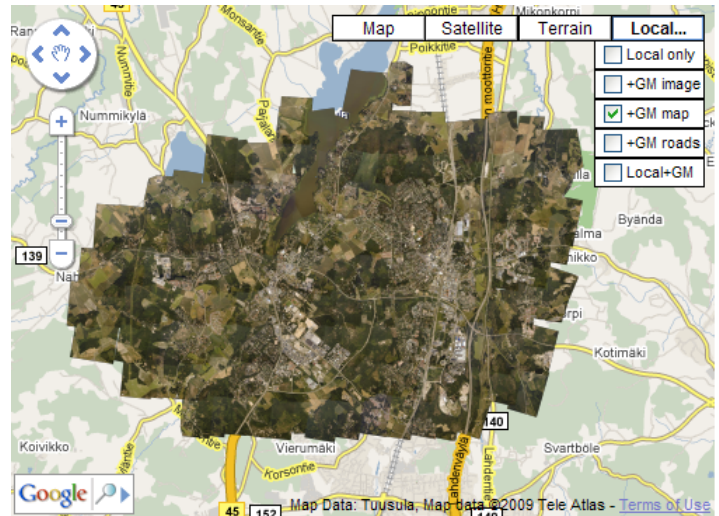


Publishing Custom Google Maps on Your Web Site

Once you have created a Google Maps Tile Overlay from your image(s) or map data in the TNTmips Auto Mosaic process, you can easily publish the Tile Overlay on your web site. Mosaic creates a sample HTML file that defines a set of custom maps (mashups) that combine the Tile Overlay with map and image layers from Google Maps. This file also sets up map selection controls to allow a choice among these custom maps or Google's standard maps. You can post this HTML file directly as a web page on your web site along with the Tile Overlay, allowing anyone to view the custom maps with your Tile Overlay in their browser. You can also edit the HTML file to change the selection of available custom maps, change the choice of controls to be included with the maps, provide additional gadgets, or add your own content to the web page. Annotated excerpts of this sample HTML file are included on these pages as a guide to its structure, providing a starting point for modifying the content.

The sample HTML file includes several JavaScript sections that load and use the Google Maps Application Programming Interface (API) to define the layers used in the various custom map combinations and load them in Google Maps. All the uses of predefined Google Maps API elements and the other code sections **mandated by Google** are highlighted in magenta in the sample file excerpts. Google provides an online Google Maps developer's guide and API reference at <http://code.google.com/apis/maps>.

To use the Google Maps API on your web site you also need a Google Maps API key. The Google Maps Key button in the Auto Mosaic process (see the Technical Guide entitled *Mosaic: Mosaic to Google Maps Tile Overlay*) takes you to the Google web page where you can sign up for your key, which is valid only within a single web domain. If you entered your API key in the Mosaic process, it is automatically incorporated in the sample HTML file and ready for use.



Google Maps custom map (mashup) with local Tuusula, Finland orthoimage Tile Overlay on top of Google's map. The sample HTML file created with the Tile Overlay in the TNTmips Mosaic process sets up a Local menu button with a series of custom map choices that show various combinations of the local Tile Overlay and Google's map and image layers.

Some Key Classes in the Google Maps API

- GTileLayer:** defines a single multiresolution tile layer to be used in a custom map; it can be from your local Tile Overlay or a layer from a standard Google map type.
- GMapType:** defines a custom map type with one or more tile layers; provides access to the standard Google Maps map types and their tile layers.
- GMap2:** core class used to define a "map" (usually a set of standard and custom map types) inside a specified HTML container in a web page.

