Google Maps API Version 2 Reference

If you only want to use the map to display your content, then you need to know these classes, types, and functions:

GMap2
GMapOptions
GInfoWindow
GInfoWindowTab
GInfoWindowOptions
GMarker
GMarkerOptions
GPolyline
GPolygon
GIcon
GPoint
GSize
GBounds
GLatLng
GLatLngBounds
GControl
GEvent
GEventListener
GXmlHttp
GXml
GXslt
GLog
GDraggableObject
GGeoStatusCode
GGeoAddressAccuracy
GClientGeocoder
GGeocodeCache
GFactualGeocodeCache
GMarkerManager
GMarkerManagerOptions
GDownloadUrl
GBrowserIsCompatible

If you want to extend the functionality of the maps API by implementing your own controls, overlays, or map types, then you also need to know these classes and types:

GMapPane
GOverlay
GControl
GControlPosition
GControlAnchor
GMapType
GMapTypeOptions
GTileLayer
GTileLayerOverlay
GCopyrightCollection
GCopyright
GProjection
GMercatorProjection

class GMap2

Instantiate class `GMap2` in order to create a map. This is the central class in the API. Everything else is auxiliary.

Constructor

Constructor	Description
`GMap2(container, opts?)`	Creates a new map inside of the given HTML container, which is typically a `DIV` element. If no set of map types is given in the optional argument `opts.mapTypes`, the default set `G_DEFAULT_MAP_TYPES` is used. If no size is given in the optional argument `opts.size`, then the size of the `container` is used. If `opts.size` is given, then the container element of the map is resized accordingly. See `class GMapOptions`.

Methods

Configuration

Methods	Return Value	Description
`enableDragging()`	*none*	Enables the dragging of the map (enabled by default).
`disableDragging()`	*none*	Disables the dragging of the map.
`draggingEnabled()`	`Boolean`	Returns `true` iff the dragging of the map is enabled.
`enableInfoWindow()`	*none*	Enables info window operations on the map (enabled by default).
`disableInfoWindow()`	*none*	Closes the info window, if it is open, and disables the opening of a new info window.
`infoWindowEnabled()`	`Boolean`	Returns `true` iff the info window is enabled.
`enableDoubleClickZoom()`	*none*	Enables double click to zoom in and out (disabled by default). (Since 2.58)
`disableDoubleClickZoom()`	*none*	Disables double click to zoom in and out. (Since 2.58)
`doubleClickZoomEnabled()`	`Boolean`	Returns `true` iff double click to zoom is enabled. (Since 2.58)
`enableContinuousZoom()`	*none*	Enables continuous smooth zooming for select browsers (disabled by default). (Since 2.58)
`disableContinuousZoom()`	*none*	Disables continuous smooth zooming. (Since 2.58)
`continuousZoomEnabled()`	`Boolean`	Returns `true` iff continuous smooth zooming is enabled. (Since 2.58)

Controls

Methods	Return Value	Description
`addControl(control, position?)`	*none*	Adds the control to the map. The position on the map is determined by the optional `position` argument. If this argument is absent, the default position of the control is used, as determined by the `GControl.getDefaultPosition()` method. A control instance must not be added more than once to the map.
`removeControl(control)`	*none*	Removes the control from the map. It does nothing if the control was never added to the map.
`getContainer()`	`Node`	Returns the DOM object that contains the map. Used by `GControl.initialize()`.

Map Types

Methods	Return Value	Description
`getMapTypes()`	`Array of GMapType`	Returns the array of map types registered with this map.
`getCurrentMapType()`	`GMapType`	Returns the currently selected map type.
`setMapType(type)`	*none*	Selects the given new map type. The type must be known to the map. See the constructor, and the method `addMapType()`.
`addMapType(type)`	*none*	Adds a new map type to the map. See section `GMapType` for how to define custom map types.
`removeMapType(type)`	*none*	Removes the map type from the map. Will update the set of buttons displayed by the `GMapTypeControl` and fire the `removemaptype` event.

Map State

Methods	Return Value	Description
`isLoaded()`	`Boolean`	Returns `true` iff the map was initialized by `setCenter()` since it was created.
`getCenter()`	`GLatLng`	Returns the geographical coordinates of the center point of the map view.

