



TWaver[®] GIS for .NET

Developer Guide

Jun 2011

Serva Software

info@servasoftware.com

<http://www.servasoftware.com>

PO Box 8143, Wichita Falls, Texas, USA 76307



For more information about Serva Software and TWaver please visit the web site at:

<http://www.servasoftware.com>

Or send e-mail to:

info@servasoftware.com

Jun, 2011

Notice:

This document contains proprietary information of Serva Software. Possession and use of this document shall be strictly in accordance with a license agreement between the user and Serva Software, and receipt or possession of this document does not convey any rights to reproduce or disclose its contents, or to manufacture, use, or sell anything it may describe. It may not be reproduced, disclosed, or used by others without specific written authorization of Serva Software.

TWaver, servasoft, Serva Software and the logo are registered trademarks of Serva Software. Java and all Java-based marks are trademarks or registered trademarks of Sun Microsystems, Inc. in the U.S.A. and other countries. Other company, brand, or product names are trademarks or registered trademarks of their respective holders. The information contained in this document is subject to change without notice at the discretion of Serva Software.

Copyright © 2011 Serva Software LLC

All Rights Reserved

Table of Contents

- TWaver GIS for .NET
 - Introduction of TWaver GIS for .NET
 - Dependencies of TWaver GIS for .NET
 - Setting Up Development Enviroment
 - Using TWaver GIS for .NET
 - Basics.
 - Add Layers for Map Control
 - Initialize the Map Control
 - Interaction
 - Using Features.
 - Query.
 - Events.
 - Controls.
 - Combine with Network.
 - Adapter.
 - Interaction with Geographic Data.

TWaver GIS for .NET

TWaver GIS for .NET provides a way to add interactive map data into your project developed for .NET framework. This development kit can display the map contents provided by WMS servers, Google Maps, Bing Maps, OpenStreetMap and MapABC. It also helps you to combine your telecom topology data with those kinds of map contents.

At the same time, this documentation is designed for people familiar with WPF, Silverlight, C# programming, and object-oriented programming concepts. If you want to be more in-depth, you should learn about WMS (Web Map Service) and WFS (Web Feature Service) of OGC (Open Geospatial Consortium), and even the structure of spatial data.

TWaver GIS for .NET provides Map component for WPF and Silverlight.

Introduction of TWaver GIS for .NET

TWaver GIS for .NET combines the power of Silverlight/WPF and Map Services to create an immersive mapping experience. Developers can use the Map Control provided by TWaver GIS to incorporate the location and spatial search features into their applications.

TWaver GIS for .Net is a software development kit (SDK) and consists of a complete set of application programming interface (API). With this kit, developers are able to combine network component provided by TWaver with map data. This kit is developed using c# under .NET Framework 4.0, and supports WPF and Silverlight.

- [Dependencies of TWaver GIS for .NET](#)
- [Setting Up Development Enviroment](#)

Dependencies of TWaver GIS for .NET

You can develop your project based on .NET Framework 4.0.

Displaying map data, you need to add twavergis.dll library in project.

Combining TWaver Network component with the map data, you need to plus twaver.dll library in project.

Setting Up Development Enviroment



Please make sure you have got TWaver GIS for .NET from Serva Software.

We recommend you use Visual Studio 2010.

Install Visual Studio 2010

Visual Studio 2010 comes with .NET Framework 4. It can be installed by using the installation media (for example, DVDs), or by downloading from the product website.

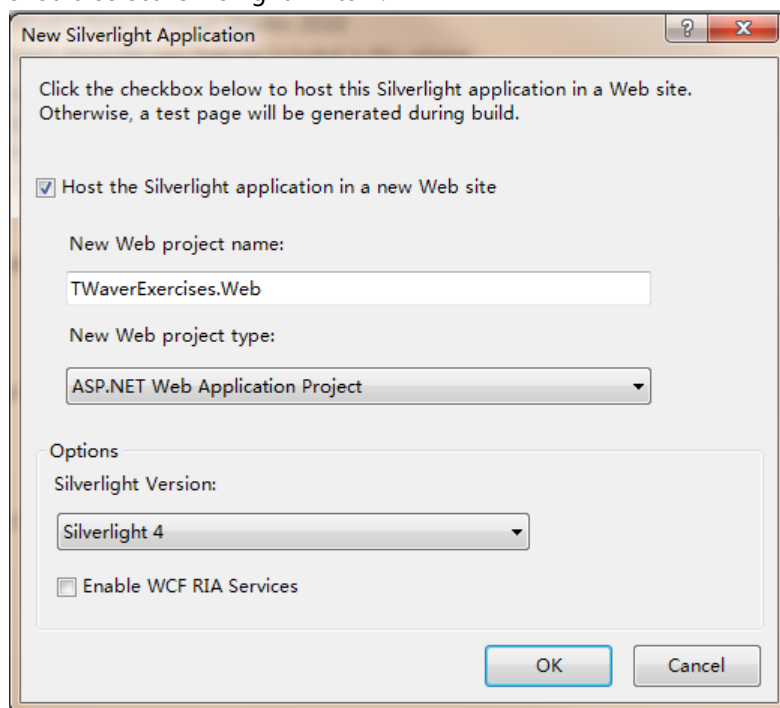
To get more details about installing Visual Studio 2010, please refer to <http://msdn.microsoft.com/en-us/library/e2h7fzkw.aspx>

If you want to develop Silverlight applications, please [install the Silverlight 4 Tools for Visual Studio 2010](#)

