pairs and the keys for the shape attributes are stored under the string array myLayer.AttributeNames.

 Sample [Mapping/IteratingShapes.aspx](#) styles a number of shapes based on the value of the country code attribute.

Importing database data

One of the above mentioned attribute keys is 'State_Abbr' which represents the state abbreviation. Consider we also have a database containing additional information about these states that we want to use on the map. A column named 'state' in this database contains the same state abbreviations. This will allow matching the map shapes with this data. The first step is to get a DataTable containing the state column along with any other data columns to bind with the map. Next, the mapLayer will be bound with this table.

```
[C#]
myLayer.ImportData(myDataTable, "State_Abbr", "state");

[Visual Basic]
myLayer.ImportData(myDataTable, "State_Abbr", "state")
```

The parameters are the DataTable, the shape attribute and the matching DataTable column name to match. All other columns in the DataTable will be added to the map layer and shapes as additional attributes.

 Sample [Mapping/ImportShapefileData.aspx](#) imports additional information about states from an access database to be used with the map layer.

Add layers to a map

Adding a layer to the map can be done easily with the following code:

```
[C#]
Chart.Mapping.MapLayerCollection.Add(myLayer);

[Visual Basic]
Chart.Mapping.MapLayerCollection.Add(myLayer)
```

Add Points Manually

Shape layers use a single type of shape such as polygons or points. Therefore, a new map layer must be created with points to add to a map. A point can be added based on a Lat/Long GPS coordinate or reversed in the form of Long/Lat as shown below:

```
[C#]
MapLayer ml = new MapLayer();
ml.AddPoint(50,100);
Chart.Mapping.MapLayerCollection.Add(ml);

[Visual Basic]
Dim ml AS new MapLayer()
ml.AddPoint(50,100)
```

```
Chart.Mapping.MapLayerCollection.Add(ml)
```

 Sample: AddMapPoint.aspx

Add Lines manually

Again, a new map layer is required to add lines to a map. Lines can be added using Lat/Long GPS coordinates or reversed in the form of Long/Lat. A line object can be specified in the AddLine parameters to specify the styling when drawn. Line caps are also supported.

 Sample: MapAddLine.aspx

3 Map Styling

Shapes Styling

The **Shape** object exposes a **Background**, **Line**, **ElementMarker**, and **Label** object which determines the shape's appearance. These objects can be manipulated individually for each shape or they can be set simultaneously for all objects through defaults.

The **Chart.Mapping** object as well as the MapLayer object contain a **DefaultShape** property which works the same as a normal DefaultSeries or DefaultElement in a regular chart.

 See sample Mapping/ShapeStyling.aspx

Thematic Mapping

Thematic mapping is a feature by which a color is applied to shapes conditionally. This feature uses the **SmartColor** object. The following sample demonstrates using a smart colors to shade US states based on population ranges.

```
[C#]
Chart.SmartPalette = SmartPalette sp = new SmartPalette();
Color[] cols = new Color[] { Color.FromArgb(2, 255, 0), Color.FromArgb(186, 253, 0) };
sp.Add("POPULATION", new SmartColor(cols[0], new ScaleRange(100000, 1000000)));
sp.Add("POPULATION", new SmartColor(cols[1], new ScaleRange(1000000, 10000000)));

[Visual Basic]
Dim sp As New SmartPalette()
Chart.SmartPalette = sp
Dim cols() As Color = { Color.FromArgb(2, 255, 0), Color.FromArgb(186, 253, 0) }
sp.Add("POPULATION", New SmartColor(cols(0), New ScaleRange(100000, 1000000)))
sp.Add("POPULATION", New SmartColor(cols(1), New ScaleRange(1000000, 10000000)))
```

 For more information on smart colors see [this tutorial \('Element Colors & Palettes' in the on-line documentation\)](#).

