Getting Started with MapObjects 2.x and C++ Builder 5

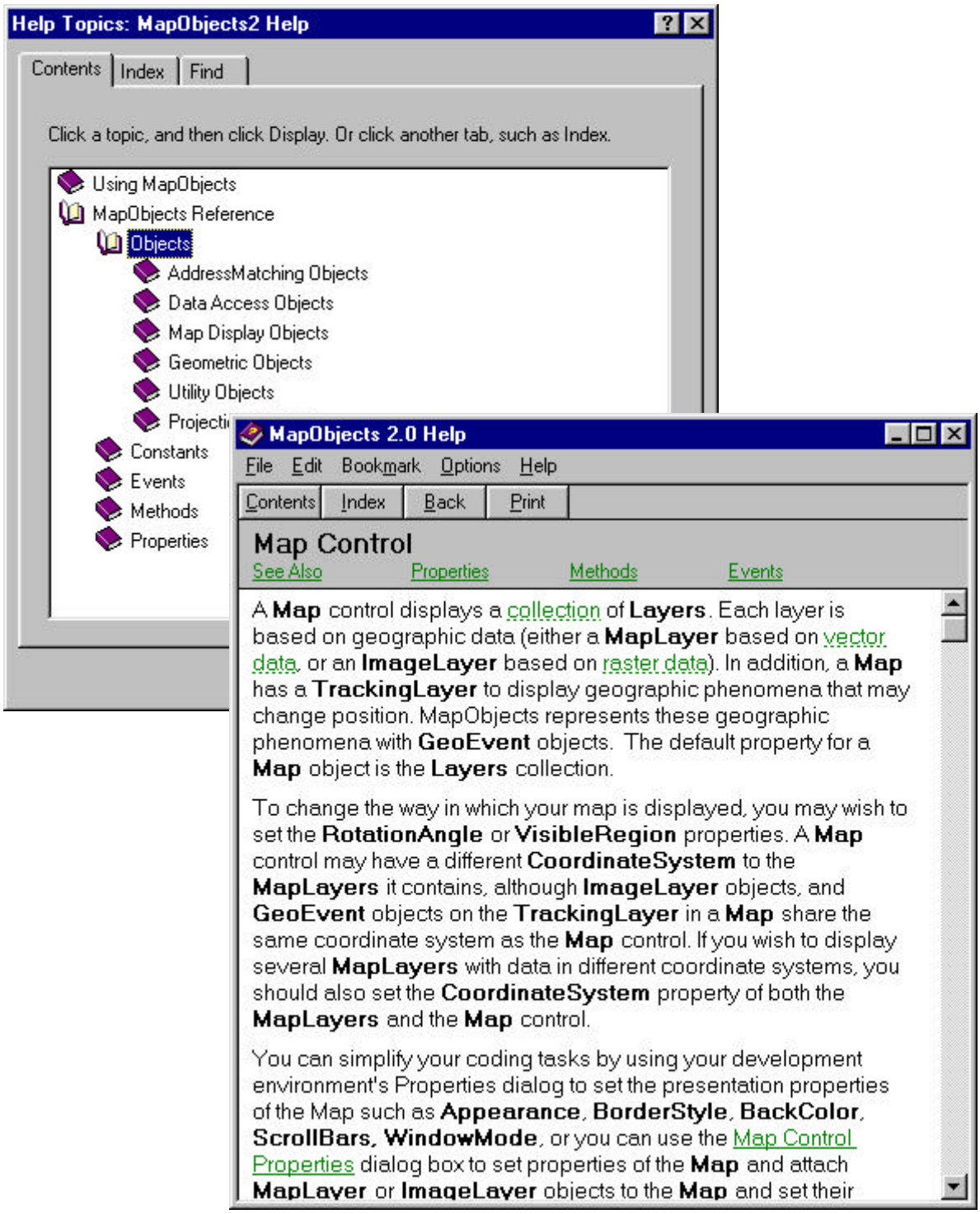
In this introductory document you will use MapObjects Version 2.X software and C++ Builder 5 to build a simple mapping application. Along the way you will learn how to:

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

The general techniques discussed here will work with any C++ development environment. The specific step-by-step examples, however, make use of Borland C++ Builder version 5. It is assumed that the user has some experience with this development environment.

Getting help

There are over forty-five objects that make up MapObjects; the map control is one of these. To find out about the various objects, open the MapObjects online help and choose **Objects** from the table of contents. The help system provides help for every object, property, method, event, and constant in MapObjects.

