

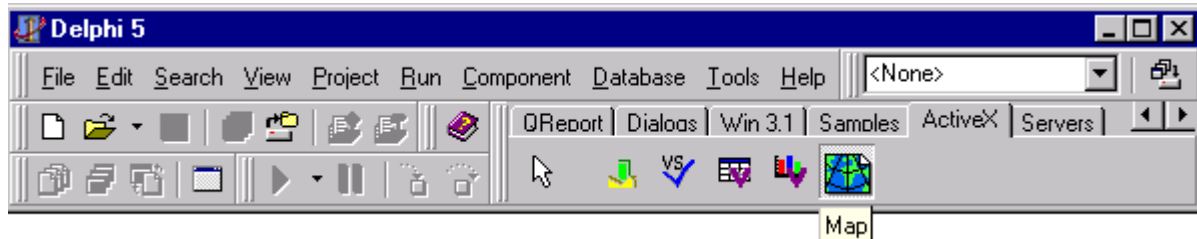
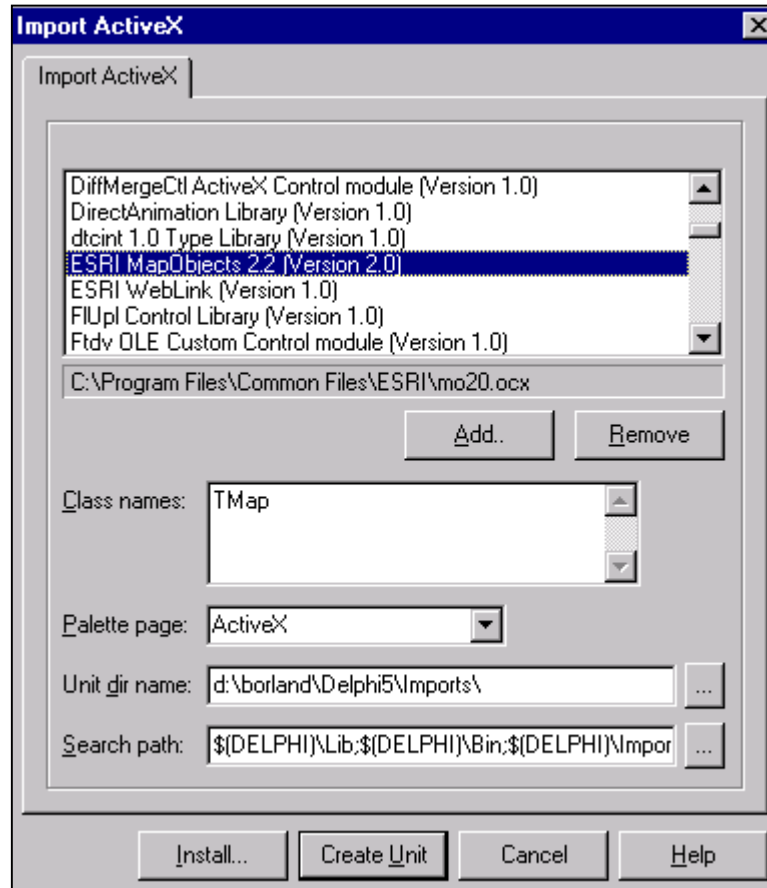
# Getting Started with MapObjects Version 2.2 in Delphi 5.0

In this introductory document you'll use MapObjects version 2.2 and Borland<sup>®</sup> Delphi 5.0 to build an application that uses maps. Along the way you will learn how to . .

- Display a map with multiple layers
- Control panning and zooming
- Create a toolbar control
- Display map layers based on scale
- Perform spatial and logical queries
- Draw simple graphics on the map
- Display features with thematic renderers
- Dynamically display real-time data with an event tracking layer
- Programmatically add data to a map

## Loading MapObjects

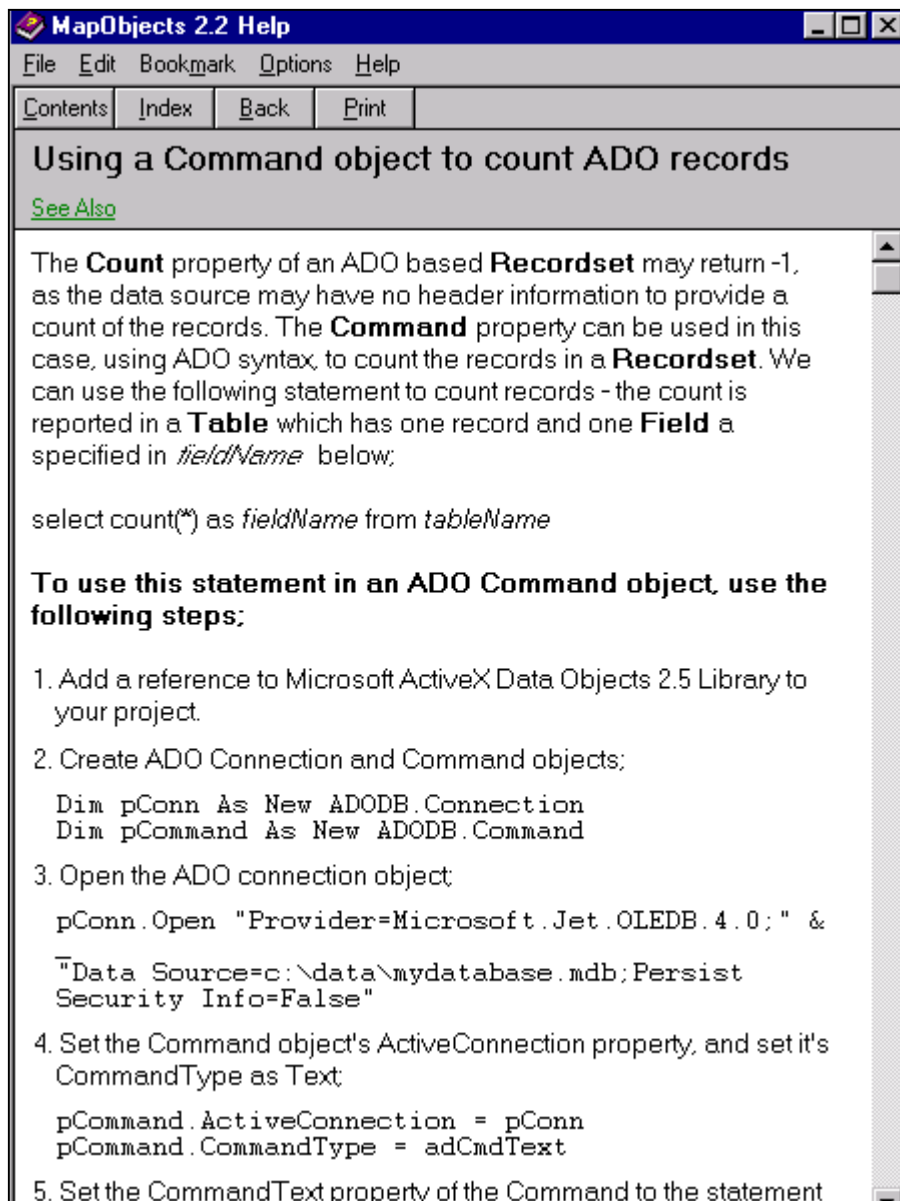
Start Delphi and choose **Import ActiveX Control** from the **Component** menu. Find the **ESRI MapObjects 2.2** reference in the list of available controls. If MapObjects 2.2 is not listed, click **Add** to include the MapObjects 2.2 reference in the list. Finally, click **Install** to rebuild the component library.



Notice that a new tool appears in the list of ActiveX icons on the toolbar of the component palette. This new tool is the MapObjects 2.2 map control.

## Getting MapObjects Help

The map control is one of the 48 objects that make up MapObjects 2.2. To find out about the various objects, open the MapObjects online help and select **Objects** in the table of contents.

