GeoTemCo

Comparative Visualization of Geospatial-Temporal Data

Overview

The most important steps for the usage of GeoTemCo are explained in this section. For this tutorial, we regard the common case of one geospatial dimension to be displayed in a map widget, one temporal dimension to be displayed in a time widget and one table widget that shows the data items' details. The accepted data standards and the formats to be fulfilled for displaying datasets with GeoTemCo are:

KML
JSON

The following 5 steps need to be done to display datasets in GeoTemCo's widgets are:

1. Include Files
2. Add Required Divs
3. Initialize Widgets
4. Load Datasources
5. Display Data

Other selected topics for setting up and using GeoTemCo are:

Map Configuration
Time Configuration
Table Configuration
Accepted Date Formats
Levels of detail for place names
Multiple Map or Time Dimensions
Publish/Subscribe-Mechanism

Please, also inspect the written source code snippets in the Live Examples' section for detailed overview.

KML

One data format which can be displayed with GeoTemCo is KML. A detailed overview is given in the KML Reference. The general structure of KML is:

```
<kml>
    <folder>
        ...
        <placemark>
            ...
        </placemark>
        <placemark>
            ...
        </placemark>
        ...
    </folder>
</kml>
```

The Folder-Tag holds a list of data items of a dataset. KML fits properly with the common case of 3 widgets (map, time, table). The following information for a single data item can be captured inside a KML's Placemark-Tag:

- **item name**: short name describing the data item

- **item description**: (detailed) description for the data item, which can also contain HTML fragments, e.g., thumbnails; it will be used for popup windows on the map
- **longitude,latitude**: location which is associated with the data item in form of float values
- **place name**: name of the place which is associated with the given latitude/longitude pair; Levels of Detail are also allowed
- **date**: time stamp which is associated with the data item (see section Accepted Date Formats)

Finally, a Placemark entry needs to be converted into the following structure:

```
<placemark>
    <name>item name</name>
    <description>item description</description>
    <address>place name</address>
    <point>
        <coordinates>longitude,latitude</coordinates>
    </point>
    <timestamp>
        <when>date</when>
    </timestamp>
</placemark>
```

Note: If a data's entry can contain special characters, e.g. its description, additionally frame it with the CDATA tag:

```
<description>
    <![CDATA[item description]]>
</description>
```

JSON

Another accepted format in JSON. Compared to KML its structure is not standardized, but it is more compact. Therefore, the size of a dataset is smaller, which makes significant performance differences especially for large datasets. The global structure of a dataset is an array with a JSON entry for each data item.

```
[
    ...
    dataItem,
    dataItem,
    dataItem,
    ...
]
```

Each dataItem contains the given geospatial and temporal information. Furthermore, there are several possibilities for a data items' description. A single data item entry looks like:

```
{
    "id": someId,
    "name": someName,
    "description": someDescription,
    "lon": someLongitude,
    "lat": someLatitude,
    "place": somePlaceName,
    "time": someDate,
    "tableContent": someValuesList
}
```

The values of the attributes are slightly different compared to KML:

- **id**: an optional (system) id for the data item, it will be used for the mapping during the user interactions between the widgets; if it is not defined, a running index will be used
- **name**: short name describing the data item
- **description**: (detailed) description for the data item, which can also contain HTML fragments, e.g., thumbnails; it will be used for popup windows on the map
- **lon**: longitude value of the data items associated location as a float value
- **lat**: latitude value of the data items associated location as a float value
- **place**: name of the place which is associated with the given latitude/longitude pair; Levels of Detail are also allowed
- **time**: a time stamp which is associated with the data item (see section Accepted Date Formats)
- **tableContent**: a list of attribute/value pairs which will be shown in the table widget

1. Include Files

GeoTemCo uses the JavaScript libraries OpenLayers for the map widget and SIMILE Timeplot for the time widget. Both sources, other required JavaScript libraries and the GeoTemCo JavaScript files are combined and minified in one GeoTemCo source file. In the Source Code section, you can download the bundle version which you extract into your client folder. Then, include the GeoTemCo sources into the head environment of the HTML-page with:

```
<head>
    ...
    <link rel="stylesheet" href="geotemco_directory/css/geotemco.css" type="text/css"/>
    <script src="geotemco_directory/geotemco-min.js"/>
    ...
</head>
```

2. Add Required Divs

Dependent on the given data source you need to place a specific number of HTML containers for initializing different widgets to show the data's metadata. For the common case, 3 divs for 3 widgets (map, time and table) are required:

```
<div id="someMapDivName"></div>
<div id="someTimeDivName"></div>
<div id="someTableDivName"></div>
```