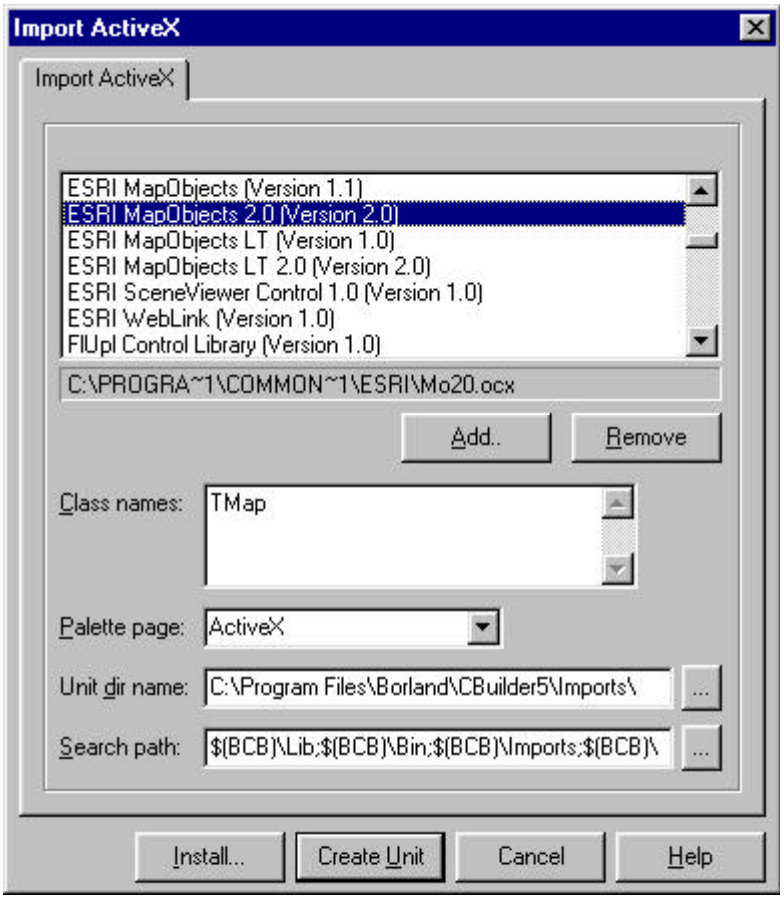


Adding a Map

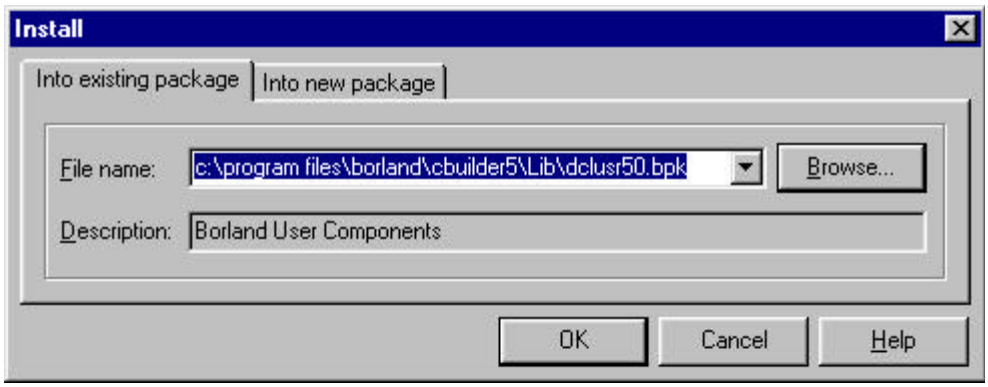
Import the Map control

To add the MapObjects ActiveX control on to your component palette:

1. Choose Component→ Import ActiveX Control
2. Scroll down and select *ESRI MapObjects 2.0* from the list of libraries. Note the settings for the Palette page and Unit directory options.



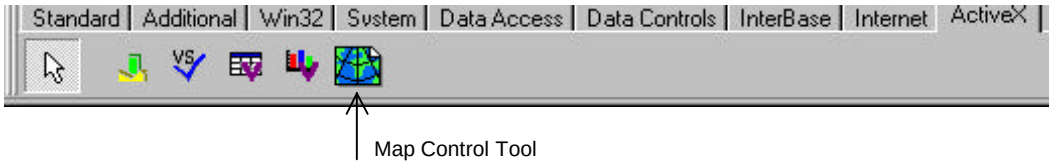
3. Select *Install* to install the MapObjects control on the component palette. This will generate the *MapObjects2*_TLB and *MapObjects2*_OCX units and then display the following dialog box:



4. Accept the defaults by clicking *OK*. The package will be rebuilt and the MapObjects components will now reside in the Borland User Components package. You will see a message box that says that the TMap component has been registered; select *OK* and exit out of the Package dialog - saving the changes made to the package when prompted. [1]

Add the map control to the main form

1. On the main window, choose the ActiveX tab on the Component palette; you should now see the MapObjects control listed among the other controls.

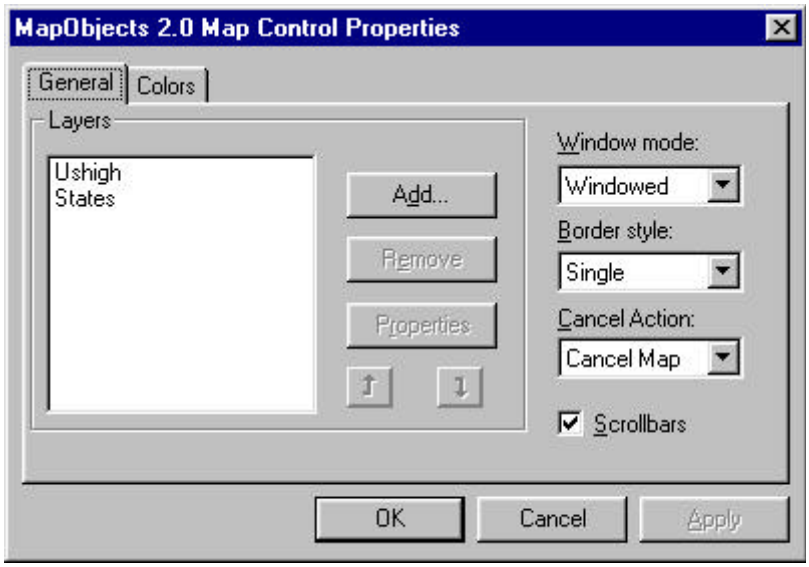


2. Either select the map control tool and drag a map onto your form or double-click the map control on the toolbox to add a new map to the form.
3. Size the map so that it fills the entire form.

Select the data to display on the map

You can specify what data to display in the map by setting properties in the map control's property sheet.

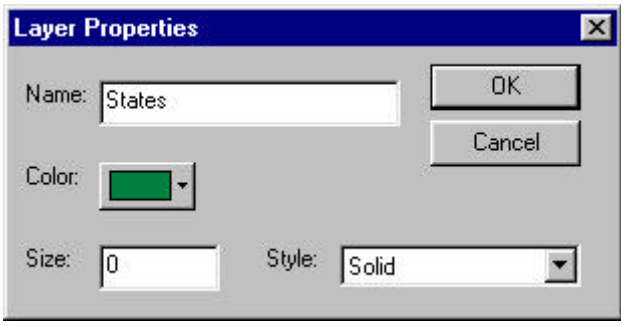
1. Right-click the mouse on the map to display the context menu.
2. Choose *Properties* to display the property sheet.
3. Click *Add* and locate the folder where the USA sample data is stored.
4. Choose the "States.shp" file and the click *Open.*
5. Add the "USHigh.shp" file in the same manner.



[1] For more information on Importing ActiveX controls, see section. 34-4 of the *Borland C++ Builder 5 Developer's Guide*

Set properties for the layers

1. Choose the States layer in the layers list and click *Properties*.
2. Click the *Color* button to select a color for the States layer.
3. Click *OK* to close the dialog box.
4. Select a color for the Ushigh layer in the same manner.
5. Close the MapObjects Map control Properties dialog box.



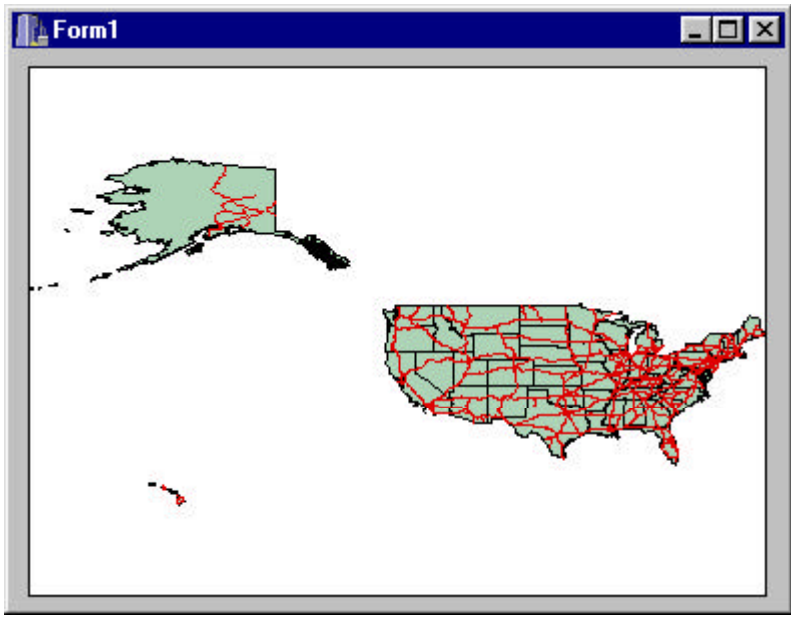
Test your application

Before you run your application, be sure to save your project.