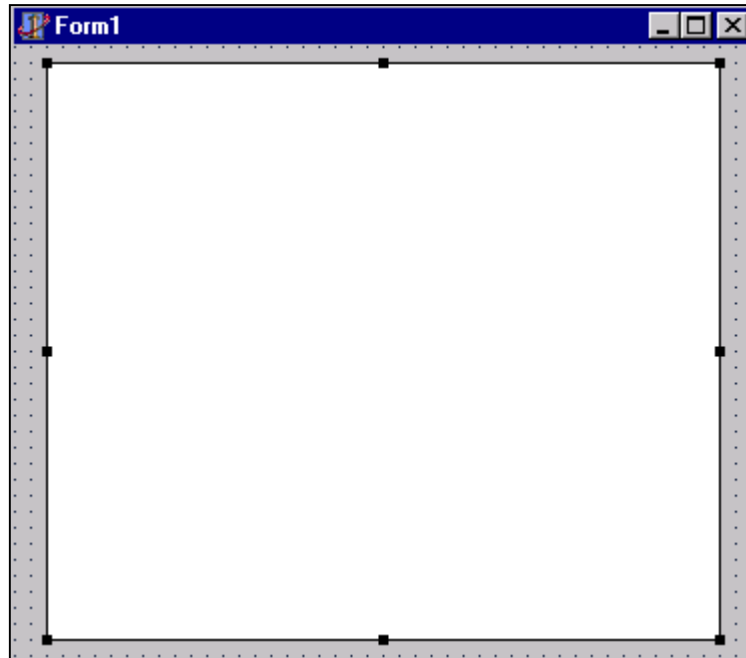


The help system provides help for every object, property, method, event, and constant in MapObjects 2.2.

## Adding a Map

### Add the Map Control to the Form

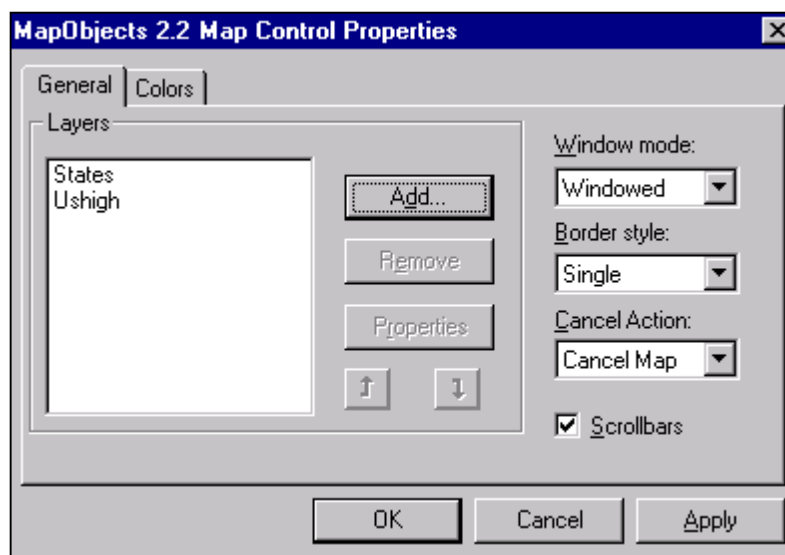
1. Double-click the map control in the ActiveX palette to add a new map to the form.
2. Resize the map to fill the form.



### Add Data to the Map

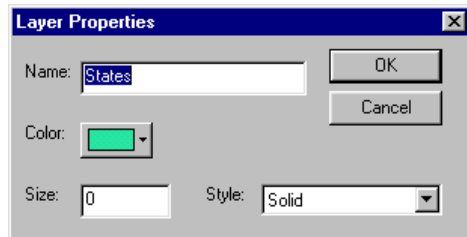
You can specify the data that is displayed in the map by setting properties in the map control's property sheet.

1. Right-click the mouse on the map to display the context menu.
2. Choose **Properties** to display the property sheet.
3. Click **Add** and locate the folder where the sample data is stored for the USA.
4. Choose the States.shp file and then click **Open**.
5. Add the file USHigh.shp in the same manner.



## Set Properties for the Layers

1. With the **MapObjects Map Control Properties** window still open, choose the States layer in the Layers list and then click **Properties**.



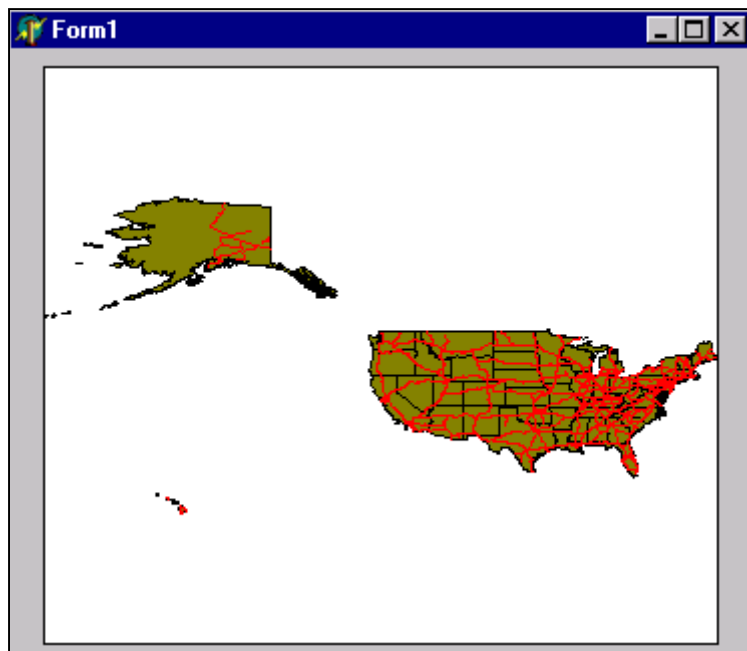
2. Click the Color button to select a color for the States layer.
3. Click **OK** to close the dialog.
4. Select a color for the USHigh layer in the same manner.
5. Click **OK** to close the property window.

## Save the Project

1. Click the **File** menu and then select **Save All**.
2. In the File Name box for the unit, type "StarterMap.pas."
3. Click **Save**.
4. In the second Save As dialog for the project, type "Starter\_Map.dpr" in the File Name box.
5. Click **Save**.

## Test your application

1. Click the Run button in the Delphi toolbar.
2. To stop running your application and return to design mode, close the form.



## Adding Pan and Zoom Controls

At this point, your application can display the map at its full extent. In this section you will add some simple pan and zoom controls that will be activated in response to mouse clicks inside the map. You will write some code that will be executed in response to the `MouseDown` event on the map.

### Respond to the MouseDown Event

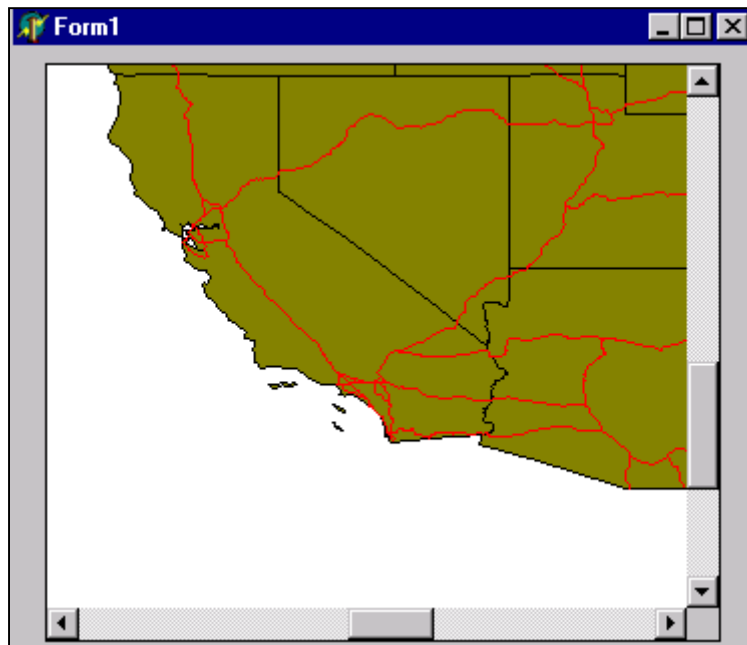
1. Select the map control then select the Events page in the Delphi Object Inspector. Double-click the **OnMouseDown** event to add an event handler.
2. Add code to the `MouseDown` procedure for `Map1`:

```
procedure TForm1.Map1MouseDown(Sender: TObject; Button: TMouseButton;
  Shift: TShiftState; X, Y: Integer);
begin
    Map1.Extent := Map1.TrackRectangle;
end;
```

*TrackRectangle* is a method that applies to a map. It tracks the movement of the mouse while the user presses the mouse button, rubber-banding a rectangle at the same time. When the user releases the mouse button, the `TrackRectangle` method returns a `Rectangle` object which the application assigns into the `Extent` property of the map, causing the map to be redrawn with a new map extent.

