

Table of Contents

- TWaver GIS for GeoTools 🏠
 - Overview
 - Introduction
 - Dependencies
 - Structures
 - First Project
 - Using the solution
 - Using GeographyMap
 - Using GeographyLayer
 - Using GeographyFeature
 - Handle the events
 - Using MapEvent
 - Using MapLayerChangedEvent
 - Using GisNetworkAdapter
 - Combination
 - Tips
 - Coordinate Transformation
 - Print Map
 - Synchronize zoom
 - Using gadgets
 - Using InterceptedLink

TWaver GIS for GeoTools

Welcome to the TWaver GIS for GeoTools Developer Guide.

- [Overview](#)
- [Introduction](#)
- [Using the solution](#)

Overview

This document is aimed to the experienced TWaver developers, who are familiar with TWaver's topology components.

TWaver GIS is a code library helping users to combine TWaver core with geographical data. With TWaver GIS, a programmer can easily combine geographic data with his applications.

With TWaver GIS, you can:

- Present, navigate, and print maps.
- Operate the map, e.g. Zoom, pan.
- Query attributes of geographic features.
- Combine geographic data with TWaver predefined components.
- Create custom simple GIS applications and embed TWaver GIS capabilities into existing applications.

TWaver GIS is an extended library of TWaver and it is built on the Java 2 platform. With this library, programmers can build Java applications with the abilities of presenting map, operating maps, and even managing the geographic features.

Naming Conventions

In this document, we use "TWaver GIS" to instead the library's name "TWaver GIS Solution for GeoTools".

Contacting Serva Software

Customer input is essential for successful product development. Contact us with comments and questions, and visit our website regularly for additional information and new product announcements.

Website

<http://www.servasoftware.com>

Mailing address

P.O. Box 8143Wichita Falls,TX. 76307

U.S.A.

Telephone and fax

Tel: 918-698-1644

Fax: 208-247-4258

E-Mails

info@servasoftware.com

Introduction















Under Java platform, there are many development kits to help programmers develops an application supporting the map. In order to reduce the complexity of development of the users who have used TWaver, TWaver development team develops a new extended library depending on GeoTools, to help TWaver users combine the network data with the [shape files](#). As this library is built on Swing technology, it can be deployed in an application or in an applet.

- [Dependencies](#)
- [Structures](#)
- [First Project](#)

Dependencies

TWaver GIS needs twaver.jar, JDK/JRE 1.4 or later.

Loading shape files, rendering the map will depends on GeoTools libraries. If you choose this solution, you should download those libraries. They are listed as below:

 geoapi-nogenerics-2.1-M2.jar	326 KB
 gt2-api-2.3.3.jar	94 KB
 gt2-coverage-2.3.3.jar	380 KB
 gt2-indexed-shapefile-2.3.3.jar	122 KB
 gt2-main-2.3.3.jar	1,632 KB
 gt2-referencing-2.3.3.jar	1,095 KB
 gt2-render-2.3.3.jar	182 KB
 gt2-shapefile-2.3.3.jar	96 KB
 gt2-shapefile-renderer-2.3.3.jar	58 KB
 jai_codec.jar	253 KB
 jai_core.jar	1,857 KB
 jsr108-0.01.jar	37 KB
 jts-1.7.1.jar	435 KB
 vecmath-1.3.1.jar	284 KB



GeoTools is an open source (LGPL) Java code library which provides standards compliant methods for the manipulation of geospatial data, for example to implement Geographic Information Systems (GIS). The GeoTools library implements Open Geospatial Consortium (OGC) specifications as they are developed. If you want to get more information about GeoTools, please check <http://geotools.codehaus.org/>.

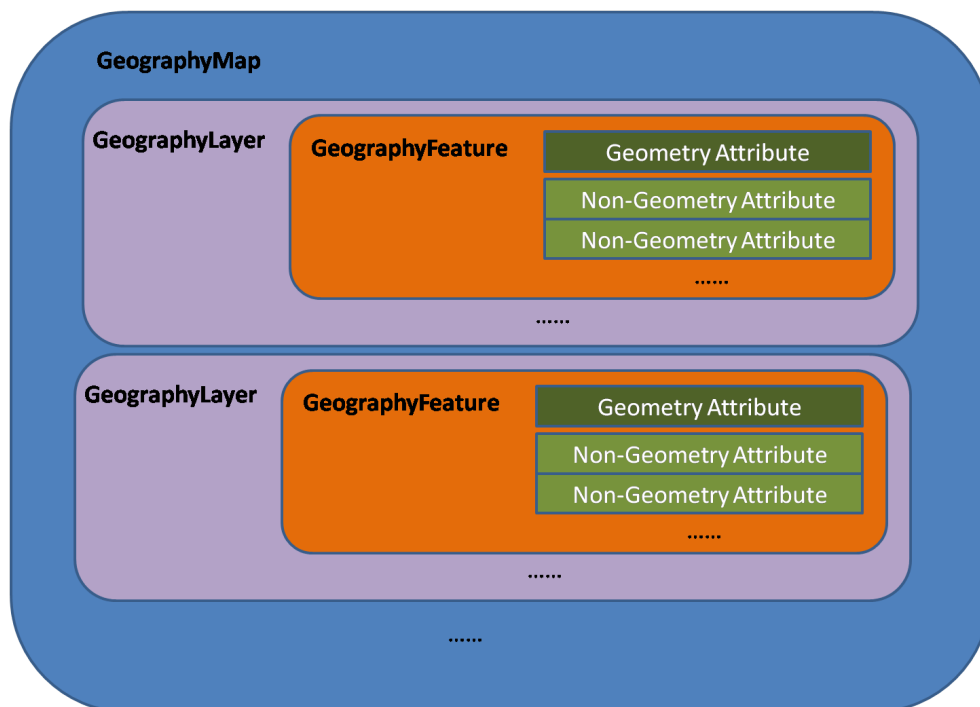
Structures

The structure of TWaver GIS is listed as below:

```
twaver.gis
|-event
  |-MapEvent
  |-MapEventListener
  |-MapLayerChangedEvent
  |-MapLayerListener
|-link
  |-InterceptedLink
  |-InterceptedLinkUI
|-gadget
  |-actionbuttons
    |...all the predefined buttons
  |-inputhandler
    |-.....all the predefined input handlers
  |-... some extended gadgets |

|-GeographyMap the class is used to define the map object
|-GeographyLayer the class is used to define the layer object
|-GeographyFeature the class is used to define the feature object
|-GisToolkits
|-TWaverGisConst
|- ... other classes
```

TWaver GIS defines 3 important interfaces to manage geographic data, GeographyMap, GeographyLayer, GeographyFeature. They are designed to present the geographic data and access their properties.



GeographyMap

This type is defined to organize a lot of geographic data as a whole like a map we used usually. It consists of geographic layers, and we can operate it, e.g. zoom in, zoom out, pan, measure distance between geographic points etc.

