ComponentOne

Maps for WinForms

ComponentOne, a division of GrapeCity
201 South Highland Avenue, Third Floor
Pittsburgh, PA 15206 USA

Website: http://www.componentone.com
Sales: sales@componentone.com
Telephone: 1.800.858.2739 or 1.412.681.4343 (Pittsburgh, PA USA Office)

Trademarks

The ComponentOne product name is a trademark and ComponentOne is a registered trademark of GrapeCity, Inc. All other trademarks used herein are the properties of their respective owners.

Warranty

ComponentOne warrants that the media on which the software is delivered is free from defects in material and workmanship, assuming normal use, for a period of 90 days from the date of purchase. If a defect occurs during this time, you may return the defective media to ComponentOne, along with a dated proof of purchase, and ComponentOne will replace it at no charge. After 90 days, you can obtain a replacement for the defective media by sending it and a check for $2 5 (to cover postage and handling) to ComponentOne.

Except for the express warranty of the original media on which the software is delivered is set forth here, ComponentOne makes no other warranties, express or implied. Every attempt has been made to ensure that the information contained in this manual is correct as of the time it was written. ComponentOne is not responsible for any errors or omissions. ComponentOne's liability is limited to the amount you paid for the product. ComponentOne is not liable for any special, consequential, or other damages for any reason.

Copying and Distribution

While you are welcome to make backup copies of the software for your own use and protection, you are not permitted to make copies for the use of anyone else. We put a lot of time and effort into creating this product, and we appreciate your support in seeing that it is used by licensed users only.

Table of Contents

Maps for WinForms Overview

ComponentOne introduces **Maps for WinForms**, a data visualization control that displays geographical data on a map. With **Maps for WinForms**, you can view images with smooth zooming, panning, and mapping between screen and geographical coordinates. The Map control also allows you to add custom elements on Map.

You can display geographical information on **Map** control from various built-in and custom sources. By default, **C1Map** provides three built-in sources, aerial, road, and hybrid views which access Microsoft Bing Maps. It also allows you to import Keyhole Markup Language (KML) files if you want to work offline. Additionally, **Maps for WinForms** makes it easy to use the online map tiles of your choice.

Help with WinForms Edition

For information on installing **ComponentOne Studio WinForms Edition**, licensing, technical support, namespaces and creating a project with the control, please visit Getting Started with WinForms Edition.

Map Key Features

Map control includes the following main features, you may find useful:

- **Draw any Geometry**
  You can draw geometries/shapes/polygons/paths with geographic coordinates on top of the map using vector layer of **Map** control. The vector layer is useful to draw the following:
    - Political borders (such as countries or states)
    - Geographic details (for example, showing automobiles or airplane routes)
    - Choropleth maps (based on statistical data, such as showing population per country)

  Vector layer can also be used instead of the regular Microsoft Virtual Earth source to show a world map representation.

- **Rich Geographical Information**
  Display rich geographical information from various sources, including Bing Map or any custom source. For example, you can build your own source for Yahoo! Maps.

- **Layers Support**
  Use layers to add your own custom elements to the maps which are linked to geographical locations. For more information, see Layers.

- **KML Support**
  The vector layer supports basic KML import/export (KML is the standard file format to exchange drawings on top of maps).

- **Display a Large Number of Elements on the Map**
  C1Map allows virtualization of local and server data. Using its virtual layer, **Map** control only displays and requests the elements currently visible.

- **Zoom, Pan, and Map Coordinates**
  Map control supports zooming and panning using the mouse or the keyboard. It also supports mapping between screen and geographical coordinates.

Quick Start: Map for Win Forms

This quick start guide familiarizes you with some of the features of **Map** control. In this section, you learn to create a new WinForms application in Visual Studio, add the **C1Map** control to the application, add vector data to it and run the application.

- **Step 1: Creating an application with C1Map control**
- **Step 2: Adding vector data to Map**
- **Step 3: Running the application**

Step 1: Creating an application with C1Map control

1. Create a new Windows Forms Application
2. Add **C1Map** control to the form in your application.
3. Set the **Dock** property of **C1Map** control to **Fill**.

Step 2: Adding vector data to Map

1. Add the following namespaces in code view:
   - using C1.FlexMap
   - using C1.Win.Map
2. Use the following code to display map in the **C1Map** control and add vector layer to it:
   - **Visual Basic**

```
'specify tile source used by the map
C1Map1.TileLayer.TileSource = New VirtualEarthRoadSource()

'create a vector layer and add it to the map
Dim layer As New C1.Win.Map.VectorLayer()
C1Map1.Layers.Add(layer)
```

   - **C#**

```
//specify tile source used by the map
c1Map1.TileLayer.TileSource = new VirtualEarthRoadSource();

//create a vector layer and add it to the map
C1.Win.Map.VectorLayer layer = new C1.Win.Map.VectorLayer();
c1Map1.Layers.Add(layer);
```

3. Add a vector placemark element to the layer using VectorPlacemark class and customize the element using the following code:
   - **Visual Basic**

```
'Create a vector placemark and add it to the layer
Dim vpl As New C1.Win.Map.VectorPlacemark()
layer.Items.Add(vpl)

'customize the vector placemark
vpl.Style.BackColor = Color.Aqua
vpl.Marker.Shape = MarkerShape.Diamond
vpl.Marker.Size = New SizeF(10, 10)

'set the geometry shape for vector placemark
vpl.Geometry = New GeoPoint(75, 25)
```

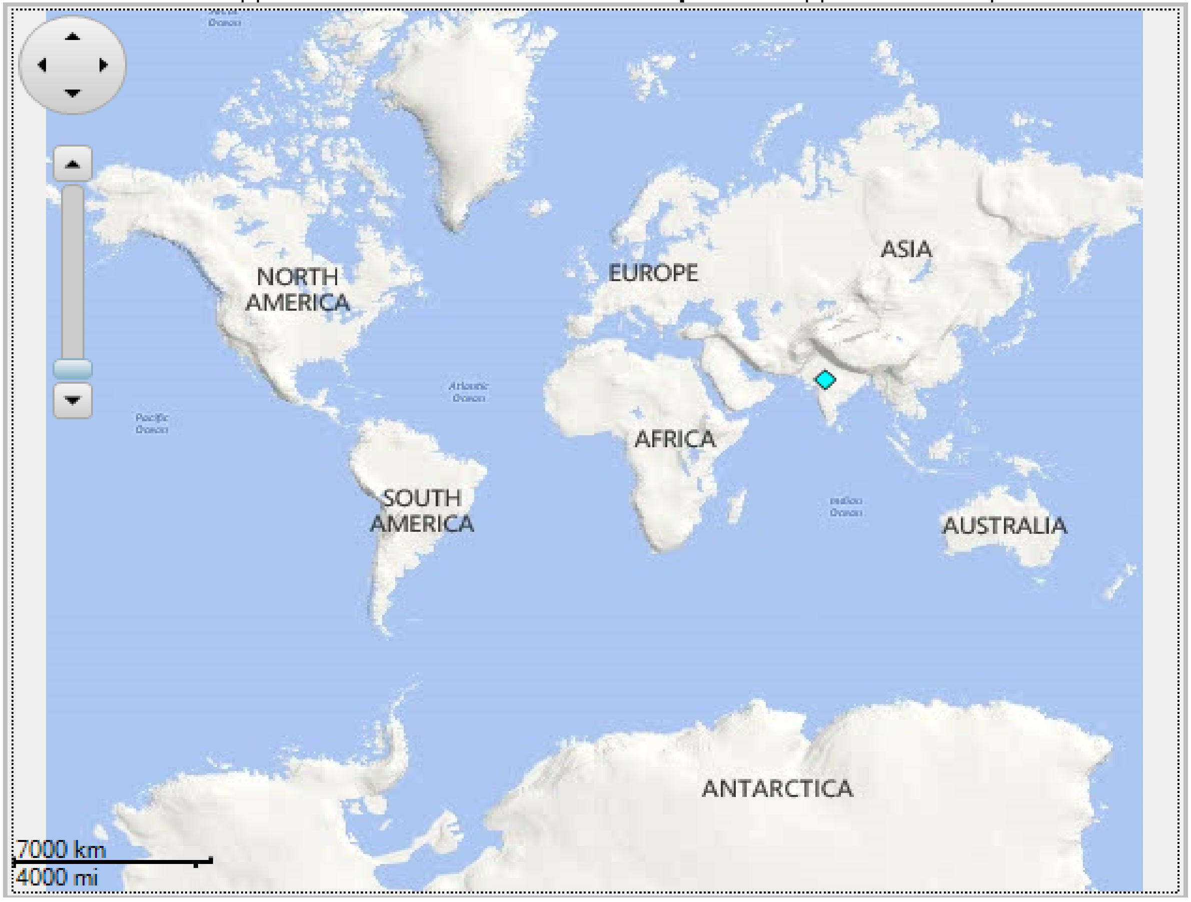
   - **C#**

```
//Create a vector placemark and add it to the layer
C1.Win.Map.VectorPlacemark vpl = new C1.Win.Map.VectorPlacemark();
layer.Items.Add(vpl);

//customize the vector placemark
vpl.Style.BackColor = Color.Aqua;
vpl.Marker.Shape = MarkerShape.Diamond;
vpl.Marker.Size = new SizeF(10, 10);

//set the geometry shape for vector placemark
vpl.Geometry = new GeoPoint(75,25);
```

You can add any other vector elements such as, **VectorPolyline** and/or **VectorPolygon** to the map according to your requirements. To know more on adding these elements to the map, refer Adding VectorPolygon and Adding VectorPolyline to map.

Step 3: Running the application

Press **F5** to run the application and observe how the **C1Map** control appears with the placemark vector added to it.



In addition to adding placemarks, you can add various types of vector data on a map. C1Maps also allows you to add markers on a map, customize them, and add image as markers as well, and add legends. For more information on further information, see Working with Map Control.