3. Initialize Widgets

Each widget needs 2 things for its initialization: its container div and a corresponding wrapper. A wrapper can be seen as an interface, which manages interaction between the different widgets. Additionally, you can give a JSON set of options for each widget.

```
var mapDiv = document.getElementById("someMapDivName");
var map = new WidgetWrapper();
var mapWidget = new MapWidget(map,mapDiv,options);

var timeDiv = document.getElementById("someTimeDivName");
var time = new WidgetWrapper();
var timeWidget = new TimeWidget(time,timeDiv,options);

var tableDiv = document.getElementById("someTableDivName");
var table = new WidgetWrapper();
var tableWidget = new TableWidget(table,tableDiv,options);
```

An overview of options you can set for map, time and table widget are given in: Map Configuration, Time Configuration and Table Configuration.
Note: There are no implementation dependencies between the widgets. All widgets only share the same data and the same interaction scenarios on the data. So, none of the widgets is mandatory! Hence, it is possible to leave out, e.g., the time widget, if there is no temporal metadata available.

4. Load Datasources

The needed function call depends on your data format choice. If you need to retrieve the files you can use the following request to get the XML tree for KML's via:

```
var kmlFile = GeoTemConfig.getKml('someKmlUrl');
```

For JSON files use:

```
var jsonFile = GeoTemConfig.getJson('someJsonUrl');
```

Having the files, you can load the data with:

```
var kmlData = GeoTemConfig.loadKml(kmlFile);
```

For JSON files use:

```
var jsonData = GeoTemConfig.loadJson(jsonFile);
```

5. Display Data

Each GeoTemCo widget always expects an array of Dataset objects for displaying. Since the aggregation procedures for map and time widget takes most of the initialization time, we waived on implementing some sort of "addDataset"- or "removeDataset"-functions. A simple way of initializing GeoTemCo with a Dataset array is:

```
// empty array for datasets
var datasets = [];

// a series of loading datasets (in this case json files)
...
var jsonFile = GeoTemConfig.getJson('someJsonUrl');
var jsonData = GeoTemConfig.loadJson(jsonFile);
var dataset = new Dataset(jsonData,'someDatasetLabel');
datasets.push(dataset);
```

The labels are used for the table's tabs. Finally, we load the data into the widgets with:

```
map.display(datasets);
time.display(datasets);
table.display(datasets);
```

After these 5 steps, the data should be shown in all widgets.

Map Configuration

Below, you can find the standard configuration of the map widget:

```
mapWidth : false, // false or desired width css definition for the map
mapHeight : '580px', // false or desired height css definition for the map
mapTitle : 'GeoTemCo Map View', // title will be shown in map header
mapIndex : 0, // index = position in location array; for multiple locations the 2nd map refer
alternativeMap : false, // alternative map definition for a web mapping service or 'false' fo
/* an example:
 {
 name: 'someMapName',
 url: '/geoserver/wms',
 layer: 'namespace:layerName'
 }
 */
googleMaps : false, // enable/disable Google maps (actually, no Google Maps API key is requir
bingMaps : false, // enable/disable Bing maps (you need to set the Bing Maps API key below)
bingApiKey : 'none', // bing maps api key, see informations at http://bingmapsportal.com/
osmMaps : true, // enable/disable OSM maps
baseLayer : 'Open Street Map', // initial layer to show (e.g. 'Google Streets')
resetMap : true, // show/hide map reset button
countrySelect : true, // show/hide map country selection control button
polygonSelect : true, // show/hide map polygon selection control button
circleSelect : true, // show/hide map circle selection control button
squareSelect : true, // show/hide map square selection control button
multiSelection : true, // true, if multiple polygons or multiple circles should be selectable
popups : true, // enabled popups will show popup windows for circles on the map
olNavigation : false, // show/hide OpenLayers navigation panel
olLayerSwitcher : false, // show/hide OpenLayers layer switcher
olMapOverview : false, // show/hide OpenLayers map overview
olKeyboardDefaults : true, // (de)activate Openlayers keyboard defaults
olScaleLine : false, // (de)activate Openlayers keyboard defaults
geoLocation : true, // show/hide GeoLocation feature
boundaries : {
    minLon : -29,
    minLat : 35,
    maxLon : 44,
    maxLat : 67
}, // initial map boundaries or 'false' for no boundaries
mapCanvasFrom : '#9db9d8', // map widget background gradient color top
mapCanvasTo : '#5783b5', // map widget background gradient color bottom
labelGrid : true, // show label grid on hover
maxPlaceLabels : 6, // Integer value for fixed number of place labels: 0 --> unlimited, 1 -->
selectDefault : true, // true, if strongest label should be selected as default
maxLabelIncrease : 2, // maximum increase (in em) for the font size of a label
labelHover : false, // true, to update on label hover
ieHighlightLabel : "color: COLOR1; background-color: COLOR0; filter:'progid:DXImageTransform.I
highlightLabel : "color: COLOR0; text-shadow: 0 0 0.4em black, 0 0 0.4em black, 0 0 0.4em bla
ieSelectedLabel : "color: COLOR1; font-weight: bold;", // css code for a selected place label
selectedLabel : "color: COLOR1; font-weight: bold;", // css code for a selected place label
ieUnselectedLabel : "color: COLOR1; font-weight: normal;", // css code for an unselected plac
unselectedLabel : "color: COLOR1; font-weight: normal;", // css code for an unselected place
ieHoveredLabel : "color: COLOR1; font-weight: bold;", // css code for a hovered place label i
hoveredLabel : "color: COLOR1; font-weight: bold;", // css code for a hovered place label
circleGap : 0, // gap between the circles on the map (>=0)
minimumRadius : 4, // minimum radius of a circle with mimimal weight (>0)
circleOutline : true, // true if circles should have a default outline
circleTransparency : true, // transparency of the circles
minTransparency : 0.4, // maximum transparency of a circle
maxTransparency : 0.8, // minimum transparency of a circle
binning : "generic", // binning algorithm for the map, possible values are: 'generic', 'squar
noBinningRadii : "dynamic", // for 'no binning': 'static' for only minimum radii, 'dynamic' f
circlePackings : true, // if circles of multiple result sets should be displayed in circle pa
binCount : 10, // number of bins for x and y dimension for lowest zoom level
showDescriptions : true, // true to show descriptions of data items (must be provided by kml/
mapSelection : true, // show/hide select map dropdown
binningSelection : false, // show/hide binning algorithms dropdown
mapSelectionTools : true, // show/hide map selector tools
dataInformation : true, // show/hide data information
overlayVisibility : false, // initial visibility of additional overlays
proxyHost : ''  //required for selectCountry feature, if the requested GeoServer and GeoTemCo
```

