GMap API Documentation

Runtian Zhai (gmap.runtianz.cn)

GMap is a web map application using the API of shipxy.com. Compared with the original API, GMap is much easier to use. It provides a way to run several map apps concurrently without explicitly using threads. Operating GMap is also safer than operating the original map API.

In this document, LAT means latitude while LNG means longitude.

Start to Use GMap

You only need to add one line to your code to use GMap.

The following code will create a Map object.

```
<script src="http://www.runtianz.cn/GMap.js" type="text/javascript"></script>
var map;
window.onload = function () {
/**
 * Displays a map on the page and returns a Map object
 * @param objectId Required, the div where the map should be placed
 * @param centerLat LAT of the center of the map, ranging from -90 to 90. A positive value means the north hemisphere. The default value is 30.
 * @param centerLng LNG of the center of the map, ranging from -180 to 180. A positive value means the east hemisphere. The default value is 120.
 * @param zoom The zoom level of the map. An integer from 1 to 18. A bigger value will display a smaller range. The default value is 5.
 * @param mapType The type of the map, which is one of the following constants:
 *              GMap.CMAP                   Ocean Map
 *              GMap.GOOGLEMAP              Google Road Map
 *              GMap.GOOGLESATELLITE        Google Satellite Map
 */
    map = new GMap.Map(objectId, centerLat, centerLng, zoom, mapType);
};
```

The above code will display a map in the div whose id is objectId, and return a Map instance.

Methods of the Map instance

Use the following methods to operate on Map instances.

```
/**
 * Sets the center and the zoom level of the map.
 * @param centerLat Required, LAT of the center of the map.
 * @param centerLng Required, LNG of the center of the map.
 * @param zoom Zoom Level of the map. Won't change if you don't pass a value.
 */
map.setCenter(centerLat, centerLng, zoom);
```

```
/**
 * Sets the zoom level of the map.
 * @param zoom Required, the zoom level of the map.
 */
map.setZoom(zoom);

/**
 * Sets the type of the map.
 * @param mapType Required, the type of the map, one of the three map type constants.
 */
map.setMapType(mapType);

/**
 * Returns an array that indicates the center of the map. The first element is the LAT of the center, and the second element is the LNG of the center.
 * @returns An array indicating the center of the map.
 */
var ans = map.getCenter();

/**
 * Returns the zoom of the map.
 * @returns An integer, the zoom of the map.
 */
var ans = map.getZoom();

/**
 * Returns the type of the map.
 * @returns One of the three map type constants, the type of the map.
 */
var ans = map.getMapType();

/**
 * Returns an array indicating the size of the map in pixels. The first element is the width of the map, and the second element is the height of the map.
 * @returns An array indicating the size of the map in pixels.
 */
var ans = map.getSize();

/**
 * Returns an array with four elements that indicates the range that the map is displaying.
 * The array is noted as [southWestLat, southWestLng, northEastLat, northEastLng]
 * @returns [southWestLat, southWestLng, northEastLat, northEastLng]
```

```
 *                southWestLat LAT of the south-west corner of the map.
 *                southWestLng LNG of the south-west corner of the map.
 *                northEastLat LAT of the north-east corner of the map.
 *                northEastLng LNG of the north-east corner of the map.
 */
var ans = map.getBounds();

/**
 * Translates the map by pixels.
 * @param east Required. An integer, the number of pixels the map should
translate in the east direction. If it is a negative value, the map will translate in the
west direction.
 * @param south Required. An integer, the number of pixels the map should
translate in the south direction. If it is a negative value, the map will translate in the
north direction.
 */
map.translate(east, south);

/**
 * Returns the pixel coordinates of a point on the map given its geographical
coordinates.
 * The top left corner of the map is (0,0).
 * @param lat LAT of the point.
 * @param lng LNG of the point.
 * @returns An array indicating the pixel coordinates of the point relative to the top
left corner of the map. The first element is the horizontal coordinate, and the second
value is the vertical coordinate.
 */
var ans = map.fromLatLngToPoint(lat, lng);

/**
 * Returns the geographical coordinates of a point on the map given its pixel
coordinates.
 * @param x Horizontal coordinate of the point.
 * @param y Vertical coordinate of the point.
 * @returns An array indicating the geographical coordinates of the point. The first
element is the LAT of the point, and the second element is the LNG of the point.
 */
var ans = map.fromPointToLatLng(x, y);
```

