MapMatrix NetGIS Application Objects Users Guide

Version 2.3

CONTENTS

1 Overview of MapMatrix NetGIS

MapMatrix NetGIS is a technology that has been designed to allow mapping and GIS information to be quickly and easily integrated into software applications. Any forms or browser based, internet, intranet or local application can utilize the MapMatrix NetGIS technology. NetGIS products sets the new standard for reducing total cost of creation, ownership and project lifecycle span for web based GIS enabled applications.

1.1 MapMatrix NetGIS Features

MapMatrix NetGIS is provides GIS information by distributing data from MapServers via Web Services to applications that have been GIS enabled utilizing the NetGIS application objects. Many GIS features are provided including:

- Display map data from a variety of data formats.
- Add custom data formats with the NetGIS Custom Data Adapters.
- Very fast display capabilities with many window and pan commands.
- Display imagery from a variety of data formats including JPEG, MrSid, Tiff, etc.
- Perform Geocoding.
- Create thematic displays.
- Allow the selection of GIS objects and retrieve tabular information for the graphical feature.
- Join disconnected databases. This allows for remote clients to apply database information that is stored separate from the GIS data.
- Perform buffer zone analysis.
- View GIS data from multiple NetGIS MapServers. This allows the application to view information from any NetGIS MapServer and view it as if it were from one source.
- Integrate GIS into existing applications without requiring a total rewrite of the application.

MapMatrix NetGIS is a very powerful system that provides the distribution of GIS applications and data in a fast and flexible fashion.

1.2 Operating System Requirements

Before you install the MapMatrix NetGIS Application Objects or WebControls your machine must meet the following requirements.

- You must be running a version of Microsoft Windows compatible with the .NET Framework and ASP.NET. At the time of this writing you can use:

- Windows NT 4.0 (with Service Pack 4 or greater installed)
- Windows 2000 (all versions supported)
- Windows XP (all versions supported)

- You must have Microsoft IIS (Internet Information Server) installed.

- You must have the Microsoft .NET Framework SDK installed

  Skip this step if you have Visual Studio.NET installed.
  If not you can visit the Microsoft .NET Framework SDK download page. At the time of this writing the .NET Framework SDK is available for download at the URL below.

  http://www.microsoft.com/downloads/details.aspx?FamilyID=9b3a2ca6-3647-4070-9f41-a333c6b9181d&DisplayLang=en

- To run the mock Service Request sample application you must have MDAC 2.6 or greater installed. At the time of this writing the latest version of MDAC can be downloaded from the URL below.

  http://www.microsoft.com/data/

1.3 The MapMatrix Architecture

The MapMatrix NetGIS Application Development System is based upon the latest in multi-tier application development architecture. MapMatrix applications are built on a true client/server architecture. The MapMatrix MapServer creates and delivers maps to client applications. The client applications are built with the MapMatrix NetGIS application products. Communication to and from the MapServer and the Client applications is performed through the MapMatrix NetGIS Web Services. Each of these are described below.

1.3.1 NetGIS MapServer

The MapServer receives and processes requests from the GIS Web Service (made upon request by SDK method calls). It is responsible for drawing maps and processing queries for attribute data. All GIS information is generated from a NetGIS MapServer. The server can be accessed through a local area network or via the Internet.

1.3.2 NetGIS MapServer Custom Data Adapters

The NetGIS MapServer reads GIS data from external sources by utilizing a specialized technique called GIS Data Late Binding. The implementation of this technique is accomplished with NetGIS Custom Data Adapters. Custom Data Adapters are software libraries that are built to read data from different data formats. If it is possible to read data through any vendor API or with custom programming, the data format can be supported and distributed with the NetGIS MapServer. These can be custom built by anyone and attached to a MapServer. This allows anyone to build a Custom Data Adapter for any format and use this GIS data with a NetGIS MapServer.

1.3.3 NetGIS Web Services

The NetGIS Web Service acts as a broker between the NetGIS MapServer and the NetGIS application objects. Web Services are easily made available across joined and disjoined networks with low risk to security and reduced maintenance requirements. The NetGIS WebService is an XML web service that provides the means for software to connect with other software applications. Any application that supports Web protocols such as HTTP, XML and SOAP can communicate with the NetGIS WebService and apply information created from a NetGIS MapServer.

1.3.4 NetGIS Application Objects

The NetGIS Application Objects supply software developers with a set of tools that provide a simple method for interacting with a NetGIS MapServer. The

NetGIS Objects provides this capability with an intuitive structure and command set. Communication from the NetGIS Objects with a MapServer is via the NetGIS Web Services. All GIS commands and queries and processed through this method.

1.3.5 NetGIS WebControls

The NetGIS WebControls builds upon the NetGIS Application Objects to provide an ASP.NET and Visual Studio.NET friendly set of components for web programmers. These controls simply many of the tasks that Web developers must face when developing applications such as state management and event handling. The NetGIS WebControls also simplify the management of retrieving GIS information from multiple MapServers.

Use of the NetGIS WebControls require a license to the NetGIS Application Objects (except for the trial version).

2 Installing and Configuring NetGIS Products

2.1 Licensing the NetGIS Products

The trial versions of the software do not require a license. For all runtime versions of the MapMatrix NetGIS products are licensed separately, for more information on these and other products are licensed please visit our website at http://www.GatewayHorizons.com or call use at 281-312-5600.

2.2 Obtaining a License Key

The trial version of the software does **not** require a license key. The trial version will only work with the Gateway Horizons GIS Web Services (it only connects to the NetGIS WebServices at www.GatewayHorizons.com). If a license is purchased the software can communicate with any MapMatrix NetGIS Web Service.

Before you install the runtime versions of the MapMatrix NetGIS software you must have a valid license key. If you are evaluating MapMatrix on trial basis you will need to go to the Gateway Horizons website at http://www.GatewayHorizons.com and fill out a new user registration form.

2.3 Installing the NetGIS Software

The NetGIS Application Objects and WebControls make use of MSI (Microsoft Installer Services). As part of this service installed packages will be audited by Windows and can be removed using Add/Remove programs from Control Panel. When you run NetGIS Setup on screen instructions will explain each step. If you experience problems with install feel free to contact Gateway Horizons support for assistance.

Upon a successful install, you will have a new program group appear on your “All Programs” menu called “NetGIS Application Objects” or “NetGIS WebControls”. Inside of these groups you will find shortcuts to this documentation, license manager control panel and sample applications.

3 The NetGIS Application Objects

The MapMatrix NetGIS Application Objects are contained in a .Net Class Library and have been designed to interact with a NetGIS MapServer and facilitate the development of distributed GIS Applications.

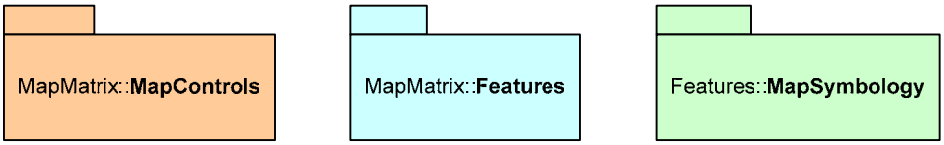
The Application Objects provides Web developers with a set of objects that can plug into Visual Studio.NET, WebMatrix or any .NET capable visual IDE. NetGIS Products set the new standard for reducing total cost of creation, ownership and project lifecycle span for web based GIS enabled applications.

These powerful objects can simplify GIS application development. With these products accessing data in a MapMatrix MapServer, any application can become GIS capable.

3.1 How to Use the Application Objects

3.1.1 The Application Objects

The MapMatrix NetGIS Application Objects are contained primarily in three different Class Libraries. The MapControls, Features and MapSymbology libraries.