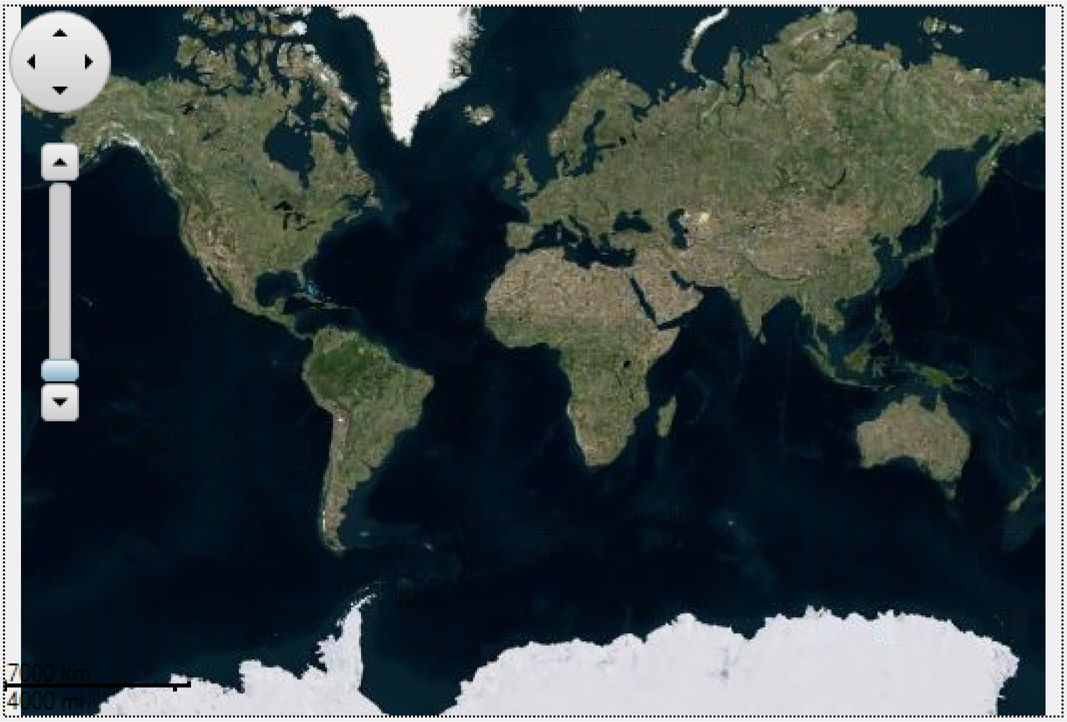
Map Control Basics

This section describes the basic concepts of **Map** control.

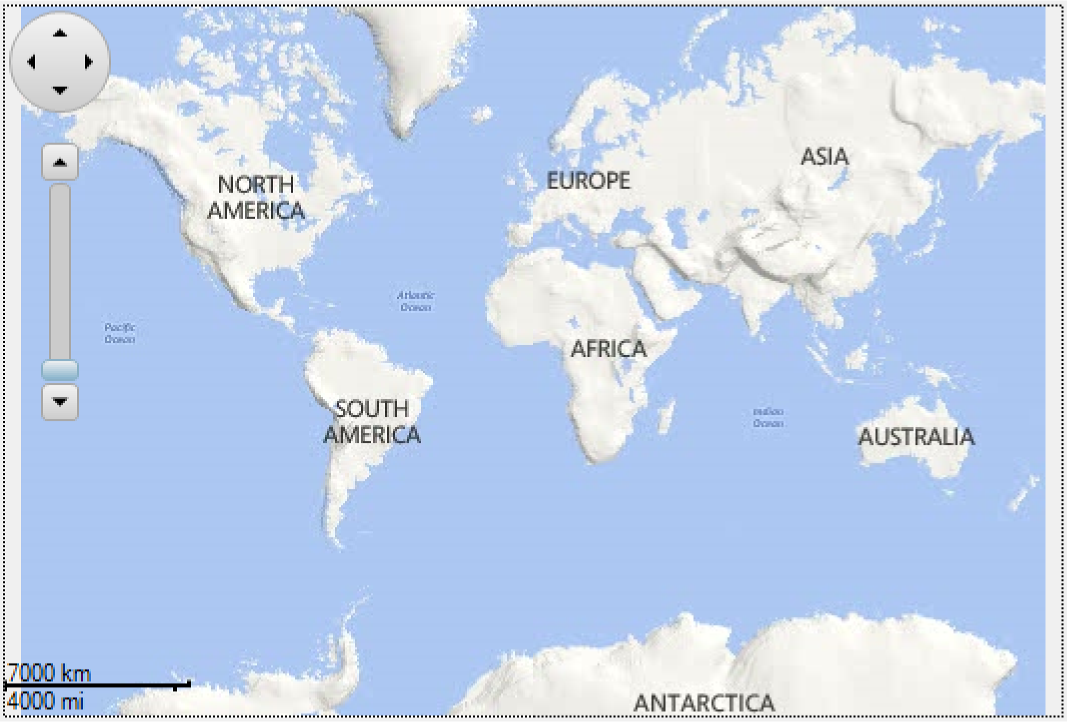
Map Source

C1Map can display geographical information on a map from several sources. You can use TileSource property to specify the tile source used by the map for its tile layer. C1Map provides three built in tile sources which are listed below along with the code required to :

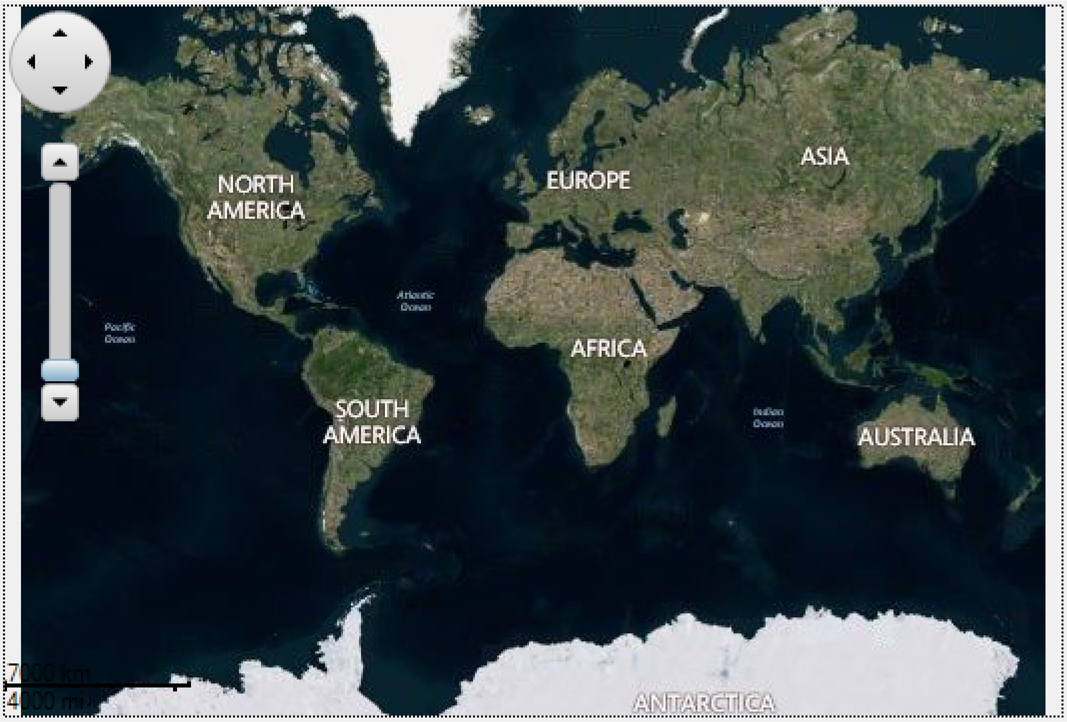
- **VirtualEarthAerialSource**



- **VirtualEarthRoadSource**



- **VirtualEarthHybridSource**



Coordinate Systems

Map control uses three coordinate systems:

- **Geographic** coordinates mark points in the world using latitude and longitude. This coordinate system is not Cartesian, which means the scale of the map may change as you pan.
- **Logical** coordinates go from 0 to 1 on each axis for the whole extent of the map, and they are easier to work with because they are Cartesian coordinates.
- **Screen** coordinates are the pixel coordinates of the Control relative to the top-left corner. These are useful for

positioning items within the control and for handling mouse events.

C1Map provides six methods for converting between these coordinate systems: ScreenToGeographic, ScreenToLogic, GeographicToScreen, GeographictoLogic, LogictoGeographic, and LogicToScreen.

Information Layers

In addition to the geographical information provided by the source, you can add layers of information to the map. C1Map includes three layers by default, namely Tile layer, Vector layer, and Virtual layer.

Legal Requirements

Map control allows you to use geographical information from **Bing Maps**. Before using this service, you should check the licensing requirements associated with it. These licensing terms can be found at:

http://www.microsoft.com/maps/product/terms.html

Layers

Layers help display geographical data on a map. C1Map supports three different layers to display the data:

- **Tile Layer**: The background layer in a map where the map tiles are displayed. For more information, see Tile Layer.
- **Vector Layer** The layer that displays vector data, like lines, placemark, and polygons, whose vertices are geographically positioned. For more information, see Vector Layer.
- **Virtual Layer** This layer displays items that are virtualized; this means they are only loaded when the region of the map they belong to is visible. For more information, see Virtual Layer.

Tile Layer

Tile layer is used to display the map tiles on a map using different tile sources. This is the first layer to be added to the map before adding any data to it. You can add the tile layer on a map using C1Map.TileLayer property. Once you add the tile layer on a map, you need to set the tile source for it using TileSource property of TileLayer class.

Adding built-in Tile Source

C1Map uses three built-in tile sources, VirtualEarthAerialSource, VirtualEarthRoadSource, and VirtualEarthHybridSource. You can use any of these to define the tile source for the tile layer.

Following code illustrates the use of C1Map.TileLayer and TileSource property:

- **Visual Basic**

```
C1Map1.TileLayer.TileSource = New VirtualEarthAerialSource()
```

- **C#**

```
c1Map1.TileLayer.TileSource = new VirtualEarthAerialSource();
```

Adding custom Tile Source

In addition to the three built-in sources, C1Map supports custom tile source that can be added to the tile layer of a map. For adding a custom tile source, you need to create your own class that implements C1.FlexMap.ITileSource interface. In the following code, we have created a class named OpenStreetTileSource that is used to add custom tile source to your map and uses the OpenStreetMap service:

- **Visual Basic**

```
' Provide custom tile source which implements C1.FlexMap.ITileSource interface
C1Map1.TileLayer.TileSource = New Map_Layers.OpenStreetTileSource()
```

- **C#**

```
// Provide custom tile source which implements C1.FlexMap.ITileSource interface
c1Map1.TileLayer.TileSource = new OpenStreetTileSource();
```

Vector Layer

The Vector layer allows you to place various objects with geographic coordinates on the map.