Putting Overlays on the Map

It's very easy to add points, polylines and polygons to the map. Every overlay on the map should have a unique id. You can operate on these overlays using these id's. Note that the id __map__ is used as the id of the entire map, so it is forbidden to use

__map__ as the id of an overlay.
The following code will put a point on the map.

```
var font = new GMap.Font('Verdana', 11, 0xff33cc, true, true, true);
var fill = new GMap.Fill(0x000000, 1.0);
var stroke = new GMap.Stroke(5, 0xff0000, 0.2);
var label = new GMap.Label('test', 0, 0, font, null, fill, stroke);
map.addPoint('point1', 30, 120, label, null, [3, 12], 3, 0, 0);
```

Strokes and Fills

Two important properties of overlays are their strokes and fills. A Stroke object defines the stroke style of an overlay, while a Fill object defines the fill style of an overlay. Both objects have default values that can be modified if you wish. If you don't pass a Stroke instance or a Fill instance when you create an overlay, the default values will be used.

Use the following constructors to create Stroke objects and Fill objects.

```
/**
 * Returns a Stroke instance.
 * @param thickness Thickness of the line, in pixels. The default value is 1.
 * @param color Color of the line, from 0x000000 to 0xffffff. The default value is
0x000000 (black).
 * @param alpha Alpha value of the line, from 0.0 to 1.0. The default value is 1.0
(opaque).
 */
var stroke = new GMap.Stroke(thickness, color, alpha);

/**
 * Returns a copy of the default Stroke instance.
 */
var stroke = new GMap.Stroke();

/**
 * Returns a Fill instance.
 * @param color Color of the fill, from 0x000000 to 0xffffff. The default value is
0x000000 (black).
 * @param alpha Alpha value of the fill, from 0.0 to 1.0. The default value is 1.0
(opaque).
 */
var fill = new GMap.Fill(color, alpha);

/**
 * Returns a copy of the default Fill instance.
 */
var fill = new GMap.Fill();
```

Use the following methods to change the default Stroke and Fill values. Note that the changes are only valid in the current page, and won't be effective if the page is refreshed.

```
/**
 * Modifies the default Stroke instance.
 * @param stroke A Stroke instance.
 */
GMap.setDefaultStroke(stroke);

/**
 * Modifies the default Fill instance.
 * @param fill A Fill instance.
 */
GMap.setDefaultFill(fill);
```

Fonts and Labels

A Label object can be bound to any overlay. Bind a label to an overlay by passing a Label instance when you create an overlay. A Font object defines the font of the text displayed on a Label object. The Font object has a default value, which will be used if you don't pass a Font instance when creating a Label object. The default value can be modified as you wish.

Use the following constructors to create Label and Font objects.

```
/**
 * Returns a Font instance.
 * @param family A string indicating the font family. The default value is Verdana.
 * @param size The size of the font in pixels. The default value is 11.
 * @param color The color of the font, from 0x000000 to 0xffffff. The default value is 0x000000 (black).
 * @param bold A boolean value indicating the bold property. The text will be bold if the value is true.
 * @param italic A boolean value indicating the italic property. The text will be italic if the value is true.
 * @param underline A boolean value indicating the underline property. The text will be underlined if the value is true.
 */
var font = new GMap.Font(family, size, color, bold, italic, underline);

/**
 * Returns a copy of the default Font instance.
 */
var font = new GMap.Font();

/**
 * Returns a Label instance.
```

```
 * @param text Required. The text on the label.
 * @param labelPosX The horizontal position of the label relative to the overlay, in pixels, with right as the positive direction. The default value is 0.
 * @param labelPosY The vertical position of the label relative to the overlay, in pixels, with bottom as the positive direction. The default value is 0.
 * @param font A Font instancec. The font that will be used by the text in the label.
 * @param href A URL string. The hyperlink of the text. The text won't have a hyperlink it href is null.
 * @param backgroundFill A Fill instance. The background fill of the label.
 * @param borderStroke A Stroke instance. The stroke of the border of the label.
 */
var label = new GMap.Label(text, labelPosX, labelPosY, font, href, backgroundFill, borderStroke);
```