1. Choose File→Save All on the main menu bar.
2. Save the unit as "StarterMap.cpp".
3. Save the project as "Starter_Map.bpr"
4. Add the line of code that is italicized below in Starter_map.cpp:

```
#include <vcl.h>
#pragma hdrstop
USERES("Starter_map.res");
USEFORM("StarterMap.cpp", Form1);
USEUNIT("C:\Program Files\Borland\CBuilder5\Imports\MapObjects2_TLB.cpp");
```

5. Build and run your application.



Adding Pan and Zoom Capability

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

Respond to the MouseDown event

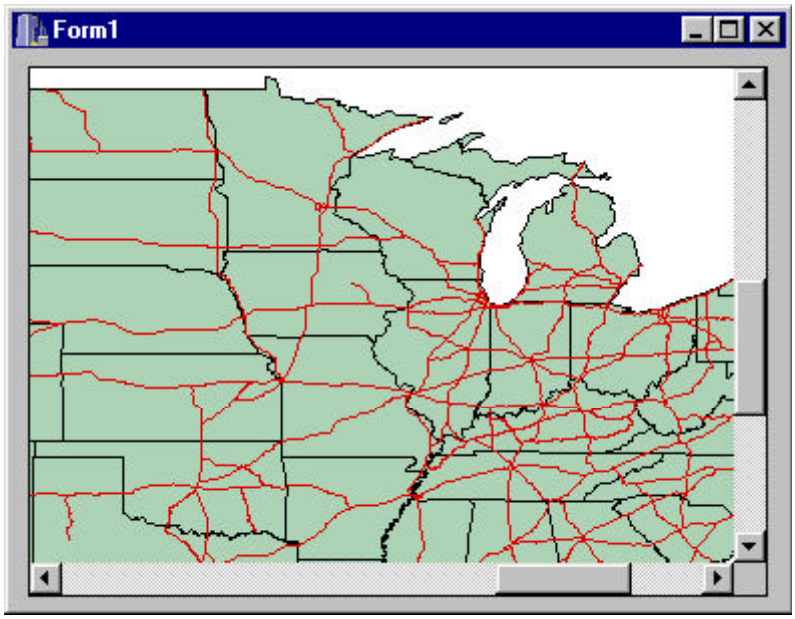
1. Select the map control on your form and then press Alt+Enter to pop up the Object Inspector window.
2. Under the *Events* tab, double-click the OnMouseDown event.
3. Add the following code to the *Map1MouseDown* procedure:

```
void __fastcall TForm1::Map1MouseDown(TObject *Sender, TMouseButton Button,
TShiftState Shift, int X, int Y)
{
     IMoRectanglePtr trackRect = Map1->TrackRectangle();
     Map1->Extent = trackRect;
}
```

TrackRectangle is a method that applies to a map. It tracks the movement of the mouse while the user holds the mouse button down, rubber-banding a rectangle at the same time. When the user releases the mouse button, the *TrackRectangle* method returns a Rectangle object. The map's *Extent* property is set to the rectangle object, causing the map to be redrawn with a new map extent.

Test your change

1. Build and run the application
2. Click the map with the left mouse button and drag a rectangle. Release the mouse button and notice that the map is redrawn at the location you specified.



Add panning and zooming out capabilities

1. Double-click on Map1's OnMouseDown event in the Object inspector to display the code window again.
2. Change the code for the *Map1MouseDown* event:

```
void __fastcall TForm1::Map1MouseDown(TObject *Sender, TMouseButton Button,
TShiftState Shift, int X, int Y)
{
   if (Button == mbLeft)
   {
      if (Shift.Contains(ssShift))
      {
        IMoRectanglePtr extRect = Map1->Extent;
        extRect->ScaleRectangle(1.5);
        Map1->Extent = extRect;
      }
      else
      {
        IMoRectanglePtr trackRect = Map1->TrackRectangle();
        Map1->Extent = trackRect;
      }
   }
   else
      Map1->Pan();
}
```

   If the left mouse button is used when the *MouseDown* event occurs, then the zooming code from the previous step will be executed. If the Shift button is also depressed, then the map will zoom out to an extent that is 1.5 times larger than the previous extent. That is, the map scale will become smaller. If the right mouse button is used, another map method, *Pan*, will be issued.

Test your change

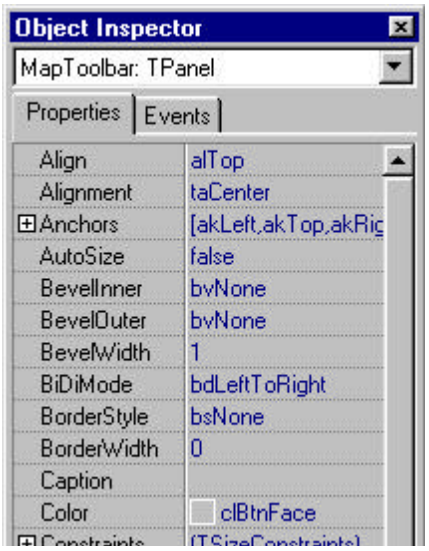
1. Build and run your application.
2. Click and drag on the map with the left mouse button to track a rectangle. Release the mouse button and notice that the map is redrawn at the location you specified.
3. Cick and drag on the map with the right mouse button to pan to a new location on the map.
4. Press the Shift button and click on the map to zoom out from the current extent.
5. Save your changes.

Using the Toolbar

Your application now supports panning and zooming, but these capabilities are somewhat hidden from the user. In this section you will add buttons to the toolbar that allow you to zoom or pan.

Adding a toolbar

1. Select the Panel control from under the Standard control tab on the Component palette and add a panel at the top of the form.
2. Using the Object Inspector, 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".
3. Resize the map so that it is not obscured by the panel.