Following are the main vector elements that can be used on the vector layer:

- **VectorPlacemark** – An object attached to the geographical point. The placemarks have scale-independent geometry where coordinates are expressed in pixel coordinates and optional label (any UIElement). Typical usage: labels, icons, marks on the map. For more information, see Adding a Label.

- **VectorPolyline** – Similar to Polygon class, however it does not need to be a closed shape. The polyline is formed using geographical coordinates. Typical usage: paths, routes. For more information, see Adding a Polyline.
- **VectorPolygon** – Similar to Polyline class, but it draws a polygon, which is a connected series of lines that form a closed shape. The polygon is formed using geographical coordinates. Typical usage: borders, regions. For more information, see Adding a Polygon.

Virtual Layer

The virtual layer displays elements over the map supporting virtualization and asynchronous data loading. It can be used to display an unlimited number of elements, as long as not many of them are visible at the same time.

The division of layer for virtualization is defined using VirtualLayer.Slices collection of map slices. Each map slice defines a minimum zoom level for its division, and the maximum zoom level for a slice is the minimum zoom layer of the next slice (or, if it is the last slice, its maximum zoom level is the maximum zoom of the map). In turn, each slice is divided in a grid of latitude/longitude divisions.

For adding a virtual layer on a map, you need to create your own class that implements IMapVirtualSource interface. We have created a class named VirtualSource which consists of the geometries of placemarks and markers required to add a virtual layer on a map. On creating your class, you would be required to add C1.Win dll to it. The following code is used to add the virtual layer on a map:

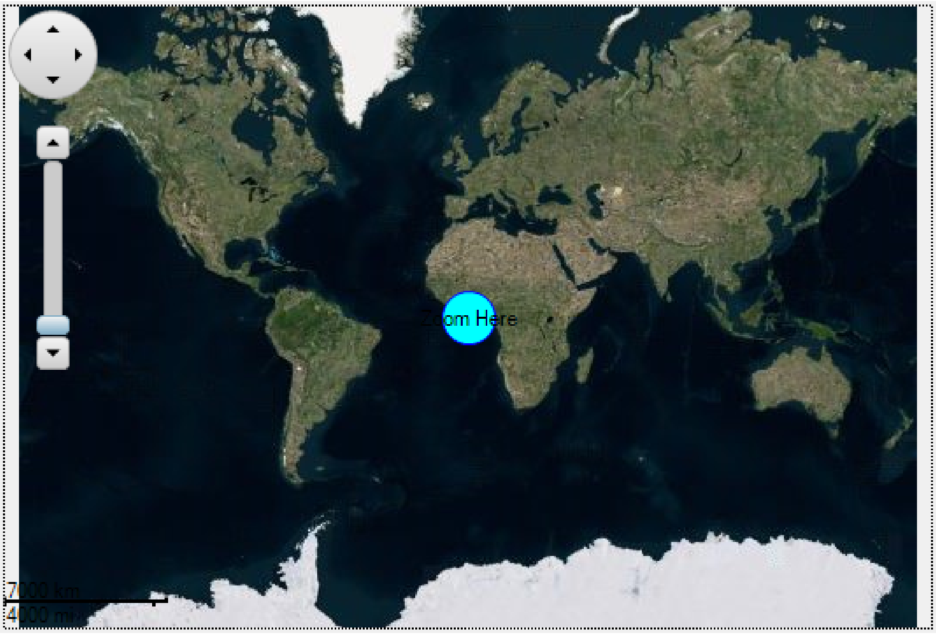
- **Visual Basic**

```
C1Map1.TileLayer.TileSource = New VirtualEarthAerialSource()

Dim vlayer = New C1.Win.Map.VirtualLayer()
C1Map1.Layers.Add(vlayer)
vlayer.LabelVisibility = LabelVisibility.Visible

' the Slices collection defines the regions that the virtual layer will get items for when needed.
vlayer.Slices.Add(New MapSlice(0, 1, 1))
vlayer.Slices.Add(New MapSlice(1, 4, 4))

' needs provide a virtual source which implements C1.Win.Map.IMapVirtualSource interface.
vlayer.ItemsSource = New VirtualSource()

' the default style
vlayer.Style.Stroke.Width = 1
vlayer.Style.Stroke.Color = Color.Blue
vlayer.Style.BackColor = Color.Aqua
```

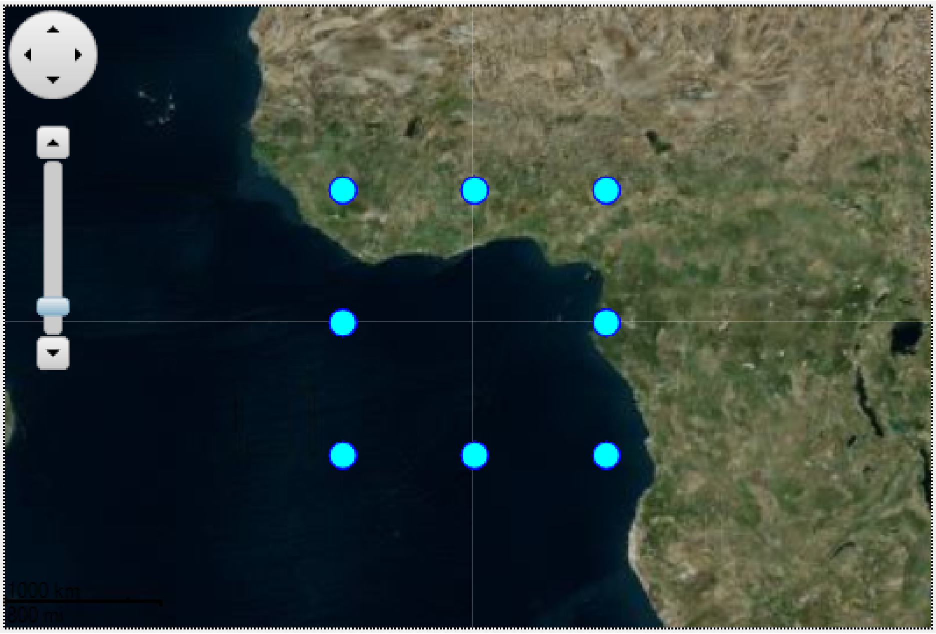
- **C#**

```
c1Map1.TileLayer.TileSource = new VirtualEarthAerialSource();

var vlayer = new C1.Win.Map.VirtualLayer();
c1Map1.Layers.Add(vlayer);
vlayer.LabelVisibility = LabelVisibility.Visible;

// the Slices collection defines the regions that the virtual layer will get items for when needed.
vlayer.Slices.Add(new MapSlice(0, 1, 1));
vlayer.Slices.Add(new MapSlice(1, 4, 4));

// needs provide a virtual source which implements C1.Win.Map.IMapVirtualSource interface.
vlayer.ItemsSource = new VirtualSource();

// the default style
vlayer.Style.Stroke.Width = 1;
vlayer.Style.Stroke.Color = Color.Blue;
vlayer.Style.BackColor = Color.Aqua;
```

The output will look like the image given below with a marker at the center on the map:



As you zoom in, you will be able to see the marker split into multiple placemarks indicating virtualization.



Markers

A marker is used to display a location on a map. C1Map allows you to use following types of built-in marker shapes:

- Circle
- Triangle
- Square
- Diamond
- Star

C1Map also allows you to use markers with custom shapes. To create these custom shapes, you first need to create a class that implements C1.Win.Map.CustomShape class. In addition, you can add images as markers on a map. To do so, you need to use C1.Win.Map.MarkerImageShape class that represents a custom marker shape which is an image. For more information, see Adding Shape Marker, Adding Custom Marker, and Adding Image as Marker.

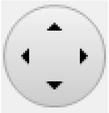
Map Tools

C1Map provides the viewing container which supports zooming and panning using multi-touch gestures. Users can double tap or stretch to zoom-in while pinching to zoom-out. Dragging will pan the map in any direction. You can enable inertial scrolling or panning by setting one property.

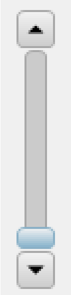
When panning or scrolling the map in the east-west direction, you can allow the map to loop back endlessly by setting the HorizontalLoop property. The default value of the property is **false**. Setting the HorizontalLoop property to **true** causes the map to not stop rendering at ±180°E/W.

The map control has the following tools:

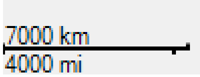
- Pan Tool



- Zoom Tool

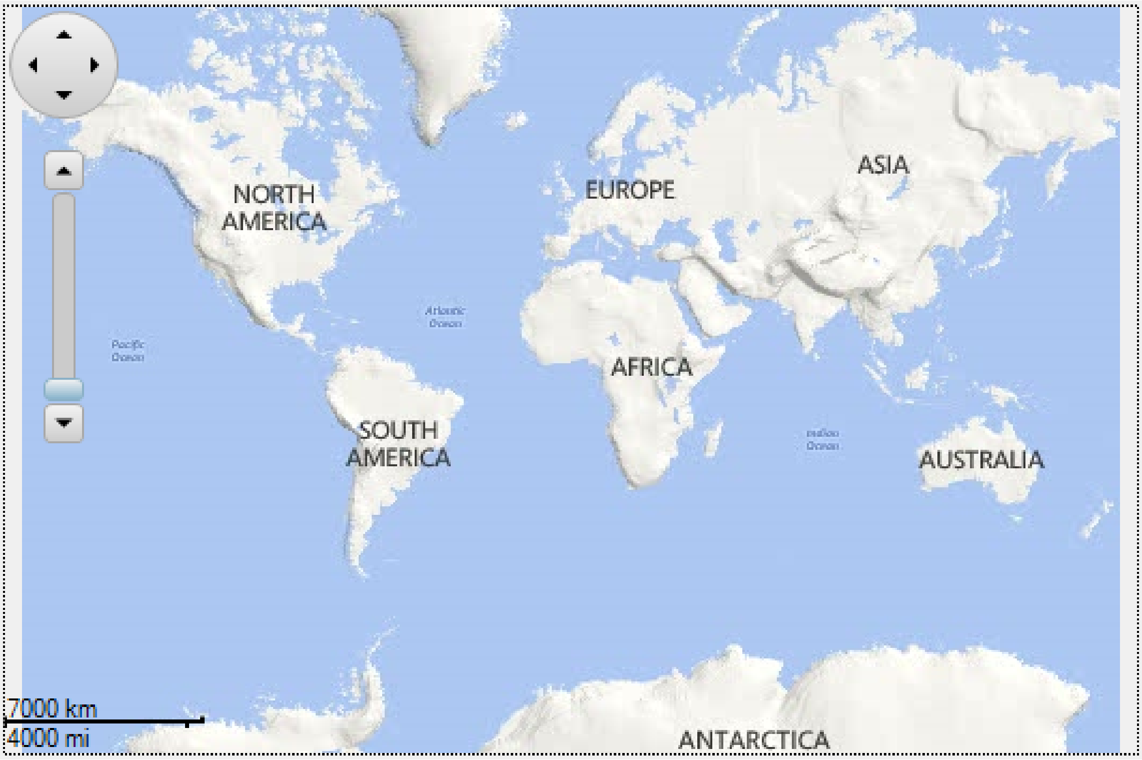


- Distance Scale Tool



C1Map allows you to customize the map tools. To do so, you can use the properties of MapPanTool, MapZoomTool, and MapDistanceScale class.

