Getting started with MapObjects in Visual Studio .NET and Visual Basic .NET

IN THIS TUTORIAL

- **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 data with an event tracking layer.**
- **Programmatically add vector and raster data to a map**

In this introductory document you will use MapObjects® and Microsoft® Visual Studio .NET® to build a simple mapping application using the Visual Basic (VB) language. No familiarity with Visual Studio .NET or VB is assumed, although some familiarity with basic programming concepts is useful. Along the way you will learn how to:

- Create a new Windows application in Visual Studio .NET, using toolbars and other controls standard in .NET.
- Add vector and raster data to a map, and perform queries on the map data you added.
- Control panning and zooming, display map layers based on scale, and render use thematic renderers to draw data based on attribute values.
- Draw simple graphics, and also dynamically display data.

MapObjects Software Developer Kit (SDK) for .NET

Before beginning this tutorial, you should check you have installed the MapObjects SDK for .NET. This kit is optionally installed as part of the MapObjects software installation.

The SDK contains signed assemblies, which by default will be located in the \DotNet\Assemblies sub folder of your MapObjects installation folder. The SDK also contains online reference documentation which integrates into the Visual Studio environment. You should also check the MapObjects readme document, for late-breaking information about the MapObjects SDK for .NET.

About the lines of code in this tutorial

In this tutorial, lines of code to be added or altered are shown in **bold** text. An elipsis (...) symbol is used to denote missing lines of code which are not relevant to the particular step. Some lines of code are shown split over two or more lines. This may include lines of code with hardcoded paths, where string variables are set with the concatenation character '+'. This is simply to allow the lines of code to be shown in this tutorial, and does not need to be copied in your code. However, all lines of code can be entered with these line continuations, if you wish.

Shortcut keys mentioned are the default settings for Visual Studio .NET.

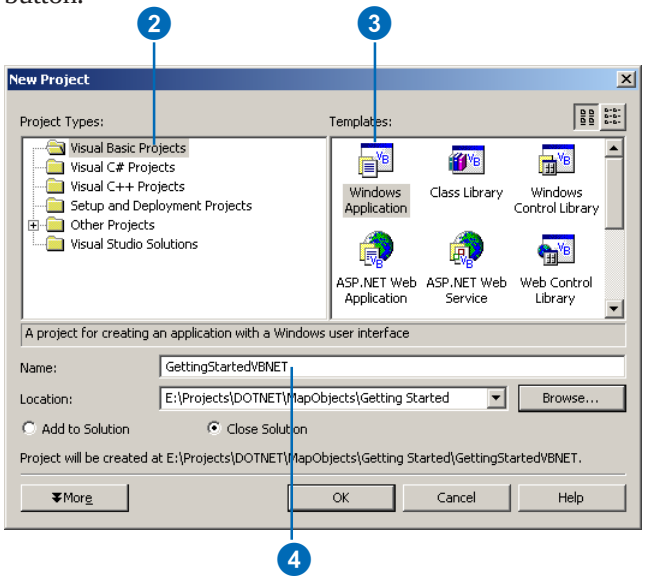
Sample data

If you accepted the defaults when installing MapObjects, the geographic data and bitmaps that this tutorial refers to can be found in the Data or Bitmaps folders, located under the MapObjects install directory. For example C:\Program Files\ESRI\MapObjects2\Samples\Data\Usa; and
C:\Program Files\ESRI\MapObjects2\Samples\Bitmaps.

Create a new Windows Application

You will start this tutorial by creating a new project in Visual Studio .NET. You will add a map to the project, and add layers to the map using the property sheet.

1. Open Visual Studio .NET, and click the New Project button on the Start Page. Alternatively, if the Start Page is not displayed, click the File menu, click New, then click Project.
2. In the left-hand pane of the New Project dialog box, select Visual Basic Projects.
3. In the right-hand pane, select Windows Application.
4. Name the project GettingStartedVBNET, and set the location to save the project by clicking the Browse button.



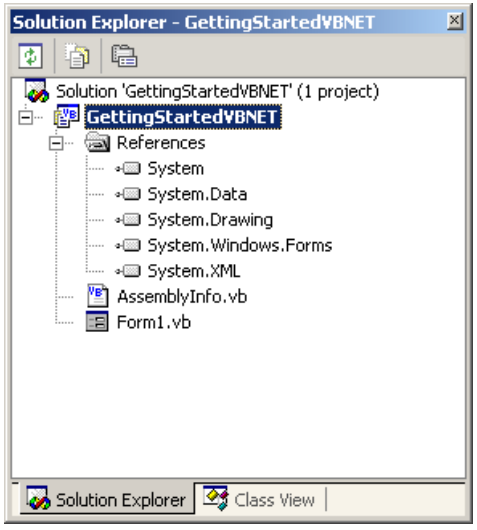
5. Click OK to create the new project.

The main document window should now display a Windows Form in design mode.

Visual Studio will, by default, create a sub-directory of the directory you selected to save your project files. The name of the new directory will be the same as the Project name you specifed. Your project is automatically added to a solution (.sln) file, which is somewhat like a project group.

You can investigate the new project by browsing the Solution Explorer window. This window shows all the files referenced by the solution file. The shortcut key to open the Solution Explorer is Ctrl+Alt+J.

You can also see a class-oriented view of your solution in the Class Explorer, shown in the same window as the Project Explorer. You will not be using the Class Explorer in this tutorial.



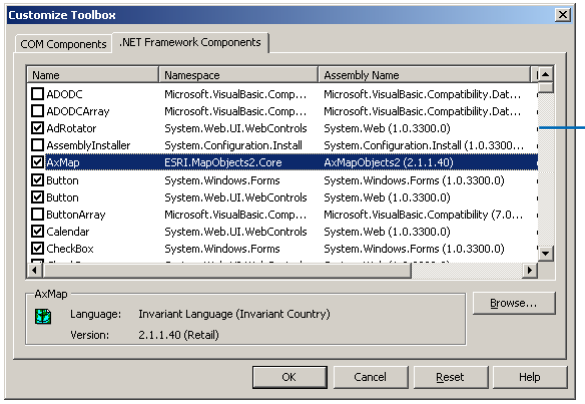
Add an AxMap control to a Form

You will need to reference the MapObjects Active X control in your project.

A .NET application cannot use ActiveX controls directly. However, the COM interoperability services provided by the .NET framework allow you to use ActiveX controls by using Runtime Callable Wrappers (RCW).

ESRI provides assemblies containing RCWs for the MapObjects ActiveX control and objects. The MapObjects ActiveX control is 'wrapped' inside a host class called AxHost, allowing a Windows Form in .NET to host the 'wrapped' map control. The name of the wrapped MapObjects map control is AxMap.

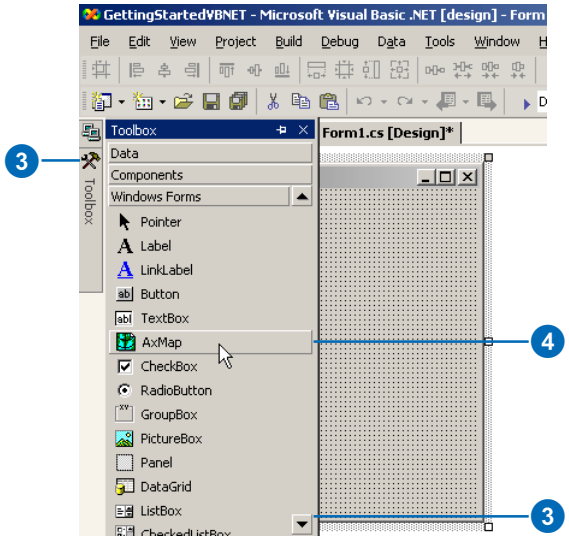
1. Click the Tools menu, and then click Customize Toolbox.
2. In the .NET Framework Components tab of the Customize Toolbox dialog, find AxMap, and check the box beside it. Click OK to close the dialog.



Note: Adding MapObjects to the Toolbox does not add a reference to the MapObjects assemblies to your project—you will perform this step next.

3. Click the Toolbox, shown at the left of the screen, to open it. Click the tab named Windows Forms. Notice the MapObjects wrapper class AxMap is now listed in the Toolbox.

   You may need to scroll down the list of controls to find the control, by clicking the black arrow at the bottom of the Toolbox tab.



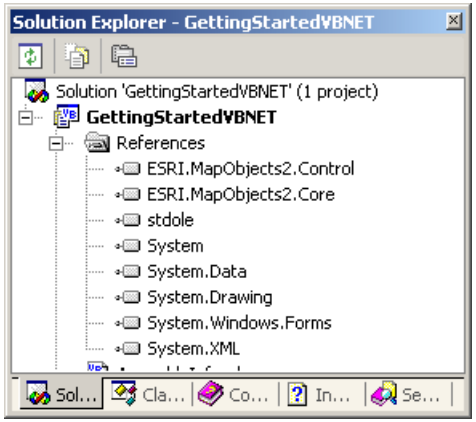
   Tip: An alternative way to open the Toolbox is to hover the mouse cursor over the Toolbox icon for a moment. The keyboard shortcut to open the Toolbox is the combination Ctrl+Alt+X.

4. Double-click AxMap in the Toolbox to add an AxMap control to the form.

When you add the AxMap class to your project by adding a map to a form, references to the MapObjects assemblies will be added to your project.

In Solution Explorer, navigate to the projects References, you will see two references:

- ESRI.MapObjects2.Core - contains RCWs for the objects in the MapObjects library.
- ESRI.MapObjects2.Control - contains a RCW for the MapObjects map control.