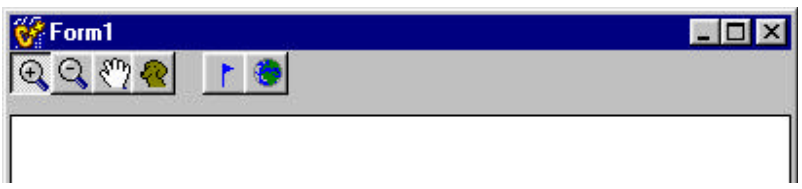


Adding buttons to the panel control

1. Select the MapToolbar panel
2. Under the Additional tab on the Component palette, select the SpeedButton control and click on the MapToolbar panel to place a button on the panel.
3. Using the Object Inspector, set the *Name* property for the new button to "ZoomInBtn"
4. Double-click on the *Glyph* property to open the property editor for glyphs.
5. Use the *Load* button to select "Zoomin.bmp" as the picture for the zoom-in button. This picture can be found under the \Samples\Bitmaps subdirectory of the folder in which you installed MapObjects. (e.g. C:\Program Files\ESRI\MapObjects\Samples\Bitmaps)
6. Add the following buttons to the panel in the same manner:

   ZoomOutBtn (zoomout.bmp) PanBtn (Pan.bmp) QueryBtn (Bex.bmp)

   FullExtentBtn (Globe.bmp) AddEventBtn (Pennant.bmp)
7. Since the first four buttons will work together like radio buttons so that only one can be selected at a time, select the first four buttons and set their *GroupIndex* property to 1.
8. Set the *Down* property for the ZoomInBtn to True to make it the selected tool by default.



Change the MouseDown event

1. Go to the *MouseDown* event for the map control. (Select the map control then double-click the OnMouseDown event in the Object Inspector.)
2. Modify the code for the Map1MouseDown procedure so that the response to the event will depend on the selected tool:

```
void __fastcall TForm1::Map1MouseDown(TObject *Sender, TMouseButton Button,
TShiftState Shift, int X, int Y)
{
   if (ZoomInBtn->Down)
   {
     IMoRectanglePtr trackRect = Map1->TrackRectangle();
     Map1->Extent = trackRect;
   }
   if (ZoomOutBtn->Down)
   {
     IMoRectanglePtr extRect = Map1->Extent;
     extRect->ScaleRectangle(1.5);
     Map1->Extent = extRect;
   }
   if (PanBtn->Down)
      Map1->Pan();
}
```

Implement the FullExtent button

1. Select the FullExtentBtn on the form and double-click the OnClick event in the Object Inspector window to create the event handler for this button.
2. Add the code to implement the FullExtent button:

```
void __fastcall TForm1::FullExtentBtnClick(TObject *Sender)
{
   IMoRectanglePtr fullExt = Map1->FullExtent;
   Map1->Extent = fullExt;
}
```

   The map has two extent properties: the FullExtent property is the full extent of the map; the Extent property keeps track of the current visible extent of the map. Setting the visible extent to the full extent results in the whole map being shown.

Test your changes

1. Build and run your application
2. Use the zooming and panning tools to move around the map.
3. Click the full extent button (the globe) to draw the map at its full extent.
4. Save your changes

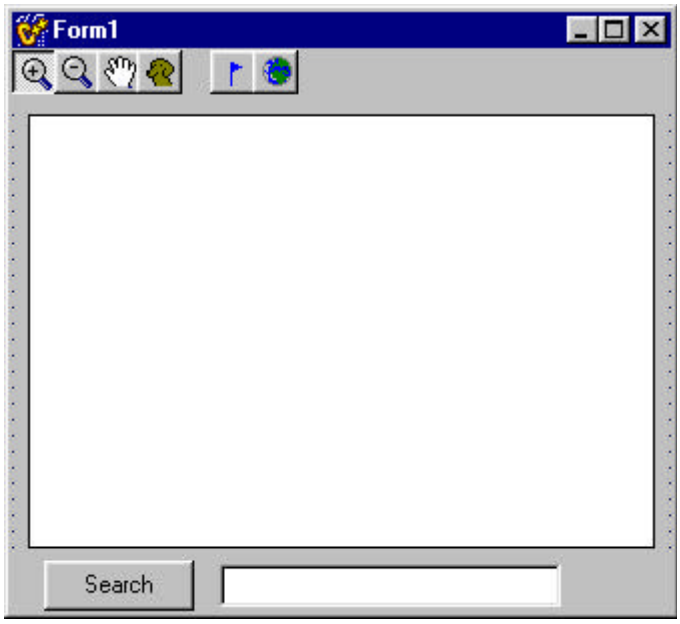
Creating a Search Toolbar

In this section you will add additional controls to your application to implement a simple function for locating a state by name. You will add a new panel control that contains a textbox and command button for searching for map features by their name attribute.

Add the panel for the Search function

Like the toolbar panel that was added to the top of the form, this panel at the bottom of the form will make it easier to handle form resizing and automatic positioning of the controls.

1. Select the Panel control from under the Standard control tab on the Component palette and add a panel at the bottom of the form.
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, under the Standard tab on the Component palette, select the Button control and click on the panel to place a button on it.
4. Set the *Caption* property of the button control to "Search". Set the *Name* of the button to "SearchStateBtn".
5. With the panel selected, select the Edit control on the component palette then click on the panel to place an edit control on it.
6. Set the *Text* property of the edit control to blank.
7. Select the button and edit controls then display the Alignment Palette by selecting " Alignment Palette" from the View menu. Choose " Align Vertical Centers", then "Center Vertically in Window" buttons to position the new controls.



Implement the Search Button

1. Double-click the search button to open up the button's click event procedure
2. Add code to the Click procedure:

```
void __fastcall TForm1::SearchStateBtnClick(TObject *Sender)
{
   IMoMapLayerPtr lyr;
   IMoLayersPtr lyrs;
   IMoRecordsetPtr recset;
   IMoFieldsPtr flds;
   IMoPolygonPtr shape;
   IMoRectanglePtr rect;

   AnsiString query;
   query = "STATE_NAME ='" + Edit1->Text + "'";
   lyrs = Map1->Layers;
   lyr = (IMoMapLayer*)(IDispatch*)lyrs->Item(WideString("States").Detach());
   recset = lyr->SearchExpression(WideString(query).Detach());
   if(!(bool)recset->EOF)
   {
      flds = recset->Fields;
      shape = (IMoPolygon*)(IDispatch*)flds->Item("Shape")->Value;
      Map1->FlashShape(shape, 3);
      rect = shape->Extent;
      rect->ScaleRectangle((1.5));
      Map1->Extent = rect;
   }
}
```

The search code first builds a simple SQL query expression using the text from the edit control. The States layer is then 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 extent of the shape is scaled and then set to be the extent of the map. The map is then explicitly redrawn using the *Refresh* method, and finally, the shape is flashed three times.

Test your changes

1. Build and run your application.
2. Type the name of a state, for example, Illinois, into the search field. Note that the first character must be a capital letter.
3. Click the Search button.
4. Exit out of the form.