### Test your change

1. Click the Run button in the Delphi toolbar.
2. Click and drag the left mouse button on the map to rubber-band a rectangle
3. Release the mouse button and notice that the map is redrawn at the location you specified.



4. Close the form to return to design mode.

## Add Panning

1. Double-click the OnMouseDown event in the Object Inspector to display the code window again.
2. Change the code for the MouseDown procedure for Map1.

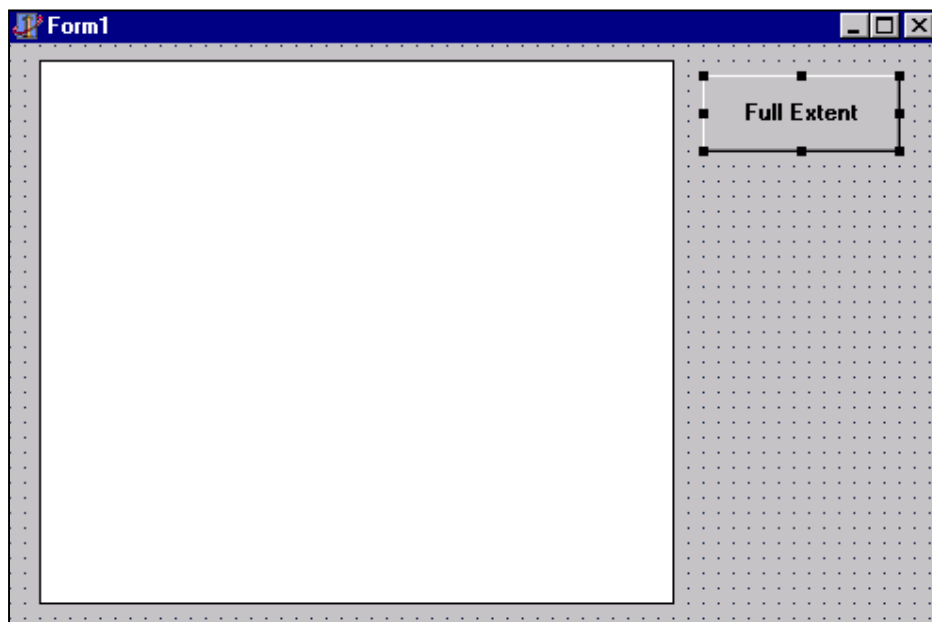
```
procedure TForm1.Map1MouseDown(Sender: TObject; Button: TMouseButton;
  Shift: TShiftState; X, Y: Integer);
begin
  if (Button = mbLeft) then
    Map1.Extent := Map1.TrackRectangle
  else
    Map1.Pan;
end;
```

Use the left mouse button to zoom in to the map; use the right mouse button to invoke the **Pan** method.

## Add a Full Extent Button

Your application now supports panning and zooming, but once the user has zoomed into the map, there is no way to get back to the full extent again. In this section you will add a button to the form which resets the map's extent to its full extent.

1. Click the Button tool on the Standard controls page in the control palette, then click the form to add a button.
2. Move the button to the upper right of the form.
3. With the button selected, click the **Properties** page in the Object Inspector window.
4. Click in the **Caption** box and type "Full Extent"..
5. Resize the map control so that it is not obscured by the button.



6. Double-click **Full Extent** to display the code window.
7. Add code for the Click event of the button.

```
procedure TForm1.Button1Click(Sender: TObject);
begin
    Map1.Extent := Map1.FullExtent;
end;
```

The *FullExtent* property of the map returns a Rectangle which defines the bounding box of all the layers of the map.

### Test Your Change

1. Click the Run button in the Delphi toolbar.
2. Click and drag the left mouse button on the map to rubber-band a rectangle.
3. Release the mouse button to reset the map extent.
4. Click the map with the right mouse button and drag to pan the map.
5. Release the mouse button to redraw the map.
6. Click **Full Extent** to redraw the map at the full extent.

### Save the Project

1. Close the form to return to design mode.
2. Click the Save All button in the Delphi toolbar to save your changes.

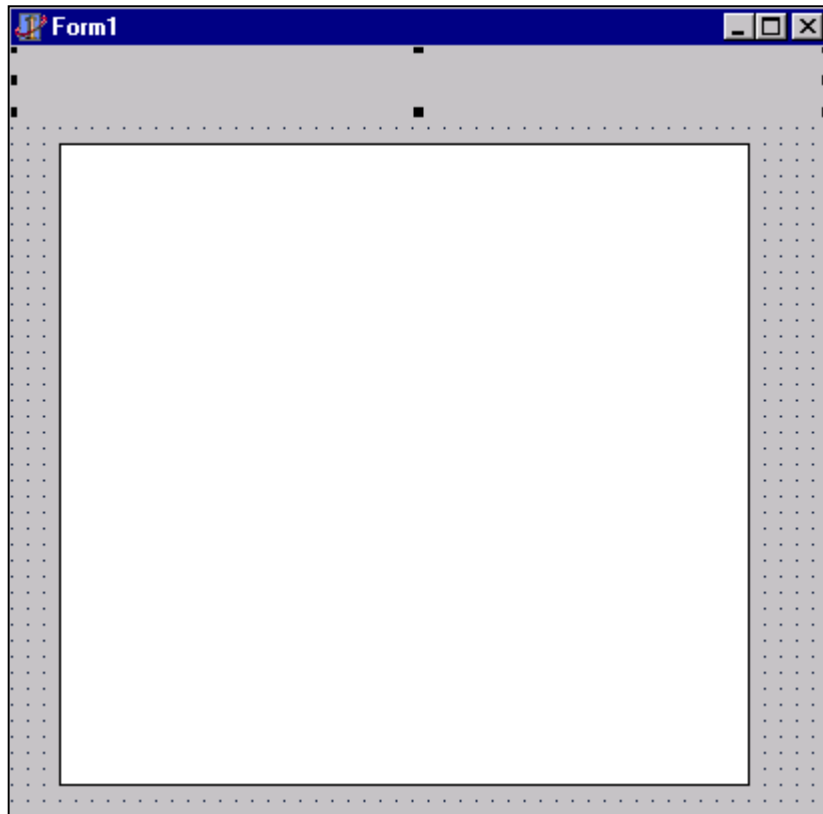
## Adding a Toolbar

Your application's pan and zoom capabilities are somewhat hidden from the user. In this section you will create a toolbar with pan and zoom buttons.

### Adding a Panel

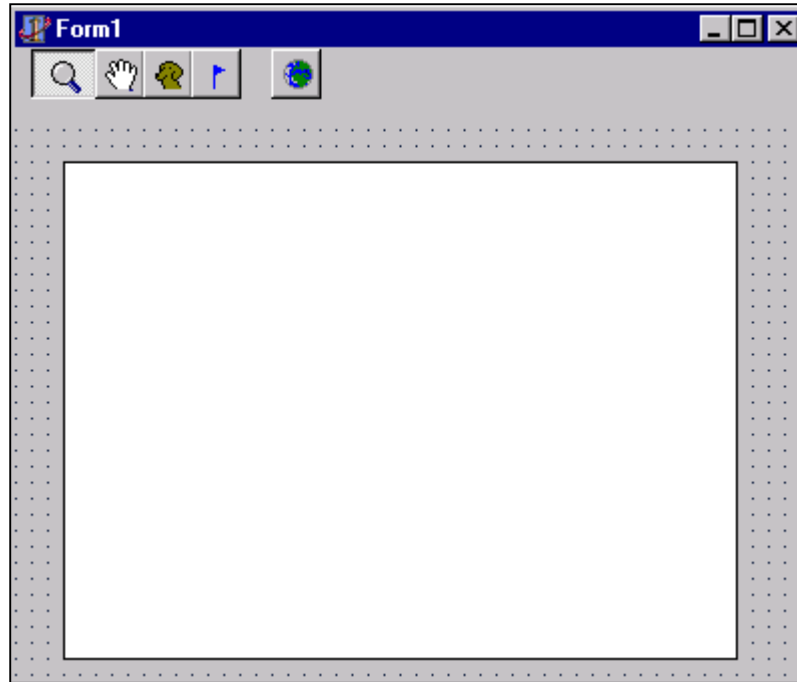
Panels are a convenient way to group controls on a form and will be used to implement the toolbar.