The MapControls library contains classes that provide communication and command capabilities to NetGIS MapServers. The Features library contains classes that define GIS Layers and Features, and the MapSymbology library provides definitions for manipulating the display symbology of GIS features. There are other class libraries within the Application objects, however, these libraries are the ones that are primarily used when creating GIS Applications. All of these libraries work together and are referred to collectively as the NetGIS Application Objects.

3.1.2 Steps in creating a GIS Enabled Application

An application can become GIS enabled by making use of the NetGIS Application Objects. There are several simple steps to follow:

1. Create a control in your application to hold an image output. You can use any control that can hold an image. A PictureBox is a good control to use as you can react to a variety of mouse events with this control.

2. Add reference to the MapMatrix Class libraries in your development project.
3. Create a Command object. This is the primary object through which the GIS environment is controlled.
4. When the application starts open a connection to a MapServer with the Command object. The Command object has a Connection object which is used to create a communication channel to a MapMatrix NetGIS WebService. The WebService is the communication bridge to and from a MapServer.
5. Configure the operating environment for you application. This includes setting the GIS Image size, adding GIS layers and setting the display symbology of the layers.
6. Add controls to your application that perform GIS functions such as Windowing, selection of elements, etc.
7. Place in the code behind each of the controls the appropriate command for the desired function.

The walk-through application in this document demonstrates each of these steps.

3.2 Opening a Connection to a MapServer

A connection to a NetGIS MapServer must be established in order to perform GIS functions with an application. A connection to a specific MapServer is created only once for the application. The connection stays open until the application is terminated.

When a connection to a MapServer is created it is done with the Connection object. The Connection object must be supplied with basic information that the WebService requires to create a new session in the MapServer. This information includes user authentication information and what basic server configuration information to use in the new GIS session.

Methods such as OpenConnection and Close can be explicitly invoked through the Connection object or can be implicitly called with the Command object. Use of the Command can simplify the creation of a new session as will be seen in the walk-through example and shown below.

C#

```
private MapMatrix.MapControls.Command cmd;

// create connection to map server
cmd = new MapMatrix.MapControls.Command(564,431,
      "www.gatewayhorizons.com", "MapMatrixWS", "MunicipalExample",
       "", "", true);
```

VB

```
Dim cmd As MapMatrix.MapControls.Command
```

```
' create connection to map server
cmd = New MapMatrix.MapControls.Command(564, 431, _
      "www.gatewayhorizons.com", "MapMatrixWS", "MunicipalExample", _
      "", "", True)
```

When the code above is placed inside of an application a new connection is established with the Gateway Horizons MapServer utilizing the MapMatrixWS WebService and loading configuration information for the MunicipalExample. For the sample no username or password are required and the connection is opened at the time this call is made (this is the final boolean parameter).

In the above sample code the ***cmd*** object should be declared where it has class scope as it will be used throughout the application. The constructor should be called only once because once a connection has been established the GIS session exists for the duration of the application. In a Windows Forms application a good location to place this code would be in the Form_Load event.

3.3 Adding Layers to an Application

With any GIS application a group of layers (also know as themes) need to be added to the application. There are two collections that identify the set of layers that are available for use or are currently being used within the application.

3.3.1 AvailableLayers Collection

The Application Objects have been made aware of layers that reside within a MapServer. This information is loaded when a connection is made (see the section above on Opening a Connection to a MapServer). The parameters that are supplied to the MapServer define what configuration information is to be loaded for the requested connection.

The layers that exist within the MapServer are loaded into a collection called AvailableLayers. This collection is a property or the Connection Object. This collection can be referenced with the following code.

C#

```
private MapMatrix.MapControls.LayerList AllLayers;

// cmd represents an open connection (See the sample code above)
AllLayers = cmd.Connection.AvailableLayers;
```

VB

```
Dim AllLayers As MapMatrix.MapControls. LayerList

' cmd represents an open connection (See the sample code above)
AllLayers = cmd.Connection.AvailableLayers
```

This collection contains a list of `MapMatrix.Features.Layer` objects that describe all of the layers that are available through this connection (on the MapServer). This is static list and no manipulation of the objects should be performed.

3.3.2 LocalLayers Collection

All layers that have been identified for use during the application are contained within the LocalLayers collection. This collection is a property of the Command object. To load a layer for use in the application a copy is made of a Layer object in the Connection object and placed into the LocalLayers collection. A method has been provided in the Command object to facilitate this process.

C#

```
// cmd represents an open connection (See the sample code above)
Layer ly = cmd.GetAvailableLayer("Roads");
```

VB

```
' cmd represents an open connection (See the sample code above)
Dim ly As Layer = cmd.GetAvailableLayer("Roads")
```

The GetAvailableLayer method returns a Clone of the requested layer. The parameter is the LayerId of a layer that exists in the AvailableLayers collection. The LayerId is the key (unique name) of the layer as defined in the configuration files in the MapServer. If the layer is not found an exception is thrown.

3.3.3 Displaying a Map

Once a connection has been established with a MapServer and GIS layers have been added to the session, a map can be displayed. This is done with the `UpdateMap` method on the Command object. The UpdateMap method requests a new image from the MapServer. The image is located at the URL or file Path in the `MapImageLocation` property of the command object. The following code shows a complete simple example of how to connect to a MapServer, add layers, and display a map on a PictureBox control in a Windows Form application.

C#

```
private MapMatrix.MapControls.Command cmd;

private void Form1_Load(object sender, System.EventArgs e)
{
// create connection to map server
cmd = new MapMatrix.MapControls.Command(564,431,
      "www.gatewayhorizons.com", "MapMatrixWS", "MunicipalExample",
       "", "", true);

// Add layers
cmd.Layers.Add(cmd.GetAvailableLayer("Roads"));
cmd.Layers.Add(cmd.GetAvailableLayer("Aerial Photography"));
```

```
cmd.Layers.Add(cmd.GetAvailableLayer("Parcels"));
// Get a map from the MapServer
cmd.UpdateMap();

// Bind the image to a PictureBox control called MapImage
MapImage.Image = Utilities.BitmapFromUrl(cmd.MapImageLocation);

}
```

VB

```
Dim cmd As MapMatrix.MapControls.Command

Private Sub Form1_Load(ByVal sender As System.Object, _
      ByVal e As System.EventArgs) Handles MyBase.Load

' create connection to map server
cmd = New MapMatrix.MapControls.Command(564, 431, _
      "www.gatewayhorizons.com", "MapMatrixWS", "MunicipalExample", _
      "", "", True)

' Add layers
cmd.Layers.Add(cmd.GetAvailableLayer("Roads"))
cmd.Layers.Add(cmd.GetAvailableLayer("Aerial Photography"))
cmd.Layers.Add(cmd.GetAvailableLayer("Parcels"))
' Get a map from the MapServer
cmd.UpdateMap()

' Bind the image to a PictureBox control called MapImage
MapImage.Image = Utilities.BitmapFromUrl(cmd.MapImageLocation)

End Sub
```

The final method call in the sample code above reads the image from the specified URL and returns a .Net Bitmap object. This object can be used as necessary, in this case it is bound assigned to the Image property of the PictureBox that has been placed on a form. The result is that the Map that has been requested is displayed on the form.

That is all that is required to GIS enable an application.

3.3.4 Manipulating the Map Display

Once a MapServer session has been established (a connection is open) and layers have been added, manipulations to the display are usually required. The manipulations include windowing, zooming, panning, changing symbology and other basic GIS functions. The manipulation of the display is described in the following sections.

3.3.5 Windowing Capabilities

The Command object has methods that support a variety of windowing commands. A list of these are shown below in an excerpt from the Reference Guide.