GeographyLayer

This type is defined to organize a lot of geographic data with similar characteristics as a collection. It consists of geographic features. Accessing directly shape files, developers can specify custom style to present the layer.

Developers can control if the layers are visible according to some specified properties, e.g. the scale of the map. At the same time, developers can query attributes' values of the features belonging to a Query Layer.

GeographyFeature

This type is used to wrap the geographic data. It consists of the geometry attribute and non-geometry attributes of the geographic meta-data.















First Project

Welcome to your first TWaver GIS project !

Getting started

Ensure you have Java and TWaver GIS library

Be sure you have got twaver.jar, twavergis.jar and Java installed. At the same time, please download the relative libraries from [GeoTools](http://www.geo-tools.org/) web site.

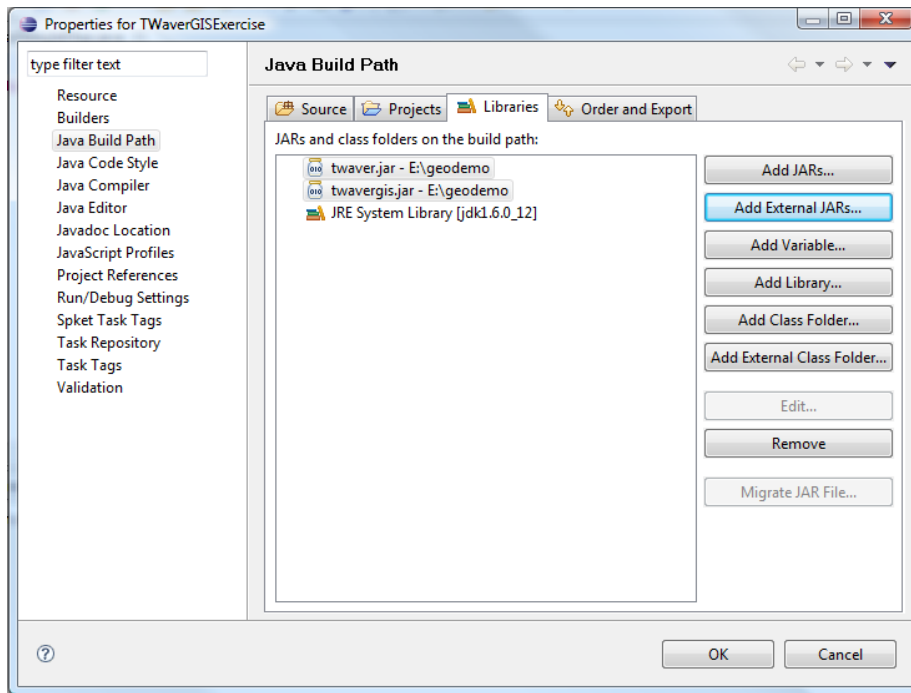
 geoapi-nogenerics-2.1-M2.jar	326 KB
 gt2-api-2.3.3.jar	94 KB
 gt2-coverage-2.3.3.jar	380 KB
 gt2-indexed-shapefile-2.3.3.jar	122 KB
 gt2-main-2.3.3.jar	1,632 KB
 gt2-referencing-2.3.3.jar	1,095 KB
 gt2-render-2.3.3.jar	182 KB
 gt2-shapefile-2.3.3.jar	96 KB
 gt2-shapefile-renderer-2.3.3.jar	58 KB
 jai_codec.jar	253 KB
 jai_core.jar	1,857 KB
 jsr108-0.01.jar	37 KB
 jts-1.7.1.jar	435 KB
 vecmath-1.3.1.jar	284 KB

jai_codec.jar and jai_core.jar can be downloaded from http://java.sun.com/products/java-media/jai/downloads/download-1_1_2.html

Import lib into your project

If you use Eclipse as your IDE, you can follow below steps.

1. Start up Eclipse
2. Open up the **File>New>Java Project** menu, and create a Java project 'TWaverGISExercise'
3. Right click 'TWaverGISExercise' in the project view, start up the **Properties** menu, select **Java Build Path>Libraries** tab, and click **Add External JARs** button to import twaver.jar,twavergis.jar...



As for other IDEs, you can do the same things like above on them.

Create your first class to get map from a shape file named "world.shp".

```
import javax.swing.JFrame;

import twaver.gis.GeographyMap;
import twaver.gis.GisNetworkAdapter;
import twaver.gis.TWaverGisConst;
import twaver.gis.tiles.Executor;
import twaver.network.TNetwork;
public class AccessMap {
    public static void accessMap(){
        JFrame frame = new JFrame("Access Shapefile through TWaverGIS");
        TNetwork network = new TNetwork();
        GisNetworkAdapter adapter = new GisNetworkAdapter(network);
        adapter.installAdapter();
        GeographyMap map = adapter.getMap();
        try {
            map.addLayer("/demo/data/world.shp");
        } catch (IOException e) {
            e.printStackTrace();
        }

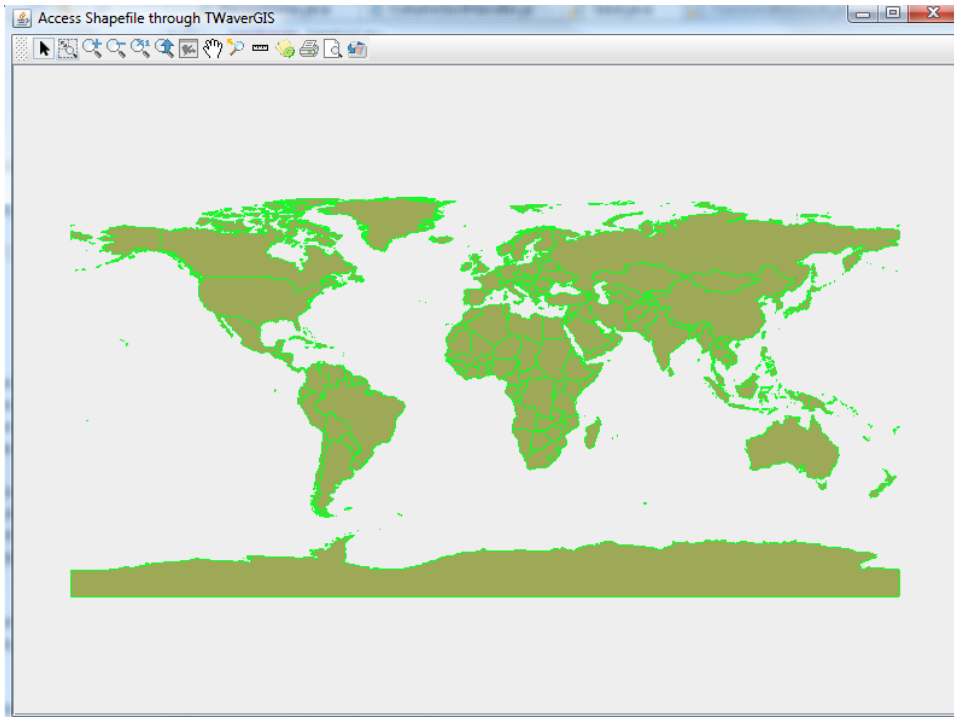
        frame.setContentPane(network);
        frame.setSize(800,600);
        frame.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
        frame.setVisible(true);
        frame.setLocationRelativeTo(null);
    }
    public static void main(String[] args){
        javax.swing.SwingUtilities.invokeLater(new Runnable(){
            public void run() {
                AccessMap.accessMap();
            }
        });
    }
}
```

```

    }
  }
}

```

Then you will get the result like below screenshot :



With above code, you will create a map object, display the map on TNetwork object and add a layer named "world" into the map.