1. Delete the **Full Extent** button from the form.
2. Select the Panel control from the Standard controls page and place a panel at the top of the form.
3. Set the **Caption** property of the panel to blank and set the **Align** property to alTop. Set the **BevelOuter** property to bvNone. Set the **Name** of the panel to "MapToolbar."
4. Resize the map so that it is not obscured by the panel.



### Adding Buttons to the Panel Control

1. Select the MapToolbar panel.
2. Click the SpeedButton control on the **Additional** palette page.
3. Click the MapToolbar panel to place a button on the panel.
4. Set the **Name** property for the new button to “ZoomBtn.”
5. Double-click the **Glyph** property in the Object Inspector to open the property editor for glyphs.
6. Use the **Load** button to select Zoom.bmp as the picture for the zoom button.
7. Add a **PanBtn** (Pan.bmp), **QueryBtn** (Bex.bmp), **AddEventBtn** (Pennant.bmp) and **FullExtentBtn**(Globe.bmp) buttons in the same manner.
8. Since the first four buttons will work together, with one selected at a time, select the first four buttons and set their **GroupIndex** property to “1.”
8. Set the **Down** property for the ZoomBtn to “True” to make it the selected tool by default.



## Change the MouseDown Event

1. Select the map control on the form and double-click the OnMouseDown event in the Property Inspector to display the code window.
2. Modify the code attached to Map1's OnMouseDown procedure.

```

procedure TForm1.Map1MouseDown(Sender: TObject; Button: TMouseButton;
  Shift: TShiftState; X, Y: Integer);
begin
  if (ZoomBtn.Down) then
    Map1.Extent := Map1.TrackRectangle;
  if (PanBtn.Down) then
    Map1.Pan;
end;

```

Selecting the first button in the toolbar sets the mouse to be a zoom tool; otherwise, if you click the map, you can use the mouse to pan.

## Implement the FullExtent Button

In this section you will re-implement the Full Extent button that you deleted.

1. Select the FullExtentBtn on the form and double-click the **OnClick** event in the Property Inspector Events page to create an event handler and open it in the code window.
2. Add code to the FullExtentBtnClick event.

```

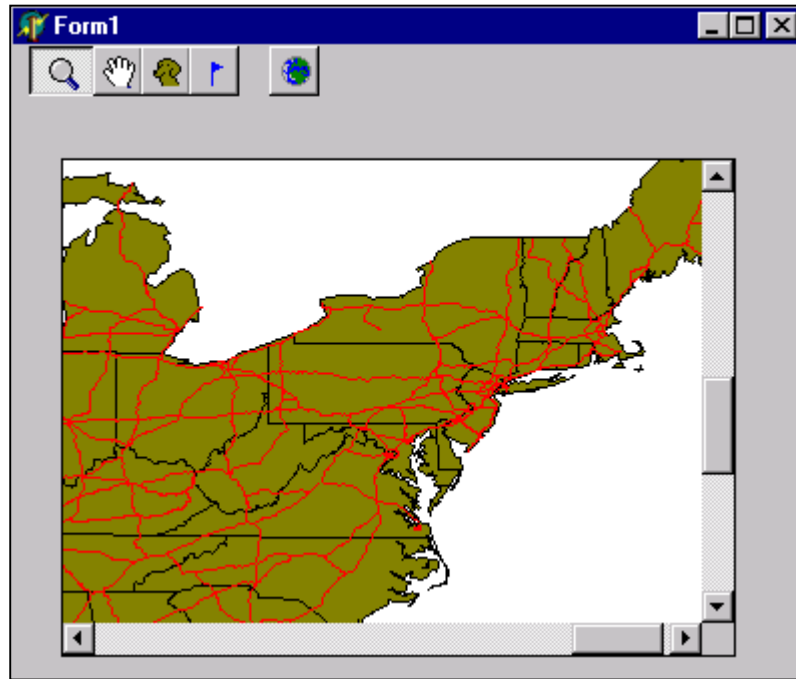
procedure TForm1.FullExtentBtnClick(Sender: TObject);
begin
  Map1.Extent := Map1.FullExtent;
end;

```

The FullExtentBtnClick event is generated whenever the FullExtentBtn is clicked.

## Test Your Changes

1. Click the **Run** button in the Delphi toolbar.
2. Click somewhere on the map to zoom into a rectangle.
3. Click the pan button in your application's toolbar.
4. Click somewhere on the map and drag to pan.



5. Click on the full extent button (the globe) in your application's toolbar to draw the map at its full extent.

## Save your changes

1. Close the form to return to design mode.
2. Click the Save All button in the Delphi toolbar to save your changes.

## Creating a Find Tool

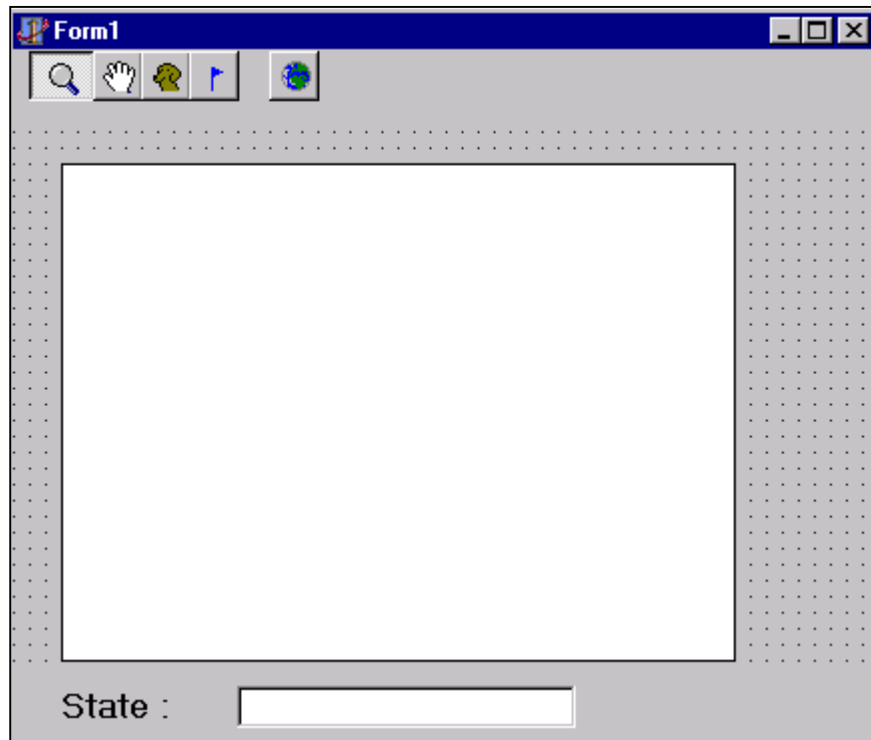
In this section you will add additional controls to your application to implement a simple function for locating a state by name.

### Add Controls to the Form

Like the toolbar panel at the top of the form, adding another panel at the bottom of the form will make it easier to handle form resizing and automatic positioning of the controls.

1. Click the Panel control on the Standard palette then click the bottom of the form to add a panel.
2. Set the **Caption** property of the panel to blank, the **Align** property to alBottom and the **BevelOuter** property to bvNone.