MoveWindowTo	Pan from center of map image to coordinate given This method will force an Update.
Pan	Pan map image by a given delta (in pixels) This method will force a Update.
PanByDirection	Pan map image in a given direction. This method will force an Update.
Zoom	Overloaded. Performs a zoom operation on the Map Window This method will force an Update.
ZoomByFactor	Zoom by a given factor. This method will force an Update.
ZoomToGround	Overloaded. Zooms to a given ground coordinate This changes the window location but not the size
ZoomToLayerLimits	Zoom to window region (in ground coordinates) of map image This method will force an Update.
ZoomToNext	Zooms to the next window in the queue Next windows are available if ZoomToPrevious was used prior to this method call
ZoomToPrevious	Zooms to the last window.
ZoomToWindow	Zoom to window region This method will force an Update.

The Application Objects do not include controls that work in .NET Windows Forms or ASPNet Web Forms. Controls are provided for in the WebControls (V2.1) and WinControls (soon to be available). However, when creating a Windows Forms application, it is quite easy to provide for user interface controls with standard .Net Windows controls. For example, if two Buttons are placed on a Windows Form for performing zoom in and out functions the code would look as follows:

C#

```
private void btnZoomIn_Click(object sender, System.EventArgs e)
{
      cmd.ZoomByFactor(.5, 282, 214);
      MapImage.Image = Utilities.BitmapFromUrl(cmd.MapImageLocation);
}
private void btnZoomOut_Click(object sender, System.EventArgs e)
{
      cmd.ZoomByFactor(1.5, 282, 214);
      MapImage.Image = Utilities.BitmapFromUrl(cmd.MapImageLocation);
}
```

VB

```
Private Sub btnZoomIn_Click(ByVal sender As System.Object, _
   ByVal e As System.EventArgs) Handles btnZoomIn.Click

   cmd.ZoomByFactor(0.5, 282, 214)
   MapImage.Image = Utilities.BitmapFromUrl(cmd.MapImageLocation)

End Sub

Private Sub btnZoomOut_Click(ByVal sender As System.Object, _
   ByVal e As System.EventArgs) Handles btnZoomOut.Click

   cmd.ZoomByFactor(1.5, 282, 214)
   MapImage.Image = Utilities.BitmapFromUrl(cmd.MapImageLocation)

End Sub
```

In the sample code above a ZoomByFactor method is being used to modify the area that the Map image will show. For zoom in the factor used is 0.5. This changes the window to one half of its current size. The coordinates that are supplied are an approximate center of an Image that has dimensions of 560 x 450 pixels. The result is that when a Zoom button is depressed the Form will display a new image that has a different window.

Other examples of windowing and panning can be seen in the walk-through application later in this document.

3.3.6 Refreshing or Resizing the Map Window

The Image that is created by the MapServer can be refreshed or resized with methods called on the command object. If a smaller or larger bitmap is desired the current size can be changed. The methods used to perform these operations are:

ResizeMap	Resize the Map Window
UpdateMap	Submit queued commands to the MapServer

ResizeMap accepts the new dimensions in pixels of the image to be created. It is important to ensure that the image that is presented (for example, the PictureBox) has the same dimensions as are issued to the MapServer. This will prevent any distortion of the image from occurring during the display of the Map.

The UpdateMap forces a call to the MapServer and all previously queued commands are processed. This includes commands such as adding layers, turning layers on or off, or changing the display symbology of layers. There are some commands that force an call to the MapServer such as windowing commands and feature selection commands.

3.3.7 Controlling Layer Display

There are several criteria that are used when layers have been added to the LocalLayers to determine whether or not they are displayed when the map is refreshed (UpdateMap).

The AvailableLayers collection contains properties that fully describe each layer as defined on the MapServer in the configuration files. These properties are retained when a layer is added to the LocalLayers collection.

If a layer is being displayed and it is required to turn that layer off the command objects methods TurnLayerOn and TurnLayerOff are used.

TurnLayerOff	Overloaded. Turn the display of a layer off An update will not automatically occur.
TurnLayerOn	Overloaded. Turn the Display of a layer on An update will not automatically occur.

It is necessary to perform an UpdateMap to see the results of the change to the layer display.

C#

```
cmd.TurnLayerOn("Roads");
cmd.TurnLayerOff("Aerial Photography");
cmd.UpdateMap();
```

VB

```
cmd.TurnLayerOn("Roads")
cmd.TurnLayerOff("Aerial Photography")
cmd.UpdateMap()
```

The Visible property on a layer can **not** be used to change the display status of a layer after it has been loaded and submitted to the MapServer. The TurnLayerOn and Off methods must be used.

3.3.8 Scale Dependent Display

Scale dependent display is a technique used to provide for a different presentation of map data at different scales. There are several reasons that this may be a desired approach with GIS applications. If a certain layer is very dense, for example, a property boundary layer, it is not necessarily desirable to show this at all display scales. If the current window are is of an entire city the property boundaries would be very dense and would take a long time to display. To provide this capability the visibility of layers can be made dependent upon the current display scale.

The Layer object contains properties that define what the scale dependency.

MaxDisplayResolution (inherited from **FeatureBase**)	Maximum distance across the display area that the feature will be shown (in ground units).
MinDisplayResolution (inherited from **FeatureBase**)	The Minimum distance across the display area that the feature will be shown (in ground units).

If in the case described above where parcels are not to be displayed above a certain display scale we simply set these properties. The following code snippet shows how to change the scale dependency of layer during the Form_Load event (as it is being added to the LocalLayers collection)

C#

```
// Change the default for roads by
// First get a reference to the layer
Layer ly = cmd.GetAvailableLayer("Roads");

// Change the properties
ly.MaxDisplayResolution = 1000;
ly.MinDisplayResolution = 0;

// Add the Layer to our LocalLayersCollection
cmd.Layers.Add(ly);
```

VB

```
' Change the default for roads by
' First get a reference to the layer
Dim ly As Layer = cmd.GetAvailableLayer("Roads")

' Change the properties
ly.MaxDisplayResolution = 1000
ly.MinDisplayResolution = 0

' Add the Layer to our LocalLayersCollection
cmd.Layers.Add(ly)
```

In the above example a new Layer is created from the AvailableLayers, the property is changed, and finally the layer is added to our session. If the image that is created is greater than 1000 ground units across, the layer will not display, even though its display status (Visible property) is on. This technique is critical to use when displaying extremely dense layers such as aerial photography.

Scale dependency can also be used to turn certain layers off when a very small window has been reached. This can be useful in cases where solid fill areas are on when a large window is shown (such as political boundaries) but it is not desirable to show these when a close window is shown. The solid fill of the area could obscure other features such as the image from an aerial photograph.

3.3.9 Display Symbology

The appearance of GIS features can be modified to virtually any appearance that is desired. The Layer object contains a Symbology collection that is used for setting that variety of display attributes that are possible. There is a robust set of definitions available for Lines, Points, Text, and Area features. The basic structure and operations will be described here.

GIS layers can have more than one set of symbology associated with a single feature type (area, line, etc). This can be used in a variety of ways such as displaying one type of symbology at one scale and another at a different scale or to display two types of representations for a single element at the same time. For example, if a solid fill was desired at a large display window for property boundaries, but only the boundary was desired at a small scale, two symbology objects would be added the Symbology collection with scale dependency.