The objects in both these assemblies reside in the namespace ESRI.MapObjects2.Core. So to declare a MapObjects Symbol for example, you would need to use the full name ESRI.MapObjects2.Core.Symbol. You will now add an imports statement to shortcut this requirement.

Add an imports statement

1. Right-click the form, and from the context menu which appears, select View Code.
2. At the top of the code window, below the existing imports statements, add the following line of code.

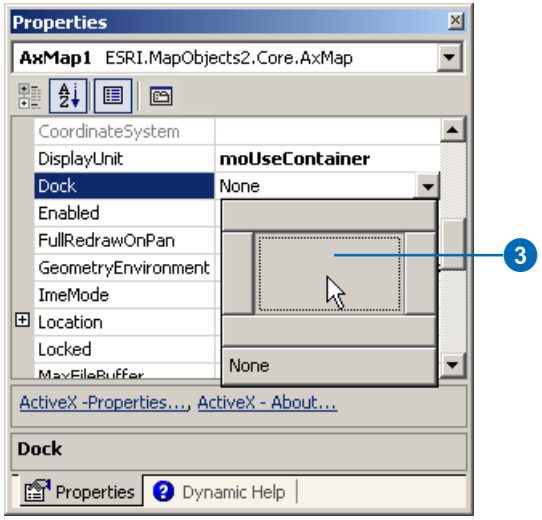
```
Imports System.Data
Imports ESRI.MapObjects2.Core
...
```

Control the resize of the AxMap control

Visual Studio provides built-in resizing functionality, which you will use to resize the map to fill the form.

1. Return to the form view by selecting the [Form1-Design] tab in the main window, then click the AxMap control on the form to select it.
2. Click the Properties window, shown by default at the bottom right of the Visual Studio .NET window, and scroll down to find the Dock property.

   Tip: The keyboard shortcut to activate the Properties window is F4.
3. Click the drop-down button next to the Dock property, and then click the central button on the displayed window, to select the Fill option.

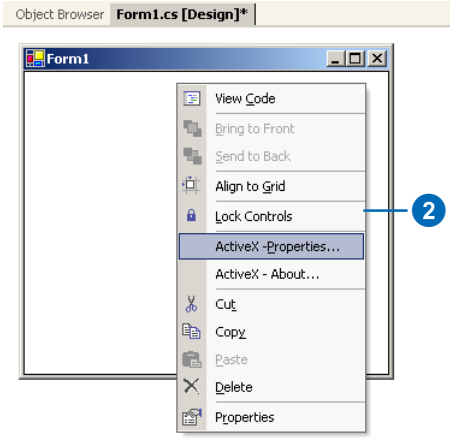


Now the map should fill the form completely.

Select the data to display on the map

You can specify the data that is displayed in the map by interacting with the AxMap control's property sheet. Later on you will perform the same task programmatically.

1. Right-click the AxMap control to display the context menu.
2. Choose ActiveX Properties to display the MapObjects property sheet.

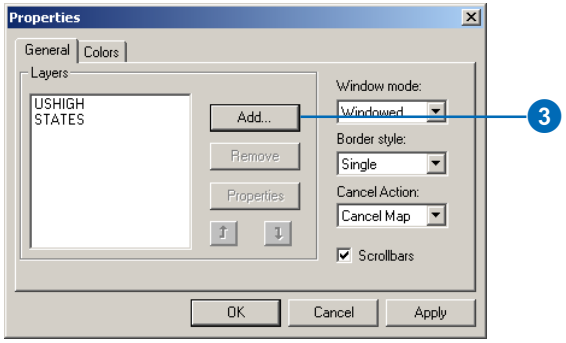


Note: Choosing the ActiveX Properties option will display the property sheet belonging to an ActiveX control, if it has one.

Alternatively, you can click the Property Sheet button in the Properties window.

3. In the Properties dialog box, click the Add button and locate the folder containing the States sample data.

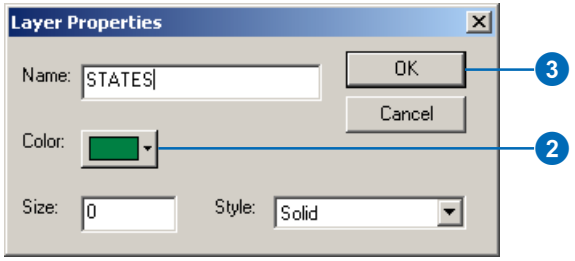
   If you selected the defaults when you installed MapObjects, the sample data will be located in C:\Program Files\ESRI\MapObjects2\Samples\Data.



4. Click the States.shp file, then click Open.
5. Add the file USHigh.shp in the same manner.

Set properties for the layers

1. Click the States layer in the list, then click Properties.
2. In the Layer Properties dialog box, click the Color button to select a color for the States layer.



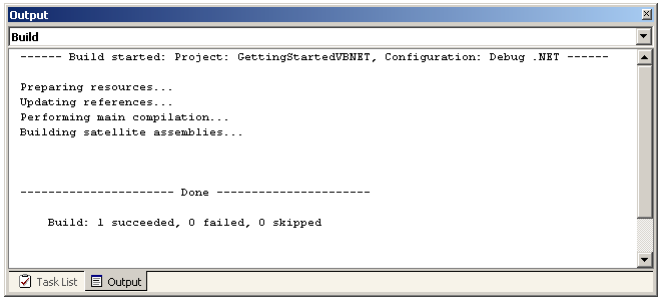
3. Click OK to close the Layer Properties dialog box.
4. Select a color for the USHigh layer in the same manner.
5. Click OK to close the MapObjects property sheet.
6. Click the File menu, then click Save All to save your project.

   Tip: The keyboard shortcut to save all items in the project is Ctrl+Shift+S.

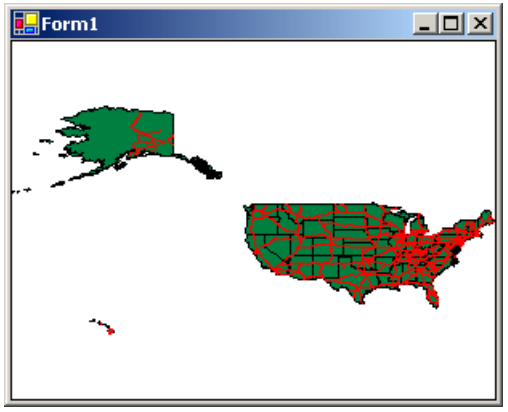
Test your application

1. Click the Start button on the Visual Studio toolbar.

   You should see the Build process progress in the Output window at the bottom of the screen.



   If the project builds correctly, the application will then run in Debug mode, and you will see your map showing USA states and highways.

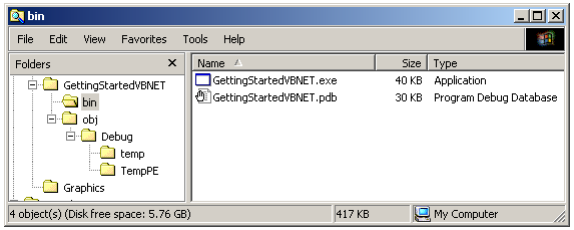


   If the Build process does not succeed, do not run the application. Return to design mode and check the Task List, shown in the window at the bottom of the screen, to see what errors are causing the problem.

   Correct the listed errors as described. If you double-click the task, the line of code causing the error will automatically be selected for you.

2. Click the Stop Debugging button on the Visual Studio toolbar to stop running your application and return to design mode.

3. You can check the results of the Build operation by looking in the sub-directories of your project.



   By default, a Debug version of your project is built. The executable file (.exe) that results from the Build operation will be stored in the \Bin directory. This directory will also contain debug information (.pdb).

   Note: The Obj sub-directory of the project directory contains temporary files used by the compiler and by Visual Studio.

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 your application will activate in response to mouse clicks inside the map.

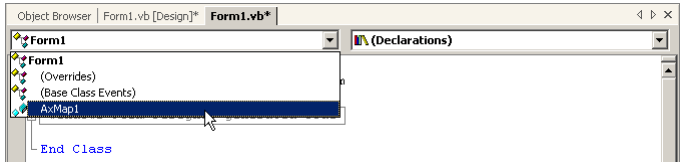
You will write some code that the application will execute in response to the MouseDownEvent event on the map.

Respond to the MouseDownEvent event

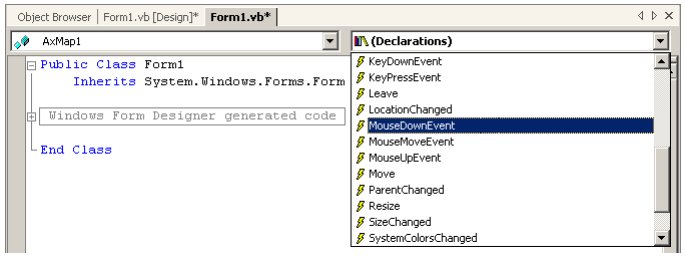
1. Select the form in the Solution Explorer, right-click it, then select View Code.

   The code window for your form is now displayed in the main document window.

2. Click the Class Name drop-down list (top left of the code window), and select AxMap1.



3. In the Method Name drop-down list, select MouseDownEvent.



A code stub for the MouseDownEvent event handler is added to the code window.

The Visual Studio environment automates the process of responding to events, hiding the details from you.

Note that the names of some MapObjects events in .NET end in 'Event'; for example 'MouseDownEvent'. This is because the .NET host class, AxHost, has a MouseDown event too—the 'Event' sufix is added to differentiate the event of the host class from the event of the underlying MapObjects Map class. See also the CtrlRefresh method, later in this tutorial.