3. With the panel selected, click the Label control on the Standard page of the component palette and click the panel to add a text label control.
4. Set the **Caption** property of the text label control to “State:.”
5. With the panel selected, click the Edit control on the Standard page of the component palette and click the panel to add an edit control.
6. Set the **Text** property of the edit control to blank.
7. Select the label and edit controls.
8. Select the **Alignment Palette** item from the **View** menu to display the Alignment Palette.
9. Click the *align vertical centers* button, then *center vertically in window* button to position the new controls.



### Edit the uses section of the code

Add a **ComObj** entry to the Uses clause for the form (at the top of the .pas file).

```
uses
  Windows, Messages, SysUtils, Classes, Graphics, Controls, Forms,
  Dialogs, OleCtrls, MapObjects_TLB, StdCtrls, ExtCtrls, Buttons,
  ComObj;
```

### Attach Code to the TextBox

1. Select the edit control, **Edit1**.
2. In the Object Inspector, select the **Events** page and double-click the OnKeyPress event to create an event handler and open the code window.
4. Add code to the KeyPress procedure.

```
Procedure TForm1. Edit1KeyPress(sender: TObject; var Key: Char);
```

```

var
  l ys      : I MoLayers;
  l ayer    : I MoMapLayer;
  recs     : I MoRecordset;
  shp      : I MoPol ygon;
  rect     : I MoRectangl e;
  fi el ds : I MoFi el ds;
  exp      : string;
begin
  // check for the enter key
  if (Key = #13) then
    begin
      // build a search expression
      exp := ' STATE_NAME ='' + Edit1.Text + ''';
      l ys := Map1.Layers;
      l ayer := I MoMapLayer(CreateO l eObj ect(' MapObj ects2. MapLayer' ));
      l ayer := I MoMapLayer(l ys. I tem(' STATES' ));
      // perform the search on the STATES layer
      recs := l ayer. SearchExpressi on(exp);

      // show the results, if any
      if (not recs.EOF) then
        begin
          fi el ds := recs. Fi el ds;
          shp := I MoPol ygon(CreateO l eObj ect(' MapObj ects2. Pol ygon' ));
          // get the shape geometry
          shp := I MoPol ygon(I Di spatch(fi el ds. I tem(' Shape' ). Val ue));
          // create a rectangle based on the extent of the state
          rect:= I MoRectangl e(CreateO l eObj ect(' MapObj ects2. Rectangl e' ));
          rect:= shp. Extent;
          rect. Scal eRectangl e((2.0));
          // zoom to the state
          Map1.Extent := rect;
          Map1.Refresh;
          // flash the state
          Map1. Fl ashShape(shp, 3);
        end;

        // suppress a beep
        Key := #0;
      end;
    end;
end;

```

The code first builds a simple SQL query expression using the value of the Edit box's Text property. Then the States layer is searched using the SearchExpression method with the result being a RecordSet object. If the value of the RecordSet's EOF property is False, the RecordSet is positioned on the first record that satisfies the search expression. In that case, the value of the Shape field is obtained for the first record. The shape is flashed three times, then the Extent of the shape is scaled and set to be the Extent of the map.

## Test Your Changes

1. Run your application.
2. Type the name of a state, for example, "Vermont," into the Edit box. Note that the first character must be in upper case.
3. Press the Enter key on your keyboard.
4. When you are finished running your application, close the form.

## Handling Resize

When you run your application and resize the form, you'll notice that the map is not automatically resized.

### Put the map control on a panel

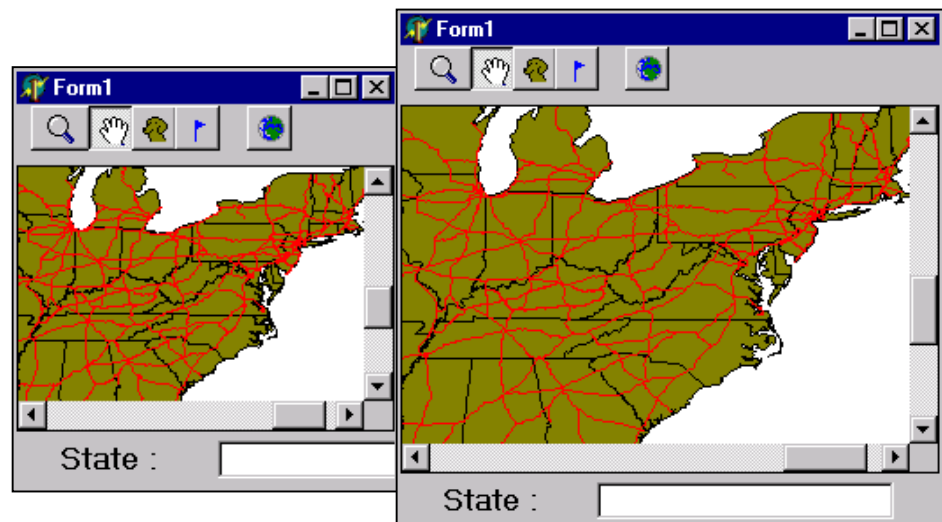
One way of resizing the map is to put the map control on a panel. Then, when the panel automatically resizes itself the map can be resized to fit in the panel.

1. Save your project.
2. Select the map control.
3. Choose **Cut** from the Edit menu.
4. Add a Panel to the form by clicking the Panel control on the control palette and then clicking the form.
5. Set the **Caption** property of the panel to blank and set the **Align** property to alClient.
6. Set the **BevelOuter** property of the panel to bvNone.
7. With the panel selected, choose **Paste** from the Edit menu to place the map control on the panel.
8. Right-click the map and choose **Properties** from the menu.
9. Ensure that the layers are still there. If they are not, delete the MapControl, add a new MapControl, and add the layers. Ensure you reassign the MouseDown event on the Map control.
10. Set the **Top** and **Left** properties of the map control to "0" (zero).

### Respond to the Resize Event

1. Select the panel added above by clicking it or by choosing it from the drop-down list in the Object Inspector.
2. Double click the **OnResize** event in the Object Inspector to add an event handler and open the code window.
3. Add code to the panel's Resize procedure.
 

```
procedure TForm1.Panel2Resize(Sender: TObject);
begin
    Map1.Width := Panel2.Width;
```



```
Map1.Height := Panel2.Height;
```

end;

When the form is resized, the controls are resized.

## Displaying Map Layers Based on Scale

In this section you will add a new layer to your map and add code which controls whether or not that layer is visible at a given time.

### Add another layer

1. Right-click the map to display the context menu. Click **Properties** to show the property sheet.
2. Click **Add** and locate the folder where the sample data is stored.
3. Navigate to the Counties.shp file and then click **Open**.
4. Click the Counties layer in the list to select it.
5. Click the down arrow to move the Counties layer below the Ushigh layer.
6. Click **Properties** to change the color of the Counties layer.
7. Click **OK** to dismiss the Layer Properties dialog.
8. Click **OK** to dismiss the property sheet.

If you run your application now you'll notice that every county in the US is displayed. At the full extent, there is no need to display that much detail, so in response to the `BeforeLayerDraw` event, you will selectively make the Counties and States layers visible or invisible depending on the current extent of the map.