By default, in Map control all these tools are placed on the left side of the map.



You can easily change the position of all these tools from the design view using PanTool, ZoomTool, and DistanceScale properties of C1Map class. This can be done in the code view as well. To know how to change the position of the map tools in code view, see Repositioning Map Tools.

Working with Map Control

Map control raises the bar on image viewing by allowing end-users to enjoy extreme close-ups with high-resolution images and smooth transitions. They also support layers that allow you to superimpose your own custom elements to the maps.

We hope till now you are familiar with the basics of the Map control and know how to use it in general.

The following topics will explain the working with Map control.

Adding Vector Data

Vector data is used as a representation of real world objects (like rivers, trees, roads, etc.) using labels, polylines, polygons, and shapes on a map surface. The following topics describe how you can add a label, polyline, polygon, and display shapes using KML file and Shapefile on a map surface.

Adding a Label

C1Map allows you to add custom elements like text label on the map surface. Label can be added on a map using placemarks. You can use the following code to add a label to a geographic point using VectorLayer and VectorPlacemark:

- **Visual Basic**

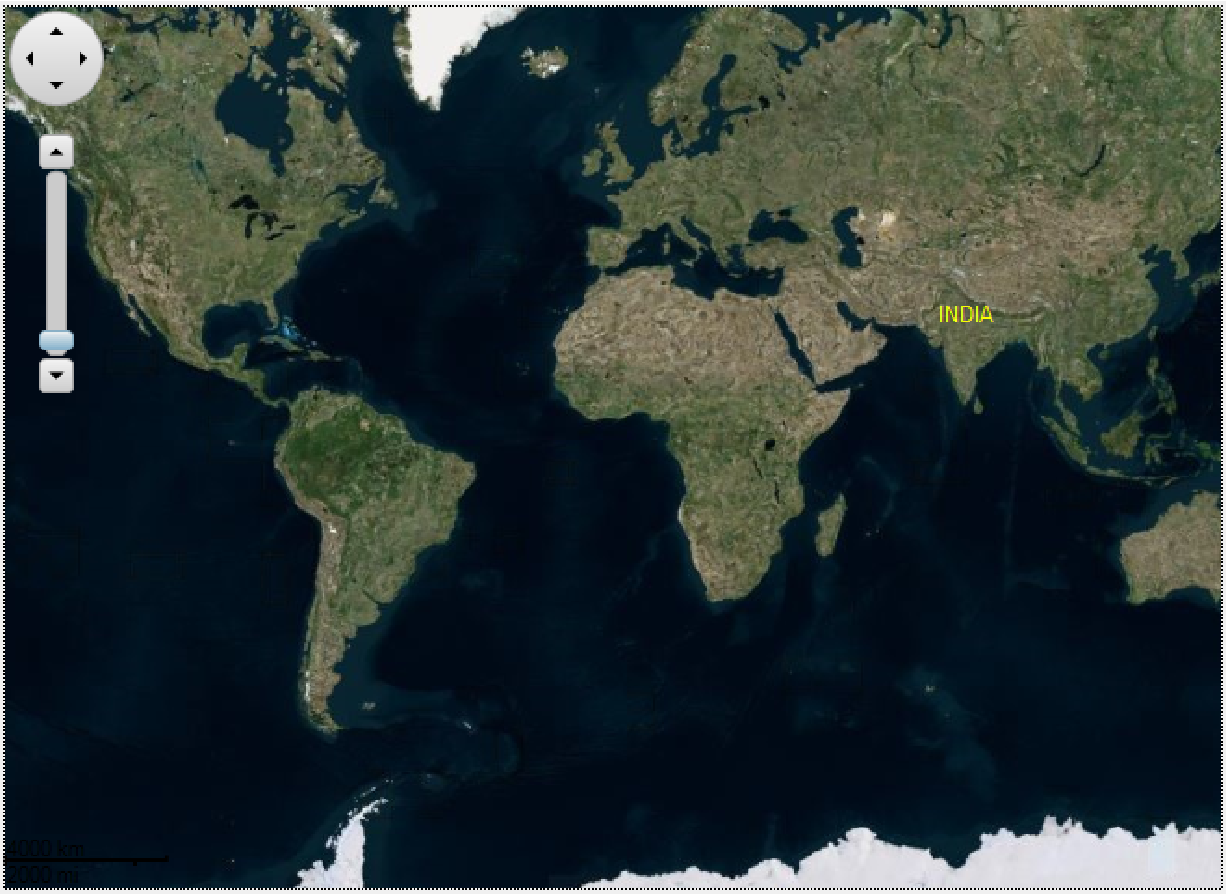
```
C1Map1.TileLayer.TileSource = New VirtualEarthAerialSource()

Dim vl = New C1.Win.Map.VectorLayer()
C1Map1.Layers.Add(vl)

Dim pm = New C1.Win.Map.VectorPlacemark()
pm.Geometry = New GeoPoint(78, 28)
vl.Items.Add(pm)

vl.LabelVisibility = LabelVisibility.Visible
pm.Marker.Caption = "INDIA"
pm.LabelStyle.ForeColor = Color.Yellow
```

- **C#**

```
c1Map1.TileLayer.TileSource = new VirtualEarthAerialSource();

var vl = new C1.Win.Map.VectorLayer();
c1Map1.Layers.Add(vl);

var pm = new C1.Win.Map.VectorPlacemark();
pm.Geometry = new GeoPoint(78,28);
vl.Items.Add(pm);

vl.LabelVisibility = LabelVisibility.Visible;
pm.Marker.Caption = "INDIA";
pm.LabelStyle.ForeColor = Color.Yellow;
```

The above code adds a label to the map, gives a caption to it, and sets the font color for the label caption.

The image given below shows **Map** control with a label added on a geographic point – the geographic coordinates of India.



C1Map also allows you to customize the label added on a map. You can use the following properties to make further customizations to the label and marker:

Here is a list of Marker Properties.

Properties	Description
Shape	Used to set the shape for a marker.
CustomShape	Specifies a custom shape for the marker.
LabelPosition	Specifies the label position relatively to the marker.
Size	Specifies the size of marker shape.

You can also use other properties to set the style for Label using LabelStyle property of VectorPlacemark class. With this property, you can change the background color of label, width, and round radius of the label borders.

Adding a Polyline

C1Map allows you to connect geographic coordinates of a map with a polyline by using VectorPolyline class. The VectorPolyline class accepts a list of GeoPoints to define the location and nodes of a polyline on a map. The following code can be used to add a polyline on the map surface and customize it:

- **Visual Basic**

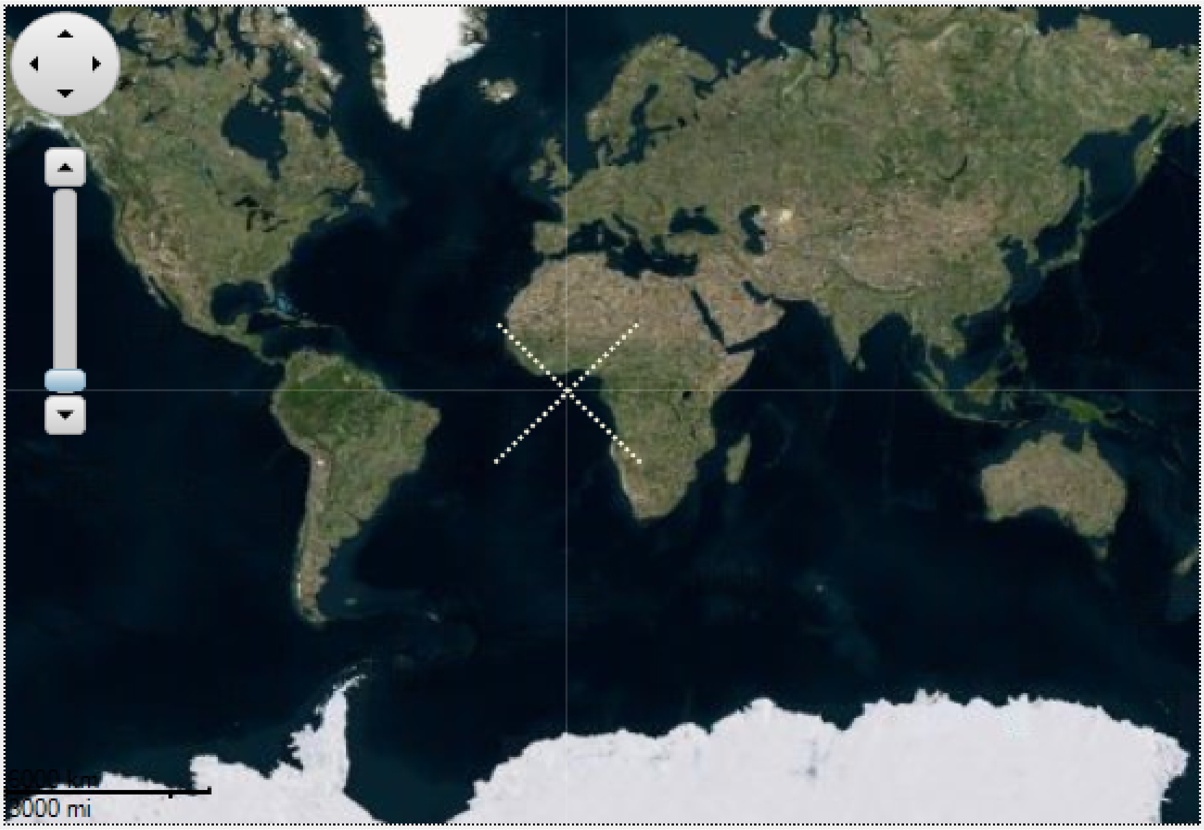
```
C1Map1.TileLayer.TileSource = New VirtualEarthAerialSource()

Dim vl = New C1.Win.Map.VectorLayer()
```

```
C1Map1.Layers.Add(vl)

Dim pline = New C1.Win.Map.VectorPolyline()
vl.Items.Add(pline)

'Provide the stroke style to the polyline
pline.Style.Stroke.Width = 2
pline.Style.Stroke.Style = DashStyle.Dot
pline.Style.Stroke.Color = Color.Beige

'Provide the geometry fro polyline
pline.Geometry = New GeoMultiLineString(New GeoLineString() _
{New GeoLineString(New GeoPoint() {New GeoPoint(20, -20),
New GeoPoint(-20, 20)}), New GeoLineString(New GeoPoint() _
{New GeoPoint(-20, -20), New GeoPoint(20, 20)})})
```