Labels

Similar to labels on normal chart elements, shape labels can contain tokens which represent the data attributes they contain. For instance, the shapes from the above example contain an attribute named 'State_Abbr' which represents the state's abbreviation. If the shape label text contains '%State_Abbr' the resulting label on the map will show the value of this attribute, depending on which state the label could be something like "IL", or "CA". This code snippet sets this label text for all the states within *myLayer*.

```
[C#]
myLayer.DefaultShape.Label.Text = "%State_Abbr";

[Visual Basic]
myLayer.DefaultShape.Label.Text = "%State_Abbr"
```

Additional data imported from other databases (See: **Mapping Data and Layers (Section 2)**) can also be represented in these labels.

 See sample Mapping/ImportShapefileData.aspx

LabelOnce Option [New in v5.0]

In cases where a single shape has multiple polygon parts representing islands for example, the control tends to place the shape's label on each polygon of the shape. Using this option will ensure only one label is used for the collection of polygons belonging to the shape.

```
[C#]
mapLayer.LabelOnce = true;

[Visual Basic]
mapLayer.LabelOnce = True
```

 **Sample:** MapLabelOnce.aspx

Shape Groups

Shape files often contain a large number of shapes which may be difficult to differentiate between and label. The **Group** object solves this by conditionally encapsulating a number of shapes. This allows settings to simultaneously be applied to all shapes within that group. A **DefaultShape** property is provided by this Group object to facilitate this feature.

The conditions that determine whether a shape belongs to a group are based on the data attributes of those shapes. For example a condition may be that only shapes with a 'population' attribute greater than 50,000 are included. The following code snippet demonstrates this example.

```
[C#]
MapLayer layer = MapDataEngine.LoadLayer( @"mapFiles/canada.shp");
dotnetCHARTING.Mapping.Group gr = new dotnetCHARTING.Mapping.Group("CNTRY_NAME", "Canada")
gr.DefaultShape.Background.Color = Color.Green;
layer.Groups.Add(gr);
Chart.Mapping.MapLayerCollection.Add(layer);
```

```
[Visual Basic]
Dim layer As MapLayer = MapDataEngine.LoadLayer("mapFiles/canada.shp")
Dim gr As New dotnetCHARTING.Mapping.Group("CNTRY_NAME", "Canada")
gr.DefaultShape.Background.Color = Color.Green
layer.Groups.Add(gr)
Chart.Mapping.MapLayerCollection.Add(layer)
```

 See sample Mapping/ShapeGrouping.aspx

Group Labels

While the styling properties of the default shape are inherited by all the shapes within the group. The Group.Label property is drawn only once on top of the entire group. Tokens are not allowed in this label's text.

```
[C#]
MapLayer layer = MapDataEngine.LoadLayer( @"mapFiles/canada.shp");
dotnetCHARTING.Mapping.Group gr = new dotnetCHARTING.Mapping.Group("CNTRY_NAME", "Canada")
gr.Label = new dotnetCHARTING.Label("Canada", new Font("Arial", 25, FontStyle.Bold), Color.Frc)
layer.Groups.Add(gr);
Chart.Mapping.MapLayerCollection.Add(layer);
```

```
[Visual Basic]
Dim layer As MapLayer = MapDataEngine.LoadLayer("mapFiles/canada.shp")
Dim gr As New dotnetCHARTING.Mapping.Group("CNTRY_NAME", "Canada")
gr.Label = New dotnetCHARTING.Label("Canada", New Font("Arial", 25, FontStyle.Bold), Color.Frc)
layer.Groups.Add(gr)
Chart.Mapping.MapLayerCollection.Add(layer)
```

 See sample Mapping/ShapeGrouping.aspx

4 Mapping Rendering

Rendering

Projections

Projections were created to emulate the surface of the earth on a flat surface such as a monitor. .netCHARTING provides two types of projections; The **Lambert Conic**, and the **Mercator** projection.