Handling Map Resizing

When you run your application and resize the form, you will notice that the map is not automatically resized. One way to resize the map is to reset its *Width* and *Height* properties when the form is resized.

1. In the Object Inspector window, double-click on the OnCanResize event for Form1.
2. Add the code to handle map resizing:

```
void __fastcall TForm1::FormCanResize(TObject *Sender, int &NewWidth,
      int &NewHeight, bool &Resize)
{
   Map1->Width = NewWidth - 10;
   Map1->Height = NewHeight - 100;
}
```

3. Save your project and test your application.

   Notice that the map control is resized together with the application window; however, you may experience an unacceptable amount of flickering. You can easily avoid the flickering by setting your Windows environment not to redraw until the window has been resized. You can do this programmatically by calling the Windows API function `SystemParametersInfo(…)` when your application is initialized.
4. Select the View→Units menu option; select "Starter_map"
5. Insert the line of code that is italicized below into the try {…} block:

```
try
{
   Application->Initialize();
   Application->CreateForm(__classid(TForm1), &Form1);
   SystemParametersInfo(SPI_SETDRAGFULLWINDOWS, FALSE,NULL, 0);
   Application->Run();
}
```

6. Build and Run your application. Note that map no longer flickers when the form is being 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 on the map to display the context menu. Click *Properties* to show the property sheet.
2. Click the *Add* button and locate the folder where the sample data is stored.
3. Select 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. Select *OK* to dismiss the Layer Properties dialog.
8. Select *OK* to dismiss the property sheet.

If you run your application now you will notice that every county in the US is displayed. At the full extent, there is no need for such 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

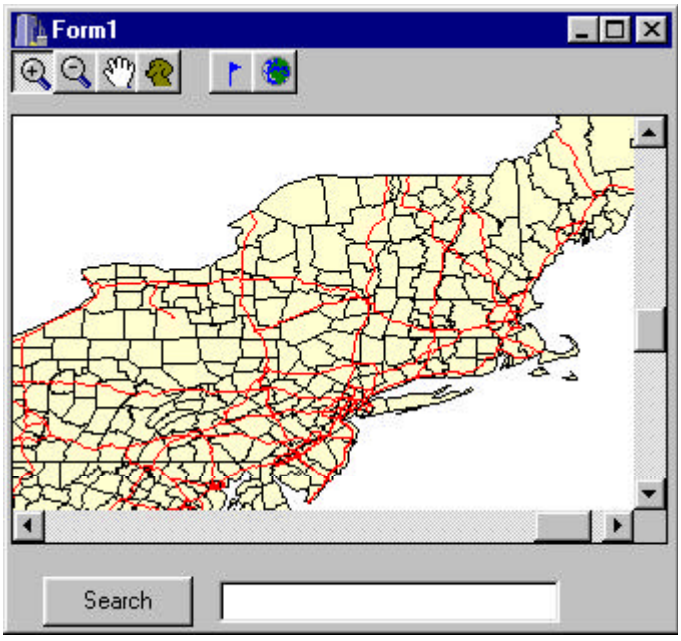
1. Use the Object Inspector to add the event handler for the *BeforeLayerDraw* event of the map control.
2. With the map control as the selected control on your form, double-click the OnBeforeLayerDraw event in the Object Inspector to create an event handler.
3. Add code to the procedure:

```
void __fastcall TForm1::Map1BeforeLayerDraw(TObject *Sender, short index,
      OLE_HANDLE hDC)
{
   IMoLayersPtr lyrs;
   IMoMapLayerPtr layer;
   IMoRectanglePtr extent;
   IMoRectanglePtr fullExtent;

   extent = Map1->Extent;
   fullExtent = Map1->FullExtent;

   BOOL showDetail = extent->Width < (fullExtent->Width / 4.0);

   lyrs = Map1->Layers;
   layer = (IMoMapLayer*)(IDispatch*)lyrs->Item(TVariant(index));

   if (index ==1)
      layer->set_Visible(showDetail);
   if (index == 2)
      layer->set_Visible(!showDetail);
}
```

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 layer will be visible and the States layer 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. Build and 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. Be sure to save your changes.

Adding a Spatial Query Tool

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

Add the query code

1. In StarterMap.h, add a private MapObjects Recordset variable to store the results of the query:

```
private:       // User declarations
   IMoRecordsetPtr g_Selection;
```

2. Open StarterMap.cpp and go to the `TForm1::Map1MouseDown` method.
3. Add code to implement the query tool:

```
IMoPointPtr mapPt;
IMoLayersPtr layers;
IMoMapLayerPtr ushigh;
IMoMapLayerPtr cntys;
IMoRecordsetPtr selHWys;
```

```
if (QueryBtn->Down)
{
   mapPt = Map1->ToMapPoint(X,Y);
   layers = Map1->Layers;
   ushigh = (IMoMapLayer*)(IDispatch*)layers->Item(WideString("USHigh"));
   cntys = (IMoMapLayer*)(IDispatch*)layers->Item(WideString("Counties"));
   // Search for a highway within N pixel tolerance
   selHWys = ushigh->SearchByDistance(mapPt, 0.1, WideString(""));
   if (!(bool)selHWys->EOF)
   {
      WideString exp("");
      // Find the counties that the selected highways intersect
      g_Selection = cntys->SearchShape(selHWys, moEdgeTouchOrAreaIntersect, exp);
      if (!(bool)g_Selection->EOF);
      {
         TVariant va;
         VariantInit(&va);
         va.vt = VT_NULL;
         Map1->TrackingLayer->Refresh(TRUE, va);
      }
   }
   else
      MessageBeep(MB_ICONQUESTION);
}
```

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, 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, g_Selection.

Add the selection display code

The MouseDown code initializes the g_Selection variable to point to the selected counties. Display the selection in response to the *AfterLayerDraw* event.