- **C#**

```
c1Map1.TileLayer.TileSource = new VirtualEarthAerialSource();

var vl = new C1.Win.Map.VectorLayer();
c1Map1.Layers.Add(vl);

var pline = new C1.Win.Map.VectorPolyline();
vl.Items.Add(pline);

//Provide the stroke style to the polyline
pline.Style.Stroke.Width = 2;
pline.Style.Stroke.Style = DashStyle.Dot;
pline.Style.Stroke.Color = Color.Beige;

//Provide the geometry fro polyline
pline.Geometry = new GeoMultiLineString(
    new GeoLineString[]
    {
        new GeoLineString(new GeoPoint[]
        {
            new GeoPoint(20, -20),
            new GeoPoint(-20, 20),
        }),
        new GeoLineString(new GeoPoint[]
        {
            new GeoPoint(-20, -20),
            new GeoPoint(20, 20),
        })
    });
```

The following image depicts **Map** control with four geographical coordinates connected by a polyline:



Adding a Polygon

C1Map allows you to add a polygon on the top of a map using VetorPolygon class. The VectorPolygon class accepts a list of GeoPoints that define the location and shape of the polygon. The following code shows how to create a polygon on a map surface:

- **Visual Basic**

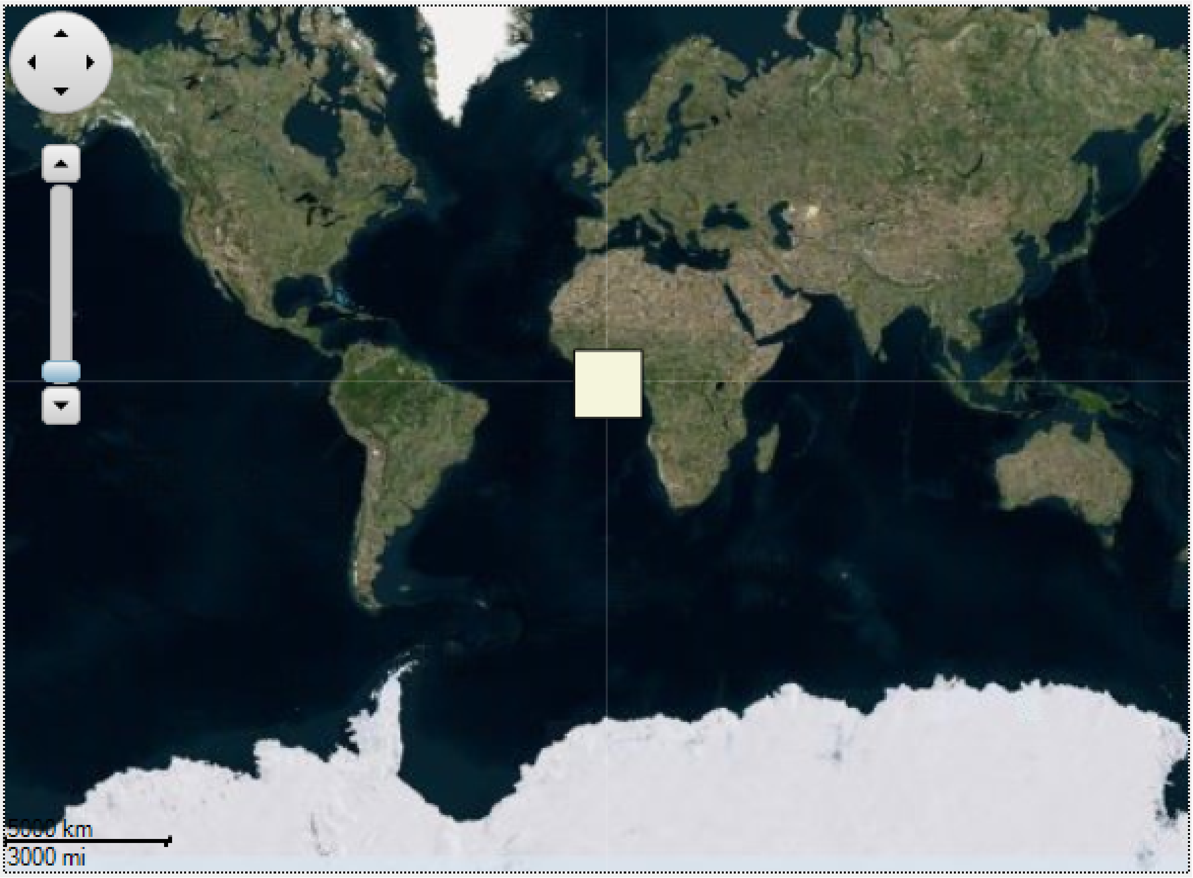
```
C1Map1.TileLayer.TileSource = New VirtualEarthAerialSource()

Dim vl = New C1.Win.Map.VectorLayer()
C1Map1.Layers.Add(vl)

' Can add the polygon vector (C1.Win.Map.VectorPolygon) into a vector layer.
Dim polygon = New C1.Win.Map.VectorPolygon()
vl.Items.Add(polygon)

' Customize the polygon
polygon.Style.BackColor = Color.Beige

' Provide the geopoint location for the polygon
polygon.Geometry = New GeoPolygon(New GeoLinearRing() _
{New GeoLinearRing(New GeoPoint() {New GeoPoint(-10, -10),
New GeoPoint(-10, 10), New GeoPoint(10, 10), New GeoPoint(10, -10)})})
```

- **C#**

```
c1Map1.TileLayer.TileSource = new VirtualEarthAerialSource();

var vl = new C1.Win.Map.VectorLayer();
c1Map1.Layers.Add(vl);
```

```
// Can add the polygon vector (C1.Win.Map.VectorPolygon) into a vector layer.
var polygon = new C1.Win.Map.VectorPolygon();
vl.Items.Add(polygon);

// Customize the polygon
polygon.Style.BackColor = Color.Beige;

// Provide the geopoint location for the polygon
polygon.Geometry = new GeoPolygon(new GeoLinearRing[]
{
    new GeoLinearRing(new GeoPoint[]
    {
        new GeoPoint(-10,-10),
        new GeoPoint(-10,10),
        new GeoPoint(10,10),
        new GeoPoint(10,-10),
    }),
});
```

The following image depicts **Map** control with four geographical coordinates connected by a polygon:



Displaying Shapes using a KML File

KML is an XML-based language for geographic visualization and annotation that was originally created for Google Earth. For more information, see https://developers.google.com/kml/documentation/.

KML import is performed by C1.FlexMap.KmlReader class that has static methods that create collection of vector objects from the supplied KML source (string or stream). The collection can be easily added to the VectorLayer.

You can also add shapes on the map using zipped KML files (KMZ). The following code uses a KMZ file to add shapes on the map:

Visual Basic

```
Dim vl = MapReader.LoadKmlFile("WorldMap.kmz", Function(vector, data)
Dim name = TryCast(data("name"), String)
vector.Tag = name

'  customizing the vector data
Dim fillColor = TryCast(data(KmlReader.StyleFillColor), String)
If fillColor IsNot Nothing Then
        vector.Style.BackColor = MapReader.GetKmlStyleColor(fillColor)
End If

Dim placemark = TryCast(vector, C1.Win.Map.VectorPlacemark)
If placemark IsNot Nothing Then
        placemark.Marker.Caption = name
        placemark.Lod = New LOD(0, 0, 2, 20)
End If

End Function)
vl.LabelVisibility = LabelVisibility.AutoHide

' can set default style for vectors and labels, instead of set the style 
through each vector
vl.Style.Stroke.Color = Color.Blue
vl.Style.Stroke.Width = 1
vl.LabelStyle.ForeColor = Color.Green

c1Map1.Layers.Add(vl)
```