Excerpts of HTML File Defining Custom Google Maps using Tile Overlay of Tuusula, Finland

```

<html>
  <head>
    <title>Google Maps Tiles</title>
    <meta http-equiv="content-type" content="text/html; charset=utf-8"/>
    <script src="http://www.google.com/jsapi?key=ABQIAAAA" type="text/javascript">
    </script>
    <script type="text/javascript">
      local JavaScript to set up the custom maps
      var map;
      var cmap=[]; // array variable to store list of custom maps to be created
      var bounds;
      var suppressedSearch=false; // function to load the map set
      function mapload(){
        if (GBrowserIsCompatible()){
          map = new GMap2(document.getElementById("mapDiv"),
            {googleBarOptions:{showOnLoad:true, onSearchCompleteCallback:function(){suppressedSearch=true;}}});
          bounds = new GLatLngBounds(new GLatLng(60.357634,24.916117), new GLatLng(60.455370,25.178711));
          // define extents rectangle in Latitude/Longitude coordinates for the local tile overlay

          var myCopyright = new GCopyright(1, bounds, 10, "Tuusula");
          var myCopyrightCollection = new GCopyrightCollection("Map Data:");
          myCopyrightCollection.addCopyright(myCopyright);
          // set required copyright message to be shown for the map extents and range of zoom levels of the local TileOverlay
        }
      }
    </script>
  </head>
  <body>
    // check that Google Maps API can be used in the current browser
    // create new map set in the "mapDiv" DIV element in the HTML and set options to configure the GoogleBar search control
  </body>
</html>

```

Loads JavaScript with all of Google's map and search APIs from their web site and specifies your Google API key. Used in conjunction with google.load method called in HTML body.

check that Google Maps API can be used in the current browser

create new map set in the "mapDiv" DIV element in the HTML and set options to configure the GoogleBar search control

define extents rectangle in Latitude/Longitude coordinates for the local tile overlay

set required copyright message to be shown for the map extents and range of zoom levels of the local TileOverlay

(next page)

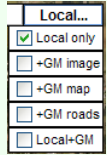
A local Tile Overlay can include tiles in either JPEG or PNG format at any tile position and zoom level. In addition, MicroImages includes a "blank" PNG tile with a "no data" message that is shown at each tile position where there is no other data. Each of these tile types is loaded as a separate tile layer (**GTileLayer** element in the Google Maps API). Loading each of these layers in Google Maps requires implementing code for three **GTileLayer** class methods that must be executed for each tile index position and zoom level: **getTileUrl** to return the URL or local directory path to the tile, **isPNG** to tell Google Maps whether the tile is in PNG format and thus can be transparent, and **getOpacity** to set an opacity value for any tile regardless of image format.

Code for Custom Map "Local only"

```
var myLayer1 = [new GTileLayer(myCopyrightCollection, 10, 20),
               new GTileLayer(myCopyrightCollection, 10, 20), new GTileLayer(myCopyrightCollection, 10, 20)];
```

```
myLayer1[0].getTileUrl = function (p,z) {
  var f;
  f = "Tuusula_Tiles/blank.png";
  return f;
};
myLayer1[0].isPng = function() { return true;};
myLayer1[0].getOpacity = function() { return 1.0; }
```

NOTE: indexing for array variables in JavaScript starts with 0
0th layer: code for loading the single blank.png blank tile for each tile position *p* and zoom level *z* so a "No data" message is shown for areas that are otherwise blank



```
myLayer1[1].getTileUrl = function (p,z) {
  var f;
  f = "Tuusula_Tiles" + "/" + z + "/" + p.y + "/" + p.x + ".png";
  return f;
};
myLayer1[1].isPng = function() { return true;};
myLayer1[1].getOpacity = function() { return 1.0; }
```

1st layer: code to get the PNG-format tile (if any) for each tile position *p* and zoom level *z*. Local tiles for each zoom level are stored in a directory named for the zoom level *z*. For each zoom level there is a subdirectory named for each available Google Maps tile row number *p.y*. Within each row number directory there is a file named for each available Google Maps tile column number *p.x*, with either a ".png" or ".jpg" file extension. The code implemented here for the **getTileUrl** method constructs the local path to each available PNG tile file from its tile position and zoom level values and provides the path to the Google Maps API so the tile can be loaded.

```
myLayer1[2].getTileUrl = function (p,z) {
  var f;
  f = "Tuusula_Tiles" + "/" + z + "/" + p.y + "/" + p.x + ".jpg";
  return f;
};
myLayer1[2].isPng = function() { return false;};
myLayer1[2].getOpacity = function() { return 1.0; }
```