To overwrite, e.g. the values for 'mapTitle' and 'googleMaps', you just need to initialize the map widget with:

```
var mapWidget = new MapWidget(map,mapDiv,{
    mapTitle: 'Some Map Title',
    googleMaps: true
});
```

Time Configuration

Below, you can find the standard configuration of the time widget:

```
timeTitle : 'GeoTemCo Time View', // title will be shown in timeplot header
timeIndex : 0, // index = position in date array; for multiple dates the 2nd timeplot refers
timeWidth : false, // false or desired width css definition for the timeplot
timeHeight : '100px', // false or desired height css definition for the timeplot
defaultMinDate : new Date(2012, 0, 1), // required, when empty timelines are possible
defaultMaxDate : new Date(), // required, when empty timelines are possible
timeCanvasFrom : '#EEE', // time widget background gradient color top
timeCanvasTo : '#EEE', // time widget background gradient color bottom
rangeBoxColor : "white", // fill color for time range box
rangeBorder : "1px solid #de7708", // border of frames
dataInformation : true, // show/hide data information
rangeAnimation : true, // show/hide animation buttons
scaleSelection : true, // show/hide scale selection buttons
linearScale : true, // true for linear value scaling, false for logarithmic
unitSelection : true, // show/hide time unit selection dropdown
timeUnit : -1 // minimum temporal unit (SimileAjax.DateTime or -1 if none) of the data
```

To overwrite, e.g. the values for 'timeTitle' and 'timeUnit', you just need to initialize the time widget with:

```
var timeWidget = new TimeWidget(time,timeDiv,{
    timeTitle: 'Some Time Title',
    timeUnit: SimileAjax.DateTime.YEAR
});
```

Table Configuration

Below, you can find the standard configuration of the table widget:

```
tableWidth : false, // false or desired width css definition for the table
tableHeight : false, // false or desired height css definition for the table
validResultsPerPage : [10, 20, 50, 100], // valid number of elements per page
initialResultsPerPage : 10, // initial number of elements per page
tableSorting : true, // true, if sorting of columns should be possible
tableContentOffset : 250, // maximum display number of characters in a table cell
tableSelectPage : true, // selection of complete table pages
tableSelectAll : false, // selection of complete tables
tableShowSelected : true, // show selected objects only option
unselectedCellColor : '#EEE' // color for an unselected row/tab
```

To overwrite, e.g. the values for 'tableSorting' and 'unselectedCellColor', you just need to initialize the table widget with:

```
var tableWidget = new TableWidget(table,tableDiv,{
    tableSorting: false,
    unselectedCellColor: 'white'
});
```

Accepted Date Formats

GeoTemCo accepts dates in the following formats in XML time (see XML Schema Part 2: Datatypes Second Edition):

- gYear (YYYY), e.g., "2012"
- gYearMonth (YYYY-MM), e.g., "2012-07"
- date (YYYY-MM-DD), e.g., "2012-07-17"
- dateTime (YYYY-MM-DDThh:mm:ssZ), e.g., "2012-07-17T19:27:32Z"

Note: We also allow negative years, which is important for historical data! In such a case, just prepend the date string with a minus character.