C#

```
// Change the default display for parcels by:
// Get a reference to the layer
Layer Parcels = cmd.GetAvailableLayer("Parcels");

// The layer can be added at any time as we work with
// a reference
cmd.Layers.Add(Parcels);

// Set custom symbology for Parcels
AreaSymbology Bigp = (AreaSymbology)Parcels.Symbology["Default"];
Bigp.UseScale = true;

Bigp.FillArea = true;
Bigp.FillColor = new RGBColor(StandardColor.Khaki);
Bigp.ShowOutline = true;
Bigp.Outline.SetColor(StandardColor.Black);
Bigp.Outline.Weight = 0.5;

// Set min and max display scale. The MaximumDisplayScale from the
// layer takes precedent over the symbology scale
// (if the Parels.MaximumDisplayScale = 1000 it is used instead
// of the following max)
Bigp.MaxScale = 99999999;
Bigp.MinScale = 2000.0;

// Create a new symbology object from the existing one
AreaSymbology Littlep = (AreaSymbology)Bigp.Clone();
Littlep.Name = "Little";

// Set the display properties
Littlep.FillArea = false;
Littlep.Outline.SetColor(StandardColor.DeepPink);
Littlep.MaxScale = 2000.0;
Littlep.MinScale = 0;

// Add the new symbology to the collection
Parcels.Symbology.Add(Littlep);
```

VB

```
' Change the default display for parcels by:
' Get a reference to the layer
Dim ly As Layer = cmd.GetAvailableLayer("Parcels")

' The layer can be added at any time as we work with
' a reference
cmd.Layers.Add(Parcels)

' Set custom symbology for Parcels
Dim Bigp As AreaSymbology = _
      CType(Parcels.Symbology("Default"), AreaSymbology)
Bigp.UseScale = True

Bigp.FillArea = True
Bigp.FillColor = New RGBColor(StandardColor.Khaki)
Bigp.ShowOutline = True
Bigp.Outline.SetColor(StandardColor.Black)
Bigp.Outline.Weight = 0.5;

' Set min and max display scale. The MaximumDisplayScale from the
' layer takes precedent over the symbology scale
' (if the Parels.MaximumDisplayScale = 1000 it is used instead
' of the following max)
Bigp.MaxScale = 99999999;
Bigp.MinScale = 2000.0;

' Create a new symbology object from the existing one
Dim Littlep As AreaSymbology = _
      CType(Bigp.Clone(), AreaSymbology)
Littlep.Name = "Little"

' Set the display properties
Littlep.FillArea = False
Littlep.Outline.SetColor(StandardColor.DeepPink)
Littlep.MaxScale = 2000.0
Littlep.MinScale = 0

' Add the new symbology to the collection
Parcels.Symbology.Add(Littlep)
```

This example shows the flexibility of the Symbology collection and its objects. Virtually any combination of colors, fonts, fills, and styles can be combined to produce a display that is visually pleasing and useful to the application. See the Reference Guide under the MapMatrix.Features class library for a complete list of properties for controlling the display symbology.

3.3.10 Identify and Searching GIS Elements

The MapMatrix NetGIS Application Objects provide capabilities for working directly with individual GIS Elements and their attributes. For many GIS applications and the integration of the GIS data with other applications it is

important to either locate an individual element or elements based upon either an attribute search or physical selection of an element or elements via a graphical pick on the map image. These capabilities are provided with the command object and its methods.

Find	Overloaded. Find elements based upon an attribute query
Identify	Overloaded. Identifies and element and returns a matching attribute list
ThematicAdd	Creates a new thematic definition against a layer

The Find method allows GIS elements to be located by attribute search, the Identify method allows GIS elements to be located based upon a graphical selected (mouse click) and the ThematicAdd creates a subset of data that can be worked with separately from the full layer.

3.3.10.1 Identify GIS Elements

The following code snippet demonstrates how to locate an element (Identify) by reacting to a MouseDown event on a PictureBox control.

C#

```
private void MapImage_MouseDown(object sender,
      System.Windows.Forms.MouseEventArgs e)
{
   // do zoom in off click coordinate
   MapMatrix.MapControls.DataTable results =
            cmd.Identify("Roads", e.X, e.Y, true);
   if (results != null)
   {
      dgIdentifyResults.DataSource =
            Utilities.ConvertToMSDataTable(results);
      MapImage.Image =
            Utilities.BitmapFromUrl(cmd.MapImageLocation);
   }
}
```

VB

```
Private Sub MapImage_MouseDown(ByVal sender As Object, _
      ByVal e As System.Windows.Forms.MouseEventArgs) _
      Handles MapImage.MouseDown

  Dim results As MapMatrix.MapControls.DataTable = _
      cmd.Identify("Roads", e.X, e.Y, True)

  If (Not results Is Nothing) Then
      dgIdentifyResults.DataSource = _
```

```
            Utilities.ConvertToMSDataTable(results)
        MapImage.Image = Utilities.BitmapFromUrl(cmd.MapImageLocation)
    End If

End Sub
```

This example demonstrates several important points.

The identify command returns a MapControls.DataTable object. This object contains any attribute information that was returned from the identify command. More than one row will be returned if the click located more than one object within its tolerance.

The MapControls.DataTable object can be converted into a System.Data.DataTable object with the Utilities.ConverToMSDataTable method. This allows the results to be bound to a data grid control as is done in the example.

A null result from the identify command means that the no items were found within the tolerance.

The e.X and e.Y are the pixel coordinates of the screen click. The Identify method requires the parameters be in image coordinates (pixels).

The boolean parameter specifies whether or not the selected elements will be displayed with a highlight. The highlight symbology can be customized just as the layer symbology by modifying the property HighlightSymbology for the Layer.

3.3.10.2 Searching GIS Elements (Find)

The following code snippet demonstrates how to locate an element by searching its attribute tables for a match.

C#

```
string sql;

int polyvalue = int.Parse(textBox1.Text);
sql =  "select * from Parcels where BASEPOLY_ = " +
            polyvalue.ToString();

MapMatrix.MapControls.DataTable results =
      cmd.Find("Parcels", sql, null, new ZoomToResultsData(
      MapMatrix.MapControls.ZoomToResultsMethod.FixedSize, 500.0));

if (results != null)
{
   dgIdentifyResults.DataSource =
            Utilities.ConvertToMSDataTable(results);
   MapImage.Image =
            Utilities.BitmapFromUrl(cmd.MapImageLocation);
```

```
}
```

VB

```
Dim sql As String

Dim polyvalue As Integer = Integer.Parse(textBox1.Text)

sql = "select * from Parcels where BASEPOLY_ = " + polyvalue.ToString()

Dim results As MapMatrix.MapControls.DataTable = _
    cmd.Find("Parcels", sql, Nothing, New ZoomToResultsData( _
    MapMatrix.MapControls.ZoomToResultsMethod.FixedSize, 500.0))

If (Not results Is Nothing) Then
   dgIdentifyResults.DataSource = _
                  Utilities.ConvertToMSDataTable(results)
   MapImage.Image = Utilities.BitmapFromUrl(cmd.MapImageLocation)
End If
```

The example shows a SQL statement being prepared with an value from a TextBox control. This query is passed into the Find command. If matches are found the map display is changed to have the resultant GIS Element in its center and a fixed size of 500 ground units. The attributes are placed into a DataTable and bound to a DataGrid control.

As with Identify the Find method makes use of HighlightSymbology for the Layer. The ZoomToResultsMethod enumeration provides a variety of types of window results that can be used.

3.3.10.3 Thematic Displays

Thematic displays are being completely rewritten to provide a higher level of performance and more robust capabilities in the next version of the Application Objects. Version 2.2 is scheduled for release in October 2003.

3.4 GeoCoding Services

GeoCoding Services provide the capability to locate a position on the ground based upon a supplied address. GeoCoding Services are provided though the GetGeoCode method on the Command object.

GetGeoCode	Processes a Geocode request

The following code snippet shows a call to the GeoCode Service.