### Respond to the BeforeLayerDraw event

1. Click the map control and double-click the **OnBeforeLayerDraw** event in the Object Inspector to create an event handler and display it in the code window.
2. Add code to the `Map1 BeforeLayerDraw` procedure:

```
procedure TForm1.Map1BeforeLayerDraw(Sender: TObject; Index: SmallInt;
  hDC: Cardinal);
var
  lys : IMoLayers;
  ly1, ly2 : IMoMapLayer;
  ext, fullExt : IMoRectangle;

begin
  lys := Map1.Layers;
  ly1 := IMoMapLayer(CreateOLEObject('MapObjects2.MapLayer'));
  ly2 := IMoMapLayer(CreateOLEObject('MapObjects2.MapLayer'));
  ly1 := IMoMapLayer(lys.Item(1));
  ly2 := IMoMapLayer(lys.Item(2));

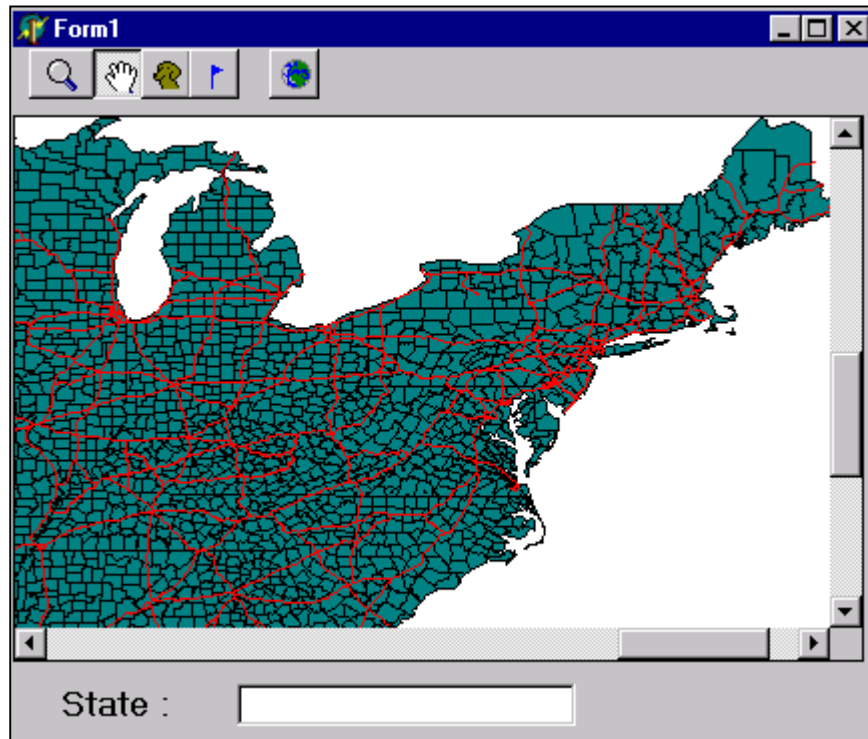
  ext := Map1.Extent;
  fullExt := Map1.FullExtent;

  ly1.Visible := ext.Width < (fullExt.Width / 4); // counties
  ly2.Visible := ext.Width >= (fullExt.Width / 4); // states
end;
```

The value of the visible property of each layer is based on the current extent of the map. If the width of the current extent is less than or equal to one fourth of the full extent of the map, then the counties will be visible and the states will be invisible. Because this code is executed in response to the BeforeLayerDraw event for each layer, the value of the visible field is calculated before drawing occurs.

### Test your changes

1. Run your application. Notice that the Counties layer is not drawn.
2. Zoom into New England and the Counties layer becomes visible.



3. Click the FullExtent button and the Counties are no longer visible.
4. Close the form.
5. Click the Save All button to save your changes.

## Adding a Spatial Query Tool

In this section you'll add a new tool to the toolbar that will perform spatial queries on the map. You'll add code to your application that will draw the results of the spatial query on the map.

### Make the Query button visible

1. Select the QueryBtn.
2. Ensure the **Visible** property of the QueryBtn is set to True and its **GroupIndex** property is set to "1".

### Add variables to the form

1. Open the Code window for the form.
2. In the private section of the form's declaration, add a variable that will be the results of the spatial query. The type of the variable is IMoRecordset.

```

type
  TForm1 = class(TForm)
  ...

private
  { Private declarations }
  gSelection : IMoRecordset;

```

### Implement the query tool

Modify the MouseDown procedure for Map1 with the following code in **bold**.

```

procedure TForm1.Map1MouseDown(Sender: TObject; Button: TMouseButton;
  Shift: TShiftState; X, Y: Integer);
var
  pt      : IMoPoint;
  recs    : IMoRecordset;
  lys     : IMoLayers;
  ly1, ly2 : IMoMapLayer;
  fields  : IMoFields;
  field   : IMoField;

const
  moEdgeTouchOrArealntersect = 6;
begin
  if (ZoomBtn.Down) then // zoom in
    Map1.Extent := Map1.TrackRectangle;
  if (PanBtn.Down) then // pan
    Map1.Pan;

  if (QueryBtn.Down) then // query
    begin
      pt := Map1.ToMapPoint(x, y);
      lys := Map1.Layers;
      ly1 := IMoMapLayer(CreateOLEObject('MapObjects2.MapLayer'));
      ly2 := IMoMapLayer(CreateOLEObject('MapObjects2.MapLayer'));
      ly1 := IMoMapLayer(lys.Item('Ushigh'));
      ly2 := IMoMapLayer(lys.Item('Counties'));

```

```

// search for a highway within a tolerance
recs := ly1.SearchByDistance(pt, Map1.ToMapDistance(10), '');
fields := recs.Fields;
field := fields.Item('Shape');
if (recs.EOF) then // nothing is found
  gSelection := nil // varNull
else // search for counties that intersect the highways
  gSelection := ly2.SearchShape(IDispatch(field.Value),
    moEdgeTouchOrAreaIntersect, '');

// trigger a redraw of the MapControl
Map1.Refresh;
end;
end;

```

When the current tool is the spatial query tool, two searches are performed. The first search is a point proximity search on the Ushigh layer. The point is obtained by converting the x and y coordinates of the event, which are in control units, into map units. If the first search is successful, the highway that is found is used as the input to the second search which is performed on the Counties layer. The result of the second search is stored in the variable, *gSelection*.

## Draw the results

1. Modify Map1's AfterLayerDraw procedure.
2. Add code to display the results of the query on top of the Counties layer.

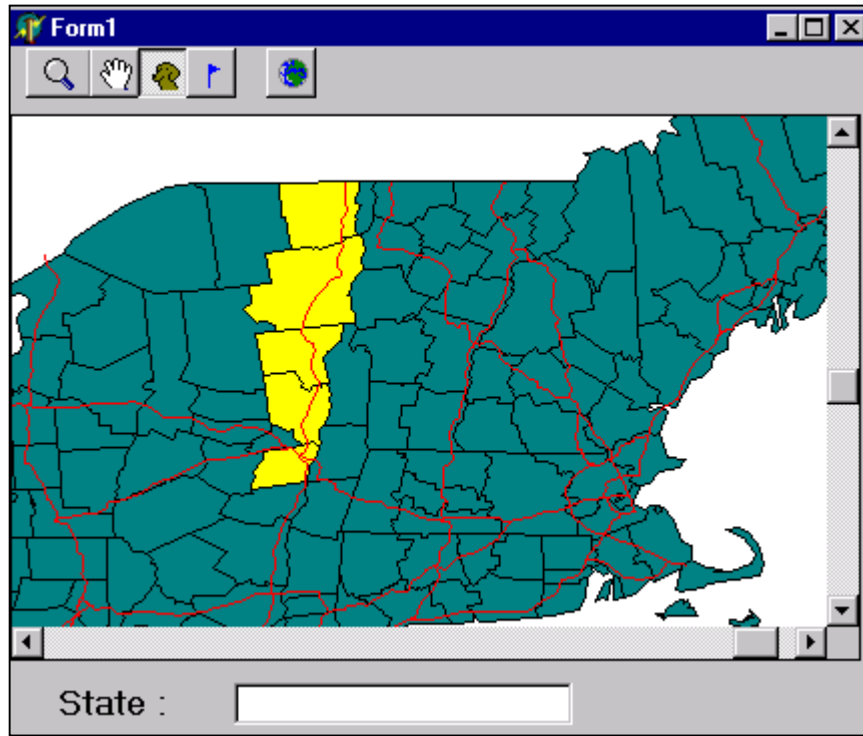
```

procedure TForm1.Map1AfterLayerDraw(Sender: TObject; Index: SmallInt;
  canceled: WordBool; hDC: Cardinal);
var
  sym : IMoSymbol;
  flds : IMoFields;
  fld : IMoField;
  shp : IMoPolygon;
begin
  if (Index = 1) then
    // counties layer
    begin
      if (not VarIsEmpty(gSelection)) then
        begin
          if (not VarIsNull(gSelection)) then
            begin
              sym := IMoSymbol(CreateComObject(Class_Symbol));
              sym.Color := clYellow;
              if gSelection.EOF then
                // Edit if no records are found
                Exit;
              flds := gSelection.Fields;
              fld := flds.Item('Shape');
              gSelection.MoveFirst;
              while (not gSelection.EOF) do
                begin
                  shp := IMoPolygon(IDispatch(fld.Value));
                  Map1.DrawShape(shp, sym);
                  // Release shp (needed in loops)
                  shp := nil;
                  gSelection.MoveNext;
                end;
            end;
          end;
        end;
      end;
    end;
end;

```

## Test your changes

1. Run your application and zoom into an area so that the Counties layer becomes visible.
2. Click the spatial query tool then click on a highway.