`getBounds()`	`GLatLngBounds`	Returns the the visible rectangular region of the map view in geographical coordinates.
`getBoundsZoomLevel(bounds)`	`Number`	Returns the zoom level at which the given rectangular region fits in the map view. The zoom level is computed for the currently selected map type. If no map type is selected yet, the first on the list of map types is used.
`getSize()`	`GSize`	Returns the size of the map view in pixels.
`getZoom()`	`Number`	Returns the current zoom level.

Modify the Map State

Methods	Return Value	Description
`setCenter(center, zoom?, type?)`	*none*	Sets the map view to the given center. Optionally, also sets zoom level and map type. The map type must be known to the map. See the constructor, and the method `addMapType()`. This method must be called first after construction to set the initial state of the map. It is an error to call other operations on the map after construction.
`panTo(center)`	*none*	Changes the center point of the map to the given point. If the point is already visible in the current map view, change the center in a smooth animation.
`panBy(distance)`	*none*	Starts a pan animation by the given distance in pixels.
`panDirection(dx, dy)`	*none*	Starts a pan animation by half the width of the map in the indicated directions. `+1` is right and down, `-1` is left and up, respectively.
`setZoom(level)`	*none*	Sets the zoom level to the given new value.
`zoomIn()`	*none*	Increments zoom level by one.
`zoomOut()`	*none*	Decrements zoom level by one.
`savePosition()`	*none*	Stores the current map position and zoom level for later recall by `returnToSavedPosition()`.
`returnToSavedPosition()`	*none*	Restores the map view that was saved by `savePosition()`.
`checkResize()`	*none*	Notifies the map of a change of the size of its container. Call this method after the size of the container DOM object has changed, so that the map can adjust itself to fit the new size.

Overlays

Methods	Return Value	Description
`addOverlay(overlay)`	*none*	Adds an overlay to the map and fires the `addoverlay` event.
`removeOverlay(overlay)`	*none*	Removes the overlay from the map. If the overlay was on the map, it fires the `removeoverlay` event.
`clearOverlays()`	*none*	Removes all overlay from the map, and fires the `clearoverlays` event.
`getPane(pane)`	`Node`	Returns a DIV that holds the object in the layer identified by `pane`. Used by `GOverlay` instances in method `GOverlay.initialize()` instances to draw themselves on the map

Info Window

Methods	Return Value	Description
`openInfoWindow(point, node, opts?)`	*none*	Opens a simple info window at the given point. Pans the map such that the opened info window is fully visible. The content of the info window is given as a DOM node.

`openInfoWindowHtml(point, html, opts?)`	*none*	Opens a simple info window at the given point. Pans the map such that the opened info window is fully visible. The content of the info window is given as HTML text.
`openInfoWindowTabs(point, tabs, opts?)`	*none*	Opens a tabbed info window at the given point. Pans the map such that the opened info window is fully visible. The content of the info window is given as DOM nodes.
`openInfoWindowTabsHtml(point, tabs, opts?)`	*none*	Opens a tabbed info window at the given point. Pans the map such that the opened info window is fully visible. The content of the info window is given as HTML text.
`showMapBlowup(point, opts?)`	*none*	Opens an info window at the given point that contains a closeup view on the map around this point.
`closeInfoWindow()`	*none*	Closes the currently open info window.
`getInfoWindow()`	`GInfoWindow`	Returns the info window object of this map. If no info window exists yet, it is created, but not displayed. This operation is not influenced by `enableInfoWindow()`.

Coordinate Transformations

Methods	Return Value	Description
`fromLatLngToDivPixel(latlng)`	`GPoint`	Computes the pixel coordinates of the given geographical point in the DOM element that holds the draggable map. You need this method to position a custom overlay when you implement the `GOverlay.redraw()` method for a custom overlay.
`fromDivPixelToLatLng(pixel)`	`GLatLng`	Computes the geographical coordinates from pixel coordinates in the div that holds the draggable map. You need this when you implement interaction with custom overlays.
`fromContainerPixelToLatLng(pixel)`	`GLatLng`	Computes the geographical coordinates of the point at the given pixel coordinates in the DOM element that contains the map on the page. You need this when you implement interaction of custom controls with the map.