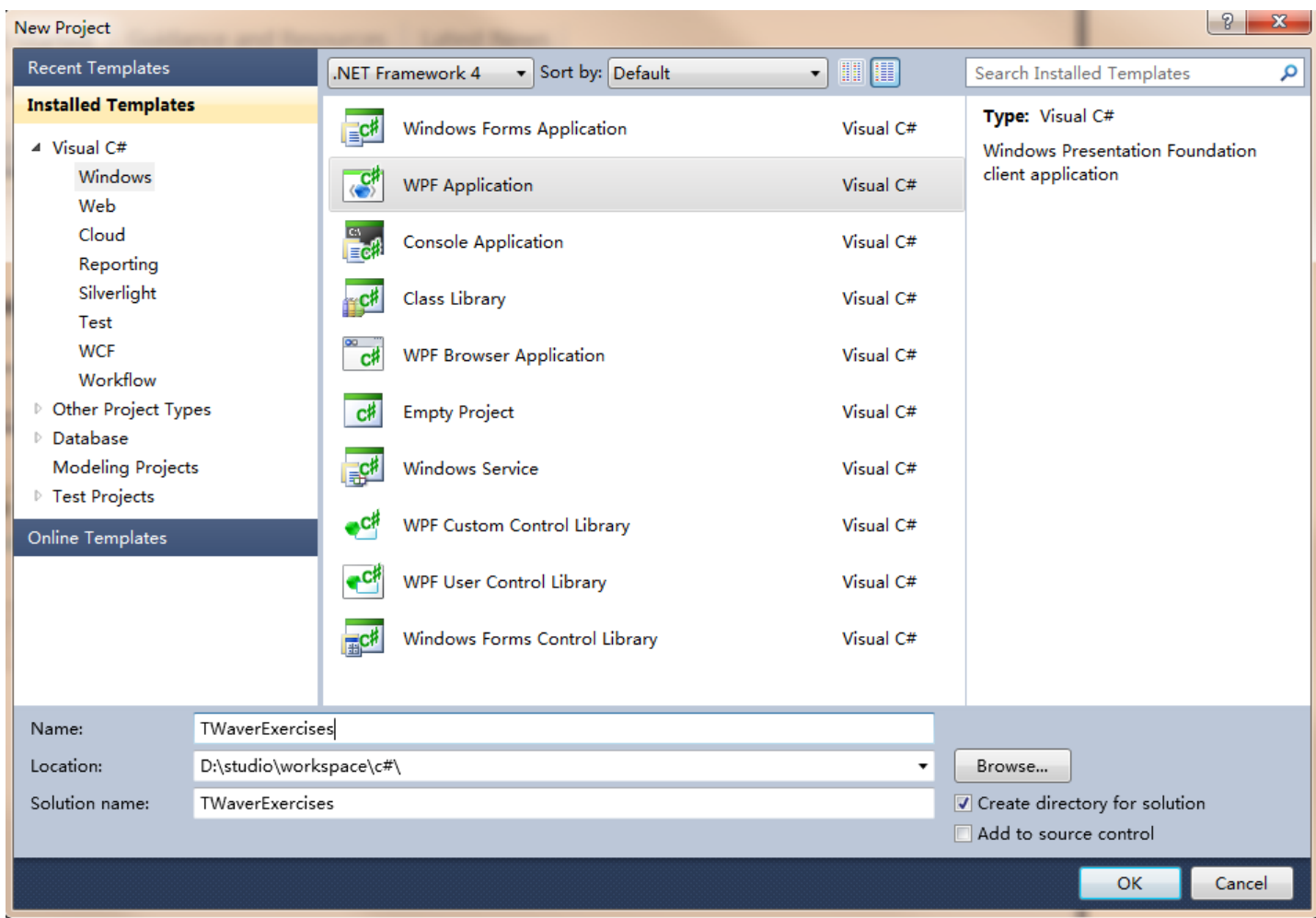
Create Your Project

The following illustration shows an example of the Silverlight project templates.

When you install the Silverlight 4 Tools for Visual Studio 2010, the option to target Silverlight 4 will appear. You should select "Silverlight 4" item.

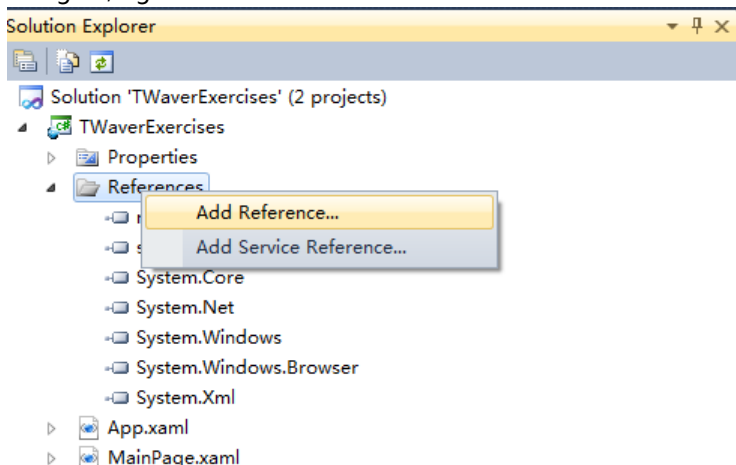


The following illustration shows an example of the WPF project templates.

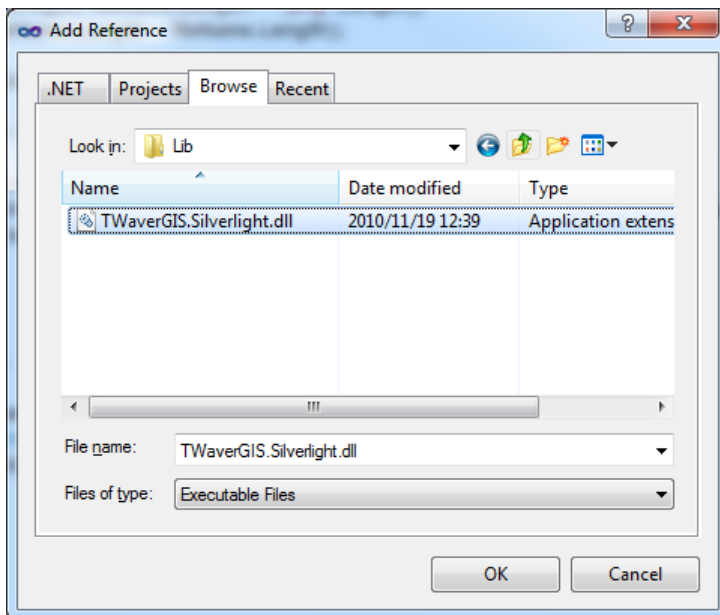


Add Reference

Before compiling code, you need to add TWaverGIS.Silverlight.dll as reference into your project. In the Project Designer, right click the References tab and choose "Add Reference..." item from popup menu.



A relative dialog box will appear as below. Click Browse tab, select the Library Path and then choose the TWaverGIS.Silverlight.dll (if you are developing a WPF project, please choose TWaverGIS.WPF.dll). Then click the "OK" button.



Writing the "Hello World" Based on TWaver GIS for .NET

The easiest way to start learning about the TWaver GIS for .NET is to refer to a simple example. We'll create a simple xaml file and a simple xaml.cs file using Visual Studio 2010 as below example, compile that file, and launch the file for visual inspection.

1.Create HelloWorld.xaml

```
<UserControl x:Class="TWaverExercises.Tour.GIS.HelloWorld"
    xmlns="http://schemas.microsoft.com/winfx/2006/xaml/presentation"
    xmlns:x="http://schemas.microsoft.com/winfx/2006/xaml"
    xmlns:d="http://schemas.microsoft.com/expression/blend/2008"
    xmlns:mc="http://schemas.openxmlformats.org/markup-compatibility/2006"
    xmlns:twavergis="clr-namespace:TWaver.GIS;assembly=TWaverGIS.Silverlight"
    mc:Ignorable="d"
    d:DesignHeight="300" d:DesignWidth="400">

    <Grid x:Name="LayoutRoot" Background="White">
        <twavergis:Map Name="map" HorizontalAlignment="Stretch" VerticalAlignment="Stretch"/>
    </Grid>
</UserControl>
```



If you are developing a WPF project, you should define twavergis with the following code:
xmlns:twavergis="clr-namespace:TWaver.GIS;assembly=TWaverGIS.WPF"