Note also that as you added the using statement in the previous section, MapObjects variables can be declared without the ESRI.MapObjects2.Core prefix, and have been written without this prefix throughout this document for brevity.

If you wish to find out more about how events are raised and handled in .NET, please read the MSDN documentation included with Visual Studio .NET.

4. Add the following lines of code to the AxMap1_MouseDownEvent function.

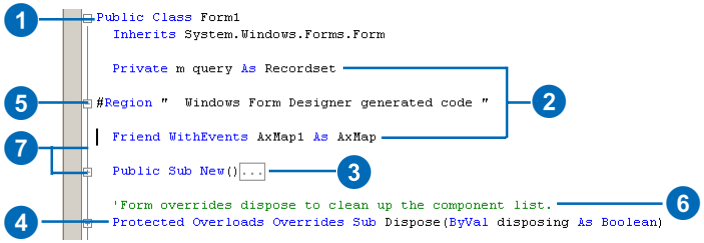
```
Private Sub AxMap1_MouseDownEvent(ByVal _
sender  As Object, ByVal e As _
MouseDownEventArgs) Handles _
AxMap1.MouseDownEvent
 AxMap1.Extent = AxMap1.TrackRectangle()
End Sub
```

Tip: If you start by typing "AxMap1", the auto-completion feature of IntelliSense will allow you to complete the item of code by pressing Tab.

The code window

When the code window for the form opens, you will see there are many lines of code already present in the code module, apart from the MouseDownEvent event handler you just created.

It may help to be familiar with certain aspects of this code, and items in the code window provided by the Visual Studio .NET IDE, before going any further.



1. Classes are declared using the class keyword—all code between the curly braces belongs to a class.

2. Class members are generally declared at the beginning of the class definition. Members of a class are often referred to as class fields in .NET, and may be prefixed with '-m'.

3. Constructor methods are called when an instance of a class is created. They are named New, and may include parameters to initialize the class.

4. Dispose methods are called when a class is no longer required and can safely clear up all it's resources; they are called at some point before the garbage collector clears the class from memory.

5. Windows Forms Designer generated code is a region (see 9) which is added by the Windows Forms Designer. It defines in code the form which you created visually. You should generally not edit this region.

6. Comments in VB.NET follow the familiar VB style of commenting. Comments are preceded by a apostrophe ('), and can appear at the start of a line, or after code.

7. Outlining is a handy way to organize your code. Lines of code between region directives can be hidden (collapsed), by clicking the adjacent plus or minus symbols. Regions can be nested.

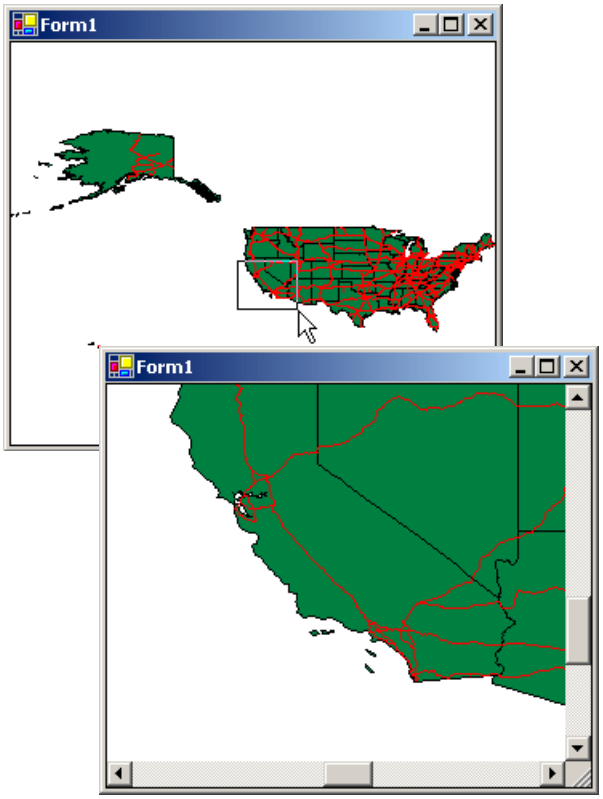
8. Class and Method name drop-down lists are found at the top of the code window, and are often used in VB .NET to construct function stubs, particularly for event handlers. The Class Name drop-down lists all the public classes declared in the module. The Method Name drop-down lists all the members of the selected class.

9. Namespaces are used to uniquely identify objects, and can be used to organize objects hierarchically, regardless of where they are defined. Objects in the ESRI.MapObjects2.Core and ESRI.MapObjects2.Control assemblies all belong to the ESRI.MapObjects2.Core namespace.

Test your changes

1. Click the Start button on the Visual Studio toolbar.
2. With the left mouse button, click the map and drag out a rectangle.
3. Release the mouse button.

   The map is redrawn at the location you specified.



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. The code assigns this Rectangle to the Extent property of the Map, causing the map to be redrawn with a new extent.

4. Click the Stop Debugging button in Visual Studio to return to design mode.

Add panning

1. Scroll the code window to find the MouseDownEvent you added previously.
2. Change the code as shown below.

```
Private Sub AxMap1_MouseDownEvent(ByVal sender
...
 If e.Button = 1 Then
   AxMap1.Extent = AxMap1.TrackRectangle()
 ElseIf e.Button = 2 Then
   AxMap1.Pan()
 End If
End Sub
```

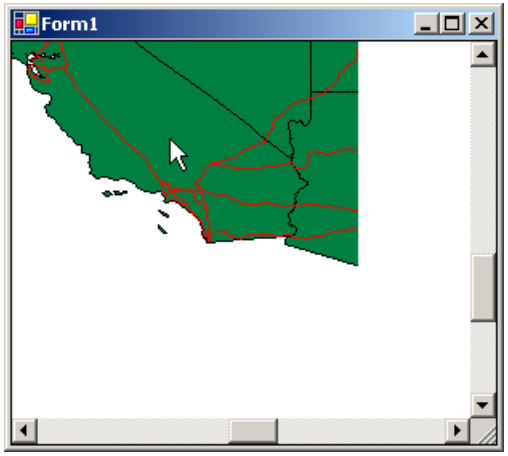
   Note: The e argument of the event contains the parameters which MapObjects passes into the event.

   The Button property of the e object contains information about which mouse button was pressed.

   If the Button value is 1, the left button was pressed, and the zooming code from the previous step will be executed. If the Button value is 2, the right button was pressed, and the code will call another method on the Map control, Pan.

Test your changes

1. Click the Start button in the Visual Studio toolbar.
2. With the left mouse button, click-drag a rectangle to zoom in.
3. With the right mouse button, click-drag to pan the map.



When you release the mouse, the map is redrawn at the new location.

Save the project

1. Click the Stop Debugging button in Visual Studio to return to design mode.
2. Click the File menu, then click Save All to save your project.

Adding a toolbar to the Form

The pan and zoom capability of your application is somewhat hidden from the user.

In this section you will create a toolbar with pan and zoom buttons. You will also add a FullExtent button.

Add a ToolBar and an ImageList

Visual Studio provides a ToolBar control that can be used in conjunction with an ImageList control, to display a collection of buttons at the top of a form.

You will add these controls to your form, and use them to control the actions which occur when the user interacts with the map.

1. Select the Form1 [Design] tab at the top of the main document window to select the Windows Forms Designer view of your form.

   Tip: You can display the Form Designer window of a form by right-clicking the form in the Solution Explorer, and selecting the View Designer option.

2. Open the Toolbox and make sure the Windows Forms tab is selected.

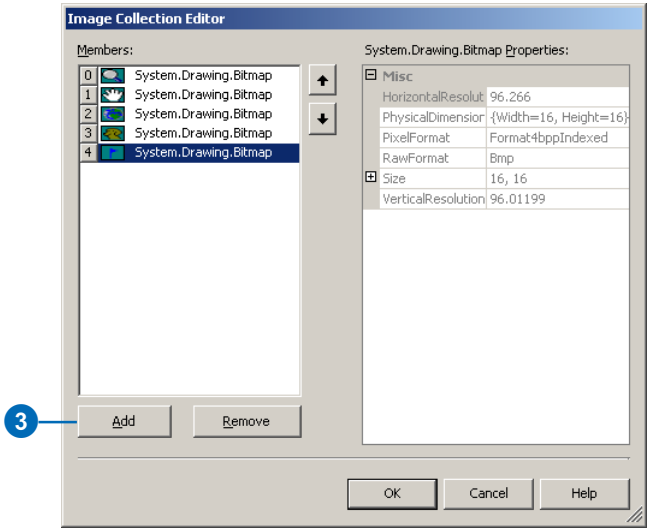
3. Double-click the ImageList in the Toolbox to add an ImageList control to the form.

   Note: The ImageList control is not visible at runtime. Visual Studio .NET places such controls in a separate area of the Forms Designer, called the component tray.

4. Double-click the ToolBar in the Toolbox to add a ToolBar control to the form, then reactivate the form by clicking somewhere on the form.

Add images to the ImageList control

1. Select ImageList1 in the component tray, shown below the Form.
2. In the Properties window, scroll down to find the Images property, then click the button next to it.
3. In the Image Collection Editor dialog box, click Add.
4. In the dialog box which appears, browse to the folder that contains the MapObjects sample bitmaps.
5. Click the Zoom.bmp file, then click Open.
6. Add the files Pan.bmp, Globe.bmp, Bex.bmp, and Pennant.bmp in the same manner.