Events

Events	Arguments	Description
`addmaptype`	`type`	This event is fired when a map type is added to the map.
`removemaptype`	`type`	This event is fired when a map type is removed from the map.
`click`	`overlay, point`	This event is fired when the map is clicked with the mouse. If the click was on a marker, then the marker is passed to the event handler in the `overlay` argument, and a `click` event is also fired on the marker. Otherwise, the geographical coordinates of the point that was clicked are passed in the `point` argument.
`movestart`	*none*	This event is fired when the map view starts changing. This can be caused by dragging, in which case a `dragstart` event is also fired, or by invocation of a method that changes the map view.
`move`	*none*	This event is fired, possibly repeatedly, while the map view is changing.
`moveend`	*none*	This event is fired when the change of the map view ends.
`zoomend`	`oldLevel, newLevel`	This event is fired when the map reaches a new zoom level. The event handler receives the previous and the new zoom level as arguments.

`maptypechanged`	*none*	This event is fired when another map type is selected.
`infowindowopen`	*none*	This event is fired when the info window opens.
`infowindowclose`	*none*	This event is fired when the info window closes. If a currently open info window is reopened at a different point using another call to `openInfoWindow*()`, then `infowindowclose` will fire first.
`addoverlay`	`overlay`	This event is fired when a single overlay is added to the map by the method `addOverlay()`. The new overlay is passed as an argument `overlay` to the event handler.
`removeoverlay`	`overlay`	This event is fired when a single overlay is removed by the method `removeOverlay()`. The overlay that was removed is passed as an argument `overlay` to the event handler.
`clearoverlays`	*none*	This event is fired when all overlays are removed at once by the method `clearOverlays()`.
`mouseover`	`latlng`	This event is fired when the user moves the mouse over the map from outside the map.
`mouseout`	`latlng`	This event is fired when the user moves the mouse off the map.
`mousemove`	`latlng`	This event is fired when the user moves the mouse inside the map.
`dragstart`	*none*	This event is fired when the user starts dragging the map.
`drag`	*none*	This event is repeatedly fired while the user drags the map.
`dragend`	*none*	This event is fired when the user stops dragging the map.
`load`	*none*	This event is fired when the map setup is complete, and isLoaded() would return true. This means position, zoom, and map type are all initialized, but tile images may still be loading. (Since 2.52)

class GMapOptions

This class represents optional arguments to the `GMap2` constructor. It has no constructor, but is instantiated as object literal.

Properties

Properties	Type	Description
`size`	`GSize`	Sets the size in pixels of the map. The container that is passed to the map constructor will be resized to the given size. By default, the map will assume the size of its container.
`mapTypes`	`Array of GMapType`	Array of map types to be used by this map. By default, `G_DEFAULT_MAP_TYPES` is used. You can use this option to restrict the set of predefined map types that is displayed on the map, or to pass your own map types to the map. See also `GMap2.addMapType()`.
`draggableCursor`	`String`	The cursor to display when the map is draggable. (Since 2.59)
`draggingCursor`	`String`	The cursor to display while dragging the map. (Since 2.59)

enum GMapPane

These constants define the layering system that is used by overlay to display themselves on the map. There are different layers for icons, shadows, the info window, the shadow on the info window, and transparent mouse event catching objects.

You need to use this type if you subclass from `GOverlay`.

Constants

Constants	Description
G_MAP_MAP_PANE	This pane is still below the shadows of the markers, directly on top of the map. It contains for instance the polylines.
G_MAP_MARKER_SHADOW_PANE	This pane contains the shadows of the markers. It is below the markers.
G_MAP_MARKER_PANE	This pane contains the markers.
G_MAP_FLOAT_SHADOW_PANE	This pane conatains the shadow of the info window. It is above the G_MAP_MARKER_PANE, so that markers can be in the shadow of the info window.
G_MAP_MARKER_MOUSE_TARGET_PANE	This pane contains transparent elements that receive DOM mouse events for the markers. Is is above the G_MAP_FLOAT_SHADOW_PANE, so that markers in the shadow of the info window can be clickable.
G_MAP_FLOAT_PANE	This pane contains the info window. It is above everything else on the map.

class GKeyboardHandler

Instantiate this class to add keyboard bindings to a map. The key bindings are the same as for the maps application.

key action up, down, left, right Moves the map continuously while the key is pressed. Two keys can be pressed simultaneously in order to move diagonally. page down, page up, home, end Pans the map by 3/4 its size in the corresponding direction, with a nice animation. This corresponds to the arrow buttons in the
GLargeMapControl
and the
GSmallMapControl
. +, - Zooms one level in or out, respectively. This corresponds to the + and - buttons in the
GLargeMapControl
and the
GSmallMapControl
.