C#

```
private void btnGetMap_Click(object sender, System.EventArgs e)
{
  // query for geocode results
```

```
  GeoCodeResultList results = cmd.GetGeoCode(
      new GeoCode(txtAddress.Text, txtZip.Text, 40, 5, 15));

  // if result set isn't empty, pull up a map
  if (results.Count > 0)
  {
      // bind results to combobox
      cbGeoCodeResults.DataSource =
            Utilities.ConvertToMSDataTable(results);
  }
  else
  {
      // if result set is empty, raise error message
      MessageBox.Show("Could not find a match for: "
      + txtAddress.Text);
      }
}
```

VB

```
Private Sub btnGetMap_Click(ByVal sender As System.Object, _
  ByVal e As System.EventArgs) Handles btnGetMap.Click
  ' query for geocode results

  Dim results As GeoCodeResultList = cmd.GetGeoCode( _
      New GeoCode(txtAddress.Text, txtZip.Text, 40, 5, 15))

  ' if result set isn't empty, pull up a map
  If (results.Count > 0) Then
     ' bind results to combobox
     cbGeoCodeResults.DataSource = _
        Utilities.ConvertToMSDataTable(results)
  Else
     ' if result set is empty, raise error message
     MessageBox.Show("Could not find a match for: " + txtAddress.Text)
  End If

End Sub
```

When making a call to GetGeoCode service you provide a GeoCode object which must contain: Address, Zip, Score, MaxResults and Spelling sensitivity. This sample supplies the necessary information and returns possible matches in a GeoGocodeResultsList (which can be converted to a DataTable) . The results can be evaluated for the best potential match score. This item can be used to re-position the map window to show the results of the Geocode operation.

C#

```
DataRow currentRow = table.Rows[0];

groundX = Convert.ToDouble(currentRow["CoordinateX"]);
groundY = Convert.ToDouble(currentRow["CoordinateY"]);

cmd.ZoomToGround(groundX, groundY, windowSize);
MapImage.Image = GetImage(cmd.MapImageLocation);
```

VB

```
Dim currentRow As System.Data.DataRow = table.Rows.Item(0)

Dim groundX As Double = Convert.ToDouble(currentRow("CoordinateX"))
Dim groundY As Double = Convert.ToDouble(currentRow("CoordinateY"))

cmd.ZoomToGround(groundX, groundY, 4500)
MapImage.Image = Utilities.BitmapFromUrl(cmd.MapImageLocation)
```

The ZoomToGround method positions the map image with the geocode point at the center of the display and a window size of 4500 ground units. The table in the above sample is created with the `Utilities.ConvertToMSDataTable` from the results from the GetGeoCode method.

The GeoCode services to provide some good capability but are being enhanced to provide a greater capabilities in the next version of the Application Objects. Version 2.2 is scheduled for release in October 2003.

4 Walkthrough – Building a .NET Address Lookup Application

This section walks you though the steps of building an Address lookup application in .NET, you will find a similar sample application, with complete source code in the \samples subdirectory where you installed the Application Objects or on our web site.

PLEASE NOTE: This walkthrough makes reference to building the Address Lookup app with Visual Studio.NET. Note that Visual Studio.NET is not required to build applications with the .NET Framework. If you do not have Visual Studio.NET you can download the .NET Framework SDK free from Microsoft's msdn website.

Visit the Gateway Horizons website http://www.GatewayHorizons.com developer zone for additional sample applications.

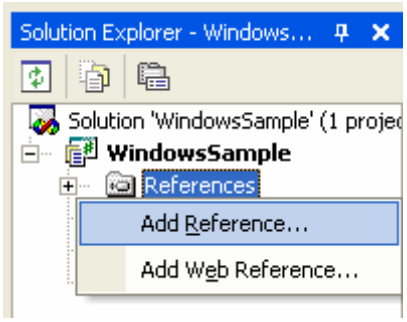
Step One – Create a new .Net Project

The first step is to create a new .Net project.

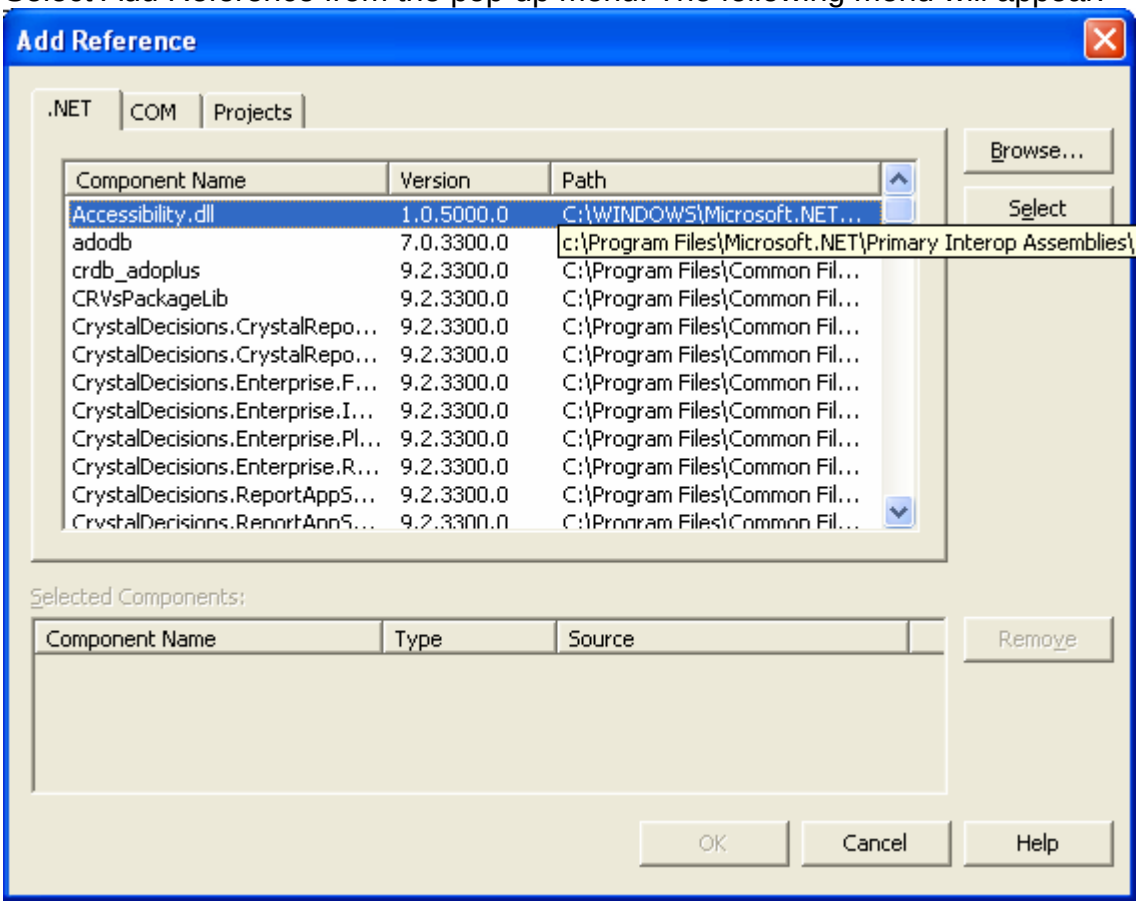
Open Visual Studio .NET and create a new .NET Windows Application project. This can be done as either a Visual Basic or C# project.