Use the following method to change the default Font value. Note that the changes are only valid in the current page, and won't be effective if the page is refreshed.

```
/**
 * Modifies the default Font instance.
 * @param font A Font instance
 */
GMap.setDefaultFont(font);
```

Add a Point

Use the following method to add a point to the map. The picture in the URL will be used to paint the point. If you don't pass a URL, the default URL will be used. You may modify the default URL as you wish.

```
/**
 * Add a point to the map.
 * @param id Required, the id of the point.
 *    If the id exists, a GMap.IdAlreadyExistError will be thrown.
 *    If the value is null, the system will generate a random string as the point's id.
 * @param lat Required, LAT of the point.
 * @param lng Required, LNG of the point.
 * @param label A Label instance, the label bound to the point. The default value is null.
 * @param imageUrl The URL of the picture file used to paint the point.
 * @param zoomlevels An array [low, high]. The point will be displayed only when the zoom level of the map is between low and high, inclusively. The default value is [1, 18].
 * @param zIndex The zIndex of the point. The default value is 3.
 * @param imagePosX The horizontal position of the painted point relative to the actual point, in pixels, with right as its positive direction. The default value is 0.
 * @param imagePosY The vertical position of the painted point relative to the actual point, in pixels, with bottom as its positive direction. The default value is 0.
```

```
 * @param isEditable A boolean value indicating whether the point can be dragged.
True if the point can be dragged.
 * @returns A string, the id of the point.
 */
map.addPoint(id, lat, lng, label, imageUrl, zoomlevels, zIndex, imagePosX, imagePosY,
isEditable);
```

Use the following method to change the default URL of the picture used to paint a point. Note that the changes are only valid in the current page, and won't be effective if the page is refreshed.

```
/**
 * Modifies the default URL when painting a point.
 * @param url A string indicating the new URL of a picture file.
 */
GMap.setDefaultPointUrl(url);
```

Add a Polyline

Use the following method to add a polyline to the map.

```
/**
* Add a polyline to the map.
 * @param id Required, the id of the polyline.
 *    If the id exists, a GMap.IdAlreadyExistError will be thrown.
 *    If the value is null, the system will generate a random string as the polyline's id.
 * @param data Required. An array indicating the position of vertices of the polyline.
Each element is an array [lat, lng] that indicates the position of a vertice.
 *                    lat LAT of the vertice
 *                    lng LNG of the vertice
 * @param stroke A Stroke instance indicating the stroke style of the line. If the
value is null, the default Stroke instance will be used.
* @param label A Label instance, the label bound to the polyline. The default value
is null.
* @param zoomlevels An array [low, high]. The polyline will be displayed only when
the zoom level of the map is between low and high, inclusively. The default value is
[1, 18].
 * @param zIndex The zIndex of the polyline. The default value is 3.
* @param isEditable A boolean value indicating whether the polyline can be dragged.
True if the polyline can be dragged.
 * @returns A string, the id of the polyline.
 */
map.addPolyline(id, data, stroke, label, zoomlevels, zIndex, isEditable);
```