The following code snippet demonstrates using the **Lambert Conic Projection**:

```
[C#]
Chart.Mapping.Projection.Type = ProjectionType.LambertConic;
Chart.Mapping.Projection.Parameters = "-10,-20,-32,49";
```

```
[Visual Basic]
Chart.Mapping.Projection.Type = ProjectionType.LambertConic
Chart.Mapping.Projection.Parameters = "-10,-20,-32,49"
```



See samples:

- Mapping/Projections.aspx

Projections and Performance

Projecting ecw files is a CPU intensive operation. To remedy this feature in a real time rendering environment, the projected ecw views can be cached as jpeg images. This works by simply providing the path and name of a jpeg file where the cached image can be stored as shown in the following code snippet.:

```
[C#]
MapLayer myLayer = MapDataEngine.LoadLayer("earth.ecw","earth_ecw.jpg");
```

```
[Visual Basic]
MapLayer myLayer = MapDataEngine.LoadLayer("earth.ecw","earth_ecw.jpg")
```

Zooming

A simple API is provided to facilitate zooming. The feature consists of a **ZoomPercentage** and **ZoomCenterPoint** properties.

Since shape and ecw files are generally based on the Longitude/Latitude coordinate system, choosing a center point of your zoom can be easily accomplished by finding the GPS coordinates of the particular location. This code snippet demonstrates a simple zoom:

```
[C#]
// Zoom 2250% into the coordinates for chicago.
Chart.Mapping.ZoomPercentage = 2250;
Chart.Mapping.ZoomCenterPoint = new PointF(41.9f,-87.65f);
```

```
[Visual Basic]
' Zoom 2250% into the coordinates for chicago.
Chart.Mapping.ZoomPercentage = 2250
Chart.Mapping.ZoomCenterPoint = New PointF( 41.9F, -87.65F)
```

The following table shows how GPS coordinates are represented in .netCHARTING:

GPS	.netCHARTING
Latitude Longitude: 41°54' N 87°39' W	new PointF(41.54f, -87.39f)
Latitude Longitude: 34° 56' S 138° 35' E	new PointF(-34.56f, 138.35f)



See sample Mapping/MapZooming.aspx

5 Map Interactivity

The mapping functionality offers some unique features to enable interactivity.

Shape Hotspots

Shapes have a Hotspot property which allows mouseover tooltips and other interactive functionality. For more information see **hotspots tutorial ('Hotspots' in the on-line documentation)**.

The most useful application of this is to set DefaultShape's hotspot properties as shown in this sample.

 Sample: MapHotspots.aspx

Click to GPS (Lat/Long) coordinates

This feature allows a pixel coordinate on the map to be translated into GPS coordinates of the point based on the map. By capturing a user's click coordinate this information can be translated into GPS coordinates to provide zooming on the map or using the GPS position to query information from other services.

```
[C#]
Chart.FileManager.SaveImage();
PointF gpsPoint = Chart.Mapping.GetLatLongCoordinates("50,100");

[Visual Basic]
Chart.FileManager.SaveImage()
Dim gpsPoint AS Chart.Mapping.GetLatLongCoordinates("50,100")
```

This feature also works with projections applied to the map.

 **Samples:**
 ClickMap.aspx
 ClickMapProjected.aspx
 ClickMapZoom.aspx

Click to Shapes array

From the click pixel position an array of Shape objects that land under the coordinate can be acquired. This can be used to select specific shapes and display its details or modify any of its properties.

```
[C#]
Chart.FileManager.SaveImage();
ArrayList shapes =
Chart.Mapping.GetShapesAtPoint("50,100");

[Visual Basic]
Chart.FileManager.SaveImage()
Dim shapes AS
Chart.Mapping.GetShapesAtPoint("50,100")
```

 Sample:
 IdentifyShape.aspx

 **Note:** The reason the SaveImage method is called is because the chart must be generated to have the information necessary to process these requests.

Ajax Zoomer

The ajax zoomer feature also supports mapping. This will allow the user to interactively zoom and scroll maps on demand.