Step Two – Add references to the NetGIS Application Objects

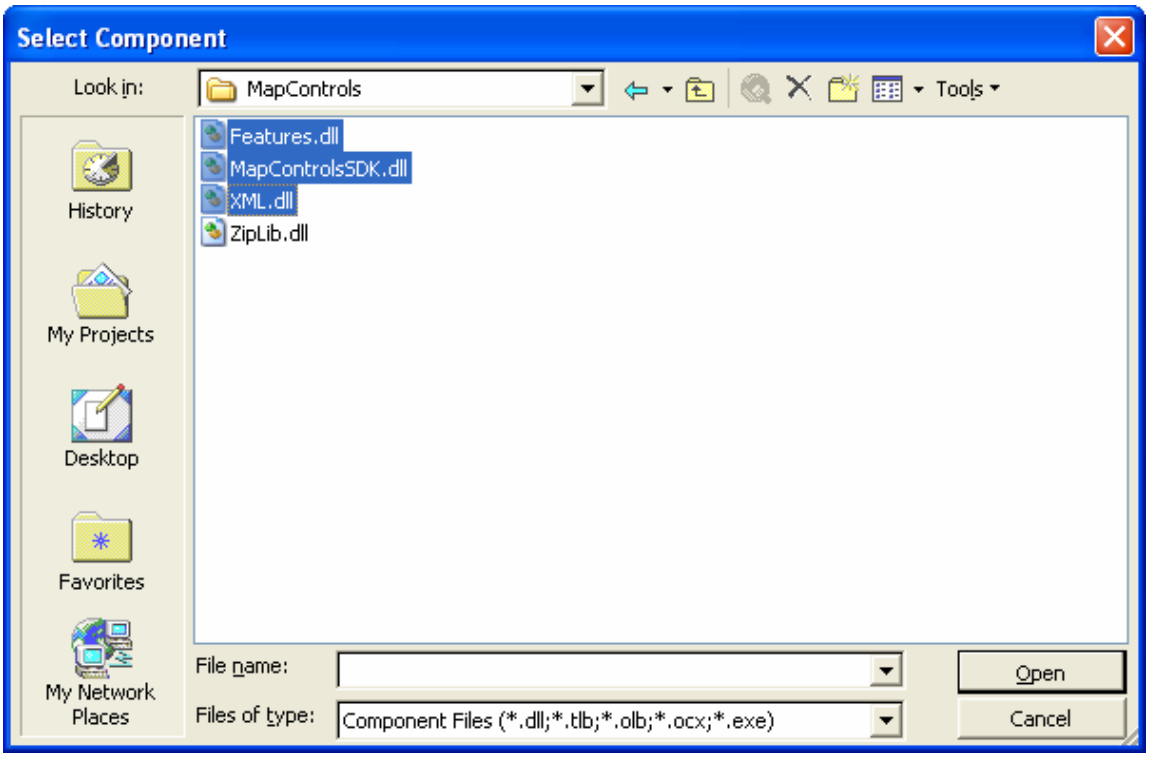
The Windows Application project must have a reference the NetGIS objects. Add them by right-clicking on the References folder in the solution explorer.



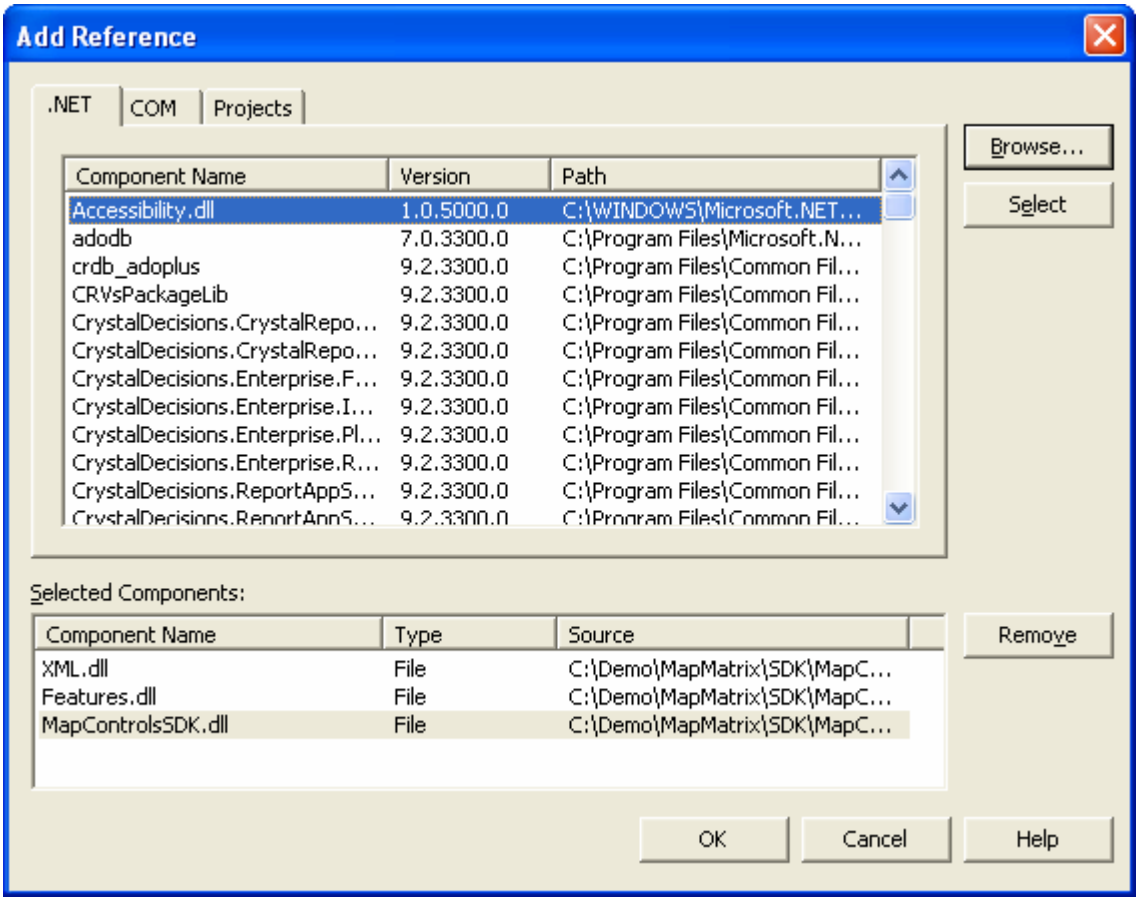
Select Add Reference from the pop-up menu. The following menu will appear.



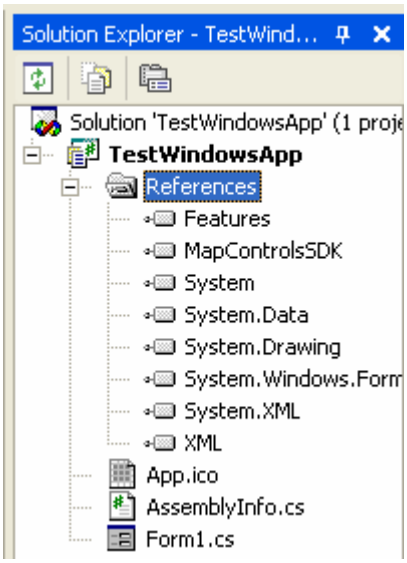
Click the Browse button and navigate to the location the contains the MapMatrix NetGIS Libraries. Select the `Features.dll`, `MapControlsSDK`.dll, and `XML.dll` files.



Click open then OK on the following dialog.



This will add the references to the Solution Explorer. Expand the References folder and you should see the files.