Add a Polygon

Use the following method to add a polygon to the map.

```
/**
* Add a polygon to the map.
  * @param id Required, the id of the polygon.
  *    If the id exists, a GMap.IdAlreadyExistError will be thrown.
  *    If the value is null, the system will generate a random string as the polygon's id.
* @param data Required. An array indicating the position of vertices of the polyline.
Each element is an array [lat, lng] that indicates the position of a vertice.
  *                  lat LAT of the vertice
  *                  lng LNG of the vertice
* @param stroke A Stroke instance indicating the stroke style of the circumference.
If the value is null, the default Stroke instance will be used.
* @param fill A Fill instance indicating the fill style of the polygon. If the value is null,
the default Fill instance will be used.
* @param label A Label instance, the label bound to the polygon. The default value
is null.
* @param zoomlevels An array [low, high]. The polygon will be displayed only when
the zoom level of the map is between low and high, inclusively. The default value is
[1, 18].
  * @param zIndex The zIndex of the polygon. The default value is 3.
* @param isEditable A boolean value indicating whether the polygon can be dragged.
True if the polygon can be dragged.
  * @returns A string, the id of the polygon.
  */
  map.addPolygon(id, data, stroke, fill, label, zoomlevels, zIndex, isEditable);
```

Control the Overlays

Use the id to control the overlays on a map. You can use the id to get the type of an overlay. There are three overlay type constants: GMap.POINT, GMap.POLYLINE and GMap.POLYGON. You can hide, show or remove an overlay using its id.

```
/**
  * Returns the type of an overlay
* @param id Required, the id of an overlay. If the id does not exist, then a
GMap.IdNotExistError will be thrown.
  * @returns One of the three type constants
  */
var ans = map.getType(id);

/**
  * Returns an array of id's of all overlays of a type on the map
  * @param type One of the three type constants
  * @returns An array of strings
  */
var ans = map.getByType(type);
```

```
/**
 * Translates the map so that (the center of) an overlay becomes the center of the map.
* @param id Required, the id of an overlay. If the id does not exist, then a GMap.IdNotExistError will be thrown.
 * @param zoom The zoom level of the map. If you don't pass a zoom value, then the system will select a best zoom level of the map according to the overlay.
 */
map.locate(id, zoom);

/**
 * Deletes an overlay, including hidden ones.
* @param id Required, the id of an overlay. If the id does not exist, then a GMap.IdNotExistError will be thrown.
 */
map.remove(id);

/**
 * Deletes all overlays of a type, including hidden ones.
 * @param type One of the three type constants
 * @returns An integer, the number of overlays that have been deleted.
 */
map.removeByType(type);

/**
 * Deletes all overlays on a map, including hidden ones.
 * @returns An integer, the number of overlays that have been deleted.
 */
map.removeAll();

/**
 * Hides an overlay and returns whether the overlay wasn't hidden.
* @param id Required, the id of an overlay. If the id does not exist, then a GMap.IdNotExistError will be thrown.
 * @returns A boolean value, false if the overlay was hidden before
 */
map.hide(id);

/**
 * Hides all overlays of a type, and returns the number of overlays that wasn't hidden but is now hidden.
 * @param type One of the three type constants
 * @returns The number of overlays that wasn't hidden but is now hidden
 */
```

```
map.hideByType(type);

/**
 * Hides all overlays on a map, and returns the number of overlays that wasn't
hidden but is now hidden.
* @returns The number of overlays that wasn't hidden but is now hidden
 */
map.hideAll();

/**
 * Shows an overlay and returns whether the overlay was hidden.
* @param id Required, the id of an overlay. If the id does not exist, then a
GMap.IdNotExistError will be thrown.
 * @returns A boolean value, true if the overlay was hidden before
 */
map.show(id);

/**
 * Shows all overlays of a type, and returns the number of overlays that was hidden
but is now shown.
 * @param type One of the three type constants
 * @returns The number of overlays that was hidden but is now shown
 */
map.showByType(type);

/**
 * Shows all overlays on a map, and returns the number of overlays that wasn't
hidden but is now hidden.
* @returns The number of overlays that was hidden but is now shown
 */
map.showAll();

/**
 * Returns whether an overlay is hidden.
* @param id Required, the id of an overlay. If the id does not exist, then a
GMap.IdNotExistError will be thrown.
 * @returns A boolean value, true if the overlay hidden.
 */
var ans = map.isHidden(id);
```

Events

You may interact with users by adding event listeners to the map. There are two types of events in GMap. Map events are events such as dragging the map and changing the map type. Overlay events are events such as hiding/showing an overlay

and clicking an overlay.