2nd layer: code to get the JPEG-format tile (if any) for each tile position *p* and zoom level *z*

```
cmmap[0] = new GMapType(myLayer1, G_SATELLITE_MAP.getProjection(), "Local only");
```

define a custom map type named "Local only" from myLayer 1 using the projection obtained from Google's satellite map layer and store it at *cmmap* array index 0

[code for custom maps myLayer2 (+GM image), myLayer3 (+GM map), and myLayer5 (+GM roads) omitted]

Code for Custom Map "Local+GM": Google Maps satellite image, local Tuusula image, and Google Maps labels/roads

```
var myLayer5 = [new GTileLayer(myCopyrightCollection, 10, 20),
               G_HYBRID_MAP.getTileLayers()[0],
               new GTileLayer(myCopyrightCollection, 10, 20),
               new GTileLayer(myCopyrightCollection, 10, 20),
               G_HYBRID_MAP.getTileLayers()[1]];
```

define array of five tile layers listed below for a custom map combining local and Google Maps "Hybrid" map layers

```
myLayer5[0].getTileUrl = function (p,z) {
  var f;
  f = "Tuusula_Tiles/blank.png";
  return f;
};
myLayer5[0].isPng = function() { return true;};
myLayer5[0].getOpacity = function() { return 1.0; }
```

0th (bottom) layer: Local blank PNG tile with "No data" message

```
myLayer5[2].getTileUrl = function (p,z) {
  var f;
  f = "Tuusula_Tiles" + "/" + z + "/" + p.y + "/" + p.x + ".png";
  return f;
};
myLayer5[2].isPng = function() { return true;};
myLayer5[2].getOpacity = function() { return 1.0; }
```

code to load the local PNG image tiles as the 2nd layer

1st layer: Google Maps satellite image layer from their "Hybrid" map

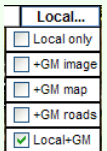
2nd layer: Local PNG tiles (if any)

3rd layer: Local JPG tiles (if any)

```
myLayer5[3].getTileUrl = function (p,z) {
  var f;
  f = "Tuusula_Tiles" + "/" + z + "/" + p.y + "/" + p.x + ".jpg";
  return f;
};
myLayer5[3].isPng = function() { return false;};
myLayer5[3].getOpacity = function() { return 1.0; }
```

code to load the local JPG image tiles as the 3rd layer

4th (top) layer: Google Maps labels and roads layer from their "Hybrid" map



```
cmmap[4] = new GMapType(myLayer5, G_HYBRID_MAP.getProjection(), "Local+GM");
```

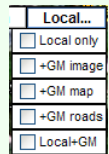
define a custom map type named "Local+GM" from myLayer 5 using the projection obtained from Google's hybrid map layer and store it at *cmmap* array index 4

(next page)

```
var myLayer6 = [new GTileLayer(myCopyrightCollection, 10, 20), G_HYBRID_MAP.getTileLayers()[0], G_HYBRID_MAP.getTileLayers()[1]];
```

```
myLayer6[0].getTileUrl = function (p,z) {  
  var f;  
  f = "Tuusula_Tiles/blank.png";  
  return f;  
};  
myLayer6[0].isPng = function() { return true;};  
myLayer6[0].getOpacity = function() { return 1.0; }
```