Without specifying the style the layer, TWaver GIS will give you a random style like above figure.

Using the solution

- [Using GeographyMap](#)
- [Using GeographyLayer](#)
- [Using GeographyFeature](#)
- [Handle the events](#)
- [Using GisNetworkAdapter](#)
- [Combination](#)
- [Tips](#)

Using GeographyMap

GeographyMap is designed as a geographic data container. It contains a list of geographic layers. It also can be used to draw a map on a image or some component.

- Create a map
TWaverGIS provides two different kinds of maps which are defined by a attribute named map type.

```
GeographyMap map = GisToolkits.createMap(GeographyMap.GEOGRAPHYMAP_TYPE_SHPFILE);
```

Above statement will create a map which can directly access [shape files](#).

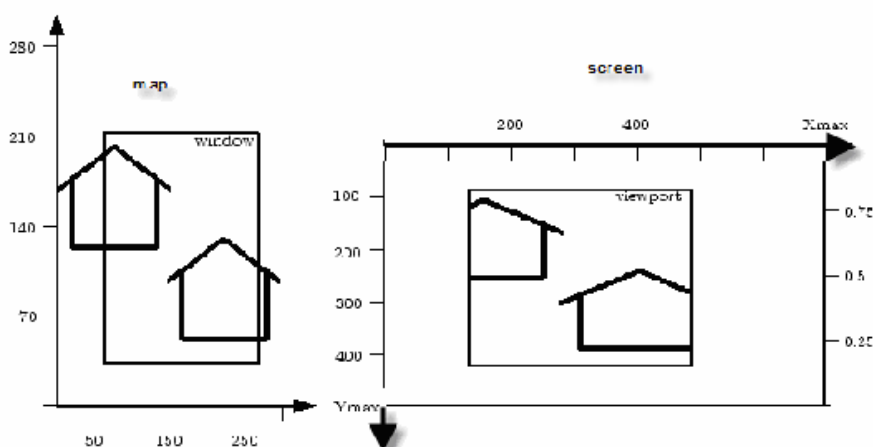
If you want to use the tile mechanism to access the data from map servers, you should use the following code to create the other kind of map.

```
GeographyMap map = GisToolkits.createMap(GeographyMap.GEOGRAPHYMAP_TYPE_TILE);
```

Of course, if you have got a map object, and you want to know the type of the map, you can invoke the method `getMapType`.

```
int type = map.getMapType();
if(GeographyMap.GEOGRAPHYMAP_TYPE_SHPFILE == type){
    System.out.println("Your map can directly access shape files");
}else if(GeographyMap.GEOGRAPHYMAP_TYPE_TILE == type){
    System.out.println("Your map can access the data from a map server");
}
```

- Present a map
TWaver GIS is mainly used to present geographic data. For presenting those data, TWaver GIS map a geographic map to the display devices. TWaver GIS defines its own viewport and window to do that.



From above figure, we know that if we want to display a map on a display device correctly, we should assign the view port volume and map window volume firstly.

```
//granted we have got a map object
//we assign a view port to the map. The view port is 800 pixels wide and 600 pixels height.
map.setViewport(new Rectangle(0,0,800,600));
```

```
//Then TWaver GIS will project the whole geographic data to a area (800 x 600) on
//the display device, such as a monitor, or an image.

//If you want to just display some parts of the geographic data on the display device,
//you should specify the map window of the map.
map.setWindow(-45,45,-30,30);
```

On the other hand, above statements is fit for the map with the type is GeographyMap.GEOGRAPHYMAP_TYPE_SHPFILE.

If you want to project the data from map servers on a display device, you should use the following code.

```
map.setViewport(new Rectangle(0,0,800,600));
//specify the center point of the map by the latitude and longitude coordinate.
//longitude is -73, and latitude is 34
map.setCenterPoint(new GeoCoordinate(-73,34));
map.setZoom(5);
//comments: By default, TWaver GIS assign the coordinate (0,0) as the center point, and 0
level as
//the default zoom level.
```

After the view port and the map window have been assigned correctly, programmers can invoke the following synchronous method to draw the specified geographic data on the screen or an image.

```
BufferedImage buffered = new BufferedImage(800, 600, BufferedImage.TYPE_INT_ARGB);
map.drawMap(buffered.createGraphics(),new Rectangle(0,0,800,600),window);
//comments: For improving the performance, TWaver GIS will paint the map asynchronously.
```



For a TWaver user, if you are using TNetwork as your view component to display your topology data, you do not need to use those above methods. TWaver GIS has wrapped them into GisNetworkAdapter which helps you present a map and operate a map, e.g. panning, zooming, positioning network elements.

- Operate a map
 - Manage layers
 - Add layers

In general, TWaver GIS supports different data source by different data engine which is defined by an Executor type. If you want to add a layer you should specify the data source path and the Executor type at first. If the data is provided by map server, the layer name should be used as the first argument of the addLayer method.

```
map.addLayer("http://10.10.163.11/maps/test.shp",Executor.EXECUTOR_TYPE_SHAPEFILE);
```

or

```
map.addLayer("file:/home/user/maps/test.shp",Executor.EXECUTOR_TYPE_SHAPEFILE);
```

or

```
map.addLayer( "/demo/data/test.shp",Executor.EXECUTOR_TYPE_SHAPEFILE);
```

or

```
//access the WMS data from a GeoServer.
GisManager.registerDefaultSetting(TWaverGisConst.MAPDRIVER_GEOSERVER,
    "http://geoserverpath:8080"+TWaverGisConst.MAPDRIVER_GEOSERVER_POSTFIX);
map.addLayer( "ny",Executor.EXECUTOR_TYPE_GEOSERVER_WMS);
```

Executor.EXECUTOR_TYPE_SHAPEFILE is the default value, so if you add a layer which is created by directly accessing the data of a shape file, the below method can be invoked, and you do not have to specify the Executor type.

```
map.addLayer( "/demo/data/test.shp" );
```

After the layers are added into the map, TWaver GIS will manage the layers by their names or index numbers. Operating a layer, programmers can specify a layer by the name or index.