To apply events to a map, you need to add an event listener to the map itself or to an overlay on the map.

```
/**
 * Adds an event listener to the map.
 * @param events A dictionary that stores a series of events.
 */
map.addMapEvent(events);

/**
 * Adds an event listener to an overlay.
* @param id Required, the id of an overlay. If the id does not exist, then a
GMap.IdNotExistError will be thrown.
* @param events A dictionary that stores a series of events.
 */
map.addOverlayEvent(id, events);

/**
 * Adds an event listener to all overlays of a type on the map.
 * @param type One of the three type constants
* @param events A dictionary that stores a series of events.
 */
map.addOverlayEventByType(type, events);

/**
 * Adds an event listener to all overlays on the map.
* @param events A dictionary that stores a series of events.
 */
map.addOverlayEventToAll(events);
```

When you remove an overlay, its event listeners will be removed as well.

Editing event dictionaries

When you add a event listener, you need to pass an event dictionary to it. For example,

```
map.addOverlayEvent('point1', {
      'onclick': function(event) { /*...*/ },
      'onmouseover': function(event) { /*...*/ }
});
```

The code will add two events to the overlay with the id ‘point1’. One will be triggered when the overlay is clicked, while the other will run when the mouse moves over the overlay. The following table displays all types of events.

Types of map events

Event Type	Triggering Condition
onchange	When the map view changes, such as zooming and dragging
onzoom	When the map zooms
onchangemaptype	When the type of the map changes
onadd	After a new overlay is added to the map
onremove	After an overlay is removed from the map
onhide	When an overlay hides (except overlays already hidden)
onshow	When an overlay shows (except overlays not hidden)
onclick	When the map is clicked
ondoubleclick	When the map is double clicked
onmousedown	When the mouse is pressed down on the map
onmouseup	When the mouse comes up on the map
onmousemove	When the mouse moves above the map
onmouseover	When the mouse comes over the map
onmouseout	When the mouse moves out of the map

Types of overlay events

Event Type	Triggering Condition
onremove	After the overlay is removed
onhide	When the overlay is hidden (except if it is already hidden)
onshow	When the overlay is shown (except if it is not hidden)
onclick	When the overlay is clicked
ondoubleclick	When the overlay is double clicked
onmousedown	When the mouse is pressed down on the overlay
onmouseup	When the mouse comes up on the overlay
onmousemove	When the mouse moves above the overlay
onmouseover	When the mouse comes over the overlay
onmouseout	When the mouse moves out of the overlay

Editing event callbacks

When an event triggered, a parameter ‘event’ which is a dictionary will be passed to the callback function. Here are the elements in the dictionary:

event[‘lat’]	The LAT of the center of the event
event[‘lng’]	The LNG of the center of the event
event[‘id’]	The id of the object triggering the event. The map’s id is __map__

The center of the event is either the position of the mouse or the center point of the map. For all mouse events, the center is the position of the mouse. Otherwise, the center is the center point of the map.

Read the following examples to learn about how to write an event.

```
map.addOverlayEvent('point1', {    // Example 1
      'onclick': function(event) {
            alert(event['lat'] + 'and' + event['lng'] + 'and' + event['id']);
```

```
    }
});
```

In example 1, whenever the user clicks an overlay called ‘point1’, a message pops out and shows the LAT and the LNG of the point, along with its id which is ‘point1’.

```
map.addMapEvent({    // Example 2
    'onmouseover': function(event) {
        alert('You are in!');
    }
});
```

In example 2, a message will pop out whenever the user moves the mouse into the map.

```
map.addMapEvent({    // Example 3
    'onhide': function(event) {
        alert('You hide something! It is ' + event['id']);
    }
});
```