3. Close the form.
4. Click the Save Project button to save your changes.

## Statistical mapping

In this section you will modify your application so that the Counties layer is drawn using the underlying attribute information.

### Attach a renderer to the Counties layer

1. Select the form (Form1) in the drop-down list in the Object Inspector and double-click the **OnActivate** event to display the code window.
1. Add code to the OnActivate procedure for the Form.

```

procedure TForm1.FormActivate(Sender: TObject);
var
  breakVal   : Double;
  i          : Integer;
  lys       : IMoLayers;
  ly1,
  ly2       : IMoMapLayer;
  cbr       : IMoClassBreaksRenderer;
  ddr       : IMoDotDensityRenderer;
  recset1, recset2 : IMoRecordset;
  stats1, stats2  : IMoStatistics;
begin
  // Render counties layer
  lys := Map1.Layers;
  ly1 := IMoMapLayer(CreateOLEObject('MapObjects2.MapLayer'));
  ly1 := IMoMapLayer(lys.Item('counties'));

  ly1.Renderer :=
  IMoClassBreaksRenderer(CreateOLEObject('MapObjects2.ClassBreaksRenderer'));
  cbr := IMoClassBreaksRenderer(ly1.Renderer);
  cbr.Field := 'MOBILEHOME';

  recset1 := IMoRecordset(ly1.Records);
  stats1 := IMoStatistics(recset1.CalculateStatistics('MOBILEHOME'));

  // calculate breaks away from the mean in both directions
  // but only add those breaks that are within the range of values
  breakVal := stats1.Mean - (stats1.StdDev * 3);
  for i := 0 to 6 do
    begin
      if (breakVal >= stats1.Min) and (breakVal <= stats1.Max) then
        begin
          cbr.BreakCount := cbr.BreakCount + 1;
          cbr.Break[cbr.BreakCount - 1] := breakVal;
        end;
      breakVal := breakVal + stats1.StdDev;
    end;

  cbr.RampColors(cLime, cRed);
  // you will later insert code here to render the states layer
end;

```

Each MapLayer object has a Renderer property. A Renderer object controls how the MapLayer is drawn. The ClassBreaksRenderer can be used to display continuous data, in this case the number of mobile homes per capita by county.

### Attach a renderer to the States layer

1. Modify the form's OnActivate procedure.
2. Insert the following code after the *RampColors* method for the counties layer and before the reserved word "end;".

```
// Render states layer
ly2 := I MoMapLayer(CreateOLEObject(' MapObjects2. MapLayer' ));
ly2 := I MoMapLayer(lys.Item(' states' ));

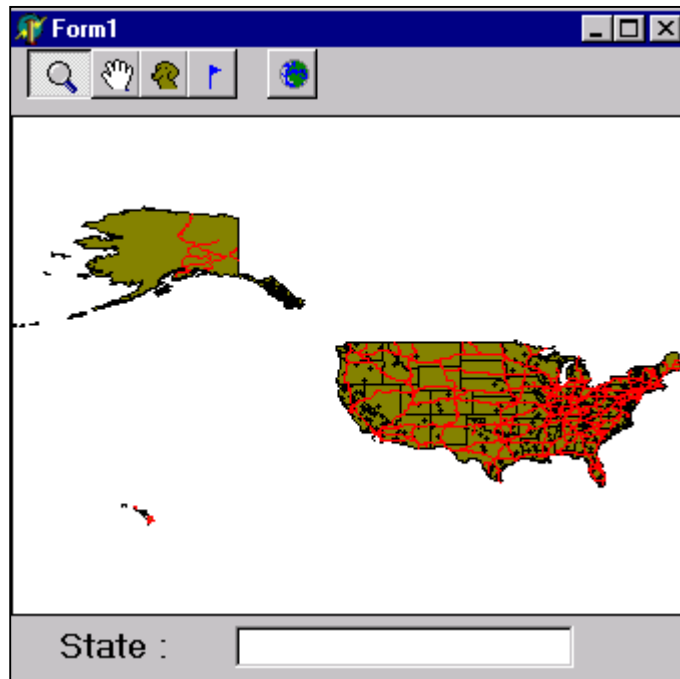
ly2.Renderer :=
IMoDotDensityRenderer(CreateOLEObject(' MapObjects2. DotDensityRenderer' ));
ddr := I MoDotDensityRenderer(ly2.Renderer);
ddr.Field := ' HOUSEHOLDS' ;

recset2 := I MoRecordset(ly2.Records);
stats2 := I MoStatistics(recset2.CalculateStatistics(' HOUSEHOLDS' ));
ddr.DotValue := stats2.Max / 40;

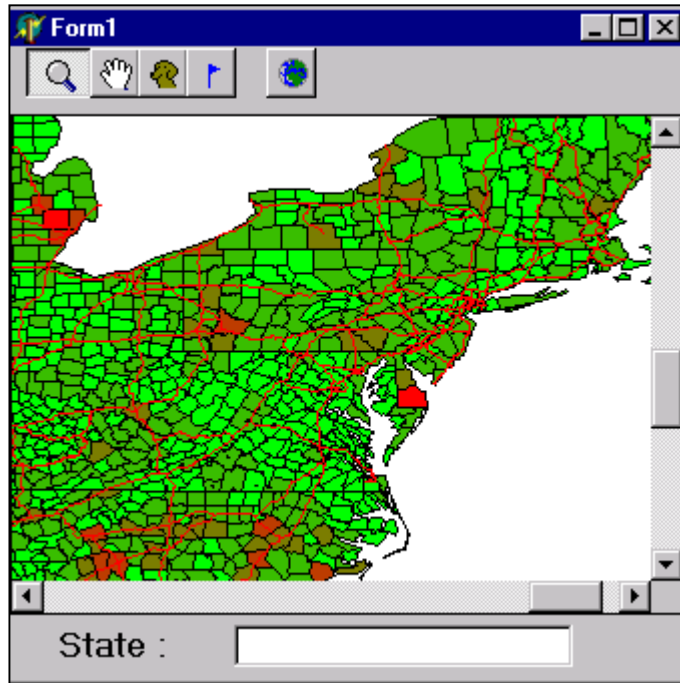
Map1.Refresh;
```

### Test your changes

1. Run your application and look at the States layer. Notice that the polygons are now drawn with dots that indicate the elevation.



- Zoom into an area so that the Counties layer becomes visible. Notice that the counties are now drawn in different colors, depending on the underlying attribute values.



- Close the form.
- Click the Save Project button to save your changes.

## Event Tracking

It is often desirable to display geographic entities on top of the map, especially if those entities have a tendency to move. For example, a vehicle tracking system would want to display vehicles on the map at the appropriate locations and update those locations over time without redrawing all the layers of the map each time a vehicle changes location.

In this section you will add an event tracking layer to your application.

### Add an event tool to your application's toolbar

- Click the AddEventBtn on the toolbar to show its properties in the Object Inspector.
- Ensure the **Visible** property is set to True and the **GroupIndex** property is set to "1".