Add some codes into HelloWorld.xaml.cs

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Net;
using System.Windows;
using System.Windows.Controls;
using System.Windows.Documents;
using System.Windows.Input;
using System.Windows.Media;
using System.Windows.Media.Animation;
using System.Windows.Shapes;
```

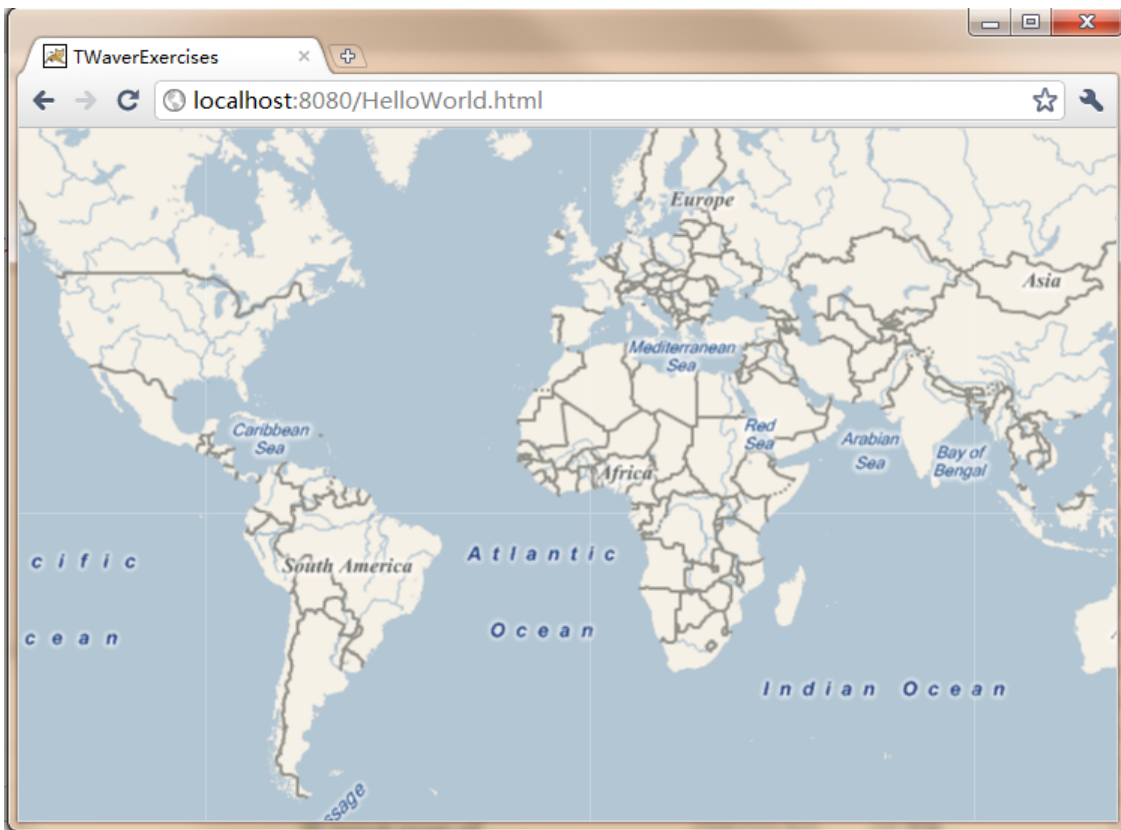
```
using TWaver.GIS.Util;


namespace TWaverExercises.Tour.GIS
{
    public partial class HelloWorld : UserControl
    {
        public HelloWorld()
        {
            InitializeComponent();
            InstallMap();
        }
        private void InstallMap()
        {
            map.AddLayer("BingMap", GISConsts.EXECUTOR_TYPE_BINGMAP);
            map.ZoomLevel = 2;
        }
    }
}
```

You should make some changes in MainPage.xaml.cs to load HelloWorld.xaml

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Net;
using System.Windows;
using System.Windows.Controls;
using System.Windows.Documents;
using System.Windows.Input;
using System.Windows.Media;
using System.Windows.Media.Animation;
using System.Windows.Shapes;
using TWaverExercises.Tour;
public MainPage()
{
    InitializeComponent();
    LayoutRoot.Children.Add(new HelloWorld());
}
```

Now you can build the project, and will get a result like the following snap shot



 Because TWaver GIS will access the map images from different map servers, to get the result, you should deploy project developed with TWaver GIS on your Web Server, like IIS-Server, Tomcat.

Using TWaver GIS for .NET

- [Basics.](#)
- [Events.](#)
- [Controls.](#)
- [Combine with Network.](#)

Basics.

Geography Coordinates

In TWaver GIS for .NET, the geography coordinate is described by `twaver.gis.geom.GeoCoordinate` class, which consists of latitude and longitude properties. Developers construct a `GeoCoordinate` object by passing its parameters in the order (latitude, longitude)

as is customary in cartography:

```
GeoCoordinat geo = new GeoCoordinate(latitude,longitude);
```

In TWaver GIS for .NET, you can control which area of the map should be displayed on screen by centering the map on a geography location.

```
map.center = new GeoCoordinate(30,-90));
```

If you also want to locate the NE(Network Element) on the map, you should set the NE's client property `GISConsts.GEOCOORDINATE`.

```
Node node = new Node();
node.SetClient(GISConsts.GEOCOORDINATE,new GeoCoordinate(20,-10));
```

Map Data Source

TWaver GIS for .NET designs build-in tile mechanism to display the map. It can access the map which is deployed by WMS servers or tiles server like GoogleMaps, BingMaps, OSM (OpenStreetMap). You can combine the data from different sources into a map, but be sure that the map data from different sources should be deployed in the same projection. TWaver GIS for .NET supports equidistant projection and mercator projection at present.

Map, Layer and Feature

It is a common sense that features compose a layer, and layers compose a map in GIS (Geography Information System). So TWaver GIS for .NET provides these kinds of elements to manage map data. The Layer object will display geography graphic as tile images and you can deal with WFS results with Feature objects. Map class is designed to contain those layer objects and handle different interactions between human and machine.

Add Layers for Map Control

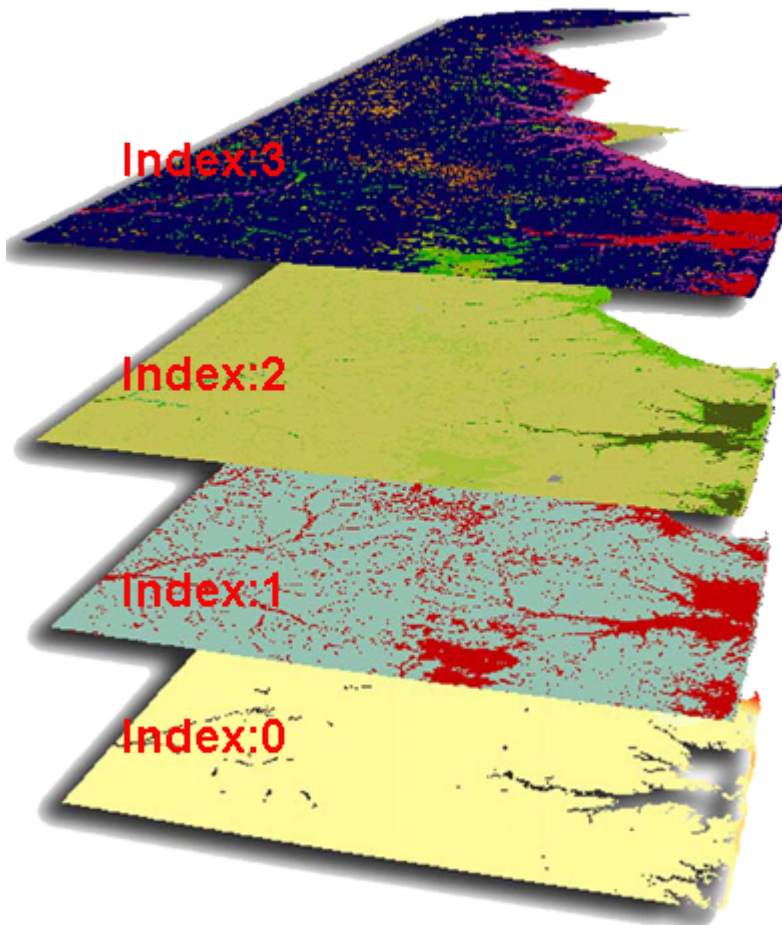
Directly add layers

You can add a layer into a map directly without declaring a layer object firstly. In this case, what you need to do is specify which kind data you should load, and where you will load the data.

```
//load a layer named server:world from a WMS
//server(URL:"http://twaver.gicp.net:9000/geoserver/wms?"),
//and the data is deployed in EPSG:4326.
map.AddLayer("serva:world",
    GISConsts.EXECUTOR_TYPE_GEOSERVER_WMS_4326,
    "http://twaver.gicp.net:9000/geoserver/wms?");
}
//load the map layer from Google
map.AddLayer("googlemap",
    GISConsts.EXECUTOR_TYPE_GOOGLEMAP);
//load map data from BingMap
map.AddLayer("BingMap",GISConsts.EXECUTOR_TYPE_BINGMAP);
```

The map can present multi-layers

The layer with lower index value will be drawn firstly.