Example 3 intends to implement the following function: whenever an overlay is hidden, a message will pop out showing the id of the overlay. However, in this example, the id in the message is always ‘__map__’ because the object that triggers the map event is always the map itself. Example 4 implements the function correctly.

```
map.addOverlayEventToAll({    // Example 4
    'onhide': function(event) {
        alert('You hide something! It is ' + event['id']);
    }
});
```

Remove an event

Use the following methods to remove events.

```
/**
 * Delete all map events of a type.
 * @param type A string. The type of the event.
 */
map.removeMapEvent(type);

/**
 * Delete all overlay events of a type of a particular overlay.
* @param id Required, the id of an overlay. If the id does not exist, then a
GMap.IdNotExistError will be thrown.
* @param type A string. The type of the event.
 */
map.removeOverlayEvent(id, type);
```

```
/**
 * Delete all overlay events of a type of all overlays of a certain type.
 * @param overlayType One of the three overlay type constants.
* @param eventType A string. The type of the event.
 */
map.removeOverlayEventByType(overlayType, eventType);

/**
 * Delete all overlay events of a type of all overlays on the map.
* @param type A string. The type of the event.
 */
map.removeAllOverlayEvent(type);
```

For example, the following code will remove all 'onclick' events that are added to the point overlays on the map.

```
map.removeOverlayEventByType(GMap.POINT, 'onclick');
```

Grouping overlays

Sometimes a user may want to treat a group of overlays as a whole. The group of overlays can be hidden or shown simultaneously, and can have common event listeners. Use the following methods to group overlays.

```
/**
 * Groups a group of overlays.
 * @param id The id of the group. If null is passed, then a random string will be
generated as the id of the group.
 * If the id already exists, a GMap.IdAlreadyExistError will be thrown.
 * @param idArray An array containing id's of the overlays that are to be grouped.
 * A GMap.IdNotExistError will be thrown if any one of the id's does not exist.
 * @returns A string. The id of the group.
 */
map.group(id, idArray);

/**
 * Cancels a group. This method does not throw errors. If the group id does not exist,
the method will return false rather than throwing an error.
 * @param id The id of the group to be cancelled.
 * @returns A boolean value indicating whether the group exists.
 */
map.ungroup(id);

/**
 * Returns an array containing the id's of all overlays in a group.
 * @param id The id of the group. If the id does not exist, a GMap.IdNotExistError
```

```
will be thrown.
  * @returns An array containing the id's of all overlays in the group.
  */
var ans = map.getGroup(id);
```

Note that whenever you delete an overlay, it will be moved out from its group, too. However, even if you deleted all overlays of a group, the group itself will not be cancelled until you call the ungroup method or the removeGroup method below.

```
/**
  * Removes all overlays in a group, and then cancel the group.
* @param id The id of the group. If the id does not exist, a GMap.IdNotExistError will
be thrown.
  */
map.removeGroup(id);

/**
  * Hides all overlays of a group.
* @param id The id of the group. If the id does not exist, a GMap.IdNotExistError will
be thrown.
  * @returns An integer indicating the number of overlays that was not hidden but is
now hidden.
  */
map.hideGroup(id);

/**
  * Shows all overlays of a group.
* @param id The id of the group. If the id does not exist, a GMap.IdNotExistError will
be thrown.
  * @returns An integer indicating the number of overlays that was hidden but is
now shown.
  */
map.showGroup(id);

/**
  * Locate a group of overlays. The system will first figure out a rectangle that exactly
contains all overlays (including hidden ones) inside the group, and the center of that
rectangle will be set as the center of the map.
* @param id The id of the group. If the id does not exist, a GMap.IdNotExistError will
be thrown.
  * @param zoom Optional. The zoom level of the map after the group is located. If
no value is passed, the system will automatically compute a best zoom level.
  */
map.locateGroup(id, zoom);
```

Group Events

Groups are actually not that useful if it only provides a way to treat a number of overlays as a whole. The key is you can write group events for a group. When several events happen together, each group event will be triggered at most once.