Data sources supported by TWaver GIS:
Vector data file: Shape file.
Map services: WMS of GeoServer, ArcGIS, MapXtreme.
Tile services of OpenStreetMap, GoogleMap, etc.

- Remove layers

In TWaver GIS, developers can find a specified layer by its layer name or its index in the map. If you want to remove some layer from the map, you can invoke the method below:

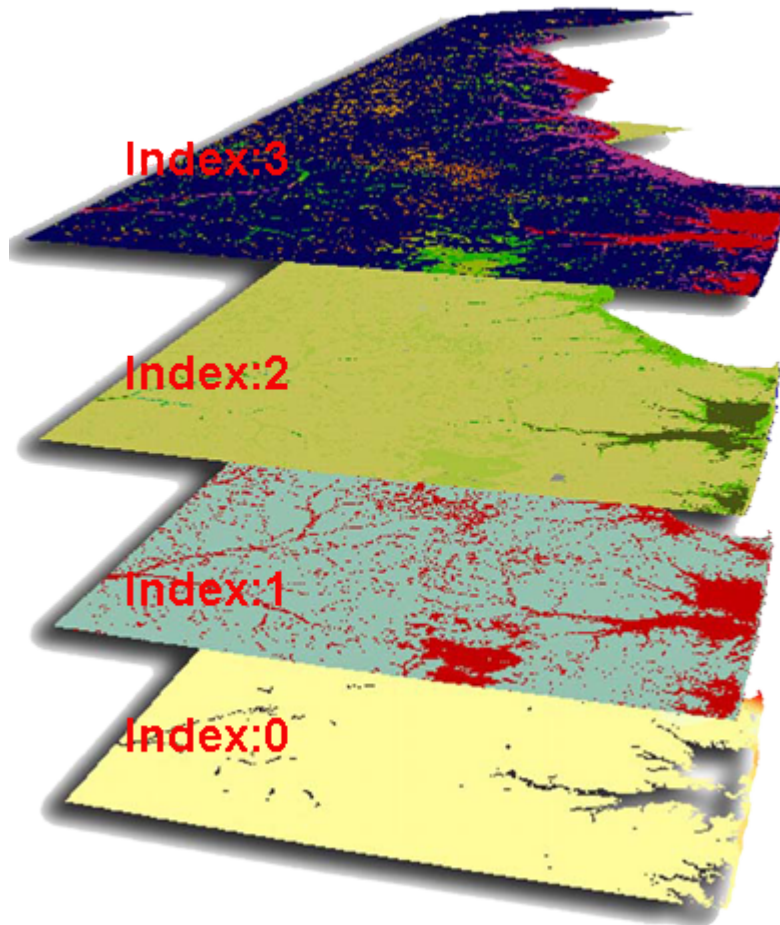
```
map.removeLayer( "ny" );
```

or

```
map.removeLayer(0);
```

- Move layers

GeographyMap presents the layers in order of their index. The layer with a lower index will be drawn firstly.



Programmers can invoke the method to move a layer upward/downward, and even can move a layer to the top/bottom of the

```
map.moveLayer(layerName, layerMovedType);
```

or

```
map.moveLayer("ny", GeographyMap.LAYERMOVE_TYPE_DOWN);
```

or

```
map.moveLayer("sh_bj", GeographyMap.LAYERMOVE_TYPE_TOTOP);
```

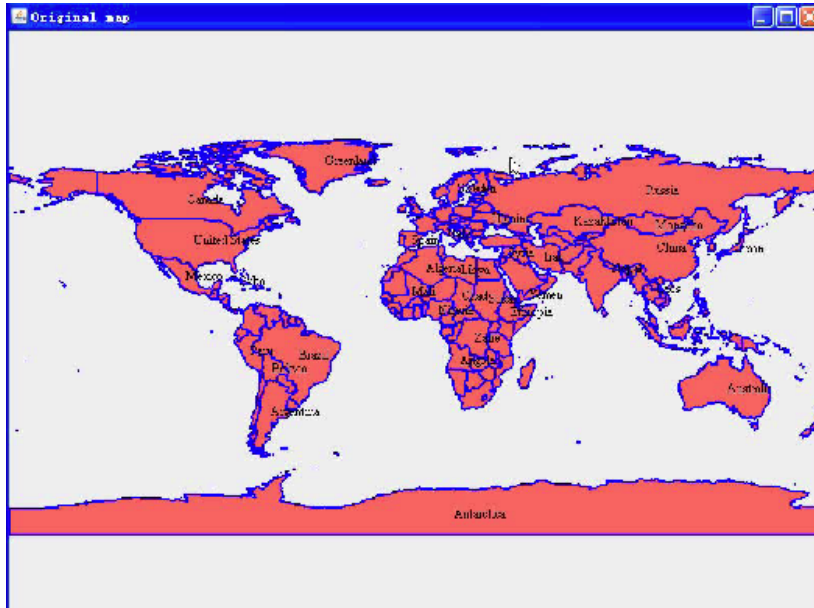
- Specify the style of a layer

When creating a map from shape files directly, programmers should specify the style of the layer, which determines how to draw the layer. GeographyMap object can help do that.

```
Font font = new Font("Dialog", Font.BOLD, 15);
map.setLayerComplexStyle("world",
```

```
Color.BLUE, Color.RED, 2, 0.6, true, font, Color.BLACK, "NAME");
```

With above statement, we can get the 'world' layer painted like below:

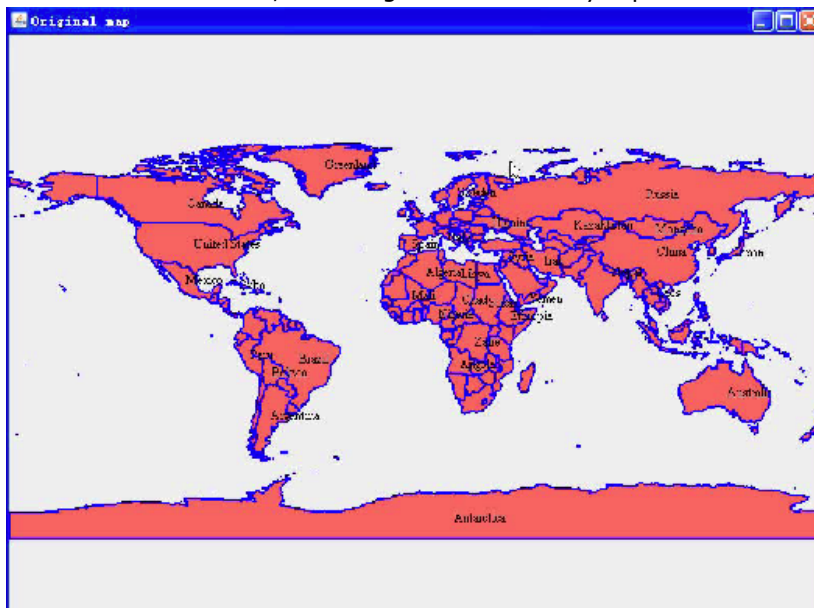


- Specify the style of a layer

When creating a map from shape files directly, programmers should specify the style of the layer, which determines how to draw the layer. GeographyMap object can help do that.

```
Font font = new Font("Dialog",Font.BOLD,15);
map.setLayerComplexStyle("world",
    Color.BLUE, Color.RED, 2, 0.6, true, font, Color.BLACK, "NAME");
```