Initialize the Map Control

Initialize the Map Control

TWaver GIS for .NET provides a Map Control to help developers accessing map data. A Map control can be defined and initialized in cs file directly.

```
Map map = new Map();
LayoutRoot.Children.Add(map);
map.Center = new GeoCoordinate(20,-70);
map.AddLayer("BingMap",GISConsts.EXECUTOR_TYPE_BINGMAP);
map.ZoomLevel = 1;
```

A Map control also be defined in a xaml file, and be initialized in cs file.

Defining the map object in a xaml file.

```
<UserControl x:Class="TWaverExercises.Tour.GIS.HelloWorld"
  xmlns="http://schemas.microsoft.com/winfx/2006/xaml/presentation"
  xmlns:x="http://schemas.microsoft.com/winfx/2006/xaml"
  xmlns:d="http://schemas.microsoft.com/expression/blend/2008"
  xmlns:mc="http://schemas.openxmlformats.org/markup-compatibility/2006"
  xmlns:twavergis="clr-namespace:TWaver.GIS;assembly=TWaverGIS.Silverlight"
  mc:Ignorable="d"
  d:DesignHeight="300" d:DesignWidth="400">

  <Grid x:Name="LayoutRoot" Background="White">
    <twavergis:Map Name="map" HorizontalAlignment="Stretch" VerticalAlignment="Stretch"/>
  </Grid>
</UserControl>
```

Initializing the map control in the relative cs file.

```
...
private void InitMap()
{
  map.AddLayer("BingMap",GISConsts.EXECUTOR_TYPE_BINGMAP);
  map.ZoomLevel = 1;
}
.....
```

Interaction

Interactions with the map

The Map Control implements some default interactions on a map.

Developers can also make interaction with the map data by invoking methods provided by TWaver GIS.

Zoom Level

TWaver GIS for .NET controls the zoom of the map by setting zoomLevel property.

```
map.ZoomLevel = 2;
```

TWaver GIS for .NET has implemented zoom in/out function in default. User can do that using the scroll wheel.

Pan

Developers can pan the map by resetting the center of the map.

```
map.Center = new GeoCoordinate(30,100);
```

TWaver GIS for .NET has implemented panning function in default and users can pan the map by dragging the map.

Conversion Between Coordinate Systems

Conversion between different coordinate systems is very important. This can help developers to get necessary geographic information of the map data.

TWaver GIS can implement the conversion between screen coordinate system and geographic coordinate system.

```
private function clickOnMap(e:MouseEvent){
    GeoCoordinate geo = map.GetGeoPointFromPointOnMap(e.localX,e.localY);
    Console.WriteLine("coordinate is "+geo);
}
private function getScreenLocation(geo:GeoCoordinate):Point{
    return map.getScreenPointFromGeoPoint(geo);
}
```


Using Features.

Features are the raw data of the map. Multi-features consist a layer, and layers are used together to represent a whole map. Interactions with a map include panning, zooming, querying features, adding features, removing features and updating features.

In most cases, developers need to get the detailed information of some features, or edit some features. TWaver GIS for .NET implements those functions through WFS (Web Feature Service) and uses transactions to update, insert or delete features. Developers can use the wrapped interface to communicate with the servers which provide WFS.

TWaver GIS for .NET supports GeoServer 1.7 or later by default.

To get more details of WFS, please refer to [Web Feature Service](#).

- [Query](#).

Query.

Query

Usually, developers want to get the features which are contained in a specified area or have some specified properties. TWaver GIS for .NET wraps the WFS queries for developers. Those queries are divided into two kinds. One is called logic query, the other is called spatial query.

Logic Query

This kind of query is always used to get the features according to the specified non-geom properties. For example, developers can get the features whose name is "NewYork" by an EQUAL logic Query. The following statements will show the classifications of logic query and how to invoke those queries. TWaver GIS for .NET designs

1. EQUAL

For example: Getting features whose name equal "twaver-OLT1" from a layer named "server:twpoint".

```
private void ShowFeatureRecordSet(List<Feature> features)
{
    foreach (Feature feature in features) {
        List<String> names = feature.PropertyNames;
        String content = feature.Fid;
        foreach (String name in names) {
            content += " " + name + ": " + feature.GetProperty(name);
        }
        Console.WriteLine(content);
    }
}

private void EqualQuery()
{
    String ns = "serva=" + "http://www.servasoftware.com/gis/";
    String server = "http://twaver.servasoft.com:8000/geoserver/wfs?";
    String ln = "serva:twpoint";
    ComparisonCondition condition =
        new ComparisonCondition(GISConsts.COMPARISON_LOGIC_TYPE_SINGLEOPERATOR);
    condition.SetOperators(new int[] { GISConsts.COMPARISON_OPERATOR_EQUAL });
    condition.SetReferenceProperties(new String[] { "name" });
    condition.SetReferenceValues(new String[] { "twaver-OLT1" });
    TWaver.GIS.OGC.Actions.Action action = new QueryAction(ns, ln, condition);
    String operation = WFSUtils.buildQueryOperator(ns, new TWaver.GIS.OGC.Actions.Action[] { action });
    Console.WriteLine(operation);
    #if SILVERLIGHT
        WFSUtils.Query(ShowFeatureRecordSet, server, operation);
    #else
        List<Feature> result = WFSUtils.Query(server, operation);
        ShowFeatureRecordSet(result);
    #endif
}
```

The result may look like the following:
twpoint.25 userid: NO.1997 name: twaver-OLT1 address: 1500 Fisher Roadm

2. GREATER THAN

For example: Getting features whose MALE property is greater than 2000000 from layer named "topp:states".

```
private void GreaterThanQuery()
{
```

```
String ns = "topp=\"http://www.openplans.org/topp\"";
String server = "http://twaver.servasoft.com:8000/geoserver/wfs?";
ComparisonCondition condition =
    new ComparisonCondition(GISConsts.COMPARISON_LOGIC_TYPE_SINGLEOPERATOR);
condition.SetOperators(new int[] { GISConsts.COMPARISON_OPERATOR_GREATER });
condition.SetReferenceProperties(new String[] { "MALE" });
condition.SetReferenceValues(new String[] { "2000000" });
TWaver.GIS.OCG.Actions.Action action = new QueryAction(ns, "topp:states", condition);
String operation = WFSUtils.buildQueryOperator(ns, new TWaver.GIS.OCG.Actions.Action[] { action });
#if SILVERLIGHT
    WFSUtils.Query>ShowFeatureRecordSet,server,operation));
#else
    List<Feature> result = WFSUtils.Query(server, operation);
    ShowFeatureRecordSet(result);
#endif
}
```

You will get the result will like [this](#) fragment:

states.1 STATE_NAME: Illinois MALE: 5552233.0 FEMALE: 5878369.0 ...