Levels of detail for place names

For places, a unique place name is shown at all zoom levels, but we are also able to separate different levels of detail. We accept 4 different level place names describing a data items' location which are shown at proper map resolutions:

- **country**: the name of the corresponding country, e.g., USA, Germany, Italy, ...

- **region**: the name of the region, which can be a political territory (e.g. states like Arizona or Utah) or countrysides (e.g., Rocky Mountains, Yellowstone, ...)
- **city**: the name of the corresponding city, e.g., Washington D.C., Berlin, Rome, ...
- **borough**: the name of an urban place (e.g., an address) or district (e.g., downtown)

For this feature, we still use the place name field. To separate the levels of detail, we use "/" as delimiter. As an example, we can map the location of the "Statue of Liberty" as:

```
USA/New York/New York City/Liberty Island
```

For missing levels, the coarser level name is shown. In such a case, leave unknown level names empty like:

```
USA//New York City/Liberty Island
```

Multiple Map or Time Dimensions

For data with multiple geospatial and/or temporal dimensions, you need to perform some modifications. For a better understanding, an example which requires 2 map and 2 time widgets. Therefore, suppose a set of excavations artifacts with:

- the excavation site, where an artifact was found (map1)
- the place, where an artifact is actually located (map2)
- the date, when the artifact was found (time1)
- the period, the artifact belongs to (time2)

Since we do not want to modify the standard KML structure, the usage of the JSON format is mandatory for such data. For multiple geospatial dimensions, we replace the values

```
{
    ...
    "lon": someLongitude,
    "lat": someLatitude,
    "place": somePlaceName,
    ...
}
```

with a location array that holds multiple places (in this case 2 places):

```
{
    ...
    "location": [
        { "lon": someLongitude1, "lat": someLatitude1, "place": somePlaceName1 },
        { "lon": someLongitude2, "lat": someLatitude2, "place": somePlaceName2 }
    ],
    ...
}
```

For multiple temporal dimensions you simply need to replace the date value with an array of dates:

```
{
    ...
    "time": [ someDate1, someDate2 ],
    ...
}
```

Each dimension requires its own div and wrapper. For the example of 2 geospatial and 2 temporal dimensions, we instantiate 2 MapWidget objects (map1, map2) and 2 TimeWidget objects (time1, time2) with:

```
var mapDiv1 = document.getElementById("someMapDivName1");
var map1 = new WidgetWrapper();
var mapWidget1 = new MapWidget(map1,mapDiv1,{
    mapIndex: 0
});
var mapDiv2 = document.getElementById("someMapDivName2");
var map2 = new WidgetWrapper();
var mapWidget2 = new MapWidget(map2,mapDiv2,{
    mapIndex: 1
});

var timeDiv1 = document.getElementById("someTimeDivName1");
var time1 = new WidgetWrapper();
var timeWidget1 = new TimeWidget(time1,timeDiv1,{
    timeIndex: 0
});
var timeDiv2 = document.getElementById("someTimeDivName2");
var time2 = new WidgetWrapper();
var timeWidget2 = new TimeWidget(time2,timeDiv2,{
    timeIndex: 1
```

For the widgets showing the second dimensions (map2, time2), you need to set the option 'mapIndex' to the dedicated array position 1. The standard mapIndex-value is 0. These are all changes for displaying multiple geospatial or temporal dimensions.

Publish/Subscribe-Mechanism

GeoTemCo uses an own mechanism for subscribing functions to be called when specific topics are published. The most important topics are:

- 'highlight': called, when the user highlights data items
- 'selection': called, when the user selected portions of the data
- 'filter': called, when the user filters for a previous performed selection; all items outside the selection are dropped

If you may add other widgets to your systems, which should be updated after such an event, you can easily subscribe for, e.g. the 'filter' topic with:

```
var object = this;
Publisher.Subscribe('filter',object,function(data){
    // any code here, e.g.:
    // object.filter(data);
});
```

The "object" variable is the object which subscribes for the topic. "data" contains all data items within the 'filter' event. You are also able to publish such an event with:

```
Publisher.Publish('filter',data,object);
```

Now, the "object" value is used to avoid calling subscribed functions from "object". If the subscribed function from "object" still should be called you can publish like:

```
Publisher.Publish('filter',data,null);
```

You can use this mechanism to publish and subscribe objects for any topic you like.

Copyright ©2012 Stefan Jänicke.
All rights reserved.
Contact us at stjaenicke{at}informatik{dot}uni-leipzig{dot}de.