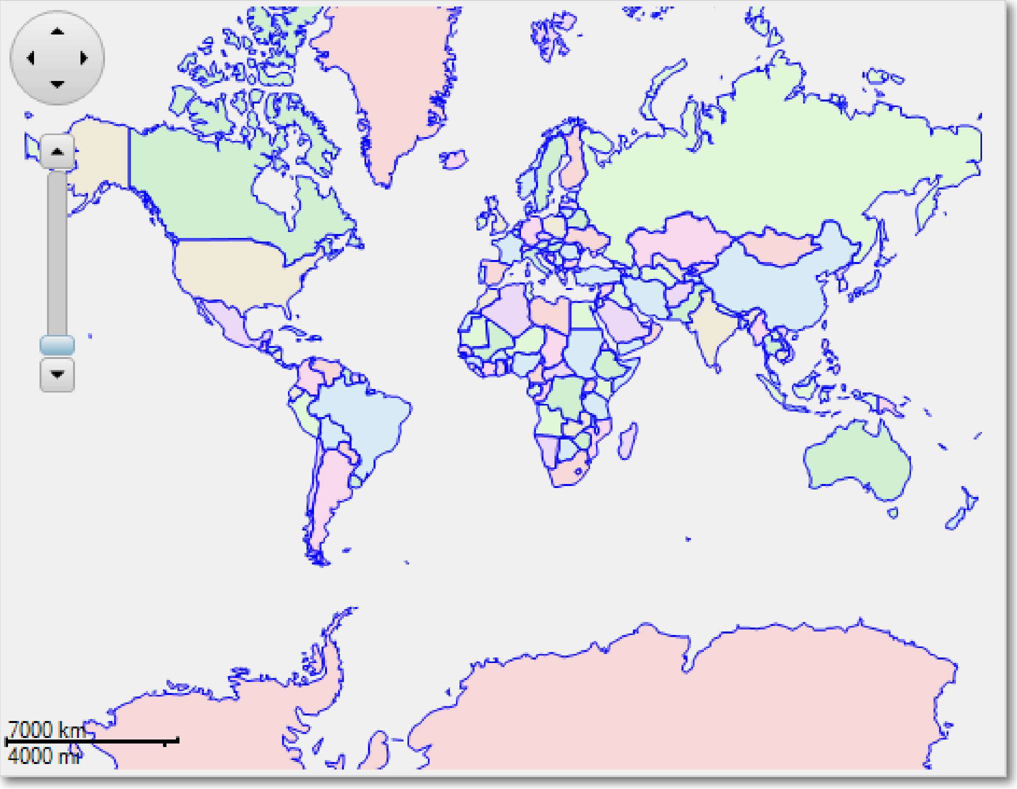
- **C#**

```
var vl = MapReader.LoadKmlFile("WorldMap.kmz", (vector, data) =>
{
    var name = data["name"] as string;
    vector.Tag = name;

    //  customizing the vector data
    var fillColor = data[KmlReader.StyleFillColor] as string;
    if (fillColor != null)
    {
        vector.Style.BackColor = MapReader.GetKmlStyleColor(fillColor);
    }

    var placemark = vector as C1.Win.Map.VectorPlacemark;
    if (placemark != null)
    {
        placemark.Marker.Caption = name;
        placemark.Lod = new LOD(0, 0, 2, 20);
    }
});
vl.LabelVisibility = LabelVisibility.AutoHide;

// can set default style for vectors and labels, instead of set the style through each vector
vl.Style.Stroke.Color = Color.Blue;
vl.Style.Stroke.Width = 1;
vl.LabelStyle.ForeColor = Color.Green;
```

```
c1Map1.Layers.Add(vl);
```

In the above code, we have referred a class named MapReader that we have created. This class reads the vector data from the KMZ file.

On adding shapes from a KMZ file, the map will look similar to the image given below:



Displaying Shapes using a Shapefile

Shapefile is a popular vector data format for geographic visualization of data. It represents data using vector elements such as polylines and polygons. This type of file is usually stored in two formats:

- .shp - A file with shp format comprises data about vector elements
- .dbf - A file with dbf format comprises the attributes of the vector elements in the corresponding shp file.

The following code loads both the shp and dbf format files using C1.FlexMap.ShapeReader class and adds the loaded data to the vector layer on a map:

Visual Basic

```
Dim layerUsa = MapReader.LoadShpFile("states.shp", "states.dbf",
Function(vector, data)
vector.Tag = data("STATE_NAME")

End Function)
c1Map1.Layers.Add(layerUsa)

layerUsa.Style.BackColor = Color.Purple
layerUsa.Style.Stroke.Color = Color.LightGray

c1Map1.Viewport.Center = New C1.Win.Interop.Point(-115, 50)
```

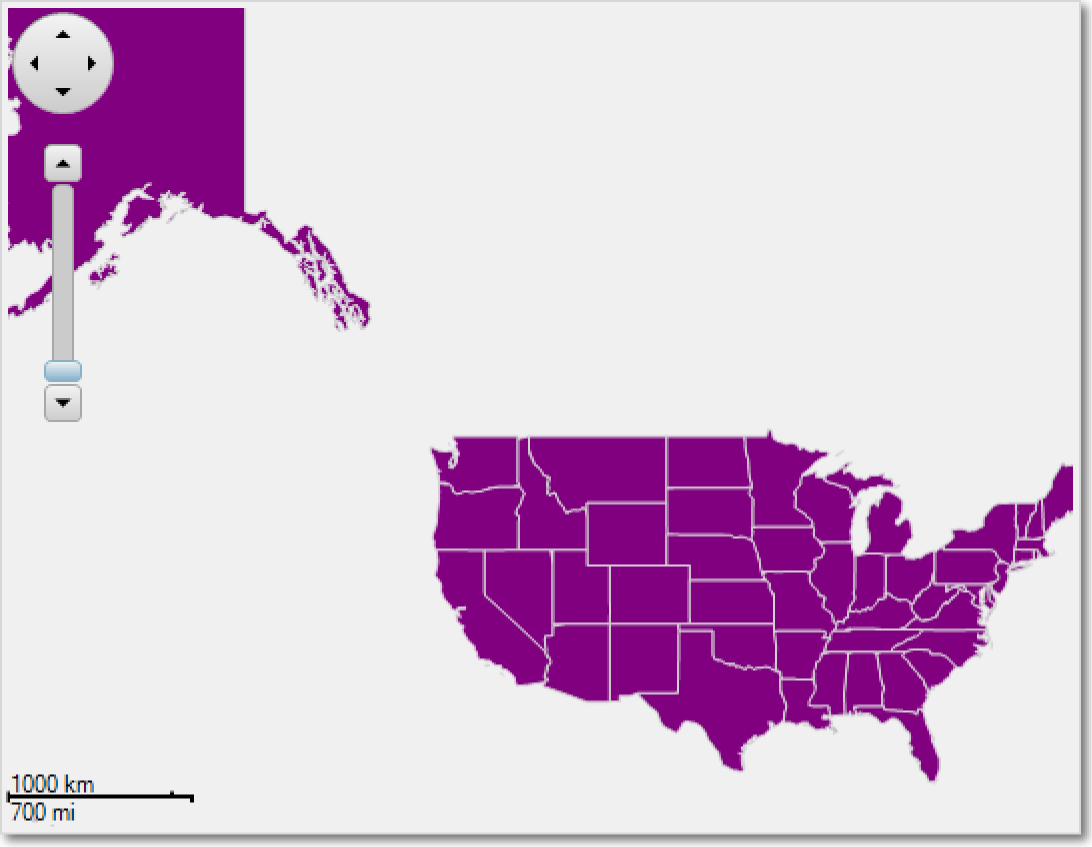
```
c1Map1.Viewport.MinZoom = 2
c1Map1.Viewport.Zoom = 2
```

- **C#**

```
var layerUsa = MapReader.LoadShpFile("states.shp", "states.dbf",
            (vector, data) =>
            {
                vector.Tag = data["STATE_NAME"];
            });
c1Map1.Layers.Add(layerUsa);

layerUsa.Style.BackColor = Color.Purple;
layerUsa.Style.Stroke.Color = Color.LightGray;

c1Map1.Viewport.Center = new C1.Win.Interop.Point(-115, 50);
c1Map1.Viewport.MinZoom = 2;
c1Map1.Viewport.Zoom = 2;
```

In the above code, we have referred a class named MapReader that we have created. This class reads the vector data from the shp and dbf files.

On adding shapes from shp and dbf files, the map will look similar to the image given below:



Adding Markers

A marker is used to display location on a map using shapes or images. The **Map** control allows you to use in-built shape markers, create your own custom markers, and use images as markers. The following topics describe how to add all these markers on a map surface.

Adding Shape Marker

For adding the in-built shape markers, you first need to add VectorPlacemark on the surface of the map. On adding the **VectorpPlacemark**, you can easily customize the marker. The following code can be used to add a marker on a map and set the shape and size of the marker:

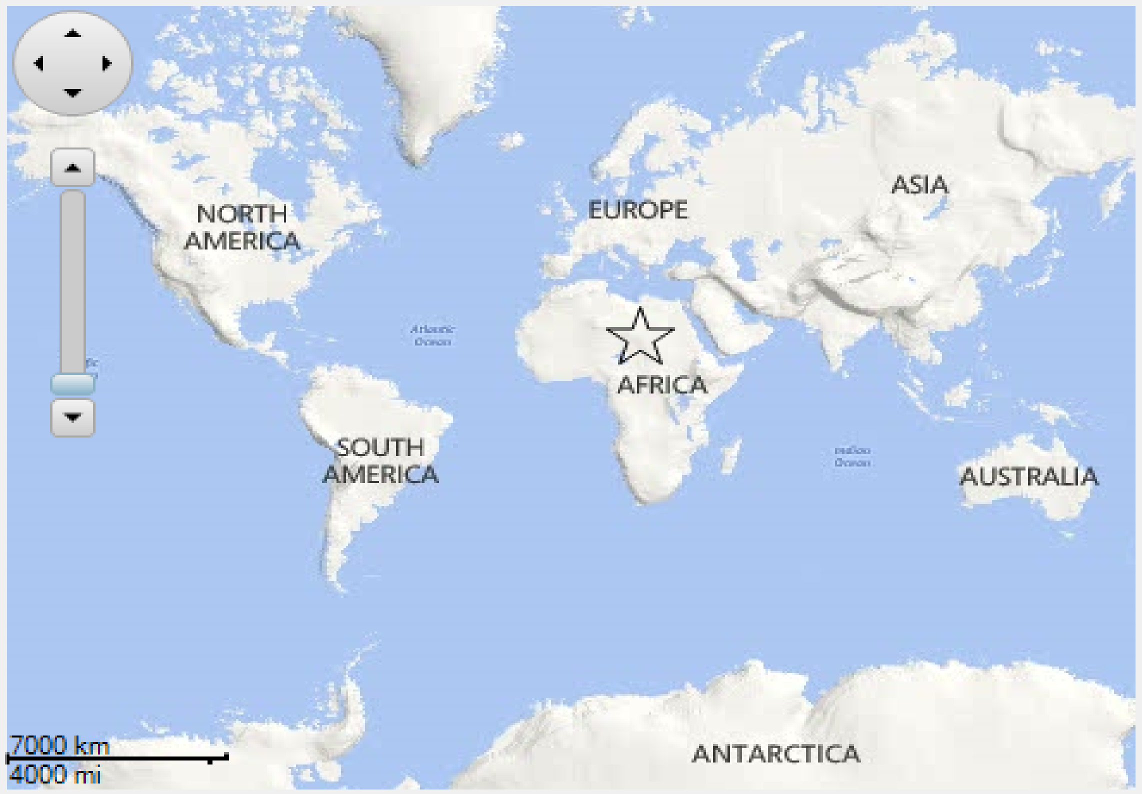
- **Visual Basic**

```
Dim vl = New C1.Win.Map.VectorLayer()
C1Map1.Layers.Add(vl)

Dim placemark = New C1.Win.Map.VectorPlacemark()
placemark.Geometry = New GeoPoint(20, 20)
vl.Items.Add(placemark)
placemark.Marker.Shape = MarkerShape.Star
placemark.Marker.Size = New SizeF(30, 30)
```

- **C#**

```
var vl = new C1.Win.Map.VectorLayer();
c1Map1.Layers.Add(vl);

var placemark = new C1.Win.Map.VectorPlacemark();
placemark.Geometry = new GeoPoint(20, 20);
vl.Items.Add(placemark);
placemark.Marker.Shape = MarkerShape.Star;
placemark.Marker.Size = new SizeF(30, 30);
```

The map with a shape marker will look similar to the image given below.