3. LESS THAN

For example: Getting features whose MALE property is less than 2000000 from layer named "topp:states".

```
private void LessThanQuery(){
    String ns = "topp=\"http://www.openplans.org/topp\"";
    String server = "http://twaver.servasoft.com:8000/geoserver/wfs?";
    ComparisonCondition condition =
        new ComparisonCondition(GISConsts.COMPARISON_LOGIC_TYPE_SINGLEOPERATOR);
    condition.SetOperators(new int[] { GISConsts.COMPARISON_OPERATOR_LESS });
    condition.SetReferenceProperties(new String[] { "MALE" });
    condition.SetReferenceValues(new String[] { "2000000" });
    TWaver.GIS.OCG.Actions.Action action = new QueryAction(ns, "topp:states", condition);
    String operation = WFSUtils.buildQueryOperator(ns, new TWaver.GIS.OCG.Actions.Action[] { action });
    #if SILVERLIGHT
        WFSUtils.Query>ShowFeatureRecordSet,server,operation));
    #else
        List<Feature> result = WFSUtils.Query(server, operation);
        ShowFeatureRecordSet(result);
    #endif
}
```

You may get the result like the following:

states.2 STATE_NAME: District of Columbia MALE: 282970.0 FEMALE: 323930.0

states.3 STATE_NAME: Delaware MALE: 322968.0 FEMALE: 343200.0

.....

4. BETWEEN

For example: Getting features whose MALE property is between 2000000 and 3000000 from layer named "topp:states".

```
private void BetweenQuery(){
    String ns = "topp=\"http://www.openplans.org/topp\"";
    String server = "http://twaver.servasoft.com:8000/geoserver/wfs?";
    ComparisonCondition condition =
        new ComparisonCondition(GISConsts.COMPARISON_LOGIC_TYPE_SINGLEOPERATOR);
    condition.SetOperators(new int[] { GISConsts.COMPARISON_OPERATOR_BETWEEN });
    condition.SetReferenceProperties(new String[] { "MALE" });
    condition.SetReferenceValues(new String[] { "2000000", "3000000" });
    TWaver.GIS.OCG.Actions.Action action = new QueryAction(ns, "topp:states", condition);
    String operation = WFSUtils.buildQueryOperator(ns, new TWaver.GIS.OCG.Actions.Action[] { action });
```

```
#if SILVERLIGHT
    WFSUtils.Query(ShowFeatureRecordSet,server,operation));
#else
    List<Feature> result = WFSUtils.Query(server, operation);
    ShowFeatureRecordSet(result);
#endif
}
```

You may get the result like the following:

```
states.5 STATE_NAME: Maryland MALE: 2318671.0 FEMALE: 2462797.0 ...
states.7 STATE_NAME: Kentucky MALE: 2195130.0 FEMALE: 2356394.0 ...
.....
```

More operator types are listed in the type of `TWaver.GIS.Util.GISConsts.COMPARISON_OPERATOR_*`.

Spatial Query

This kind of query is used to get the features according to some specified spatial conditions. Every feature has two main kinds of properties. They are geometric property and non-geometric property. Spatial query will focus on geometric property which describes the shape of a feature.

1. BBOX

This operator identifies all geometries that spatially interact with the bounding box constraints specified by the user.

```
Geom geom = new EnvelopeGeom(new GeoCoordinate[]{new GeoCoordinate(-10,-10),new
    GeoCoordinate(10,10)});
Condition condition = new SpatialCondition(geom);
condition.SetOperatorType(GISConsts.SPATIAL_OPERATOR_TYPE_BBOX);
TWaver.GIS.OGC.Actions.Action action = new QueryAction(ns,"serva:twpoint",condition);
String operation = WFSUtils.buildQueryOperator(ns,new TWaver.GIS.OGC.Actions.Action[]{action});
```

2. INTERSECTS

This operator gets the features whose geometries will intersect with the specified shape.

```
var bl:GeoCoordinate = new GeoCoordinate(36.37,-106.12);
var ur:GeoCoordinate = new GeoCoordinate(43.143,-95.85);
var geom:Geom = new EnvelopeGeom([bl,ur]);
var condition:SpatialCondition = new SpatialCondition(geom);
condition.SetOperatorType(GISConsts.SPATIAL_OPERATOR_TYPE_INTERSECT);
var action:Action = new QueryAction(ns,"topp:states",condition);
WFSUtils.query(server,WFSUtils.buildQueryOperator(ns,[action]),showFeatureRecordSet);
```

3. CONTAIN

This operator gets the features whose geometries contain the specified shape.

```
Geom geom = new EnvelopeGeom(new GeoCoordinate[]
    {new GeoCoordinate(38.17933,-100.70152),new GeoCoordinate(38.85622,0-98.44523)});
SpatialCondition condition = new SpatialCondition(geom);
condition.SetOperatorType(GISConsts.SPATIAL_OPERATOR_TYPE_CONTAIN);
var action:Action = new QueryAction(ns,"topp:states",condition);
```

4. DWITHIN

This operator gets the features whose geometric property's values are within a specified distance of the specified literal geometric value.

In TWaver GIS for .NET, the default unit of the specified distance is cm.

```
PointGeom geom = new PointGeom(new GeoCoordinate(10,10));
Condition condition = new SpatialCondition(geom);
condition.dwithinDistance = 10000;//unit is cm
condition.SetOperatorType(GISConsts.SPATIAL_OPERATOR_TYPE_DWITHIN);
QueryAction action = new QueryAction(ns,"serva:twpoint",condition);
```

Above code will get the features which are 10000 cms away from the specified point (10,10).

More operator types are listed in the type of TWaver.GIS.Util.GISConsts.SPATIAL_OPERATOR_TYPE_*.

FID Query

Every feature has a built-in property named "FID". This property is a Primary Key. Users can query by FID directly.

```
private function QueryByFID():void{
    String ns = "serva=\"http://www.servasoftware.com/gis\"";
    String server = "http://twaver.servasoft.com:8000/geoserver/wfs?";
    String ln = "serva:twpoint";
    ComparisonCondition condition =
        new ComparisonCondition(GISConsts.COMPARISON_QUERY_BY_FID);
    condition.SetReferenceProperties(["twpoint.3"]);
    QueryAction action = new QueryAction(ns,ln,condition);
    String operation = WFSUtils.buildQueryOperator(ns,new TWaver.GIS.OCG.Actions.Action[]{action})
    List<Feature> features = WFSUtils.query(server,operation);
}
```

Complex Query

Select features with specified properties.

In above examples, you will get the Feature objects filled with all the property values. In some cases, users may want to just get the features with several specified properties. You can use the code as following:

```
private void Query()
{
    String ns = "serva=\"http://www.servasoftware.com/gis\"";
    String server = "http://twaver.gicp.net:9000/geoserver/wfs?";
    ComparisonCondition condition =
        new ComparisonCondition(GISConsts.COMPARISON_LOGIC_TYPE_SINGLEOPERATOR);
    condition.SetOperators(new int[]{GISConsts.COMPARISON_OPERATOR_GREATER});
    condition.SetReferenceProperties(new String[]{"MALE"});
    condition.SetReferenceValues(new String[]{"2000000"});
    QueryAction action = new QueryAction(ns,"topp:states",condition);
    action.PropertyManager.SetProperty("MALE");
    String operation = WFSUtils.buildQueryOperator(ns,new TWaver.GIS.OCG.Actions.Action[]{action});
    List<Feature> features = WFSUtils.query(server,operation);
    ShowFeatureRecordSet(features);
}
```

You can get the results like the following:

```
states.1 MALE: 5552233.0
states.5 MALE: 2318671.0
states.7 MALE: 2195130.0
```

```
states.9 MALE: 3030948.0
```

Use complex condition

1. Use OR operator

```
private void QueryWithOrOperator()
{
    Condition condition = new
    ComparisonCondition(GISConsts.COMPARISON_LOGIC_TYPE_MULTIOPERATORS_OR);
    condition.SetOperators(new int[]
    {GISConsts.COMPARISON_OPERATOR_EQUAL,GISConsts.COMPARISON_OPERATOR_EQUAL});
    condition.SetReferenceProperties(new String[]{"name","userid"});
    condition.SetReferenceValues(new String[]{"n2","NO.1"});
    QueryAction action = new QueryAction(ns,ln,condition);
    String operation = WFSUtils.buildQueryOperator(ns,new TWaver.GIS.OCG.Actions.Action[]{action});
    WFSUtils.query(server,operation,showFeatureRecordSet);
}
```

You will get all the features whose name property equals "n2" or userid property equals "No.1".