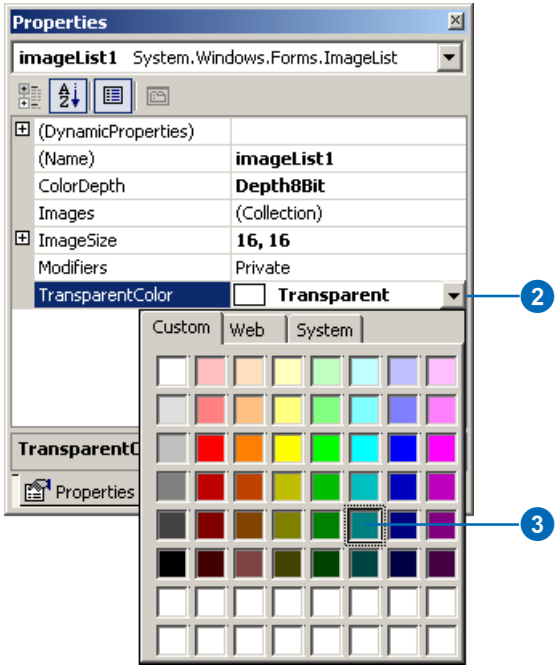


7. Click OK to dismiss the Image Collection Editor dialog box.

Set the TransparentColor of the ImageList

Setting the TransparentColor property of an ImageList control specifies a color that will act as a mask for any images contained by the control. The mask color will not be drawn, resulting in an image with a transparent background.

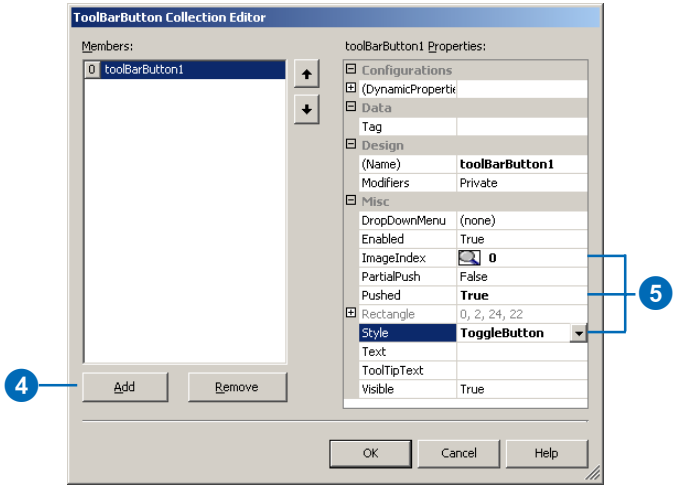
1. Click ImageList1 in the component tray to select it.
2. In the Properties window, click the pull down next to the TransparentColor property.
3. Click the Custom tab, and select the Teal color, as shown.



Add buttons to the ToolBar

You can associate the ToolBar control with an ImageList control to provide the graphic images for the buttons which you will add.

1. Return to the Form Designer view, and click the ToolBar control on the form, to select it.
2. In the Properties window, click the button next to the ImageList property, and select ImageList1.
3. Scroll back up to find the Buttons property, and click the pull down next to this property.
4. In the ToolBarButtons Collection Editor dialog box, click the Add button.



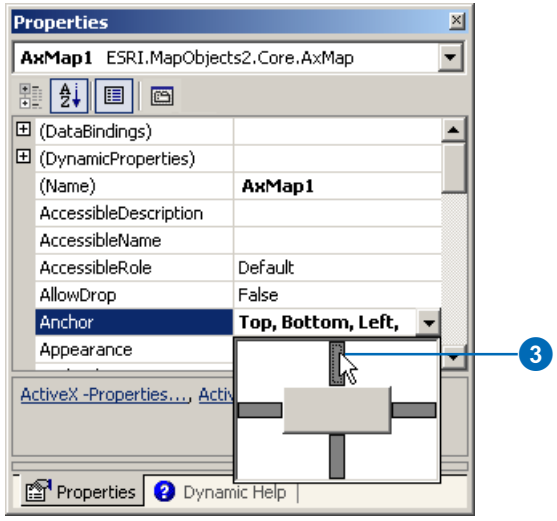
5. Set the buttons Style property to ToggleButton, its ImageIndex property to 0, and its Pushed property to True.
6. Add a second button, and set its Style to ToggleButton and it's ImageIndex to 1.

7. Add a third button, and set its ImageIndex to 2. Leave the Style as PushButton.
8. Click OK to dismiss the dialog and add the buttons to the ToolBar.

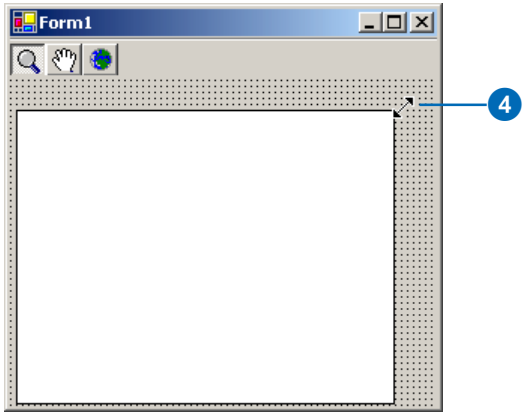
Resize the AxMap control

You may have noticed that the top of the AxMap control is now obscured by the ToolBar. You will change the properties of the AxMap control to avoid this conflict.

1. Click the AxMap control on the form to select it.
2. In the Properties window, click the pull-down next to the Dock property and select None.
3. Scroll upwards and click the pull down next to the Anchor property. Select the top, left, right and bottom bars, then press Enter to confirm the selection.



4. Return to the Form Designer, select the AxMap control, and resize it to so that it is not covered by the ToolBar.



Change the MouseDown event

You will now change the code you previously added to the Form, to make the pan and zoom functionality dependant on which button in the ToolBar is selected.

1. Select the Form1.cs tab at the top of the main document window to select the code window view of your form.
2. Scroll down to find the axMap1_MouseDownEvent procedure and edit the code as shown.

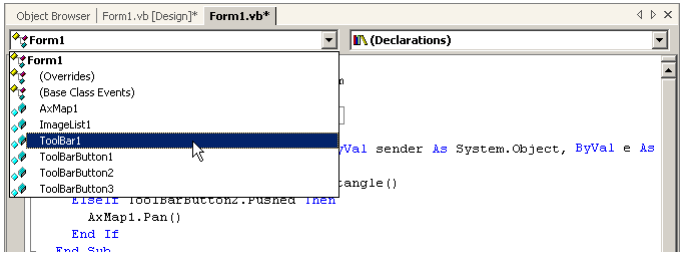
```
Private Sub AxMap1_MouseDownEvent(ByVal sender
...
   If ToolBarButton1.Pushed Then
     AxMap1.Extent = AxMap1.TrackRectangle()
   ElseIf ToolBarButton2.Pushed Then
     AxMap1.Pan()
   End If
End Sub
```

Add code to respond to clicking a ToolBar button

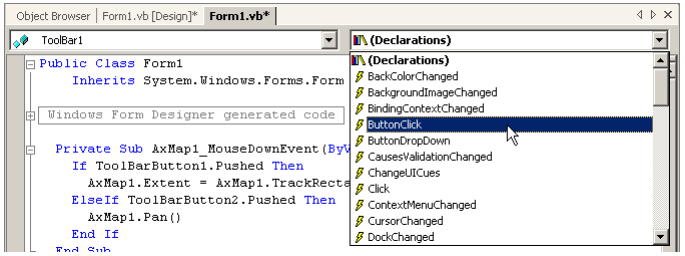
Your application now supports panning and zooming, but once the user has zoomed into the map, there is no way to get back to the full extent again.

You will complete the pan and zoom functionality by adding code to respond to the third ToolBarButton by zooming the map to its full extent. You will also ensure that only one button at a time may be selected on the ToolBar.

1. Return to the code window of the form.
2. Click the Class Name drop-down list (top left of the code window), and select ToolBar1.



3. In the Method Name drop-down list, select the ButtonClick event.



An event handler for the ButtonClick event on the ToolBar is added to the code window.

4. Add the following lines of code to the ToolBar1_ButtonClick procedure.

```
Private Sub ToolBar1_ButtonClick(ByVal sender _
As Object, ByVal e As System.Windows.Forms. _
ToolBarButtonClickEventArgs) _
Handles ToolBar1.ButtonClick
 If e.Button Is ToolBarButton3 Then
   AxMap1.Extent = AxMap1.FullExtent
 Else
   SetSelected(e.Button)
 End If
End Sub
```

This code specifies that for any button other than the FullExtent button, the SetSelected procedure is called. The FullExtent button is not a toggle button, so should not change the state of any other button.

5. Below the ButtonClick procedure, add the following function procedure.

```
Private Sub SetSelected(ByVal currBtn As _
ToolBarButton)
 Dim btn As ToolBarButton
 For Each btn In ToolBar1.Buttons
   If Not (btn Is currBtn) Then
     btn.pushed = False
   End If
 Next
End Sub
```

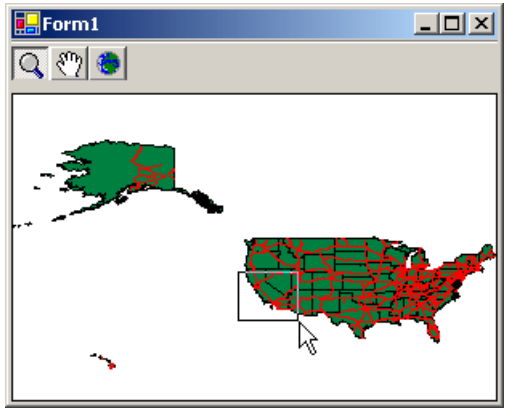
This procedure will ensure that only one tool on the toolbar at a time is selected.