The way of writing a group event is the same as writing a normal event:

```
/**
 * Adds a group event listener to a group.
* @param id The id of the group. If the id does not exist, a GMap.IdNotExistError will be thrown.
 * @param events A dictionary that stores a series of events.
 */
map.addGroupEvent('group1', events);

/**
 * Removes all group events of a type of a group.
* @param id The id of the group. If the id does not exist, a GMap.IdNotExistError will be thrown.
 * @param type A string. The type of the event.
 */
map.removeGroupEvent(id, type);
```

For group events, the value of event['id'] which used to be '__map__' or the id of an overlay is now the id of the group.

Why do we need group events? Consider the following example. We require that whenever a point on the map is hidden, a message will pop out informing us. One way to implement this is to add event listeners to all points on the map.

```
var map;
window.onload = function () {
	map = new GMap.Map('mapDiv');
	map.addPoint('point1', 29, 115, new GMap.Label('point1'), null, [1,18], 3, 0, 0);
	map.addPoint('point2', 25, 130, new GMap.Label('point2'), null, [1,18], 3, 0, 0);
	map.addPolyline('line', [[29,115], [25,130]], new GMap.Stroke(20, 0x000000, 1.0));
	map.addOverlayEventByType(GMap.POINT, {
		'onhide': function() {
			alert('You hide a point!');
		}
	});
	map.hideByType(GMap.POINT);
};
```

Run this code and two messages will pop out, which is not what we want. We only want to see one message if several points are hidden together.

Group events provide a simple way to implement this function.

```
var map;
window.onload = function () {
	map = new GMap.Map('mapDiv');
	map.addPoint('point1', 29, 115, new GMap.Label('point1'), null, [1,18], 3, 0, 0);
	map.addPoint('point2', 25, 130, new GMap.Label('point2'), null, [1,18], 3, 0, 0);
	map.addPolyline('line', [[29,115], [25,130]], new GMap.Stroke(20, 0x000000,
1.0));
	map.group('points', map.getByType(GMap.POINT));
	map.addGroupEvent('points', {
		'onhide': function() {
			alert('You hide a point!');
		}
	});
	map.hideByType(GMap.POINT);
};
```

Here we put all points into a group called ‘points’, and add a group event to this group. Run this code and only one message will pop out.

In another situation, we manually delete several points, but also want to see only one message popping out.

```
var map;
window.onload = function () {
	map = new GMap.Map('mapDiv');
	map.addPoint('point1', 29, 115, new GMap.Label('point1'), null, [1,18], 3, 0, 0);
	map.addPoint('point2', 25, 130, new GMap.Label('point2'), null, [1,18], 3, 0, 0);
	map.addPolyline('line', [[29,115], [25,130]], new GMap.Stroke(20, 0x000000,
1.0));
	map.group('points', map.getByType(GMap.POINT));
	map.addGroupEvent('points', {
		'onhide': function() {
			alert('You hide a point!');
		}
	});
	map.hide('point1');
	map.hide('point2');
};
```

Run this code and two messages will pop out, because GMap thinks that the two hiding operations don’t happen simultaneously. To fix this problem, use the following run method.

```
/**
 * Run a callback function that will trigger any group event at most once.
 * @param fun A callback function to be called.
```

```
 * @returns The result the callback function returns.
 */
map.run(fun);
```

Run the following code:

```
var map;
window.onload = function () {
	map = new GMap.Map('mapDiv');
	map.addPoint('point1', 29, 115, new GMap.Label('point1'), null, [1,18], 3, 0, 0);
	map.addPoint('point2', 25, 130, new GMap.Label('point2'), null, [1,18], 3, 0, 0);
	map.addPolyline('line', [[29,115], [25,130]], new GMap.Stroke(20, 0x000000,
1.0));
	map.group('points', map.getByType(GMap.POINT));
	map.addGroupEvent('points', {
		'onhide': function() {
			alert('You hide a point!');
		}
	});
	map.run(function() {
		map.hide('point1');
		map.hide('point2');
	});
};
```

Now only one message pops out.

Thread Security and GMap Lock

Loading the map application may cause some time (especially when the network is slow). Therefore, the loading process is completed in another thread called the “map creating thread”. Any methods that are called in the main thread when the map is still loading will be put into a waiting queue, and will be called in order when the map gets ready.