1. Use AND operator

```
private void QueryWithANDOperator()
{
    Condition condition = new
    ComparisonCondition(GISConsts.COMPARISON_LOGIC_TYPE_MULTIOPERATORS_AND);

    condition.SetOperators([GISConsts.COMPARISON_OPERATOR_EQUAL,GISConsts.COMPARISON_OPERATOR_EQUAL]);
    condition.SetReferenceProperties(new String[]{"name","userid"});
    condition.SetReferenceValues(new String[]{"n2","NO.1"});
    QueryAction action = new QueryAction(ns,ln,condition);
    String operation = WFSUtils.buildQueryOperator(ns,new TWaver.GIS.OCG.Actions.Action[]{action});
    List<Feature> features = WFSUtils.query(server,operation);
    ShowFeatureRecordSet(features);
}
```

You will get the features whose name property equals "n2" and userid property equals "No.1".

Events.

Map Events Overview

An application driven by events can get supports from TWaver GIS. The programs interested in certain events will register event listeners for those events and execute code when those events are received.

Event listeners

Events in the TWaver GIS are defined within specific classes (MapEvent, LayerEvent) that contain an enumeration of all events specific to the API for .NET. These events are related to state changes of the Maps API environment itself.

Map Control can fire out those events.

To register for notification of these events, use the AddMapEventHandler or AddLayerEventHandler.

Monitor the state of the map

When some states of the map are changed, the map will fire a MapEvent. For example, a MapEvent whose type is MapEvent_MAP_CENTERPOINT_CHENGED will be fired when the center of the map is changed.

Using MapEvent

If you are interested in the zoom changing of the map, you can use this snippet:

```
private void InitMap()
{
    map.AddMapEventHandler(MonitorStateChanged);
}
private void MonitorStateChanged(Object sender, MapEvent evt)
{
    if(MapEvent.MAP_ZOOM_CHANGED.Equals(evt.EventType)
    {
        Console.WriteLine("ZoomLevel is Changed, Current ZoomLevel is "+map.ZoomLevel;
    }
}
```

When moving the map, developers are able to get the MapEvent.MAP_CENTERPOINT_CHENGED event. You can handle this event by register relative listeners.

```
private void MonitorStateChanged(Object sender, MapEvent evt)
{
    if(MapEvent.MAP_CENTERPOING_CHANGED.Equals(evt.EventType)
    {
        Console.WriteLine("Center is Changed, Current Center is "+ map.Center);
    }
}
```

Using LayerEvent

When a layer is added or removed, a LayerEvent will be fired to notify the program.

```
private void InitMap()
{
    map.AddLayerEventHandler(MonitorLayerEvent);
}
```

```

}
private void MonitorLayerEvent(Object sender, LayerEvent evt)
{
    IGeographyLayer layer = evt.Layer as IGeographyLayer;
    switch(evt.EventType)
    {
        case LayerEvent.LAYER_ADDED:
            Console.WriteLine(layer.Name + " is Added");
            break;
        default:
            break;
    }
}

```



For a complete list of map event constants, see the MapEvent, and LayerEvent reference documentation.

Monitor UI events

In order to correctly display the geographic data on a map, the map object should be set correctly view port property. Usually this property should be dynamically changed according to map's size. So developers can use the following code to do that:

```

private void InitMap()
{
    map.SizeChanged += SizeChangedHandler;
}
private void SizeChangedHandler(Object sender, SizeChangedEventArgs evt)
{
    Console.WriteLine("Map's size is changed");
}

```

Another event is MouseEvent on a map object. Users want to know the correct latitude and longitude of a screen point where the mouse is clicked. Developers can implement this requirement using the following code:

```

private void InitMap()
{
    map.MouseLeftButtonDown = MousePressOnMap;
}
private void MousePressOnMap(Object sender, MouseButtonEventArgs evt)
{
    Point position = evt.GetPosition(map);
    Console.WriteLine("current location is " + map.GetGeoPointByMouse(position.X, position.Y));
}

```

Removing Event Listeners

When an event listener is no longer needed, you should remove it.

```

map.RemoveMapEventHandler(MonitorStateChanged);
or
map.RemoveLayerEventHandler(MonitorLayerEvent);

```


Controls.

Navigator Control

As a necessary tool of a map, the Navigator helps users to pan the map and zoom in or zoom out the map. Developers can lay the Navigator component on a map with the following code:

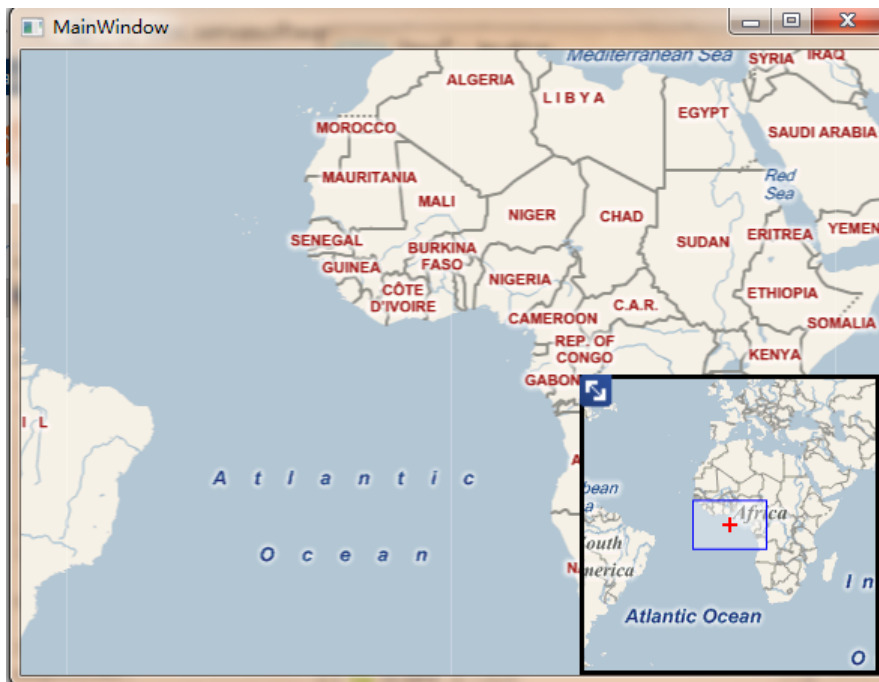
```
private void InitMap()
{
    map.AddLayer("BingMap", GISConsts.EXECUTOR_TYPE_BINGMAP);
    map.ZoomLevel = 1;
    // install the navigator component
    map.InstallNavigator(new Point(15,20));
}
```



Overview Control

Overview control is designed to help users get an overview of the map under some zoom level. Developers can anchor the Overview control at the bottom right corner of the map component.

```
private void InitMap()
{
    map.AddLayer("BingMap", GISConsts.EXECUTOR_TYPE_BINGMAP);
    map.ZoomLevel = 1;
    // install the navigator component
    map.InstallOverview();
}
```



Magnifier Control

TWaver GIS provides a interested Magnifier control. Users can get more details of the map which is under the magnifier.

```
private void InitMap()
{
    map.AddLayer("BingMap", GISConsts.EXECUTOR_TYPE_BINGMAP);
    map.ZoomLevel = 1;
    map.InstallMagnifier();
}
```

With above fragment, a plain magnifier will appear where your mouse pass over, and more detailed map



images will present on the magnifier.