Test your changes

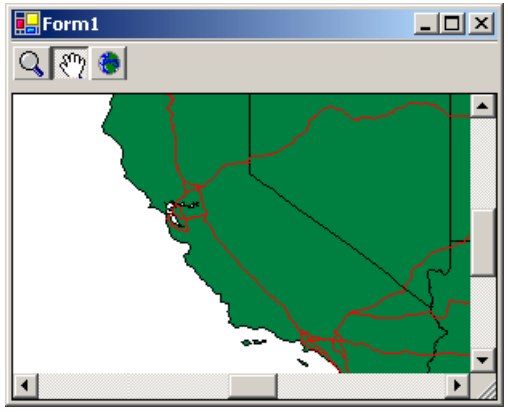
1. Click the Start button on the Visual Studio toolbar.

   The zoom button on your ToolBar should be activated by default when you start the application.

2. Click the map with the left mouse button and drag out a rectangle.
3. Release the mouse button to redraw the map.



4. Click the map with the right mouse button and drag to pan the map.



5. Release the mouse button to redraw the map.

   The application should behave the same as before, except the left/right mouse clicks are replaced by selecting the buttons on the ToolBar.
6. Click the Full Extent button to redraw the map at the full extent.

Save the project

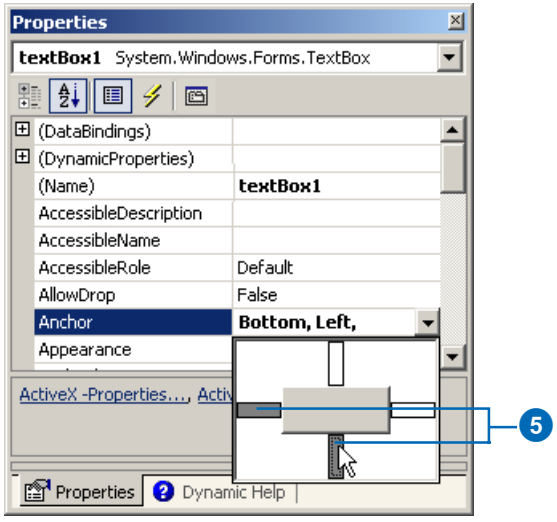
1. Click the Stop Debugging button in Visual Studio to return to design mode.
2. Click the File menu, then click Save All to save your project.

Creating a find tool

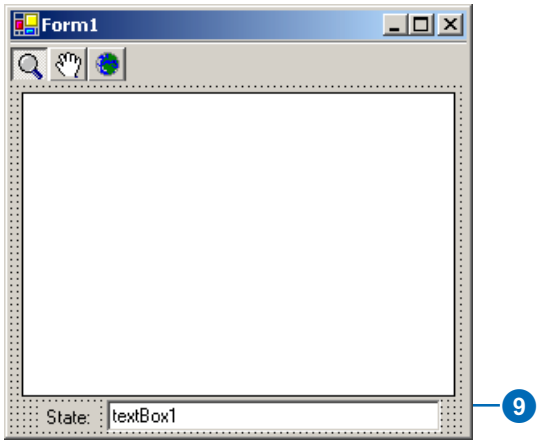
In this section you will add additional controls to your application to implement a simple function for locating a state by name.

Add controls to the form

1. Select the Form1 [Design] tab at the top of the main document window to select the Forms Designer view.
2. Open the Toolbox, make sure the Windows Forms tab is selected, then double-click the Label control entry in the Toolbox to add a label control to the form.
3. In the Properties window for the Label control, set the Text property of the Label control to be "State".
4. Scroll up in the Properties window to find the Anchor property and click the pull-down next to it.
5. Select only the bottom and left bars, then press Enter.



6. Double-click the TextBox in the Toolbox to add a TextBox to the form.
7. In the Properties window, clear the Text property of the TextBox.
8. Scroll up to find the Anchor property, click the pull down next to it and set the property equal to Bottom, Left, Right.
9. Reposition the Label and TextBox controls at the bottom of the Form, and resize the AxMap control so that it is not obscured by the new controls, as shown.

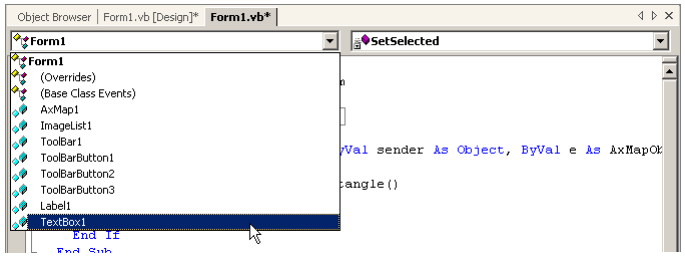


Attach code to the TextBox

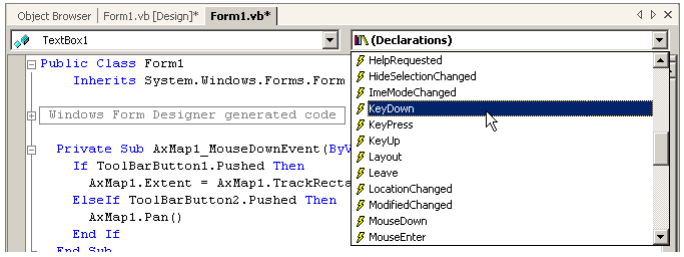
You will use the text the user types into the TextBox to perform a logical query.

You will add code to ensure the query is only performed when the user presses Enter in the TextBox.

1. Return to the code window of the form.
2. Click the Class Name drop-down list (top left of the code window), and select TextBox1.



3. In the Method Name drop-down list, select the KeyDown event.



An event handler for the KeyDown event on the TextBox is added to the code window.

4. Add the following lines of code to the TextBox1_KeyDown procedure.

```
Private Sub TextBox1_KeyDown(ByVal sender As _
Object, ByVal e As System.Windows.Forms. _
KeyEventArgs) Handles TextBox1.KeyDown
   If e.KeyCode = Keys.Return Then
     Dim exp As String = "STATE_NAME = '" + _
       TextBox1.Text + "'"
     Dim lyr As MapLayer = _
       AxMap1.Layers.Item("States")
     Dim recs As Recordset = _
       lyr.SearchExpression(exp)

     If Not recs.EOF Then
       Dim res As Polygon = _
         recs.Fields.Item("Shape").Value
       Dim ext As ESRI.MapObjects2.Core. _
         Rectangle = res.Extent
       ext.ScaleRectangle(2.0)
       AxMap1.Extent = ext
       AxMap1.CtlRefresh()
       AxMap1.FlashShape(res, 3)
     End If
   End If
 End Sub
```

Note that the Rectangle object is fully referenced using the full namespace. This is because the System.Drawing namespace, for which there is an imports statement at the top of the form, also has a Rectangle object, and therefore a full reference is required in this case to define the Type.

The code first builds a simple SQL query expression using the text in the TextBox control, then searches the States layer using the SearchExpression method. The result is a Recordset object.

If the value of the Recordset's EOF property is False, the code positions the Recordset on the first record that satisfies the search expression.

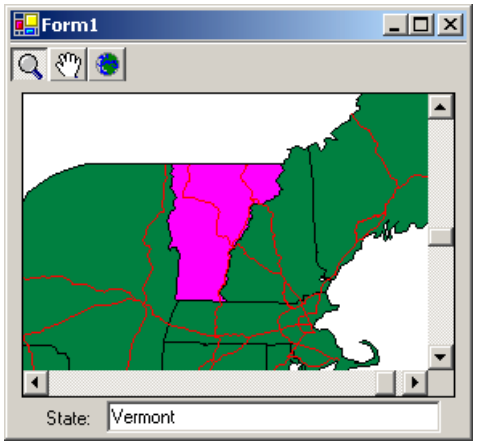
The code then gets the value of the Shape field for the first record. The code scales the Extent of the shape and then sets it to be the extent of the map.

Finally, the map is redrawn using the CtlRefresh method, and the shape is flashed three times. Note that the CtrlRefresh method is used to redraw the map. This is because the .NET host class, AxHost, has a method called Refresh too—the 'Ctrl' prefix is added to differentiate the method of the host class from the method of the underlying MapObjects Map class.

Note also that as you added the using statement in the previous section, MapObjects variables can be declared without the ESRI.MapObjects2.Core prefix, and have been written without this prefix throughout this document for brevity.

Test your changes

1. Click the Start button in the Visual Studio toolbar.
2. Type the name of a state, e.g. Vermont, into the TextBox, remembering the search is case-sensitive.
3. Press the Enter key.



The map zooms in to the selected state, and flashes the state.

4. Click the Stop Debugging button on the Visual Studio toolbar.
5. Click the File menu, then click Save All to save your project.

Displaying map layers based on scale

Your map currently appears the same, regardless of the scale at which it is being displayed.

In this section you will add a new layer to your map and add code that controls whether or not that layer is visible at a given time, depending on the current display scale.

Add another layer

1. Return to the Forms Designer window, and right-click on the AxMap control to display the context menu.
2. Choose ActiveX Properties to display the property sheet.
3. Click the Add button, and locate the folder where the sample data is stored.
4. Click the Counties.shp file, then click Open.
5. Click the Counties layer in the Layers list to select it.
6. Click the down arrow to move the Counties layer below the USHigh layer.
7. Click the Properties button, and in the Layer Properties dialog box, change the color of the Counties layer.
8. Click OK to dismiss the Layer Properties dialog.
9. Click OK to dismiss the property sheet.

If you run your application now you will notice that it displays every county in the United States.

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. Return to the code window of the form.
2. Click the Class Name drop-down list (top left of the code window), and select AxMap1.
3. In the Method Name drop-down list, select BeforeLayerDraw.