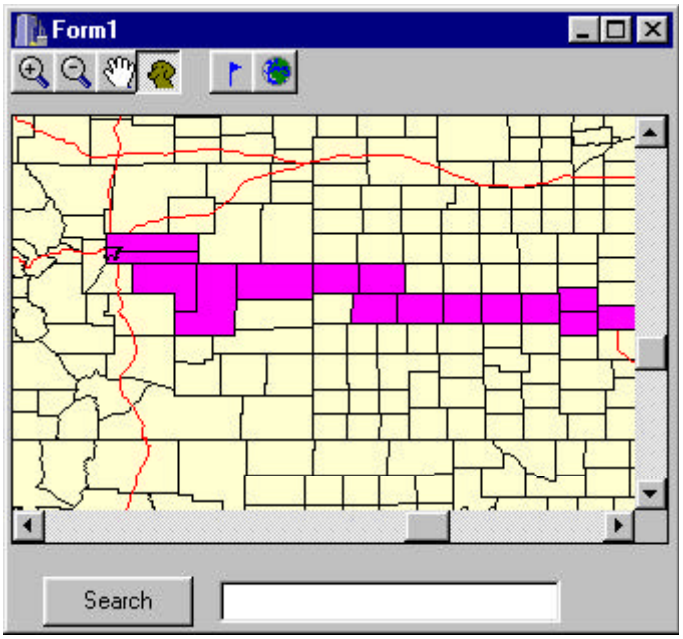
1. Use the Object Inspector to add the AfterLayerDraw handler to TForm1.
2. Add the following code to `TForm1::Map1AfterLayerDraw`:

```
void __fastcall TForm1::Map1AfterLayerDraw(TObject *Sender, short index,
      TOLEBOOL canceled, OLE_HANDLE hDC)
{
   if (index!=1)
      return;  //Only draw after counties
   if ((IDispatch*)g_Selection == 0)
      return;

   //Draw the selected counties in a highlight color
   IMoSymbolDisp sym;
   sym = (IDispatch*)CreateOleObject("MapObjects2.Symbol");
   sym->Color = moMagenta;

   g_Selection->MoveFirst();
```

```
    IMoFieldsPtr flds(g_Selection->Fields);
    IMoFieldPtr shpFld(flds->Item("Shape"));
    IMoPolygonPtr shape;

    while(!(bool)g_Selection->EOF)
    {
       shape = (IDispatch*)shpFld->Value;
       Map1->DrawShape(shape, (IDispatch*)sym);
       g_Selection->MoveNext();
   }
}
```

Test your changes

1. Build and Run your application
2. Zoom in so that the counties are shown
3. Click the spatial query tool, then click on a highway.



4. You may notice an unacceptable amount of flickering as the screen is redrawn. This is because all three layers have been forced to redraw, which, in this instance is unnecessary, as the underlying layers have not actually been changed. There are a number of ways to avoid this. For example, you could use the *RefreshRect* method on the map control and pass a rectangle as a parameter representing the specific area to be drawn. Alternatively, you could draw the selected shapes within the *AfterTrackingLayerDraw* event handler. This uses what is called the "bitmap backing store" which is used to refresh the map control background if no changes to the layers have been made. As no actual changes to our layers have been made, this is the ideal place to highlight a recordset on screen.

5. To draw the selection in the *AfterTrackingLayerDraw* event, add the AfterTrackingLayerDraw handler to TForm1 using the Object Inspector.
6. Cut the code from `TForm1::Map1AfterLayerDraw` and paste it into `TForm1::Map1AfterTrackingLayerDraw`.

   Ensure you remove the following lines:

   ```
   if (index!=1)
       return;  //Only draw after counties
   ```
7. Within the `TForm1::Map1MouseDown` event, modify the code implementation for the spatial query tool by replacing the lines:

   ```
   if (!(bool)g_Selection->EOF);
       Map1->Refresh();
   ```

   with:

   ```
   if (!(bool)g_Selection->EOF);
   {
           TVariant va;
           VariantInit(&va);
           va.vt = VT_NULL;
           Map1->TrackingLayer->Refresh(TRUE, va);
   }
   ```
8. Finally, to allow the roads to show through the highlighted selection of Counties, insert the following two lines right before setting the symbol object's color (i.e. `sym->Color = moMagenta;`) in the `TForm1::Map1AfterTrackingLayerDraw` event:

   ```
   sym->SymbolType = moFillSymbol;
   sym->Style = moLightGrayFill; //experiment with different fill styles
   ```
9. Rebuild and run your application to see the difference.

A Note on OLE Automation Memory Management:

In the selection drawing code, you will notice that CreateOleObject() is used. Until now, we have initialized all of our wrapper classes by passing a pointer to an existing object.

```
IMoRectanglePtr extent;
extent = Map1->Extent;
```

or:

```
IMoFieldsPtr flds(g_Selection->Fields);
```

Since there is not an existing symbol that we can attach our IMoSymbolPtr to, we need to create one from scratch. To create a new integer from scratch you would use the following code:

```
Int* pInt = new int;
```

To create a new OLE automation object from scratch, in this case a MapObjects symbol, use the following code:

```
IMoSymbolDisp sym;
sym = (IDispatch*)CreateOleObject("MapObjects2.Symbol");
```

Working with DataConnection Objects

In the previous section, you specified the MapLayer objects interactively using the Map control's property sheet. In this section you will add code to your application that creates MapLayer objects programmatically using the Data Access objects.

Remove the existing layers

1. Toggle to the form and then right-click 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 close the dialog box.

Add a procedure to initialize the map

1. In StarterMap.h, add a private function called "InitializeMap":

```
private:       // User declarations
   IMoRecordsetPtr g_Selection;
   void InitializeMap(void);
```

2. At the end of StarterMap.cpp, implement the function. Be sure to check the *Database* property of the DataConnection object to ensure that it is pointing to the directory where the USA sample data is stored.

```
void TForm1::InitializeMap()
{
   IMoDataConnectionPtr dc;
   IMoGeoDatasetPtr gds;
   IMoMapLayerPtr lyr;

   dc = (IDispatch*)CreateOleObject("MapObjects2.DataConnection");
   lyr = (IDispatch*)CreateOleObject("MapObjects2.MapLayer");

   dc->Database = WideString("C:\\ESRI\\MapObjects2\\Samples\\Data\\USA");

   if (!(bool)dc->Connect())
   {
      Application->MessageBox("Check path to DataConnection", NULL, MB_OK);
      return;
   }

   gds = dc->FindGeoDataset(WideString("States"));
   lyr->GeoDataset = gds;
   lyr->Symbol->Color = moLimeGreen;
   Map1->Layers->Add(lyr);

   lyr.Release();
   lyr = (IDispatch*)CreateOleObject("MapObjects2.MapLayer");
```

```
    gds = dc->FindGeoDataset(WideString("Counties"));
    lyr->GeoDataset = gds;
    lyr->Symbol->Color = moPaleYellow;
    Map1->Layers->Add(lyr);

    lyr.Release();
    lyr = (IDispatch*)CreateOleObject("MapObjects2.MapLayer");

    gds = dc->FindGeoDataset(WideString("USHigh"));
    lyr->GeoDataset = gds;
    lyr->Symbol->Color = moRed;
    Map1->Layers->Add(lyr);
}
```

3. Call the above function from the Form's constructor (at the top of StarterMap.cpp):

```
__fastcall TForm1::TForm1(TComponent* Owner)
        : TForm(Owner)
{
   InitializeMap();
}
```