Constructor

Constructor	Description
GKeyboardHandler(map)	Installs keyboard event handler for the map passed as argument.

interface GOverlay

This interface is implemented by the GMarker, GPolyline, GTileLayerOverlay and GInfoWindow classes in the maps API library. You can implement it if you want to display custom types of overlay objects on the map. An instance of GOverlay can be put on the map with the method GMap2.addOverlay(). The map will then call the method GOverlay.initialize() on the overlay instance to display itself on the map initially. Whenever the map display changes, the map will call GOverlay.redraw() so that the overlay can reposition itself if necessary. The overlay instance can use the method GMap2.getPane() to get hold of one or more DOM container elements to attach itself to.

Constructor

Constructor	Description
GOverlay()	This constructor creates dummy implementations for the methods. Still, when inheriting from this class, your derived class constructor should call this constructor for completeness.

Static Methods

Static Methods	Return Value	Description
`getZIndex(latitude)`	`Number`	Returns a CSS `z-index` value for a given latitude. It computes a z index such that overlays further south are on top of overlays further north, thus creating the 3D appearance of marker overlays.

Abstract Methods

Abstract Methods	Return Value	Description
`initialize(map)`	*none*	Called by the map after the overlay is added to the map using `GMap2.addOverlay()`. The overlay object can draw itself into the different panes of the map that can be obtained using `GMap2.getPane()`.
`remove()`	*none*	Called by the map after the overlay is removed from the map using `GMap2.removeOverlay()` or `GMap2.clearOverlays()`. The overlay must remove itself from the map panes here.
`copy()`	`GOverlay`	Returns an uninitialized copy of itself that can be added to the map.
`redraw(force)`	*none*	Called by the map when the map display has changed. The argument `force` will be `true` if the zoom level or the pixel offset of the map view has changed, so that the pixel coordinates need to be recomputed.

class GInfoWindow

`GInfoWindow` has no constructor. It is created by the map and accessed by its method `GMap2.getInfoWindow()`.

Methods

Methods	Return Value	Description
`selectTab(index)`	*none*	Selects the tab with the given index. This has the same effect as clicking on the corresponding tab.
`hide()`	*none*	Makes the info window invisible. NOTE: This doesn't close the info window. It can be made visible again using `show()`.
`show()`	*none*	Makes the info window visible if its currently invisible.
`isHidden()`	`Boolean`	Returns `true` iff the info window is hidden. This includes the state that it's closed.
`reset(point, tabs, size, offset?, selectedTab?)`	*none*	Resets the state of the info window. Each argument may be `null` and then its value will not be changed from the current value.
`getPoint()`	`GLatLng`	Returns the geographical point at which the info window is anchored. The tip of the window points to this point on the map, modulo the pixel offset.

`getPixelOffset()`	`GSize`	Returns the offset, in pixels, of the tip of the info window from the point on the map at whose geographical coordinates the info window is anchored.
`getSelectedTab()`	`Number`	Returns the index, starting at 0, of the current selected tab.
`getTabs()`	`Array of GInfoWindowTabs`	Returns the array of tabs in this info window. (Since 2.59)
`getContentContainers()`	`Array of Node`	Returns the array of DOM nodes that hold the content of the tabs of this info window. (Since 2.59)

Events

Events	Arguments	Description
`closeclick`	*none*	This event is fired when the info window close button is clicked. An event handler for this event can implement to close the info window, by calling the `GMap2.closeInfoWindow()` method.

class GInfoWindowTab

An array of instances of this class can be passed as the `tabs` argument to the methods `GMap2.openInfoWindowTabs()`, `GMap2.openInfoWindowTabsHtml()`, `GMarker.openInfoWindowTabs()`, and `GMarker.openInfoWindowTabsHtml()`. If the array contains more than one element, the info window will be shown with tabs. Every `InfoWindowTab` object contains two items: `content` defines the content of the info window when the tab is selected, and `label` defines the label of the tab. The properties are passed as arguments to the constructor. For the `openInfoWindowTabs()` methods, `content` is a DOM Node. For the methods `openInfoWindowTabsHtml()`, `content` is a string that contains HTML text.