   An event handler for the BeforeLayerDraw on the Map control is added to the code window.
4. Add the following lines of code to the AxMap1_BeforeLayerDraw procedure.

```
Private Sub AxMap1_BeforeLayerDraw(ByVal _
sender As Object, ByVal e As _
BeforeLayerDrawEventArgs) _
Handles AxMap1.BeforeLayerDraw
 Dim lyr As MapLayer = _
   AxMap1.Layers.Item(e.index)
 Dim ratio As Double = AxMap1.Extent.Width / _
   (AxMap1.FullExtent.Width / 5)

 If lyr.Name.ToLower() = "counties" Then
   lyr.Visible = (ratio <= 1.0)
 ElseIf lyr.Name.ToLower() = "states" Then
   lyr.Visible = (ratio > 1.0)
 End If
End Sub
```

Note: The code changes the value of the visible property of each layer, based on the current extent of the map.

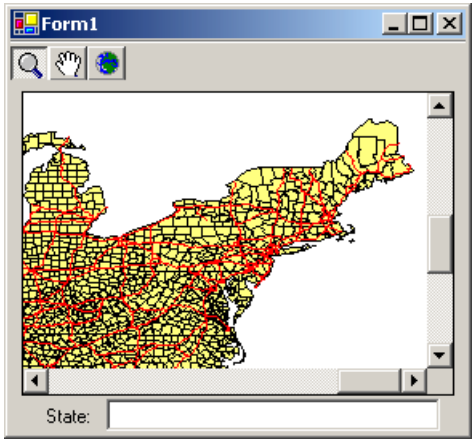
If the width of the current extent is less than or equal to one-fifth of the full extent of the map, then the counties will be visible and the states will be invisible.

Because this code executes in response to the BeforeLayerDraw event for each layer, the code changes the value of the Visible property before drawing occurs.

Test your changes

1. Click the Start button on the Visual Studio toolbar.
   The Counties layer is not visible.
2. Zoom into New England.

   The Counties layer becomes visible.



3. Click the FullExtent button

   The Counties layer is no longer visible.
4. Click the Stop Debugging button in Visual Studio to return to design mode.
5. Click the File menu, then click Save All to save your project.

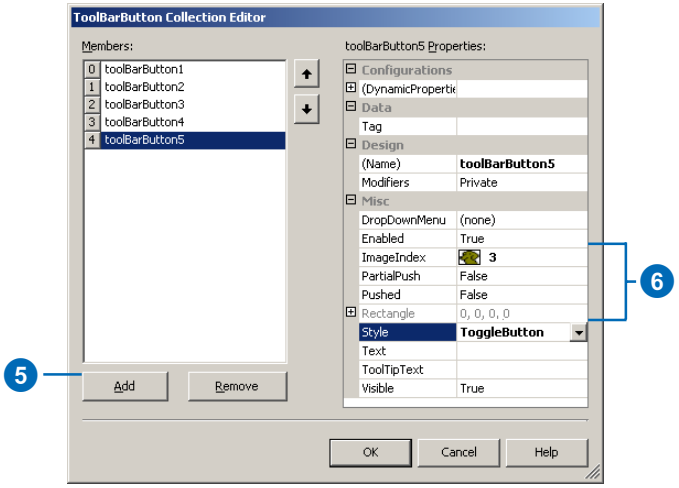
Adding a spatial query tool

In this section you will add a new tool to the toolbar 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 a query button to the toolbar

1. Return to the Form Designer view, and click the ToolBar to select it.
2. In the Properties window, click the Properties button, then click the button next to the Buttons property.
3. In the ToolBarButtons Collection Editor dialog box, click the Add button.
4. Set the new button's Style property to Separator.
5. Click the Add button again.



6. Set the new button's Style property to ToggleButton, and its ImageIndex property to 3.
7. Click OK to dismiss the ToolBarButton Editor dialog box, and add the buttons to the ToolBar.

Add a member variable to the form

1. Return to the code window of the form.
2. Scroll the code window to find the beginning of the declaration of the form class.
3. Add a member variable of type MapObjects Recordset, as shown below, to store the results of the spatial query.

```
Public Class Form1
 Inherits System.Windows.Forms.Form

 Private m_query As Recordset
 ...
```

The results of the spatial query must be stored in a member variable, as the variable is accessed from two different procedures. The spatial query is performed in response to the MouseDownEvent on the AxMap control. The results are displayed in the AfterLayerDraw event of the AxMap control.

Implement the query tool

You will now change the code you previously added to respond to the MouseDown event. It will now account for the new query tool being the current tool.

1. In the code window view of your form, scroll down to find the axMap1_MouseDownEvent procedure.

2. Edit the procedure as shown below.

```
Private Sub AxMap1_MouseDownEvent(ByVal sender
...
 If ToolBarButton1.Pushed Then
   ...
 ElseIf ToolBarButton5.Pushed Then
   Dim pt As ESRI.MapObjects2.Core.Point = _
     AxMap1.ToMapPoint(e.x, e.y)
   Dim highLayer As MapLayer = _
     AxMap1.Layers.Item("UShigh")
   Dim highRecs As Recordset = _
     highLayer.SearchByDistance(pt, _
     AxMap1.ToMapDistance(2), "")

   If highRecs.EOF Then
     m_query = Nothing
   Else
     Dim cntyLayer As MapLayer = _
       AxMap1.Layers.Item("Counties")
     Dim highLine As Line = _
       highRecs.Fields.Item("Shape").Value
     m_query = cntyLayer.SearchShape(highLine, _
       SearchMethodConstants. _
       moEdgeTouchOrAreaIntersect, "")
   End If
   AxMap1.CtlRefresh()
 End If
End Sub
```

Note another use of a fully qualified variable, necessary due to the Imports statement you added previously—the System.Drawing namespace also has a Point object.

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 code obtains the point by converting the x and y coordinates of the event from control units, to map units.

If the first search is successful, the highway found is used as the input to the second search, performed on the Counties layer. The result of the second search is stored in the member variable m_query.

Draw the results

1. Return to the code window of the form.
2. Click the Class Name drop-down list (top left of the code window), and select AxMap1.
3. In the Method Name drop-down list, select the AfterLayerDraw.

   An event handler for the AfterLayerDraw event on the Map control is added to the code window.

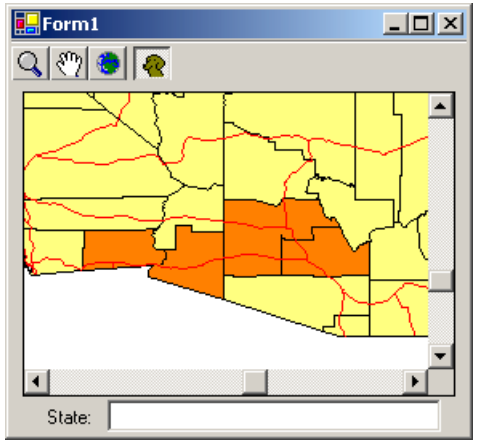
4. Add the following lines of code to the AxMap1_AfterLayerDraw procedure.

```
Private Sub AxMap1_AfterLayerDraw(ByVal _
sender As Object, ByVal e As _
AfterLayerDrawEventArgs) _
Handles AxMap1.AfterLayerDraw
 Dim lyr As MapLayer = _
   AxMap1.Layers.Item(e.index)
 If lyr.Name.ToLower() = "counties" Then
   If Not (m_query Is Nothing) Then
     If Not (m_query.EOF) Then
       Dim sym As Symbol = New SymbolClass()
       sym.SymbolType = _
         SymbolTypeConstants.moFillSymbol
       sym.Color = Convert.ToUInt32( _
         ColorConstants.moOrange)
       AxMap1.DrawShape(m_query, sym)
     End If
   End If
 End If
End Sub
```

Test your changes

1. Click the Start button on the Visual Studio toolbar, and zoom into an area so that the Counties layer becomes visible.
2. Click the spatial query tool, then click on a highway.

   The counties intersecting the highway are highlighted in orange.



3. Click the Stop Debugging button in Visual Studio to return to design mode.
4. Click the File menu, then click Save All to save your project.

Statistical mapping

Currently the map displays no information about the layers it contains.

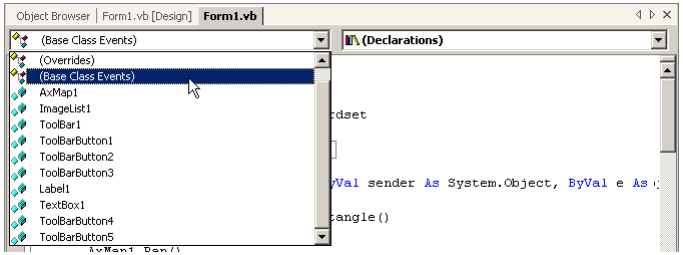
In this section you will modify your application so that it draws the Counties layer using the underlying attribute information.

Attach a renderer to the Counties layer

You will use the MapObjects ClassBreaksRenderer to represent continuous data—in this case, the number of mobile homes per capita by county, stored in the 'Mobilehome' field.

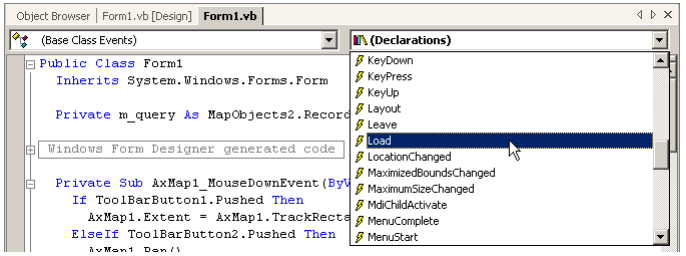
Also, the colors of the ClassBreaksRenderer will be ordered by this attribute.