Adding Custom Marker

You can easily add a custom marker on a map surface using Map control. For doing so, you need to create a class that implements C1.Win.Map.CustomShape class for creating a custom shape. The following code can be used to add a custom marker on a map:

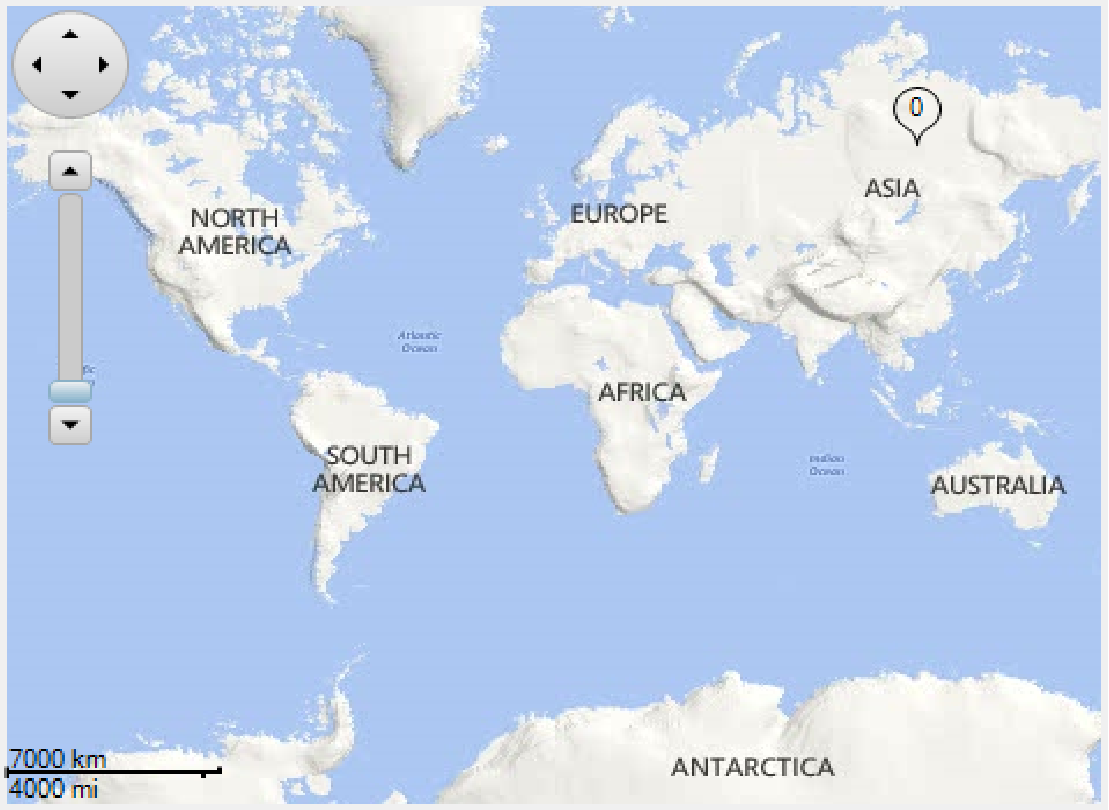
- **Visual Basic**

```
Dim vl = New C1.Win.Map.VectorLayer()
C1Map1.Layers.Add(vl)

Dim placemark = New C1.Win.Map.VectorPlacemark()
placemark.Geometry = New GeoPoint(110, 65)
vl.Items.Add(placemark)

' needs set Shape to MarkerShape first
placemark.Marker.Shape = MarkerShape.[Custom]
placemark.Marker.Size = New SizeF(20, 27.32F)

' then set the CustomShape.
Dim shape = New SampleCustomShape(0)
placemark.Marker.CustomShape = shape
```

- **C#**

```
var vl = new C1.Win.Map.VectorLayer();
c1Map1.Layers.Add(vl);

var placemark = new C1.Win.Map.VectorPlacemark();
placemark.Geometry = new GeoPoint(110, 65);
vl.Items.Add(placemark);

// needs set Shape to MarkerShape first
placemark.Marker.Shape = MarkerShape.Custom;
placemark.Marker.Size = new SizeF(20, 27.32f);

// then set the CustomShape.
var shape = new SampleCustomShape(0);
placemark.Marker.CustomShape = shape;
```

The map with a custom marker will look similar to the image given below.



Adding Image as Marker

If you want to show images as markers on a map, you need to use C1.Win.Map.MarkerImageShape class that represents a custom marker shape. The following code can be used to add an image as a marker on a map:

- **Visual Basic**

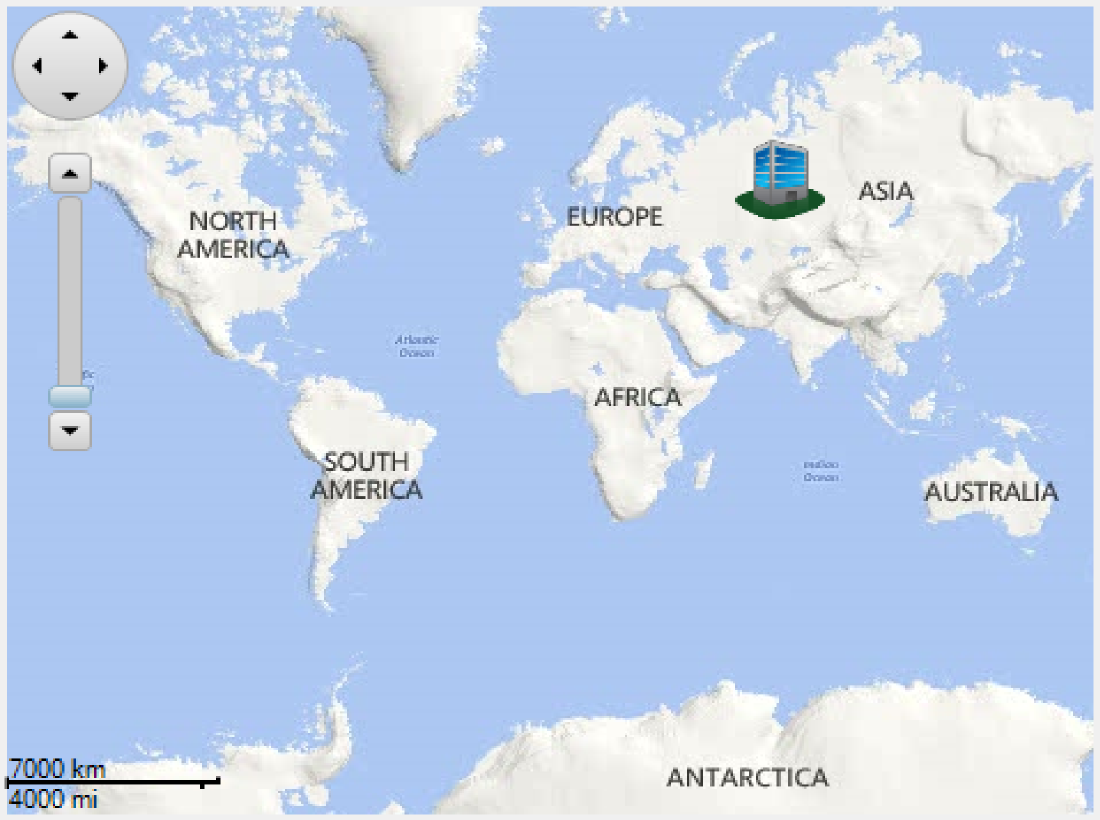
```
Dim vl = New C1.Win.Map.VectorLayer()
C1Map1.Layers.Add(vl)

Dim placemark = New C1.Win.Map.VectorPlacemark()
placemark.Geometry = New GeoPoint(70, 60)
vl.Items.Add(placemark)

' needs set the Shape to MarkerShape.Custom first.
placemark.Marker.Shape = MarkerShape.[Custom]
placemark.Marker.Size = New SizeF(40, 40)

' then set the CustomShape
Dim shape = New MarkerImageShape()
placemark.Marker.CustomShape = shape

shape.Image = Image.FromFile("../../office.png")
```

- **C#**

```
var vl = new C1.Win.Map.VectorLayer();
c1Map1.Layers.Add(vl);

var placemark = new C1.Win.Map.VectorPlacemark();
placemark.Geometry = new GeoPoint(70, 60);
vl.Items.Add(placemark);
```

```
// needs set the Shape to MarkerShape.Custom first.
placemark.Marker.Shape = MarkerShape.Custom;
placemark.Marker.Size = new SizeF(40, 40);

// then set the CustomShape
var shape = new MarkerImageShape();
placemark.Marker.CustomShape = shape;

shape.Image = Image.FromFile("office.png");
```

The map with an image as a marker will look similar to the image given below.