Constructor

Constructor	Description
`GInfoWindowTab(label, content)`	Creates an info window tab data structure that can be passed in the `tabs` argument to `openInfoWindowTabs*()` methods.

class GInfoWindowOptions

Instances of this class are used in the `opts?` argument to the methods `openInfoWindow()`, `openInfoWindowHtml()`, `openInfoWindowTabs()`, `openInfoWindowTabsHtml()`, and `showMapBlowup()` of classes `GMap2` and `GMarker`. There is no constructor for this class. Instead, this class is instantiated as javascript object literal.

Properties

As the name of this class indicates, all properties are optional.

Properties	Type	Description
`selectedTab`	`Number`	Selects the tab with the given index, starting at 0, instead of the first tab (with index 0).
`maxWidth`	`Number`	Maximum width of the info window content, in pixels.

onOpenFn	Function	Function is called after the info window is opened and the content is displayed.
onCloseFn	Function	Function is called when the info window is closed.
zoomLevel	Number	Pertinent for showMapBlowup() only. The zoom level of the blowup map in the info window.
mapType	GMapType	Pertinent for showMapBlowup() only. The map type of the blowup map in the info window.

class GMarker

A GMarker marks a position on the map. It implements the GOverlay interface and thus is added to the map using the GMap2.addOverlay() method.

A marker object has a point, which is the geographical position where the marker is anchored on the map, and an icon. If the icon is not set in the constructor, the default icon G_DEFAULT_ICON is used.

After it is added to a map, the info window of that map can be opened through the marker. The marker object will fire mouse events and infowindow events.

Constructor

Constructor	Description
GMarker(point, icon?, inert?)	Creates a marker at point with icon or the G_DEFAULT_ICON. If the inert flag is true, then the marker is not clickable and will not fire any events. (Deprecated since 2.50)
GMarker(point, opts?)	Creates a marker at point with options specified in GMarkerOptions. By default markers are clickable & have the default icon G_DEFAULT_ICON. (Since 2.50)

Methods

Before these methods can be invoked, the marker must be added to a map.

Methods	Return Value	Description
openInfoWindow(content, opts?)	*none*	Opens the map info window over the icon of the marker. The content of the info window is given as a DOM node. Only option GInfoWindowOptions.maxWidth is applicable.
openInfoWindowHtml(content, opts?)	*none*	Opens the map info window over the icon of the marker. The content of the info window is given as a string that contains HTML text. Only option GInfoWindowOptions.maxWidth is applicable.
openInfoWindowTabs(tabs, opts?)	*none*	Opens the tabbed map info window over the icon of the marker. The content of the info window is given as an array of tabs that contain the tab content as DOM nodes. Only options GInfoWindowOptions.maxWidth and InfoWindowOptions.selectedTab are applicable.
openInfoWindowTabsHtml(tabs, opts?)	*none*	Opens the tabbed map info window over the icon of the marker. The content of the info window is given as an array of tabs that contain the tab content as Strings that contain HTML text. Only options InfoWindowOptions.maxWidth and InfoWindowOptions.selectedTab are applicable.
showMapBlowup(opts?)	*none*	Opens the map info window over the icon of the marker. The content of the info window is a closeup map around the marker position. Only options InfoWindowOptions.zoomLevel and InfoWindowOptions.mapType are applicable.

`getIcon()`	`GIcon`	Returns the `icon` of this marker, as set by the constructor.
`getPoint()`	`GLatLng`	Returns the geographical coordinates of the point at which this marker is anchored, as set by the constructor or by `setPoint()`.
`setPoint(point)`	*none*	Sets the geographical coordinates of the point at which this marker is anchored.
`enableDragging()`	*none*	Enables the marker to be dragged and dropped around the map. To function, the marker must have been initialized with `GMarkerOptions.draggable = true`.
`disableDragging()`	*none*	Disables the marker from being dragged and dropped around the map.
`draggable()`	`Boolean`	Returns true if the marker has been initialized via the constructor using `GMarkerOptions.draggable = true`. Otherwise, returns false.
`draggingEnabled()`	`Boolean`	Returns true if the marker is currently enabled for the user to drag on the map.

Events

All these events fire only if the marker is not inert (see constructor).