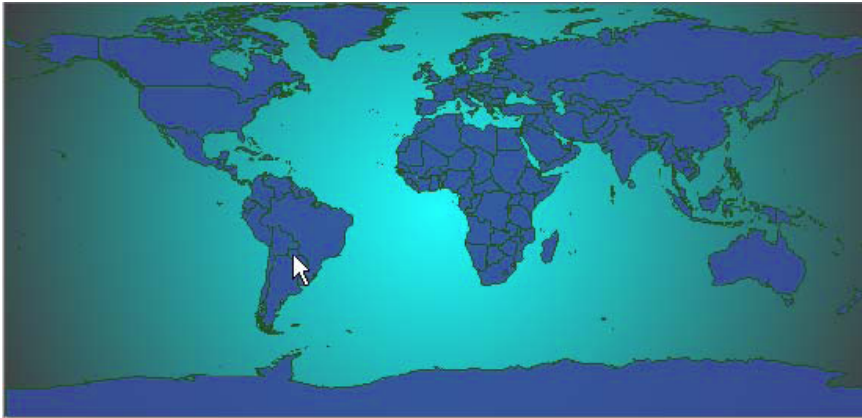
With above statement, we can get the 'world' layer painted like below:



- Manage the background of the map
 - Set up the grounding color of the map

In general, some color should be specified as the grounding color of a map. GeographyMap has a method named `setGroundingColor` to do that.

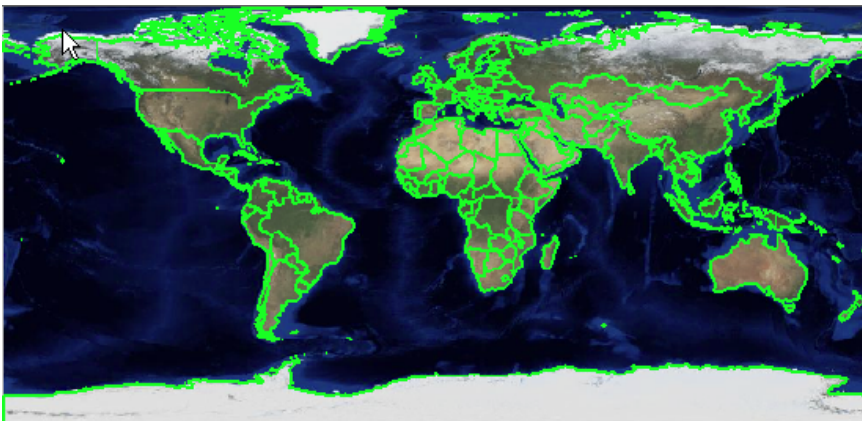
```
map.setGroundingColor(Color.CYAN.brighter());
map.setGroundingStyle(GeographyMap.GROUNDING_STYLE_GRADIENT_ROUND);
```



- ° Raster data is used as the background
Besides color can be set as the background, raster data also can be used to do that.

```
map.setLayerComplexStyle("world",
    Color.GREEN.brighter(), Color.GRAY.brighter(),
    2, 0.3, false, null, null, null);
map.addLayer("/demo/data/world.jpg")
```

OK, enjoy the effect.



Using GeographyLayer

Many geographic layers make up of a geographic map. GeographyLayer is used to describe those layers. A GeographyLayer object manages the style which is used to define how to draw that layer, and a GeographyLayer object manages a lot of geographic data a.k.a. features.

- Set up the style of the layer

The effect of presenting the map depends on the theme of the layers. When we use the data from map servers, the styles have been set by the administrators of the servers. But if programmers load the data directly from shape files, in order to get a colorful map, programmers should assign good style to the layers. Programmers can assign the style either by GeographyMap object or by GeographyLayer object.

The following methods are contained in GeographyLayer object.

```
public void setLayerComplexStyle(Color strokeColor,
    Color fillColor, double width, double alpha, boolean needText,
    Font font, Color fontColor, String property);
public void setLayerComplexStyle(Color strokeColor,
    Color fillColor, double width, double alpha, boolean needText,
    Font font, Color fontColor, boolean withHalo, Color haloColor, String property);
public void setLabelVisible(String attributeName, boolean visible, Color fontColor, Font
    font);
public void addCustomRule(String attributeName, int operator,
    Object value, Color strokeColor, double width, Color fillColor, double alpha);
//comments: More details, please refer to API documents.
```



Only loading data directly from shape files, programmers need to design the style of layers.

By invoking those methods or mixed-use them, programmers can create great themes.

```
//just simply use setlayerComplexStyle method
//and get the same effect as the effect in last chapter
GeographyLayer layer = map.getLayer(0);
layer.setLayerComplexStyle(Color.BLUE, Color.RED, 2, 0.6,
    true, new Font("Default",Font.BOLD,10), Color.BLACK, "NAME");
```

If you want to get halo effect on the labels, you can invoke the second method in above list.

```
layer.setLayerComplexStyle(Color.GRAY, Color.GREEN,
    2, 1, true, new Font("Default",Font.BOLD,10),
    Color.BLACK,true, Color.WHITE, "NAME");
```

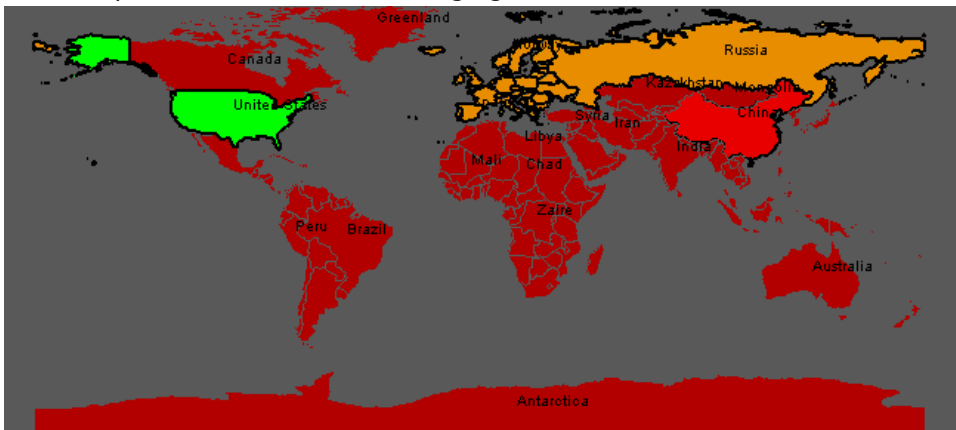
Now, you can get the same effect as below:



Of course, if you want to get more complex effect, you can mixed-use those methods.

```
GeographyLayer layer = map.getLayer("world");
layer.setLabelVisible("NAME", true, Color.BLACK, new Font("Default", Font.BOLD, 10));
layer.addCustomRule(null, TWaverGisConst.OPERATOR_NULL, null, Color.GRAY.darker(), 0,
    Color.RED.darker(), 1);
layer.addCustomRule("NAME", TWaverGisConst.OPERATOR_EQUAL, "United States", Color.BLACK, 2,
    Color.GREEN, 1);
layer.addCustomRule("NAME", TWaverGisConst.OPERATOR_EQUAL, "China", Color.BLACK, 2, Color.RED,
    0.7);
layer.addCustomRule("REGION", TWaverGisConst.OPERATOR_EQUAL, "Europe", Color.BLACK, 2,
    Color.ORANGE, 0.7);
layer.updateDisplay();
//comments: do not forget to invoke updateDisplay method to note the layer.
```

What do you think about the following figure?



- Get features by specified attributes
GeographyLayer object creates many feature objects when loading the shape file. Programmers have to get the features from the corresponding layer. As of getting features, TWaver GIS provides the following methods:

```
public GeographyFeature[] getAllFeatures();
public GeographyFeature[] getFeatures(double longitude, double latitude);
public Collection getFeaturesByAttribute(String attributename, Object value);
//comments: more details, please refer to the corresponding API documents.
{note}
```

At present, only the data is directly loaded from shape files, above methods can work correctly. If the data comes from map servers, TWaver GIS needs to use WFS or other kinds of map services, and [this](#) part of features will be provided soon.

{note}

In general, programmers want to get some feature from a layer by some attribute value. With the following code, you can achieve what you want.

```
GeographyLayer layer = map.getLayer("world");
Iterator iterator = layer.getFeaturesByAttribute("NAME", "US").iterator();
while(iterator.hasNext){
    GeographyFeature feautre = (GeographyFeature)iterator.next();
    //do something to visit the feature's contents
}
```

- Use zoom layer

It is very common that a map contains a lot of layers and every layer has many many features. With such a large amount of data, the performance of presenting a map is not good enough. In this case, it becomes a good solution that according to the zoom scale of the map, developers determine which layer can be visible, and which layer should be hidden. So at some scale level, it is possible that only those necessary layers are drawn and the performance is greatly improved.

In order to get that ability, programmers can invoke the method `setScaleFactor(scaleFactor)` contained in `GeographyLayer`.

```
layer.setScaleFactor(100000);
```

Above code means that only when the scale is bigger than 1:100000 this layer can be visible.

Using GeographyFeature

A map consists of many layers, and a layer consists of a lot of geographic features. In many cases, developers want to get the information of some specified feature. GeographyFeature is designed to present those features. In current, GeographyFeature is readable only. All the GeographyFeature instances are created at the moment when the shape file is loaded into map. Developers can get the features by query the Querying Layer with the relative attributes.

As mentioned in last chapter, programmers can get features from the relative layer. When getting the feature, programmers can traverse the attributes of the feature, or make some other use.

```
Collection result = layer.getFeaturesByAttribute(attributeName, v);
if (result.size() > 0) {
    Object features = result.toArray();
    Rectangle2D bounds = null;
    for (int i = 0; i < features.length; i++) {
        GeographyFeature feature = (GeographyFeature) features[i];
        if (i == 0){
            bounds = feature.getBounds();
        }
        else{
            bounds.add(feature.getBounds());
            System.out.println(feature.getAttribute("NAME"));
        }
    }
    if ((bounds.getWidth() == 0) && (bounds.getHeight() == 0)) {
        map.setWindow(bounds.getMinX() - 0.02, bounds.getMinY() - 0.02,
            bounds.getMinX() + 0.02, bounds.getMinY() + 0.02);
    } else{
        map.setWindow(bounds.getMinX(), bounds.getMinY(), bounds.getMaxX(), bounds.getMaxY());
    }
}
```

Handle the events

- [Using MapEvent](#)
- [Using MapLayerChangedEvent](#)

Using MapEvent

Being operated, GeographyMap object will fire some corresponding map event to indicate the operation finishes. Any observers will receive that event and should do something relative.

A MapEvent is passed to every MapListener object which is registered to receive the "interesting" map events using the map's addMapListener method. A MapEvent object is identified by event type which can be got by invoking MapEvent's getEventType method.

For example, if we set the center coordinates of a map, a map event with the type MAP_WINDOW_CHANGED is generated.

```
map.addMapListener(new MapListener(){
    public void mapChanged(MapEvent evt){
        GeographyMap map = evt.getMap();
        int type = evt.getEventType();
        if(MapEvent.MAP_WINDOW_CHANGED == type){
            int zoom = map.getZoom();
            System.out.println("received map event:"+evt.getEventTypeDescription()+
                ", current zoom is "+zoom);
        }
    }
});
```

Using MapLayerChangedEvent

When some layer of a map is removed, moved, even is set visible or not, the map object will generate some corresponding MapLayerChangedEvent object. The event will be passed to the registered event listener.

For example:

```
map.addMapLayerChangedListener(new MapLayerListener(){
    public void layerChanged(MapLayerChangedEvent evt){
        System.out.println(evt.getLayerName()+
            " ----- event type is "+evt.getEventTypeDescription());
    }
});
```


Using GisNetworkAdapter

GisNetworkAdapter is designed as a wrapper. With this wrapper, programmers can turn a normal TNetwork instance into an instance with GIS supports. After wrapped, the specified TNetwork instance can position topological elements according to their geographic coordinates, require the relative geographic information of a specified coordinate pair, and present the geographic data as the background.

For example:

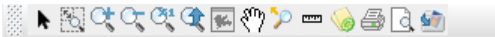
```
GisNetworkAdapter adapter = new GisNetworkAdapter(network);
adapter.installAdapter();
```

With above code, the specified network object will be able to display a map, and locate elements stored in the network's databox at appropriate screen position according to their geographic coordinates. And the network will get a pre-designed toolbar, which help users to operate the map data.

When the GIS supports are not necessary, programmers can remove these supports by invoking GisNetworkAdapter's unInstallAdapter method.

```
adapter.unInstallAdapter();
```

Commonly, programmers need a convenient toolbar in their UI program. TWaver GIS provides the pre-implemented toolbar. After the installAdapter is invoked, a toolbar named TWaverGisConst.TOOLBAR_GISSTAND will be installed either.