You can change the magnifier's effect by invoking the method `GISManager.RegisterGlobalSetting`

```
GISManager.RegisterGlobalSetting(Settings.MAGNIFIER_EFFECT,
```

```
GISConsts.MAGNIFIER_EFFECT_CURVATURE);;
```

With above codes, the magnifier will display the detailed map image in a way with a little curvature.



Remove controls

You can remove these controls by invoking the relative methods of the Map class.

They are listed as below :

UninstallNavigator, UninstallOverview, and UninstallManifier.

Combine with Network.

TWaver GIS for .NET will help users show their topology network to combine logic topology data with geographic data.

Overlap Network component on a map

Users who are familiar with TWaver Network component, can easily overlap NEs (network element) on a map by put GISConsts.GEOCOORDINATE client property.

Work together with spatial data base

If there are huge amounts of NEs in an application, TWaver GIS recommends you to insert them into spatial data base as resources. Users can deploy all these resources by [GeoServer](#). TWaver GIS for .NET helps user to access these data through WMS and WFS.

- [Adapter.](#)
- [Interaction with Geographic Data.](#)

Adapter.

Class Adapter is designed as glue to combine logic topology network data with geographic data. An Adapter object can locate NEs on a map, and will relocate NEs when the map's states are changed. Users can implement the combination requirement by referring to the example code:

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using System.Windows;
using System.Windows.Controls;
using System.Windows.Data;
using System.Windows.Documents;
using System.Windows.Input;
using System.Windows.Media;
using System.Windows.Media.Imaging;
using System.Windows.Navigation;
using System.Windows.Shapes;
using TWaver.GIS;
using TWaver.GIS.Util;
using TWaver.Network;
using TWaver;
using TWaver.Network.Interaction;
using TWaver.GIS.Extent;
using TWaver.GIS.Geometry;
namespace TWaverGISExercises
{
    /// <summary>
    /// Interaction logic for MainWindow.xaml
    /// </summary>
    public partial class MainWindow : Window
    {
        private int DEFAULTZOOM = 5;
        private GeoCoordinate DEFAULTCENTER = new GeoCoordinate(46, -70);

        private Adapter adapter;
        private Network network;
        public MainWindow()
        {
            InitializeComponent();
            InitMap();
            InitNetwork();
            Combine();
        }
        private void InitMap()
        {
            map.AddLayer("BingMap", GISConsts.EXECUTOR_TYPE_BINGMAP);
            map.ZoomLevel = 5;
            map.Center = DEFAULTCENTER;
        }
        private void InitNetwork()
        {
            network = new Network();
            FillTopoData();
            InitNetworkInteraction();
        }
        private void FillTopoData()
        {
            ElementBox box = network.ElementBox;
            Node node = new Node();
            node.SetClient(GISConsts.GEOCOORDINATE,
```

```

        new GeoCoordinate(45.3, -70));
node.Name = "top";
box.Add(node);
SubNetwork subnetwork = new SubNetwork();
subnetwork.Name = "Subnetwork with GIS";
subnetwork.SetClient("zoom", 10);
subnetwork.SetClient("layerName", GISConsts.TILEMAP_LAYERNAME_GOOGLEMAPPLAYER);
subnetwork.SetClient("layerType", GISConsts.EXECUTOR_TYPE_GOOGLEMAP);
subnetwork.SetClient(GISConsts.GEOCOORDINATE,
    new GeoCoordinate(44.3, -78));
box.Add(subnetwork);

Node child = new Node();
child.Name = "Node 001";
child.SetClient(GISConsts.GEOCOORDINATE,
    new GeoCoordinate(44.3, -73));
subnetwork.AddChild(child);
box.Add(child);
child = new Node();
child.Name = "Node 002";
child.SetClient(GISConsts.GEOCOORDINATE,
    new GeoCoordinate(44.3, -72));
subnetwork.AddChild(child);
box.Add(child);

SubNetwork subnetworkWithoutGIS = new SubNetwork();
subnetworkWithoutGIS.Name = "Subnetwork without GIS";
subnetworkWithoutGIS.SetClient(GISConsts.GEOCOORDINATE,
    new GeoCoordinate(44.3, -72.5));
box.Add(subnetworkWithoutGIS);
node = new Node();
node.Location = new Point(100, 100);
node.Name = "NO GIS Node 001";
node.Parent = subnetworkWithoutGIS;
box.Add(node);
node = new Node();
node.Name = "NO GIS Node 002";
node.Location = new Point(400, 200);
node.Parent = subnetworkWithoutGIS;
box.Add(node);
}

private void InitNetworkInteraction()
{
    network.Interaction += InteractWithNetwork;
}

private void InteractWithNetwork(InteractionEvent evt)
{
    switch (evt.Kind)
    {
        case InteractionEvent.DOUBLE_CLICK_ELEMENT:
            Element element = (Element)network.GetElement(evt.MouseEvent);
            break;
        case InteractionEvent.ENTER_SUBNETWORK:
            EnterSubNetwork(evt);
            break;
        case InteractionEvent.UP_SUBNETWORK:
            QuitSubNetwork(evt);
            break;
        default:
            break;
    }
}

private void EnterSubNetwork(InteractionEvent evt)
{

```

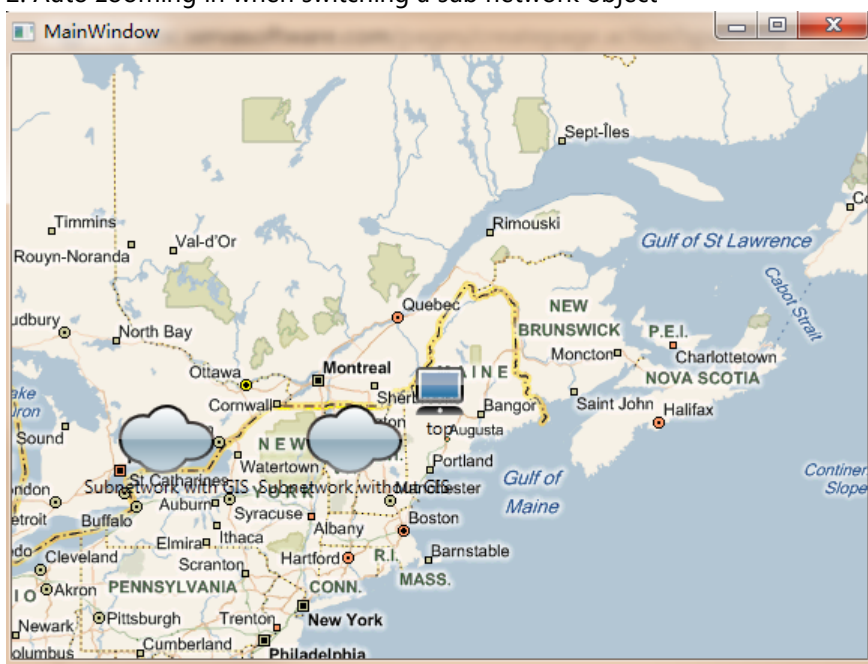
```