1. Return to the code window of the form, click the Class Name drop-down list (top left of the code window), and select Base Class Events.



   Base Class Events are the public events on the Form class, from which your form class inherits.

2. In the Method Name drop-down, select the Load event.



   An event handler for the Load event on the Form is added to the code window.

3. Add the following lines of code to the Form_Load procedure.

```
Private Sub Form1_Load(ByVal sender As Object,
ByVal e As System.EventArgs) Handles MyBase.Load
 SetupCounties()
 SetupStates()
End Sub
```

4. Now add the procedure below to the code window, just below the Form1_Load procedure.

```
Private Sub SetupCounties()
 Dim ctLyr As MapLayer = _
   AxMap1.Layers.Item("Counties")
 Dim ctRnd As ClassBreaksRenderer _
   = New ClassBreaksRendererClass()
 ctLyr.Renderer = ctRnd
 ctRnd.Field = "MOBILEHOME"

 Dim stats As Statistics = ctLyr. _
   Records.CalculateStatistics("MOBILEHOME")
 Dim breakVal As Double = _
   stats.Mean - (stats.StdDev * 3)
```

```
Dim i As Integer
For i = 0 To 6
  If (breakVal >= stats.Min And _
    breakVal <= stats.Max) Then
    Dim breakCount As Integer = _
      ctRnd.BreakCount
    ctRnd.BreakCount = breakCount + 1
    ctRnd.Break(breakCount) = breakVal
  End If
  breakVal = breakVal + stats.StdDev
Next
cntyRnd.RampColors(Convert.ToUInt32 _
  (ColorConstants.moLimeGreen), Convert. _
  ToUInt32(ColorConstants.moRed))
End Sub
```

This procedure will render the Counties MapLayer according to mobile home statistics.

A Statistics object is used to set the class breaks, based on the standard deviation value. Only breaks inside the actual range of values stored in the attribute are added to the Renderer.

Attach a renderer to the States layer

1. Add the procedure below to the code window, just below the SetupCounties procedure you just added.

```
Private Sub SetupStates()
 Dim stLyr As MapLayer = _
   AxMap1.Layers.Item("States")
 Dim stRnd As ValueMapRenderer = _
   New ValueMapRendererClass()

 Dim stRecs As Recordset = stLyr.Records
 stLyr.Renderer = stRnd
 stRnd.Field = "SUB_REGION"
 Dim regions As Strings = New StringsClass()
 regions.Unique = True
 While Not stRecs.EOF
   regions.Add(stRecs.Fields. _
     Item("SUB_REGION").ValueAsString)
   stRecs.MoveNext()
 End While
 stRnd.ValueCount = regions.Count

 Dim i As Integer
 For i = 0 To regions.Count - 1
   stRnd.Value(i) = regions.Item(i)
 Next
End Sub
```

This procedure will render the States MapLayer according to region. The colors of each different region will be random.

The code uses a Strings object to collect the unique values of the SUB_REGION attribute. These values are added to the ValueMapRenderer.

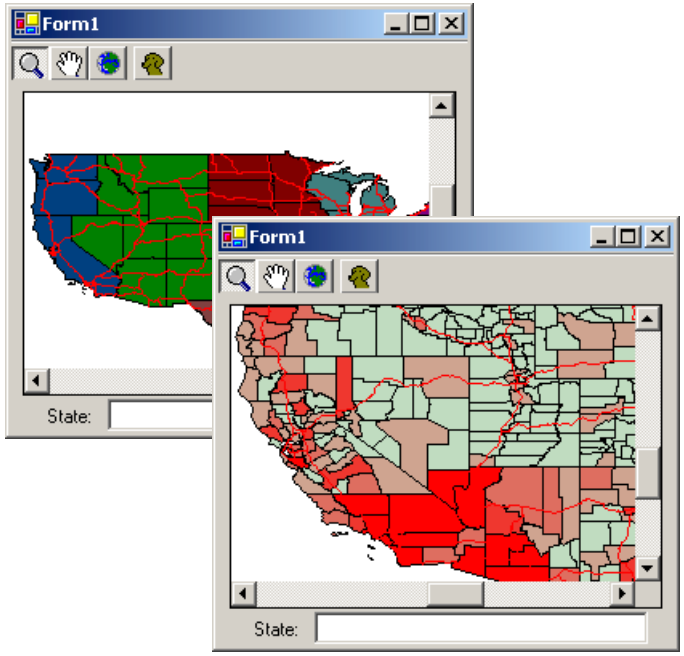
Test your changes

1. Click the Start button on the Visual Studio toolbar and look at the States layer.

   Instead of drawing each state the same color, the application now draws states colored according to the sub region the state belongs to.

2. Zoom into an area so that the Counties layer becomes visible.

   Instead of drawing each county the same color, the application now draws the counties in different colors, depending on the underlying attribute values. The states colored red indicate a higher number of mobile homes.



3. Click the Stop Debugging button in Visual Studio to return to design mode.
4. Click the File menu, then click Save All.

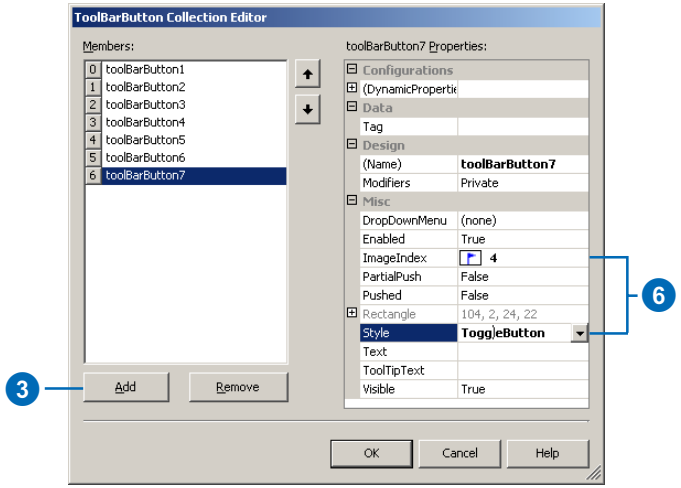
Event tracking

Some applications must 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 update vehicle locations frequently 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 to simulate this requirement.

Add an event tool to your application's toolbar

1. Return to the Form Designer view, and click the ToolBar to select it.
2. In the Properties window, select the Properties button, then click the button next to the Buttons property.
3. In the ToolBarButtons Collection Editor dialog box, click the Add button.



4. Set the new button's Style property to Separator.
5. Click the Add button again.
6. Set the new buttons Style property to ToggleButton, and its ImageIndex property to 4.
7. Click OK to dismiss the dialog and add the buttons to the ToolBar.

Implement the event tool

You will again change the code which responds to the MouseDown event, to account for the new event tool.

1. Return to the code window view of your form.
2. Scroll up to find the AxMap1_MouseDownEvent procedure, and edit the code as shown.

```
Private Sub AxMap1_MouseDownEvent(ByVal sender
...
 If ToolBarButton1.Pushed Then
   ...
 ElseIf ToolBarButton7.Pushed Then
   Dim pt As ESRI.MapObjects2.Core.Point = _
     AxMap1.ToMapPoint(e.x, e.y)
   Dim sym As Symbol = _
     AxMap1.TrackingLayer.Symbol(0)
   sym.SymbolType = _
     SymbolTypeConstants.moPointSymbol
   sym.Size = 5
   sym.Style = _
     MarkerStyleConstants.moTriangleMarker
   AxMap1.TrackingLayer.AddEvent(pt, 0)
 End If
End Sub
```

Test your changes

1. Click the Start button on the Visual Studio toolbar and zoom in a little.
2. Click the event tool, then click in the map a few times to add a few events.
3. Click the Stop Debugging button in Visual Studio to return to design mode.

Add a timer to your form

To trigger the movement of the events, a Timer control will be used. The events will be moved randomly around the map, at intervals triggered by the Timer.

1. Select the Form1 [Design] tab at the top of the main document window to select the Forms Designer view of your form.
2. Open the Toolbox and make sure the Windows Forms tab is selected.
3. Double-click the Timer in the Toolbox to add a Timer control to the form.
4. Return to the Forms Designer window, and make sure the Timer is selected.

   Note: The Timer control is also invisible at run time, and is therefore shown in the component tray.
5. In the Properties window, set the Interval property to 500.
6. Return to the code window of the form, click the Class Name drop-down list, and select Timer1.
7. In the Method Name drop-down, select the Tick event.

   An event handler for the Tick event on the Timer is added to the code window.
8. Add the following lines of code to the Timer1_Tick procedure.

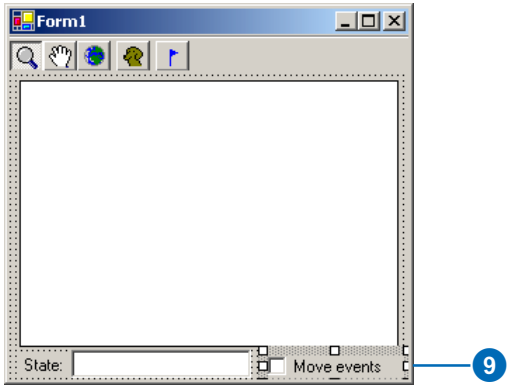
```
Private Sub Timer1_Tick(ByVal sender As Object, _
ByVal e As System.EventArgs) Handles Timer1.Tick
 Dim evCount As Integer = _
   AxMap1.TrackingLayer.EventCount
 Dim dist As Double = AxMap1.Extent.Width / 25
 Dim rnd As System.Random = New Random()

 Dim i As Integer
 For i = 0 To evCount - 1
   Dim gEvt As GeoEvent = _
     AxMap1.TrackingLayer.Event(i)
   gEvt.Move((rnd.NextDouble() - 0.5) * dist, _
     (rnd.NextDouble() - 0.5) * dist)
 Next
End Sub
```

Add a CheckBox to your form

To control the timer, you will add a CheckBox control.