Combination

TWaver GIS is designed to help programmers combine TWaver with geographic data easily. In general, TWaver GIS program development consists of several steps presented as below.

- 1.Create a TNetwork componnet;
- 2.Create a GisNetworkAdapter, and wrap the TNetwork component.
- 3.Create network elements with longitude/latitude coordinates, and insert them into the data box of TNetwork component.
- 4.Put TNetwork component into a container and show out.



If you are not familiar with using TNetwork, please refer to [TWaver Developer Guide](#)

```
import java.awt.Color;
import java.awt.Font;
import javax.swing.JFrame;
import twaver.Link;
import twaver.Node;
import twaver.TDataBox;
import twaver.TWaverConst;
import twaver.gis.GeographyLayer;
import twaver.gis.GisNetworkAdapter;
import twaver.gis.TWaverGisConst;
import twaver.gis.link.InterceptedLink;
import twaver.gis.utils.GeoCoordinate;
import twaver.network.TNetwork;
public class Exercise {
    public void combine(){
        JFrame frame = new JFrame("Combination");
        TNetwork network = new TNetwork();
        GisNetworkAdapter adapter = new GisNetworkAdapter(network);
        adapter.installAdapter();
        try {
            adapter.getMap().addLayer("file:/E:/studio/maps/world/world_adm0.shp");
            GeographyLayer layer = adapter.getMap().getLayer(0);
            layer.setLayerComplexStyle(Color.GRAY, Color.GREEN, 2, 1, true, new
Font("Default",Font.BOLD,10),
                Color.BLACK,true, Color.WHITE, "NAME");
            layer.setLayerComplexStyle(Color.BLUE, Color.RED, 2, 0.6, true, new
Font("Default",Font.BOLD,10), Color.BLACK, "NAME");
            layer.addCustomRule(null, TWaverGisConst.OPERATOR_NULL, null, Color.GRAY.darker(), 5,
Color.GRAY.darker(), 1);
            layer.addCustomRule(null, TWaverGisConst.OPERATOR_NULL, null, Color.WHITE.darker(), 3,
Color.WHITE.darker(), 1);
            layer.updateDisplay();
        } catch (Exception e) {
            e.printStackTrace();
        }
        TDataBox box = network.getDataBox();
        Node node = new Node();
        box.addElement(node);
        node.putClientProperty(TWaverGisConst.GEOCOORDINATE,
            new GeoCoordinate(-73.12,34.01));
        Node messageNode = new Node();
        messageNode.putClientProperty(TWaverGisConst.GEOCOORDINATE,
            new GeoCoordinate(120.11,33.98));
        box.addElement(messageNode);
        Link l = new InterceptedLink(node, messageNode);
        l.putLinkStyle(TWaverConst.LINK_STYLE_DASH);
    }
}
```

```
box.addElement(1);
frame.setContentPane(network);
frame.setSize(400,300);
frame.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
frame.setVisible(true);
frame.setLocationRelativeTo(null);
}
public static void main(String[] args){
    javax.swing.SwingUtilities.invokeLater(new Runnable(){
        public void run() {
            Exercise.combine();
        }
    });
}
```

Tips

- [Coordinate Transformation](#)
- [Print Map](#)
- [Synchronize zoom](#)
- [Using gadgets](#)
- [Using InterceptedLink](#)

Coordinate Transformation

TWaver GIS provides methods for coordinate transformations between longitude/latitude and screen coordinates.

```
GeoCoordinate convertScreenToLatLong(GeographyMap map,Point2D screenPoint)
GeoCoordinate convertScreenToLatLong(GeographyMap map,double screenX,double screenY)
Point2D convertLatLongToScreen(GeographyMap map,double longitude,double latitude)
Point2D convertLatLongToScreen(GeographyMap map,GeoCoordinate co)
```

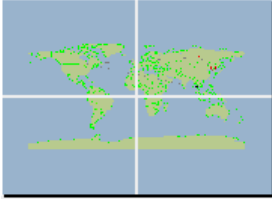
With above listed methods, programmers can meet the requirement of coordinate transformation.

For example, if you want to monitor the motion of the mouse, and print the longitude/latitude coordinates of the mouse's track, you can use the code as below:

```
TNetwork network = new TNetwork();
GisNetworkAdapter adapter = new GisNetworkAdapter(network);
adapter.installAdapter();
network.getCanvas().addMouseMotionListener(new MouseMotionListener() {
    public void mouseDragged(MouseEvent e){
    }
    public void mouseMoved(MouseEvent e){
        GeoCoordinate co = GisToolkits.convertScreenToLatLong(map,e.getPoint());
        if (co != null){
            System.out.println("\n"+co.getLongitude()+" "+co.getLatitude()+"\n");
        }
    }
});
```