Repositioning Map Tools

The tools in the Map control are positioned on the left side of a map. Although, you can change their position according to your requirements. Following code illustrates repositioning of the map tools:

- **Visual Basic**

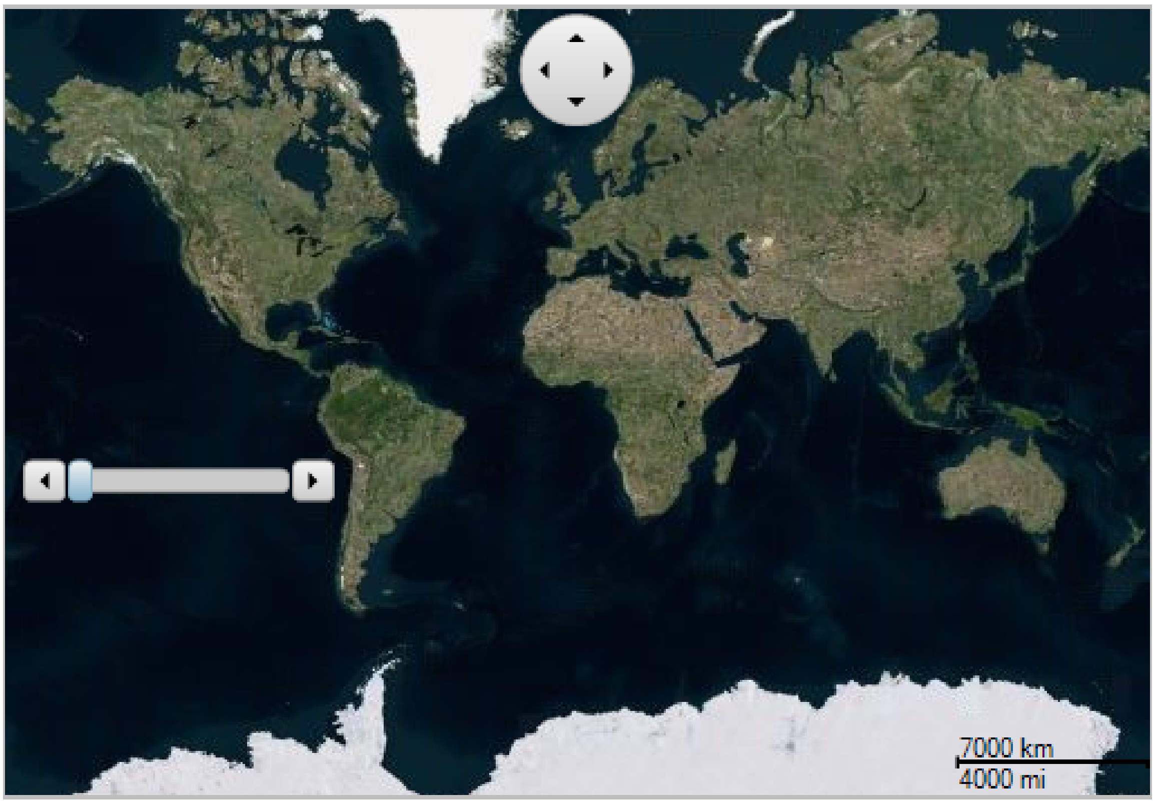
```
C1Map1.TileLayer.TileSource = New VirtualEarthAerialSource()
C1Map1.DistanceScale.Alignment = ContentAlignment.BottomRight
C1Map1.PanTool.Alignment = ContentAlignment.TopCenter
C1Map1.ZoomTool.Alignment = ContentAlignment.MiddleLeft
C1Map1.ZoomTool.Orientation = Orientation.Horizontal
```

- **C#**

```
c1Map1.TileLayer.TileSource = new VirtualEarthAerialSource();
c1Map1.DistanceScale.Alignment = ContentAlignment.BottomRight;
c1Map1.PanTool.Alignment = ContentAlignment.TopCenter;
```

```
c1Map1.ZoomTool.Alignment = ContentAlignment.MiddleLeft;
c1Map1.ZoomTool.Orientation = Orientation.Horizontal;
```

On repositioning the tools on a map, the map will look similar to the image given below.



C1Map also allows you to set the custom location and size of a map tool by setting the Bounds property. Following code demonstrates the use of Bounds property to set the custom location and size of the Pan tool:

- **Visual Basic**

```
C1Map1.PanTool.Bounds = New Rectangle(100, 100, 100, 100)
```

- **C#**

```
c1Map1.PanTool.Bounds = new Rectangle(100, 100, 100, 100);
```

Adding Legends

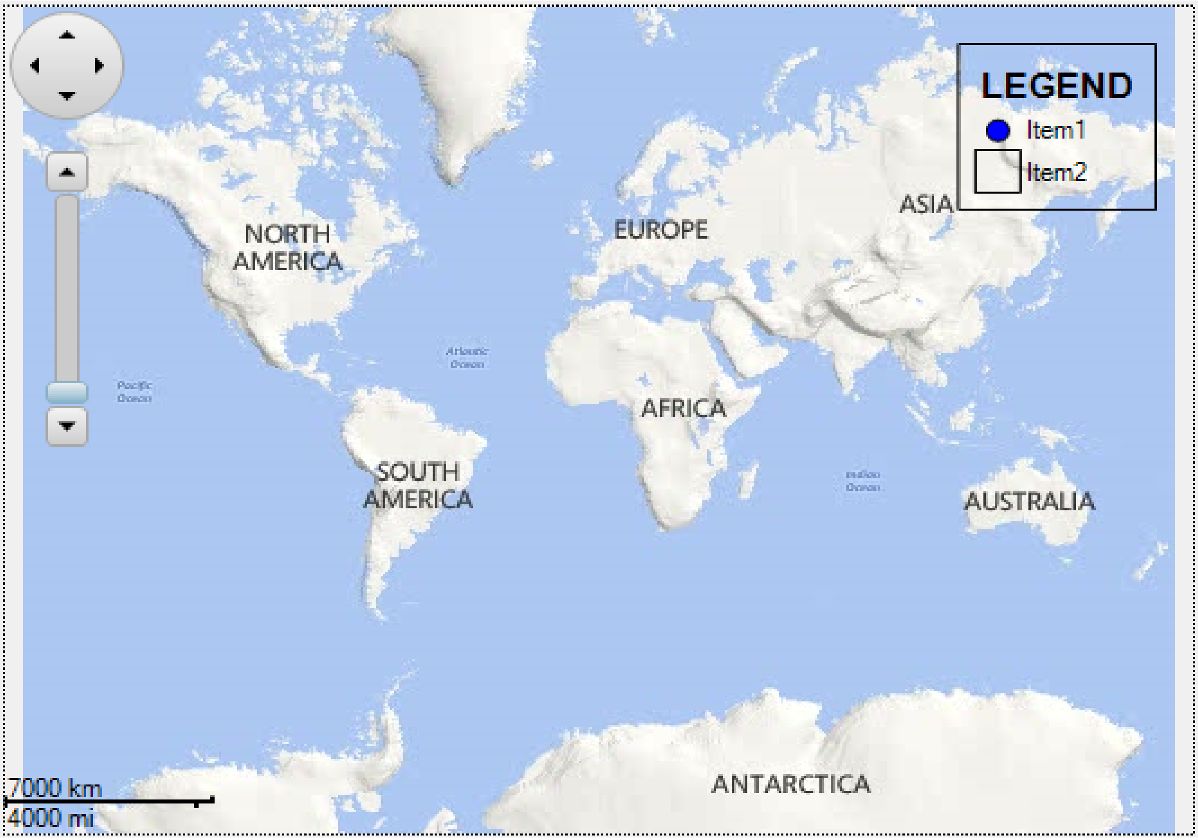
With C1Map, you can add legends to describe the vector elements and markers on a map. Legends can be displayed on a map using MapLegend class. On adding a Legend, you are required to add the legend items on it which can be displayed using MapLegendItem class. The following code can be used to add and customize the legend and legend items on a map:

- **Visual Basic**

```
C1Map1.TileLayer.TileSource = New VirtualEarthRoadSource()

Dim legend = New C1.Win.Map.MapLegend()
C1Map1.Legends.Add(legend)
legend.Alignment = ContentAlignment.TopRight
legend.Margin = New Padding(0, 20, 20, 0)
legend.Padding = New Padding(5)
legend.Layout = MapLegendLayout.Column
legend.Style.Border.Width = 1

legend.Title.Caption = "LEGEND"
legend.Title.Position = DockStyle.Top
legend.Title.Padding = New Padding(5)
legend.Title.Style.Alignment = ContentAlignment.MiddleCenter
legend.Title.Style.Font = New Font("Arial", 12, FontStyle.Bold)

Dim item1 = New C1.Win.Map.MapLegendItem()
legend.Items.Add(item1)
item1.Label = "Item1"
item1.Kind = MapLegendItemKind.Marker
item1.Shape = MarkerShape.Circle
item1.Style.BackColor = Color.Blue

Dim item2 = New C1.Win.Map.MapLegendItem()
legend.Items.Add(item2)
item2.Kind = MapLegendItemKind.Rectangle
item2.Label = "Item2"
item2.Size = New SizeF(20, 20)
item2.Style.Alignment = ContentAlignment.MiddleLeft
```

- **C#**

```
c1Map1.TileLayer.TileSource = new VirtualEarthRoadSource();

var legend = new C1.Win.Map.MapLegend();
c1Map1.Legends.Add(legend);
legend.Alignment = ContentAlignment.TopRight;
legend.Margin = new Padding(0, 20, 20, 0);
legend.Padding = new Padding(5);
legend.Layout = MapLegendLayout.Column;
legend.Style.Border.Width = 1;

legend.Title.Caption = "LEGEND";
legend.Title.Position = DockStyle.Top;
legend.Title.Padding = new Padding(5);
legend.Title.Style.Alignment = ContentAlignment.MiddleCenter;
legend.Title.Style.Font = new Font("Arial", 12, FontStyle.Bold);

var item1 = new C1.Win.Map.MapLegendItem();
legend.Items.Add(item1);
item1.Label = "Item1";
item1.Kind = MapLegendItemKind.Marker;
item1.Shape = MarkerShape.Circle;
item1.Style.BackColor = Color.Blue;

var item2 = new C1.Win.Map.MapLegendItem();
legend.Items.Add(item2);
item2.Kind = MapLegendItemKind.Rectangle;
item2.Label = "Item2";
item2.Size = new SizeF(20, 20);
item2.Style.Alignment = ContentAlignment.MiddleLeft;
```


LEGEND
● Item1
☐ Item2
NORTH AMERICA
SOUTH AMERICA
EUROPE
AFRICA
ASIA
AUSTRALIA
ANTARCTICA
Atlantic Ocean
Pacific Ocean
Indian Ocean
7000 km
4000 mi