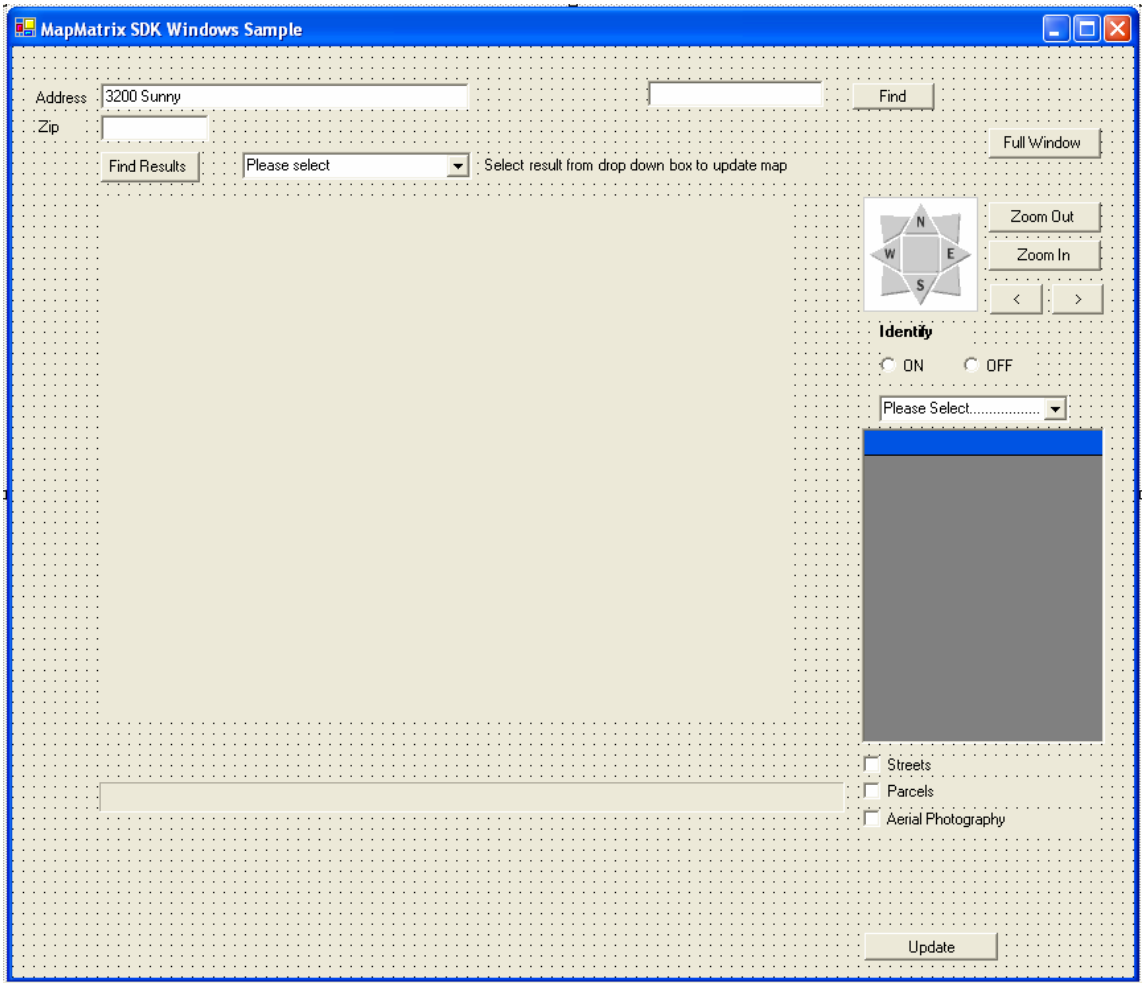
You are now ready to develop the application.

Step Three – Add Controls to the Form

Using the form designer, setup your form like the one pictured below.

1. Drag a Label onto the Form. Set its **`Text`** to “Address”.
2. Drag another label and set its **`Text`** to “Zip”.
3. Place another label with its **`Text`** = “Select result from list to go to an address.”
4. Place a TextBox next to Address label and name it txtAddress.
5. Place another TextBox next to Zip label and name it txtZip.
6. Place a Button below txtZip and name it btnGetMap and make its **`Text`** = “Find Map”.
7. Place a ComboBox next to the Find button and name it cbGeoCodeResults. Size it so it can contain a large string (this will contain potential matches). Set its Text to “Please select”.
8. Place a PictureBox below the Find button and name it MapImage. Set its Width = 520 and Height = 386.
9. Place a TextBox to the right of txtAddress and name it textBox1.
10. Place a Button to the right of the above, name it button1 and make its Text “Find”.
11. To the right of the Find button place another button named btnZoomFull make its Text = “Full Window”.
12. To the right of the MapImage place a PictureBox sized 89,88 name it pbCompass. If you have loaded the samples set its image to rose.png.
13. To the right of the pbCompass place a button, name it btnZoomOut and make its Text = “Zoom Out”.
14. Below Zoom Out place a button, name it btnZoomIn and make its Text = “Zoom In”.
15. Below the Zoom In button place a button, name it btnPrevious and make its Text = “<”.
16. To the right of btnPrevious place a button name it btnNext and make its Text = “>”.
17. Below the pbCompass place a label and make its Text = “Identify”.
18. Below the Identify label place a RadioButton, name it rbIdentifyOn and make its Text = “On”.
19. To the right of the On identify button place a RadioButton, name it rbIdentifyOff and make its Text = “Off”.
20. Below the radio buttons place a ComboBox, name it cbIdentifyLayer, add in its Items Collection “Roads” and “Parcels”. Make its Text = “Please Select….”.
21. Below the Identify ComboBox place a DataGrid. Name it dgIdentifyResults and size it 184, 239.
22. Below the text place 3 checkboxes names chkRoads, chkParcels and chkAerials. Make there text Roads, Parcels and Aerial Photography.
23. Below the MapImage place ProgressBar, name it pbRenderStatus and set its Visible property to False.

24. At the bottom right of the form place a button, name it cmdUpdateMap and make its Text = “Update”.



Step Four – Adding References to the Code

Open the code view for the Form. You can do this by clicking one of the code icons (main toolbar or above solution explorer) or by right clicking the form and selecting View Code or by double clicking the page.

At the class level add the appropriate references to the MapMatrix class libraries.

C#

```
using MapMatrix.MapControls;
using MapMatrix.Features;
using MapMatrix.Features.MapSymbology;
```

VB

```
Imports MapMatrix.MapControls
Imports MapMatrix.Features

Imports MapMatrix.Features.MapSymbology
```

Step Five – Initialize the Map

When an application is first started several things must be done to establish what data the application works with. The NetGIS Mapping Web Services Map Server must be connected to. In order for map information to display, layers must be loaded to the current session. When the initial connection is made to the Map Server a list of layers that are on the Map Server are placed in the AvailableLayers Collection. A layer is added only the first time that the page is loaded (it is persisted for the duration of the session). Because of this you should only add the layer if the page is not being loaded from a post back. The modified Page_Load event is shown below.