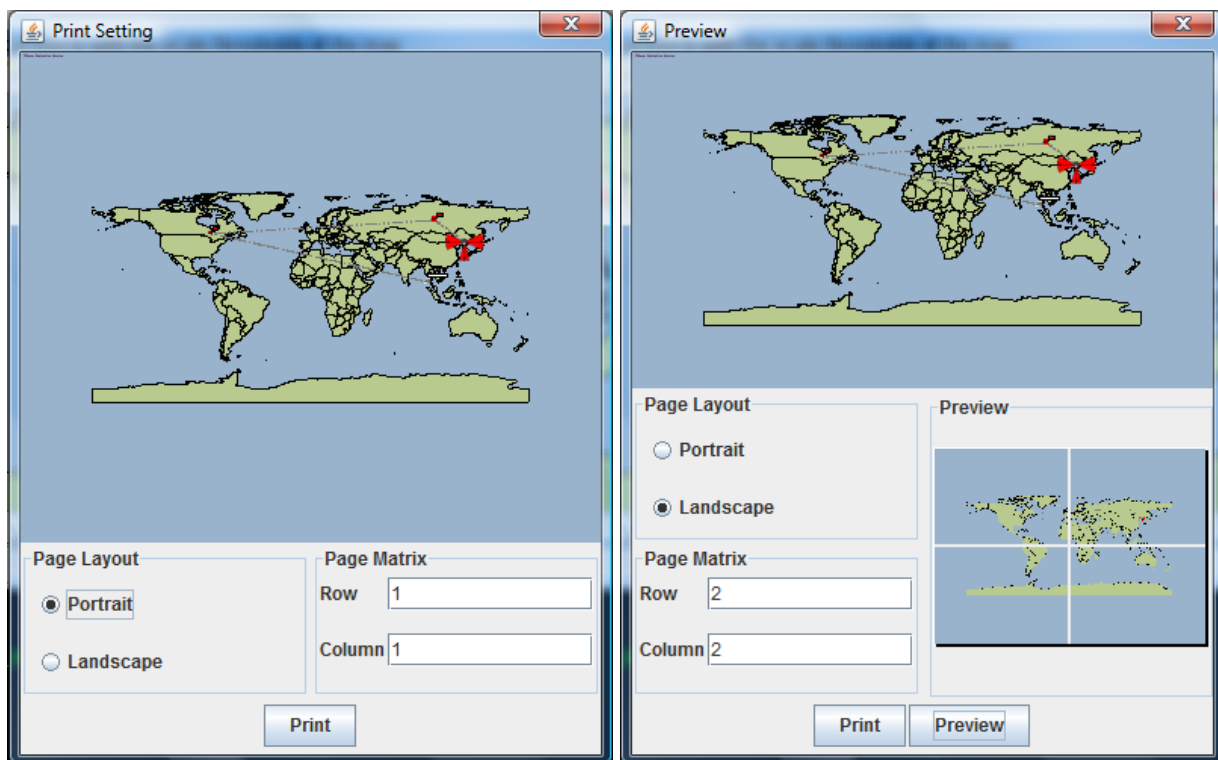
Print Map


TWaver GIS provides the ability of printing map with the manner of WYSIWYG (What You See Is What You Get). TWaver GIS can prints a map with the portrait or landscape layout. And it also print the map in page grids pattern.



TWaver GIS also provides public method to pop up a print setting dialog or print preview dialog which can help users choose the print options.

```
GisToolkits.showPrintDialog(TNetwork network, GeographyMap map, boolean isPreview)
```



 Extended ability: There is a print regional diagram in the top of the print or print preview dialog box. That diagram is used to tell users which area will be printed. By default the whole of the diagram will be printed. But if the user is not satisfied with the area, and he wants to print a part of the area of the diagram, he can just select a rectangle area on the diagram directly as the new print region. He does not have to close current setting dialog, return back to the main interface, and select the new region.



Synchronize zoom

In general, programmers need to locate some network element on a map. When programmers use the `GisNetworkAdapter` to wrap the `TNetwork` component, the network element's size will remain unchanged as the map is zoomed out/in by default.

But in some cases, the customers need the size changed together with the map's zoom level.

To meet that kind of demand, TWaver GIS provides a mechanism named "threshold". The threshold is described by a pair of integers. One stands for the threshold level, and the other integer stands for the denominator of the map's current scale. Programmers can register several thresholds of the scale of the map with the method `registerThresholds` contained in `GeographyMap` type. This method needs two parameters, separately are the threshold level and the relative scale.

After invoking that method, the `GeographyMap` object will keep a map between the scale and threshold level. If the real scale of the map is no smaller than the registered scale of level A, and is bigger than registered scale of level B. The threshold level of that real scale is level B. If the real scale is bigger than the maximal registered scale of the thresholds, the threshold level is level 1. If the real scale is smaller than the minimal registered scale, the threshold level is the registered maximal level.

As you operate the map, if the scale changes and meets some threshold - bigger or smaller than the denominator stored in the threshold pair - TWaver GIS will automatically synchronize the network element's size with the map's scale. TWaver GIS will keep the original size of a network element at level 1. If you register N thresholds into a `GeographyMap` object, and the current threshold level is X, the element's size will be $(N-X+1)/N$ of its original size.

For example, you register a group of thresholds with the following code:

```
public void registerThresholds() {
    GeographyMap map = getMap();
    int[] thresholdScales = { 16165314, 3692668, 2000000, 1000000, 500000, 300000, 100000, 80000 };
    for (int i = 0; i < thresholdScales.length; i++) {
        //register the threshold pair, (level, denominator of the relative scale)
        map.registerThresholds(thresholdScales.length - i, thresholdScales[i]);
    }
}
```

With above code, eight threshold pairs have been registered into the map. They are listed as below:

Threshold Level	Relative scale
1	1:80000
2	1:100000
3	1:300000
4	1:500000
5	1:1000000
6	1:2000000
7	1:3692668
8	1:16165314

Assuming that you are viewing a map with the scale 1:1500000, when you zoom in the map, if the scale jumps over 1:1000000, the network element which is located on the map will change its size. The size will become into four eighths of the original size.

Using gadgets

A special package named `twaver.gis.gadget` is defined to provide some little gadgets.

As last chapter mentioned, TWaver GIS provides predefined toolbars. All the buttons on the toolbars and the relative actions are included in this package.

In some cases, programmers also need to create a print setting dialog or a status bar at the bottom of the interface. Those kinds of gadgets are provided in the package.

If you want to display a print setting dialog, you can use the following code:

```
GisToolkits.showPrintDialog(network, map, true);
```

In above statement, `network` means the relative `TNetwork` object, `map` means the specified `GeographyMap` object which you want to print, and `true` means that a print preview dialog will pop out. If you want to display the print dialog without preview setting, you can assign `false` to that parameter.

```
GisToolkits.showPrintDialog(network, map, false);
```

Please refer to [Print Map](#) to get more details about print dialog.

TWaver GIS provides a status bar. It can be used to monitor the mouse's motion, and display current scale of the specified map, the longitude/latitude coordinate of the mouse and cooperating with the default measuring distance input handler to display the distance between specified points. You can use the method as below to display that status bar:

```
JPanel mainPanel = new JPanel(new BorderLayout());
....
mainPanel.add(new StatureBar(map,network.getCanvas()),BorderLayout.SOUTH);
```

You will get the following effect:

Scale : 1:24141900	Longitude: E115°25'24"	Latitude: N74°31'9"	Distance : 4644.27 km
--------------------	------------------------	---------------------	-----------------------

Using InterceptedLink

In some cases, TWaver users perhaps want to set the style of a link as 'TWaverConst.LINK_STYLE_DASH' in order to describe some state of the link. If the link is very long, TWaver will not be able to draw this link efficiently, because TWaver have to calculate the path of the whole long link. Programmers, who want to combine TNetwork component with the location service, will often meet this problem.

For example, you locate two network elements on different positions, which longitude/latitude coordinates respectively are (-73.12,34.01) and (120.11,33.98). There is a link between these two elements. If the scale of the map is changed into 1:10000, which is a big scale, the link will stretch very very long.

TWaver GIS provides a new kind of link which is named InterceptedLink so solve this problem. No matter how long a InterceptedLink is, TWaver will just calculate its visible part on the screen. That can improve the performance greatly.

```
TDataBox box = network.getDataBox();
Node node = new Node();
box.addElement(node);
node.putClientProperty(TWaverGisConst.GEOCOORDINATE,
    new GeoCoordinate(-73.12,34.01));
Node messageNode = new Node();
messageNode.putClientProperty(TWaverGisConst.GEOCOORDINATE,
    new GeoCoordinate(120.11,33.98));
box.addElement(messageNode);
Link l = new InterceptedLink(node, messageNode);
l.putLinkStyle(TWaverConst.LINK_STYLE_DASH);
box.addElement(l);
```