        map.RemoveAllLayers();
        SubNetwork subnetwork = (SubNetwork)network.CurrentSubNetwork;
        Object layerName = subnetwork.GetClient("layerName");
        if (!(layerName is String))
        {
            return;
        }
        map.AddLayer("BingMap", GISConsts.EXECUTOR_TYPE_BINGMAP);
        int zoom = Convert.ToInt32(subnetwork.GetClient("zoom"));
        map.ZoomLevel = zoom;
        GeoCoordinate centerPoint = (GeoCoordinate)subnetwork.GetClient(GISConsts.GEOCOORDINATE);
        map.Center = centerPoint;
    }
    private void QuitSubNetwork(InteractionEvent evt)
    {
        map.RemoveAllLayers();
        SubNetwork subnetwork = (SubNetwork)network.CurrentSubNetwork;
        if (subnetwork == null)
        {
            map.AddLayer("BingMap", GISConsts.EXECUTOR_TYPE_BINGMAP);
            map.ZoomLevel = DEFAULTZOOM;
            map.Center = DEFAULTCENTER;
        }
    }
    private void Combine()
    {
        adapter = new Adapter();
        adapter.BindNetworkWithTWaverMap(map, network);
    }
}

```

With above codes you can implements the following functions:

1. Combining network elements with geography data
2. Auto zooming in when switching a sub network object



Interaction with Geographic Data.

As mentioned in previous chapters, when a huge amounts of NEs are required to be displayed in an application, those NEs' shapes should be stored in a spatial data base. TWaver GIS for .NET will display these NEs as geography features on the screen. When users want to interact with some NEs, users can get those features by querying and displaying those features as TWaver's network element on Network component.

For example, a resource management system is required to manage several thousand fibers. Each fiber is divided into many fragments. If you want to display all the data by TWaver Element objects, you possibly meet performance issues. In this situation, displaying these NEs as features on a map will be a good solution. The following code tells you how to insert NEs into a spatial data base, and interact with the features.

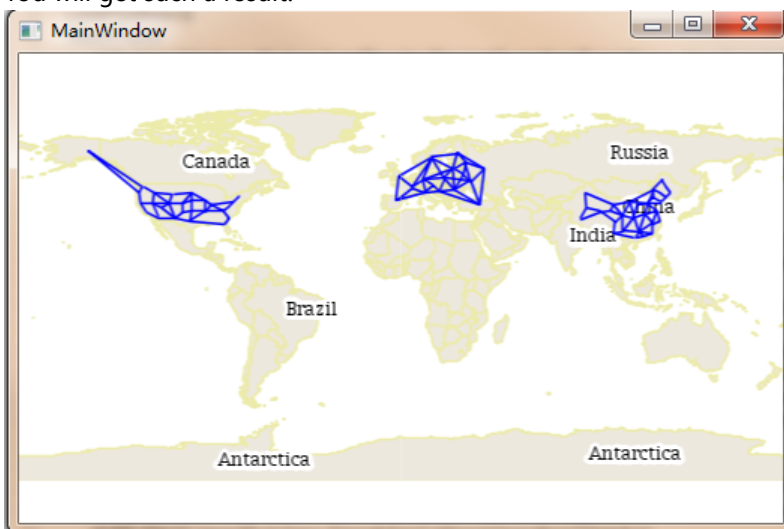
Combine Map component with Network component

Please refer to the previous introduction about [Adapter](#).

```
private void InitMap()
{
    map.AddLayer("serva:world", GISConsts.EXECUTOR_TYPE_GEOSERVER_WMS_4326,DemoConsts.wmsServer );

    map.AddLayer(DemoConsts.FIBERTEMPLATELAYER,GISConsts.EXECUTOR_TYPE_GEOSERVER_WMS_4326,DemoConsts.wmsServer);
}
```

You will get such a result:



Display features with TWaver Element objects.

When you need do interact with some specified NEs, you can get the features from the spatial database and display these features by TWaver's elements.

```
private void FillDataWithFeatures(List<Feature> features)
{
    FillNetwork(network, features);
}
private void InitNetwork()
{
    network = new Network();
    Adapter adapter = new Adapter();
    adapter.BindNetworkWithTWaverMap(map, network);
    BBoxQuery(DemoConsts.FIBERTEMPLATELAYER, new GeoCoordinate(10, -150),
```

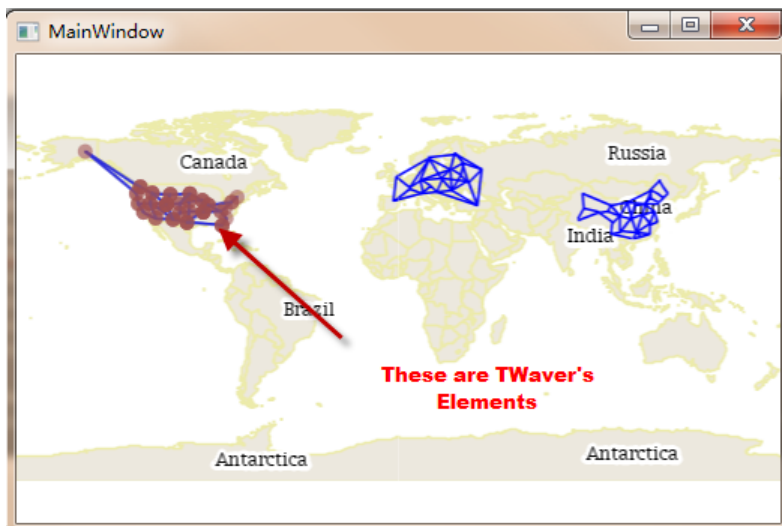


```

        new GeoCoordinate(90, -60), FillDataWithFeatures);
    }
    public static void FillNetwork(Network network, List<Feature> features)
    {
        FeatureConverter converter = new FeatureConverter();
        network.ElementBox.Clear();
        foreach (Feature feature in features)
        {
            Object result = converter.ToTarget(feature);
            if (result is Element)
            {
                Element element = result as Element;
                if (element != null)
                {
                    network.ElementBox.Add(element);
                }
            }
            else if (result is List<Node>)
            {
                List<Node> nodes = result as List<Node>;
                foreach(Node node in nodes)
                {
                    network.ElementBox.Add(node);
                }
            }
        }
    }
    public static void BBoxQuery(String layerName, GeoCoordinate bl, GeoCoordinate ur, FeaturesHandler handler)
    {
        List<GeoCoordinate> bounds = new List<GeoCoordinate>();
        bounds.Add(bl);
        bounds.Add(ur);
        Geom geom = new EnvelopeGeom(bounds);
        SpatialCondition condition = new SpatialCondition(geom);
        condition.OperatorType = (GISConsts.SPATIAL_OPERATOR_TYPE_BBOX);
        QueryAction action = new QueryAction(DemoConsts.SERVA_NAMESPACE, layerName, condition);
        #if SILVERLIGHT
            WFSUtils.Query(handler, DemoConsts.wfsServer,
                WFSUtils.buildQueryOperator(DemoConsts.SERVA_NAMESPACE, new TWaver.GIS.OCG.Actions.Action[]
                { action }));
        #else
            List<Feature> features = WFSUtils.Query(DemoConsts.wfsServer,
                WFSUtils.buildQueryOperator(DemoConsts.SERVA_NAMESPACE, new TWaver.GIS.OCG.Actions.Action[] {action}));
            if (handler != null)
            {
                handler(features);
            }
        #endif
    }
}

```

Then you can get a result like the following shot:



Appendix

```
public class DemoConsts
{
    //public const
    public static String server = "http://twaver.servasoft.com:8000/geoserver/wfs?";
    public static String toppSpace = "topp=\"http://www.openplans.org/topp\"";
    public static String usLayer = "topp:states";
    public static String SERVA_NAMESPACE = "serva=\"http://www.servasoftware.com/gis\"";
    public static String USONWORLD = "usonworld";
        public static String TERMINALLAYER = "serva:twpoint";
        public static String FIBERTEMPLATELAYER = "serva:twline";
        public static String EDITLAYER = "serva:twLineString";

    public static String wfsServer = "http://twaver.servasoft.com:8000/geoserver/wfs?";
    public static String wmsServer = "http://twaver.servasoft.com:8000/geoserver/wms?";
}
```