4. Run your application and test your changes.

Statistical Mapping

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

Attach a renderer to the Counties layer

1. Navigate to the Form1's constructor (at the top of StarterMap.cpp).
2. Add code to attach a ClassBreaksRenderer to the Counties layer:

```
__fastcall TForm1::TForm1(TComponent* Owner)
        : TForm(Owner)
{
   //Add Data to map
   InitializeMap();

   //Set up county renderer
   IMoLayersPtr lyrs(Map1->Layers);
   IMoMapLayerPtr counties;
   IMoClassBreaksRendererPtr cbRenderer;

   counties = (IMoMapLayer*)(IDispatch*)lyrs->Item(WideString("Counties"));
   cbRenderer = (IDispatch*)CreateOleObject("MapObjects2.ClassBreaksRenderer");
   cbRenderer->Field = WideString("Mobilehome");

   //calculate breaks away from the mean in both directions
   //but only add those breaks that are withing the range of values
   IMoRecordsetPtr recs;
```

```
    IMoStatisticsPtr stats;

    recs = counties->Records;
    stats = recs->CalculateStatistics(WideString("Mobilehome"));

    double breakVal = stats->Mean - (stats->StdDev * 3);
    double min = stats->Min;
    double max = stats->Max;
    for (int i = 0; i < 7; i++)
    {
       if (min <= breakVal && breakVal <= max)
       {
          int breakCount = cbRenderer->BreakCount;
          cbRenderer->BreakCount = breakCount + 1;
          cbRenderer->set_Break(breakCount, breakVal);
       }
       breakVal = breakVal + stats->StdDev;
    }
    cbRenderer->RampColors(moPaleYellow, moOrange);
    counties->Renderer = cbRenderer;
}
```

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.

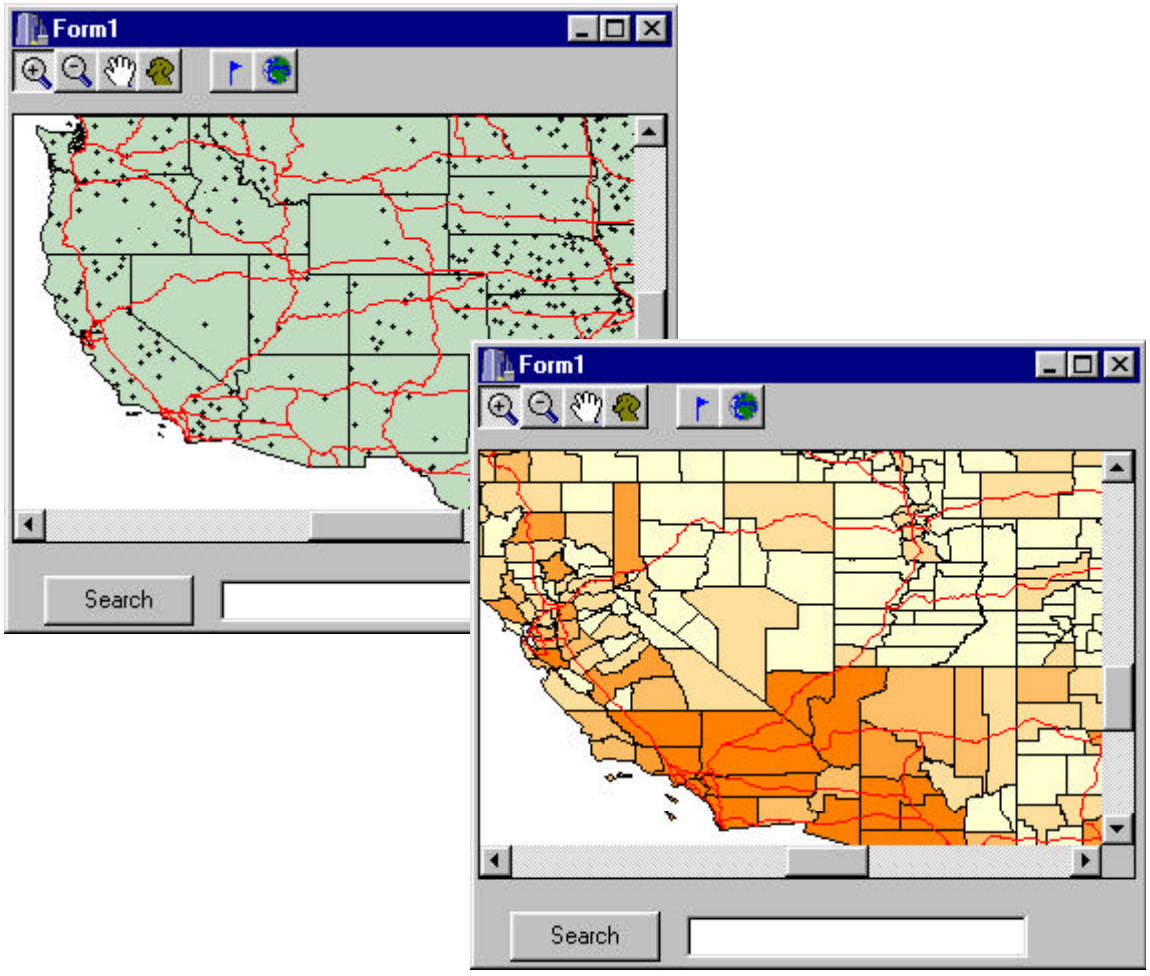
Attach a renderer to the States layer

1. Navigate to the Form's constructor (at the top of StarterMap.cpp).
2. Append the following code after the code for the Counties renderer:

```
//
//Set up States renderer
//
IMoMapLayerPtr states;
IMoDotDensityRendererPtr ddRenderer;
states = (IMoMapLayer*)(IDispatch*)lyrs->Item(WideString("States"));
ddRenderer = (IDispatch*)CreateOleObject("MapObjects2.DotDensityRenderer");
ddRenderer->Field = WideString("NO_FARMS87");
recs = states->Records;
stats = recs->CalculateStatistics(WideString("NO_FARMS87"));
ddRenderer->set_DotValue(stats->Max/100.0);
states->Renderer = ddRenderer;
```

Test your changes

1. Build and Run your application. Look at the States layer. Notice that the polygons are now drawn with dots that indicate the number of farms.
2. 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 representing the number of mobile homes in each county.



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 the vehicle changes location.

In this section you will add an event tracking layer to your application (though you may have already used this layer to highlight your selected counties).