C#

```
private void Form1_Load(object sender, System.EventArgs e)
{
   // create connection to map server
   cmd = new MapMatrix.MapControls.Command(564,431, _
      "www.gatewayhorizons.com", "MapMatrixWS", _
      "MunicipalExample", "", "", true);

   // add layers
   try
   {
      // Change the default color for roads by
      // First get a reference to the layer
      Layer ly = cmd.GetAvailableLayer("Roads");
      // Next get a reference to the Default symbology
      LineSymbology sym = (LineSymbology)ly.Symbology["Default"];
      // Change the color
      sym.SetColor(StandardColor.RosyBrown);

      // Add the Aerial Photography Layer
      cmd.Layers.Add(cmd.GetAvailableLayer("Aerial Photography"));

      // When adding the roads, use the object reference we created
      // above where the symbology was changed
      cmd.Layers.Add(ly);

      // Finally, add the Parcels

      cmd.Layers.Add(cmd.GetAvailableLayer("Parcels"));

      PerformUpdate();
   }
      catch (Exception exc)
   {
      MessageBox.Show(exc.Message);
      return;
   }

   MapImage.Visible = true;
```

```
    label3.Visible = true;

    btnZoomIn.Visible = true;
    btnZoomOut.Visible = true;
    pbCompass.Visible = true;
    lblIdentify.Visible = true;
    rbIdentifyOn.Visible = true;
    rbIdentifyOff.Visible = true;
    dgIdentifyResults.Visible = true;
    cbIdentifyLayer.Visible = true;

    chkRoads.Visible = true;
    chkParcels.Visible = true;
    chkAerials.Visible = true;

    chkRoads.Checked = true;
    chkParcels.Checked= true;
    chkAerials.Checked = true;

}
```

VB

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

  ' create connection to map server
  cmd = New MapMatrix.MapControls.Command(564, 431, _
      "www.gatewayhorizons.com", "MapMatrixWS", _
      "MunicipalExample", "", "", True)

  ' add layers
  Try

    ' Change the default color for roads by
    ' First get a reference to the layer
    Dim ly As Layer = cmd.GetAvailableLayer("Roads")
    ' Next get a reference to the Default symbology
    Dim sym As LineSymbology = _
        CType(ly.Symbology("Default"), LineSymbology)
    ' Change the color
    sym.SetColor(StandardColor.RosyBrown)

    ' Add the Aerial Photography Layer
    cmd.Layers.Add(cmd.GetAvailableLayer("Aerial Photography"))

    ' When adding the roads,
    ' use the object reference we created above where
    ' the symbology was changed
    cmd.Layers.Add(ly)

    ' Finally, add the Parcels
    cmd.Layers.Add(cmd.GetAvailableLayer("Parcels"))

    PerformUpdate()

    Catch exc As Exception
```

```
            MessageBox.Show(exc.Message)
            Return
        End Try


        MapImage.Visible = True
        label3.Visible = True

        btnZoomIn.Visible = True
        btnZoomOut.Visible = True
        pbCompass.Visible = True
        lblIdentify.Visible = True
        rbIdentifyOn.Visible = True
        rbIdentifyOff.Visible = True
        dgIdentifyResults.Visible = True
        cbIdentifyLayer.Visible = True

        chkRoads.Visible = True
        chkParcels.Visible = True
        chkAerials.Visible = True

        chkRoads.Checked = True
        chkParcels.Checked = True
        chkAerials.Checked = True

End Sub
```

This code makes a reference to the Command object and adds layers to the Layers collection. If a Layer is in the Layers collection it can be displayed or manipulated with other GIS commands. A single command is issued to send the requests to the Map Server and the MapImage is refreshed. Other miscellaneous properties for the Forms controls are set. The application can now be run and you should see a map.

Step Six – Add the Address Lookup to the Find button

The address lookup will be performed when the Find button is used. To perform this task we must add code to the btnGetMap click event. The simplest way to get to the code section for the click event is to open the WebForm in design mode and double click the find button. This will open the code page with the following code.

C#

```
private void btnGetMap_Click(object sender, System.EventArgs e)
{

}
```

VB

```
Private Sub btnGetMap_Click(ByVal sender As System.Object, ByVal e As
System.EventArgs) Handles btnFind.Click
```

```
End Sub
```

In this method the GeoCode request (address lookup) will be submitted to the MapServer, the results will be stored in the dropdown DataSource property. The results will be shown in the dropdown. The modified code is shown below.

C#

```
private void btnGetMap_Click(object sender, System.EventArgs e)
{
   // query for geocode results
   GeoCodeResultList results = cmd.GetGeoCode(
      new GeoCode(txtAddress.Text, txtZip.Text, 40, 5, 15));

   // if result set isn't empty,
   // pull up a map and display map related UI controls
   if (results.Count > 0)
   {
      // bind results to combobox
      cbGeoCodeResults.DataSource =
            Utilities.ConvertToMSDataTable(results);
      cbGeoCodeResults.ValueMember = "Address";
      cbGeoCodeResults.DisplayMember = "Address";

      bLoadDropDownComplete = true;

      cbGeoCodeResults.Visible = true;
      // make hidden controls visible

   }
   else
   {
      // if result set is empty, raise error message
      MessageBox.Show("Could not find a match for: "
            + txtAddress.Text);
   }
}
```

VB

```
Private Sub btnGetMap_Click(ByVal sender As System.Object, ByVal e As
System.EventArgs) Handles btnGetMap.Click

   ' query for geocode results
   Dim results As GeoCodeResultList = cmd.GetGeoCode( _
      New GeoCode(txtAddress.Text, txtZip.Text, 40, 5, 15))

   ' if result set isn't empty, pull up a map and display map
   ' related UI controls
   If (results.Count > 0) Then
      ' bind results to combobox
      cbGeoCodeResults.DataSource = _
            Utilities.ConvertToMSDataTable(results)

      cbGeoCodeResults.ValueMember = "Address"
      cbGeoCodeResults.DisplayMember = "Address"

      bLoadDropDownComplete = True

      cbGeoCodeResults.Visible = True

   Else
      ' if result set is empty, raise error message
      MessageBox.Show("Could not find a match for: " + txtAddress.Text)
   End If

End Sub
```

Step Seven – Change window to selected address

The code in the previous step populates the drop down control with a set of possible candidates that match the address that was keyed in. To relocate the window to another address in the drop down code must be entered to the SelectIndexChanged event. To code in this event double click the drop down from the designer. The following code will appear.

C#

```
private void cbGeoCodeResults_SelectedIndexChanged(object sender,
System.EventArgs e)
{

}
```

VB

```
Private Sub cbGeoCodeResults_SelectedIndexChanged(ByVal sender As
System.Object, ByVal e As System.EventArgs) Handles
cbGeoCodeResults.SelectedIndexChanged

End Sub
```

When the application detects a _SelectedIndexChanged event it will perform a change ZoomToGround method call.

C#

```
private void cbGeoCodeResults_SelectedIndexChanged(object sender,
System.EventArgs e)
{
      if (bLoadDropDownComplete)
      {
            ShowStatusBar(20);
            System.Data.DataTable table =
                  (System.Data.DataTable)cbGeoCodeResults.DataSource;
            DataRow currentRow =
                  table.Rows[cbGeoCodeResults.SelectedIndex];
            groundX = Convert.ToDouble(currentRow["CoordinateX"]);
            groundY = Convert.ToDouble(currentRow["CoordinateY"]);
            ShowStatusBar(50);
            windowSize = 4500;
            cmd.ZoomToGround(groundX, groundY, windowSize);
            ShowStatusBar(100);
            MapImage.Image = GetImage(cmd.MapImageLocation);
            HideStatusBar();
      }
}
```

VB

```
Private Sub ddAddresses_SelectedIndexChanged(ByVal sender As
System.Object, ByVal e As System.EventArgs) Handles
ddAddresses.SelectedIndexChanged
    If (bLoadDropDownComplete) Then

        ShowStatusBar(20)

        Dim table As System.Data.DataTable = _
                  cbGeoCodeResults.DataSource
        Dim currentRow As System.Data.DataRow = _
                  table.Rows.Item(cbGeoCodeResults.SelectedIndex)

        groundX = Convert.ToDouble(currentRow("CoordinateX"))
        groundY = Convert.ToDouble(currentRow("CoordinateY"))

        ShowStatusBar(50)
        windowSize = 4500

        cmd.ZoomToGround(groundX, groundY, windowSize)
        ShowStatusBar(100)
        MapImage.Image = GetImage(cmd.MapImageLocation)

        HideStatusBar()

    End If
End Sub
```

Key an address in (600 Wild Rose) and after the initial display select another entry from the drop down. The map display will change to a location for the new address.