Define a default layer to use for the "Local" map selection menu button if all Local menu checkbox options are toggled off; displays the Google Maps Hybrid Map. Also serves as a parent for the other Local menu options (which are related to this layer as children).



```
cmap[5] = new GMapType(myLayer6, G_HYBRID_MAP.getProjection(), "Local...");
```

```
map.enableGoogleBar();
```

enable the GoogleBar search control for the map set

```
var ui = new GMapUIOptions(new GSize(400,400));  
ui.controls.menupatypecontrol = false;  
ui.controls.maptypecontrol = false;  
ui.controls.scalecontrol = false;  
map.setUI(ui);
```

create and set a collection of user interface control options for the map set; turn off the default map type and menu map type controls (a different map selection control will be enabled subsequently) and turn off the default scale bar control

```
for (var i = 0; i < cmap.length; i++) {  
  cmap[i].getMinimumResolution = function() {return (10);}  
  cmap[i].getMaximumResolution = function() {return (18);}  
  cmap[i].isCustom = true;  
  map.addMapType(cmap[i]);  
}
```

iterate through the array of custom maps created; for each one, define the minimum resolution (zoom level), maximum resolution, declare each one as a custom map type, and add to the map set

```
var mapTypesControl = new GHierarchicalMapTypeControl();
```

construct a "hierarchical" map selection control with buttons and nested menus of checkbox items; the default Google Maps map types are automatically added as buttons to this control

```
for (var i = 0; i < cmap.length - 1; i++) {  
  mapTypesControl.addRelationship(cmap[cmap.length-1], cmap[i]);  
};
```

iterate through the array of custom maps; using the last custom map type as the parent, add the remaining custom maps to the control as children (menu checkbox items)

```
map.setCenter(new GLatLng(60.406502,25.047414), 12, cmap[2]);  
map.addControl(mapTypesControl);  
map.enableContinuousZoom();  
map.enableScrollWheelZoom();
```

set center of the map set, add the previously-defined user interface control options, and enable continuous smooth zooming and zooming using the mouse scroll wheel

```
GEvent.addListener(map, "move", function() {forceBounds();});
```

register a Google Maps event handler for map panning to call custom function to keep map center in view

```
function forceBounds() {  
  if (map.getCurrentMapType().isCustom != true) {  
    return;  
  }  
  if (bounds.contains(map.getCenter())) {  
    return;  
  }  
  var C = map.getCenter();  
  var X = C.lng();  
  var Y = C.lat();  
  var maxX = bounds.getNorthEast().lng();  
  var maxY = bounds.getNorthEast().lat();  
  var minX = bounds.getSouthWest().lng();  
  var minY = bounds.getSouthWest().lat();  
  if (X < minX) {X = minX;}  
  if (X > maxX) {X = maxX;}  
  if (Y < minY) {Y = minY;}  
  if (Y > maxY) {Y = maxY;}  
  map.setCenter(new GLatLng(Y,X));  
  if (suppressedSearch){  
    alert("Your search results appear to be beyond the extents of this map data");  
    suppressedSearch = false;}  
}
```

function called when the map is being moved by manual panning or by results of a GoogleBar search; forces the center of a custom map to remain in the browser view

if center of new map view is still within bounds of the custom map, do nothing

get latitude and longitude of center of new map view and latitude or longitude of each edge of the custom map bounds; if new map center is outside one or more of these bounds, reset the center latitude and/or longitude to the nearest custom map boundary

if suppressed recentring is in response to a GoogleBar search, show an alert message

end of forceBounds function

end of mapload function

```
</script>  
</head>
```

```
<body onload="mapload()" onunload="GUNload()">
```

register our mapload function to be called when page is loaded, and Google's function to dismantle Google Maps event handlers and structures when the page is unloaded

```
<div id="mapDiv" style="width:100%;height:100%" ></div>
```

DIV element to contain the map, set to fill the entire web page; you can reduce its size if you want to add your own content to the page

```
<script type="text/javascript">  
  google.load("maps", "2");  
</script>
```

load version 2 of Google's Maps API from the API set registered earlier

```
</body>  
</html>
```