Implement the event tool

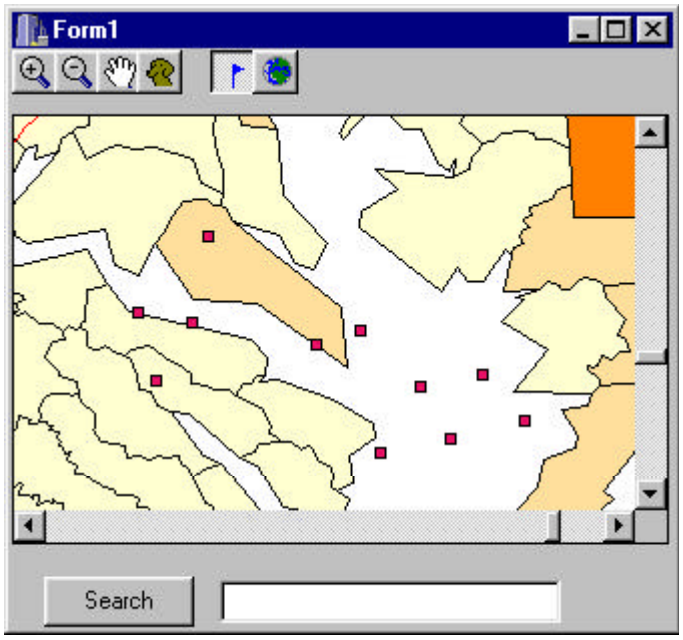
1. Select the AddEventBtn on the toolbar.
2. Open the Object Inspector and set the *GroupIndex* property of the AddEventBtn control to 1.
3. Add code to implement the Event tool to `TForm1::Map1MouseDown`:

```
IMoTrackingLayerPtr tLayer(Map1->TrackingLayer);
if (AddEventBtn->Down)
```

```
{
   mapPt = Map1->ToMapPoint(X,Y);
   tLayer->AddEvent((IDispatch*)mapPt, 0);
}
```

Test the event tool

1. Build and run your application
2. Zoom into an area
3. Click the event tool, then click on the map to add events.



Add a timer to your form

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

1. Select the Timer control on the System tab on the Component palette
2. Click the form to add a Timer control to it.
3. In the Object Inspector window, set the *Enabled* property of Timer1 to false.
4. Double-click the OnTimer event for Timer1 to open the code window.
5. Add code to the `TForm1::Timer1Timer` event:

```
void __fastcall TForm1::Timer1Timer(TObject *Sender)
{
   //Offset all of the events by a random amount
   IMoRectanglePtr extent(Map1->Extent);
   double maxDist = extent->Width/20.0;
   IMoTrackingLayerPtr tLayer(Map1->TrackingLayer);
   int eventCount = tLayer->EventCount;

   IMoGeoEventPtr event;
```

```
    for (int i = 0; i < eventCount; i++)
    {
       event = tLayer->get_Event(i);
       double xOffset = maxDist * ((double)rand()/(double)RAND_MAX - 0.5);
       double yOffset = maxDist * ((double)rand()/(double)RAND_MAX - 0.5);
       event->Move(xOffset, yOffset);
    }
}
```

Add a Checkbox control to your search panel

A CheckBox will be used to control whether or not real-time data collection is simulated

1. Select the bottom panel control.
2. Under the Standard tab on the Component palette, select the CheckBox control and click on the bottom panel to place a checkbox on it.
3. Set the *Caption* property of CheckBox1 to "Data Collection".
4. In the Object Inspector window, double-click the OnClick event for CheckBox1 to open the code window.
5. Add code to `the TForm1::CheckBox1Click:`

```
Timer1->Enabled = CheckBox1->Checked;
```

Test your changes

1. Build and run your application
2. Zoom into an area
3. Select the event tool, then click on the map to add events.
4. Click the checkbox. Notice that the events start moving randomly on top of the map.
5. Click the checkbox again to stop the events

Changing the GeoEvent symbol

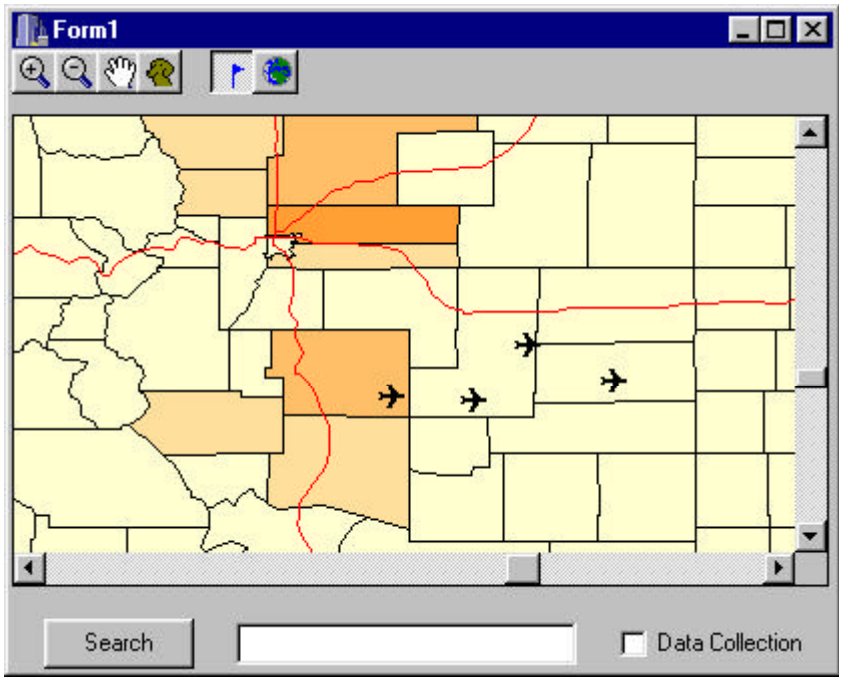
We will now change the default GeoEvent symbol to that of a small aircraft.

1. Append the following code to TForm1::TForm1:

```
//set tracking layer symbol

IMoTrackingLayerPtr tLyr(Map1->TrackingLayer);
tLyr->SymbolCount = 1;

IMoSymbolPtr sym(tLyr->get_Symbol(0));
TFont* stdFont;
_di_IFontDisp olefont;
stdFont = new TFont();
stdFont->Name = "Wingdings";
GetOleFont(stdFont, olefont);
delete stdFont;
sym->Font = (IDispatch*)olefont;
```

```
sym->Color = moBlack;
sym->Style = moTrueTypeMarker;
sym->Size = 14;
sym->CharacterIndex = 81;
```

2. Rebuild and run your application. You should find that instead of the default GeoEvent symbol you will have a small aircraft.



Changing the Mouse Icon

Finally, to give the user additional visual feedback as to what tool they currently have selected, we can make use of the MapObjects mouse pointers.

1. In the Object Inspector window, double-click the *OnMouseMove* event for Map1
2. Add code to `the TForm1::Map1MouseMove event:`

```
void __fastcall TForm1::Map1MouseMove(TObject *Sender, TShiftState Shift,
      int X, int Y)
{
   if (QueryBtn->Down)
      Map1->MousePointer = moIdentify;
   if (AddEventBtn->Down)
      Map1->MousePointer = moDefault;
   if (PanBtn->Down)
      Map1->MousePointer = moPan;
   if (ZoomOutBtn->Down)
      Map1->MousePointer = moZoomOut;
   if (ZoomInBtn->Down)
      Map1->MousePointer = moZoomIn;
}
```