The GMap Lock is useful for the following situation: You don’t want the user of your application to do anything before the map is loaded. If a user does too many operations (for example, click a button many times), all these operations will be processed once the map is ready and may cause the application to go down.

To avoid this, use the following method to enable GMap Lock.

```
/**
 * Enables GMap Lock.
 */
GMap.lock();
```

By default GMap Lock is not enabled, and will stay enabled once you enable it. There is only one GMap Lock on a page. For example:

```
var map;
window.onload = function () {
	map = new GMap.Map('mapDiv');
	map.addPoint('point1', 29, 115, null, [1,18], 3, 0, 0, new GMap.Label('point1'));
	map.addPoint('point2', 25, 130, null, [1,18], 3, 0, 0, new GMap.Label('point2'));
};
```

In this code, two addPoint methods are called before the map is loaded. Therefore, these calls will be stored in the waiting queue, and will be processed once the map is ready. Run this code and two points will be shown on the map after the map is loaded.

```
var map;
window.onload = function () {
	map = new GMap.Map('mapDiv');
	GMap.lock();
	map.addPoint('point1', 29, 115, null, [1,18], 3, 0, 0, new GMap.Label('point1'));
	map.addPoint('point2', 25, 130, null, [1,18], 3, 0, 0, new GMap.Label('point2'));
};
```

If you enable GMap Lock, all operations before the map getting ready are ignored. Therefore, there are no points on the map by running this code.

Sometimes, developers may want to add some overlays to the map, but still want to activate GMap Lock. Use GMap.onready to do that. GMap.onready is a callback function that will be called after all maps on a page is ready, ignoring GMap Lock.

For example:

```
var map;
window.onload = function () {
	map = new GMap.Map('mapDiv');
	GMap.lock();
	map.addPoint('point1', 29, 115, null, [1,18], 3, 0, 0, new GMap.Label('point1'));
	GMap.onready = function() {
		map.addPoint('point2', 25, 130, null, [1,18], 3, 0, 0, new
GMap.Label('point2'));
	};
};
```

Now ‘point1’ will not be shown as it is not even created. ‘point2’ will be shown.

Moreover, NEVER write something like this:

```
var map;
window.onload = function() {
	map = new GMap.map('mapDiv');
	map.addPoint('point', 30, 120);
	if (map.getType('point') === GMap.POINT) {
		// Do something important...
```

```
    }
};
```

When you call map.getType('point'), the map is not loaded, so the call will be stored in the waiting queue. Its return value when the map is not ready is something unknown (which is in fact a temporary token object), and the if condition will never be true, and the important code beneath it will never be run.
GMap provides a way to fix this problem:

```
/**
 * Provides an if-else clause for the waiting queue.
 * @param pred Required, a callback function returning a boolean value that acts as the condition.
 * @param clause Optional, a callback function that will be called if pred returns true. Can be null.
 * @param elseclause Optional, a callback function that will be called if pred returns false. Can be null.
 */
map.if(pref, clause, elseclause);

/**
 * Provides a while clause for the waiting queue.
* @param pred Required, a callback function returning a boolean value that acts as the condition.
 * @param clause Required, a callback function that will be repeatedly called until pred returns false.
 */
map.while(pred, clause);
```

You can write something like this:

```
map.if(function() {
    return map.getType('point') === GMap.POINT;
}, function() {
    // If pred is true...
    alert('This is a point.');
}, function() {
    // Else...
    alert('This is not a point.');
})
```

Although GMap provides the above two methods, it is strongly recommended to write codes as simple as possible when the map is not loaded. If you really need to write complex codes, you can use the run method to wrap them.

```
var map;
window.onload = function() {
```

```
    map = new GMap.map('mapDiv');
    map.addPoint('point', 30, 120);
    map.run(function() {
        if (map.getType('point') === GMap.POINT)
            alert('This is a point.');
        else
            alert('This is not a point.');
    });
};
```