1. Return to the Forms Designer view, and double-click the CheckBox button in the Toolbox to add a CheckBox to the form.
2. Move the CheckBox to as shown below.



3. In the Properties window, click the Properties button.
4. Scroll down to find the Text property of the CheckBox, and change its value to Move Events.
5. Scroll back up to find the Anchor property, and change its value to Bottom, Right.
6. Return to the code window of the form, click the Class Name drop-down list, and select ToolBar1.
7. In the Method Name drop-down list, select the ButtonClick event.

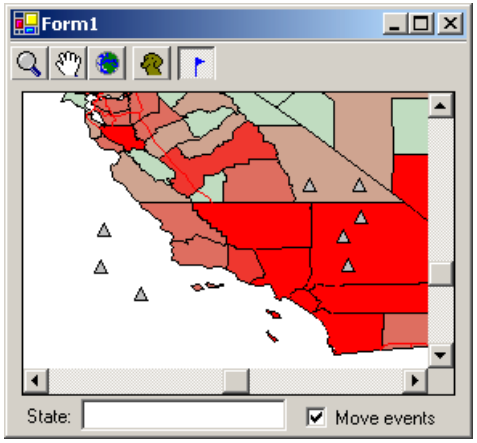
   An event handler for the ButtonClick event on the ToolBar is added to the code window.
8. Add code to the CheckBox1_CheckedChanged procedure as shown.

```
Private Sub CheckBox1_CheckedChanged(ByVal _
sender As Object, ByVal e As System.EventArgs) _
Handles CheckBox1.CheckedChanged
 If CheckBox1.Checked Then
   Timer1.Start()
 Else
   Timer1.Stop()
 End If
End Sub
```

Test your changes

1. Click the Start button on the Visual Studio toolbar.
2. Click and drag a rectangle on the map to zoom in to the map a little.
3. Click the event tool on the ToolBar, then click a few times on the map, to add a few events.
4. Click the Move events check box to select it.

   The events start moving randomly on top of the map.



5. Click the check box again to stop the events.

Save the project

1. Click the Stop Debugging button in Visual Studio to return to design mode.
2. Click the File menu, then click Save All to save your project.

Working with DataConnection objects

In each of the previous sections, you have worked with MapLayer objects that were specified interactively, using the AxMap control's property sheet.

In this section, you will add code to your application that creates MapLayer objects programmatically, using a DataConnection object. First you will remove the existing layers from the map, so they are not duplicated when the application is run.

Remove the existing layers

1. Return to the Form Designer view of your form.
2. Right-click the mouse on the AxMap control to display the context menu.
3. Choose Active X Properties to display the MapObjects property sheet.
4. Click on the UShigh layer in the Layers list, then click Remove to delete the highways layer.
5. Remove the Counties and States layers in the same manner.
6. Click OK to dismiss the MapObjects property sheet.

Add a procedure that will initialize the map

1. Return to the code window of the form.
2. Scroll down the code window, and add the following procedure below the last existing procedure, before the end of the class.

```
Private Sub InitializeMap()
 Dim dc As DataConnection = _
   New DataConnectionClass()
 dc.Database = "C:\Program Files\ESRI" + _
   "\MapObjects2\Samples\Data\USA"

 If dc.Connect() Then
   Dim layer As MapLayer = New MapLayerClass()
   layer.GeoDataset = _
     dc.FindGeoDataset("States")
   AxMap1.Layers.Add(layer)

   layer = New MapLayerClass()
   layer.GeoDataset = _
     dc.FindGeoDataset("Counties")
   AxMap1.Layers.Add(layer)

   layer = New MapLayerClass()
   layer.GeoDataset = _
     dc.FindGeoDataset("USHigh")
   layer.Symbol.Color = Convert.ToUInt32( _
     ColorConstants.moRed)
   AxMap1.Layers.Add(layer)

 Else
   MessageBox.Show( _
     "The data could not be located.")
   Application.Exit()
 End If

End Sub
```

3. Add a call to your procedure at the beginning of the Form1_Load procedure.

```
Private Sub Form1_Load(ByVal sender As _
Object, ByVal e As System.EventArgs) _
Handles MyBase.Load
 InitializeMap()
 SetupCounties()
 SetupStates()
End Sub
```

Test your changes

1. Click the Start button in the Visual Studio toolbar.

   The map should appear as before, but the colors of the MapLayers are specified in the code, and may be different to the colors you selected in the Layer Properties dialog box.

2. Click the Stop Debugging button in Visual Studio to return to design mode.

3. Click the File menu, then click Save All to save your project.

Working with ImageLayer objects

In each of the previous sections, you have worked with MapLayer objects based upon vector data sources.

In this section, you will see how to add layers to your map that are based on images. MapObjects allows you to use a wide range of image types, including such common image types as windows bitmaps (.bmp), tagged image file format (.tiff), and CompuServe bitmaps (.gif).

For a full, up-to-date list of the image formats you can use in a map, see "Supported Image Formats" in the "Using MapObjects" section of the online help.

Adding an ImageLayer in code

Previously, you added a MapLayer programmatically, using the DataConnection object.

Now you will add an ImageLayer programmatically. To do this, you use the File property of the ImageLayer object.

1. Return to the code window of the Form.
2. Scroll down the code window to find the InitializeMap procedure, and add edit it as shown below, to add an ImageLayer after adding the other MapLayers.

```
Private Sub InitializeMap()
 ...
 End If
 Dim imgLayer As ImageLayer = _
   New ImageLayerClass()
 imgLayer.File = "C:\Program Files\ESRI\" + _
   "MapObjects2\Samples\Data" + _
   "\Washington\Wash.bmp"
 AxMap1.Layers.Add(imgLayer)
End Sub
```

Set the coordinate system of the map

The USA data you added previously uses a projected coordinate system to determine how the data in these layers is projected to a flat screen.

ImageLayers however cannot be projected by MapObjects. If you run your application at this point, you will find the MapLayers you added previously do not align correctly with the ImageLayer which you added in the last step. You can however, project the data from the existing layers to match up with the ImageLayer.

You will set the coordinate system of the map to be the same as the coordinate system of the Washington ImageLayer. This will re-project the USA MapLayer data on the fly, to correctly align with the ImageLayer.

There is a projection (.prj) file stored in the Washington directory which you can use to create a coordinate system object applicable to the image.

For more information about coordinate systems and projection of data, see "About projections and coordinate systems" in the "Using MapObjects" section of the online help.

1. In the InitializeMap procedure, add the lines of code as shown to the bottom of the procedure.

   This procedure will create a MapObjects projected coordinate system (a ProjCoordSys object), and apply it to the map.

```
Private Sub InitializeMap()
 ...
 axMap1.Layers.Add (imgLayer)

 dc.Disconnect()
 dc.Database = "C:\Program Files\ESRI\" + _
   "MapObjects2\Samples\Data\Washington"
 If dc.Connect() Then
   Dim pcs As ProjCoordSys = _
     New ProjCoordSysClass()
   pcs = dc.FindCoordinateSystem("Roads.prj")
   AxMap1.CoordinateSystem = pcs
 Else
   MessageBox.Show(_
     "Could not project the MapLayers.", _
     "Display Warning")
 End If
End Sub
```

Test your changes

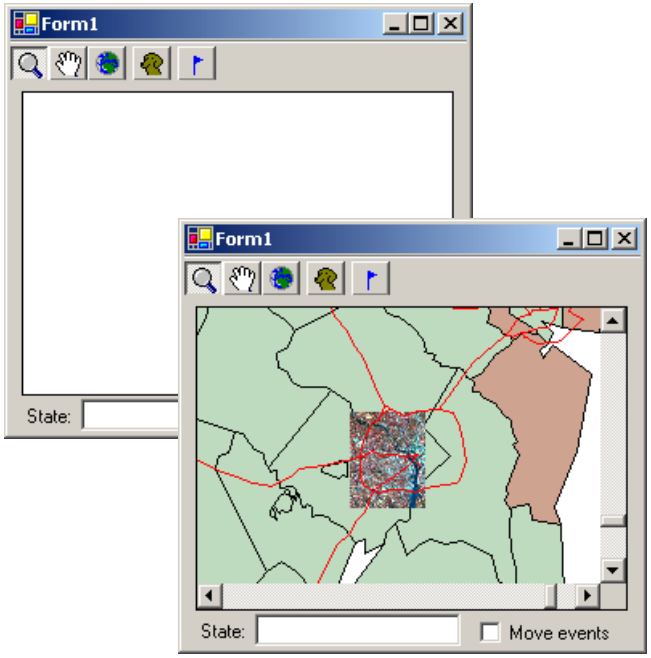
1. Click the Start button in the Visual Studio toolbar.

   The map should appear, but this time is reprojected to the new coordinate system.

   Try zooming in to find the Washington ImageLayer.

   You might like to add your own tool to locate the correct extent (try using the Extent property of the ImageLayer).

   You could also rearrange the order of the MapLayers to display the US Highways over the top of the ImageLayer.



2. Click the Stop Debugging button in Visual Studio to return to design mode.
3. Click the File menu, then click Save All to save your project.