### Implement the event tool

- With the map control selected, double-click the OnMouseDown event in the Object Inspector to display the code window.
- Change the MouseDown procedure for Map1 with the following code in **bold>:**

```

procedure TForm1.Map1MouseDown(Sender: TObject; Button: TMouseButton;
  Shift: TShiftState; X, Y: Integer);
var
  pt      : IMoPoint ;
  recs    : IMoRecordSet;
  lvs     : IMoLayers ;

```

```

ly1,
ly2  : I MoMapLayer;
fields : I MoFields ;
field  : I MoField ;
tLayer : I MoTrackingLayer ;

const
    moEdgeTouchOrAreaIntersect = 6;

begin
    if (ZoomBtn.Down) then // zoom
        Map1.Extent := Map1.TrackRectangle;
    if (PanBtn.Down) then // pan
        Map1.Pan;

    if (QueryBtn.Down) then // query
        begin
            pt := Map1.ToMapPoint(x, y);
            lys := Map1.Layers;
            ly1 := I MoMapLayer(CreateOLEObject(' MapObjects2. MapLayer' ));
            ly2 := I MoMapLayer(CreateOLEObject(' MapObjects2. MapLayer' ));
            ly1 := I MoMapLayer(lys.Item(' Ushigh' ));
            ly2 := I MoMapLayer(lys.Item(' Counties' ));

            // search for a highway within a tolerance
            recs := ly1.SearchByDistance(pt, Map1.ToMapDistance(10), '');
            fields := recs.Fields;
            field := fields.Item(' Shape' );
            if (recs.EOF) then // nothing is found
                gSelection := nil // varNull
            else // search for counties that intersect the highways
                gSelection := ly2.SearchShape(IDispatch(field.Value),
                    moEdgeTouchOrAreaIntersect, '');

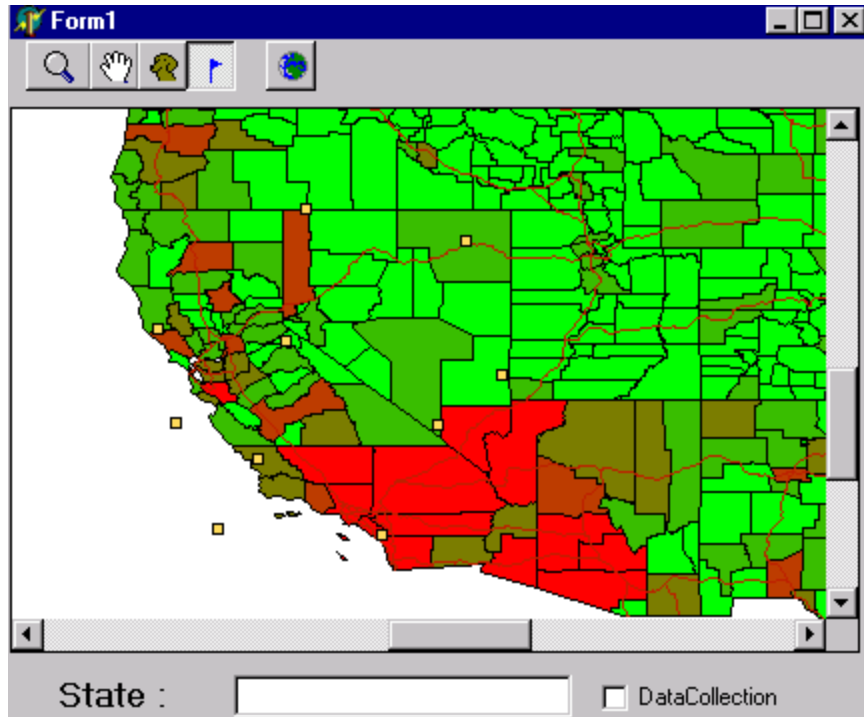
            // trigger a redraw of the MapControl
            Map1.Refresh;
        end;

        if (AddEventBtn.Down) then // add an event
            begin
                pt := Map1.ToMapPoint(x, y);
                tLayer := Map1.TrackingLayer;
                // remember to insert tLayer in var declarations
                tLayer.AddEvent(pt, 0);
            end;
        end;
end;

```

## Test the event tool

1. Run your application and zoom into an area



2. Click the event tool, then click in the map to add events.
3. Close the form.

### Add a timer to your form

To trigger the movement of the events, a timer control will be used.

1. Click the Timer control on the **System** page of the control palette.
2. Click the form to add a Timer control, Timer1.
3. Double-click the **OnTimer** event in the Object Inspector to open the code window.
4. Add code to the Timer procedure.

```

procedure TForm1.Timer1Timer(Sender: TObject);
var
  maxDist      : Double;
  nEventCount,
  iIndex       : Integer;
  geoEvt       : IMoGeoEvent;
  rect         : IMoRectangle;
  tLayer       : IMoTrackingLayer;
begin
  rect := IMoRectangle(Map1.Extent);
  maxDist := rect.Width / 20;
  tLayer := Map1.TrackingLayer;
  nEventCount := tLayer.EventCount;

  for iIndex := 0 to nEventCount - 1 do
    begin
      geoEvt := IMoGeoEvent(tLayer.Event[iIndex]);
      // move each event randomly
      geoEvt.Move(maxDist * (Random - 0.5), maxDist * (Random - 0.5));
    end
  end

```

```

        geoEvtnt := nil;
    end;
end;

```

## Add a CheckBox to your form

To control the timer, a CheckBox control will be used.

1. Select Timer1 control and set its **Enabled** property to False.
2. Add a CheckBox control from the **Standard** controls page to the bottom of the form.
3. Set the **Caption** property of the check box to "Data Collection".
4. In the Object Inspector, double-click the **OnClick** event for the check box to open the code window.
5. Add code to CheckBox1's Click procedure.

```

procedure TForm1.CheckBox1Click(Sender: TObject);
begin
    Timer1.Enabled := CheckBox1.Checked;
end;

```

## Test your changes

1. Run your application.
2. Zoom into an area.
3. Click the event tool, then click in the map to add events.
4. Click the Data collection check box. The events start moving randomly on top of the map.
5. Click the check box again to stop the events.

## Working with DataConnection objects

In each of the previous sections you have worked with MapLayer objects that were specified interactively using the Map control's property sheet. In this section you will add code to your application which creates MapLayer objects programmatically using a DataConnection object.

### Remove the existing layers

1. Right-click the mouse on the map to display the context menu.
2. Choose **Properties** to display the property sheet.
3. Click on the Ushigh layer, then click **Remove** to delete the layer.
4. Remove Counties and States in the same manner, then click **OK**.

### Add a procedure which will initialize the map

1. Open the Code window.
2. In the form's private declaration section, declare a procedure.

```

{ Private declarations }
gSelection: _Recordset;
procedure InitializeMap;

```

3. Add the procedure to the end of the code.

```

procedure TForm1.InitializeMap;
var
    dc : IMoDataConnection;

```

```

Layer : IMoMapLayer;
sym   : IMoSymbol;
lys   : IMoLayers;
begin
dc := MoDataConnection(CreateOLEObject('MapObjects2.DataConnection'));
dc.Database := 'c:\Program Files\ESRI\MapObjects2\Samples\Data\USA';

if (dc.Connect) then
begin
Layer := IMoMapLayer(CreateOLEObject('MapObjects2.MapLayer'));
Layer.GeoDataset := IMoGeoDataset(dc.FindGeoDataset('States'));
sym := Layer.Symbol;
sym.Color := clYellow;
lys := Map1.Layers;
lys.Add(Layer);

Layer := IMoMapLayer(CreateOLEObject('MapObjects2.MapLayer'));
Layer.GeoDataset := dc.FindGeoDataset('Counties');
lys := Map1.Layers;
lys.Add(Layer);

Layer := IMoMapLayer(CreateOLEObject('MapObjects2.MapLayer'));
Layer.GeoDataset := dc.FindGeoDataset('ushigh');
// sym.Color := clRed;
lys.Add(Layer);
end
else
raise Exception.Create('The data could not be located. ');
end;

```

3. Add a call (see **bold** below) to your procedure in the Form's **OnActivate** procedure.

```

procedure TForm1.FormActivate(Sender: TObject);
var
breakVal : Double;
i        : Integer;
lys      : IMoLayers;
ly1,
ly2     : IMoMapLayer;
cbr      : IMoClassBreaksRenderer;
ddr      : IMoDotDensityRenderer;
recset1,
recset2  : IMoRecordset;
stats1,
stats2   : IMoStatistics;
begin
// Initialize the map
InitializeMap;
// counties layer

```

.....

4. Run your application to test the changes.  
5. Save your changes.