Events	Arguments	Description
`click`	*none*	This event is fired when the marker icon was clicked. Notice that this event will also fire for the map, with the marker passed as the first argument to the event handler there.
`dblclick`	*none*	This event is fired when the marker icon was double-clicked. Notice that this event will not fire for the map, because the map centers on double-click as a hardwired behavior.
`mousedown`	*none*	This event is fired when the DOM mousedown event is fired on the marker icon. Notice that the marker will stop the mousedown DOM event, so that it doesn't cause the map to start dragging.
`mouseup`	*none*	This event is fired for the DOM mouseup on the marker. Notice that the marker will not stop the mousedown DOM event, because it will not confuse the drag handler of the map.
`mouseover`	*none*	This event is fired when the mouse enters the area of the marker icon.
`mouseout`	*none*	This event is fired when the mouse leaves the area of the marker icon.
`infowindowopen`	*none*	This event is fired when the info window of the map was opened through this marker.
`infowindowclose`	*none*	This event is fired when the info window of the map that was opened through this marker is closed again. This happens when either the info window was closed, or when it was opened on another marker, or on the map.
`remove`	*none*	This event is fired when the marker is removed from the map, using `GMap2.removeOverlay()` or `GMap2.clearOverlays()`.
`dragstart`	*none*	If the marker is enabled for dragging, this event is fired when the marker dragging begins.
`drag`	*none*	If the marker is enabled for dragging, this event is fired when the marker is being dragged.
`dragend`	*none*	If the marker is enabled for dragging, this event is fired when the marker ceases to be dragged.

class GMarkerOptions

Instances of this class are used in the `opts?` argument to the constructor of the `GMarker` class. There is no constructor for this class. Instead, this class is instantiated as a javascript object literal.

Properties

As the name of this class indicates, all properties are optional.

Properties	Type	Description
`icon`	`GIcon`	Chooses the Icon for this class. If not specified, `G_DEFAULT_ICON` is used. (Since 2.50)
`dragCrossMove`	`Boolean`	When dragging markers normally, the marker floats up and away from the cursor. Setting this value to `true` keeps the marker underneath the cursor, and moves the cross downwards instead. The default value for this option is `false`. (Since 2.63)
`title`	`String`	This string will appear as tooltip on the marker, i.e. it will work just as the `title` attribute on HTML elements. (Since 2.50)
`clickable`	`Boolean`	Toggles whether or not the marker is clickable. Markers that are not clickable or draggable are inert, consume less resources and do not respond to any events. The default value for this option is `true`, i.e. if the option is not specified, the marker will be clickable. (Since 2.50)
`draggable`	`Boolean`	Toggles whether or not the marker will be draggable by users. Markers set up to be dragged require more resources to set up than markers that are clickable. Any marker that is draggable is also clickable and bouncy by default. The default value for this option is `false`. (Since 2.61)
`bouncy`	`Boolean`	Toggles whether or not the marker should bounce up and down after it finishes dragging. The default value for this option is `false`. (Since 2.61)
`bounceGravity`	`Number`	When finishing dragging, this number is used to define the acceleration rate of the marker during the bounce down to earth. The default value for this option is `1`. (Since 2.61)

class GPolyline

This is a map overlay that draws a polyline on the map, using the vector drawing facilities of the browser if they are available, or an image overlay from Google servers otherwise.

Constructor

Constructor	Description
`GPolyline(points, color?, weight?, opacity?)`	Creates a polyline from an array of vertices. The `color` is given as a string that contains the color in hexadecimal numeric HTML style, i.e. #RRGGBB. The `weight` is the width of the line in pixels. The `opacity` is given as a number between 0 and 1. The line will be antialiased and semitransparent.

Factory Methods

Factory Methods	Return Value	Description

`fromEncoded(color?, weight?, opacity?, points, zoomFactor, levels, numLevels)`	`GPolyline`	Creates a polyline from encoded strings of aggregated points and levels. `color` is a string that contains a hexadecimal numeric HTML style, i.e. #RRGGBB. `weight` is the width of the line in pixels. `opacity` is a number between 0 and 1. `points` is a string containing the encoded latitude and longitude coordinates. `levels` is a string containing the encoded polyline zoom level groups. `numLevels` is the number of zoom levels contained in the encoded `levels` string. `zoomFactor` is the magnification between adjacent sets of zoom levels in the encoded `levels` string. Together, these two values determine the precision of the `levels` within an encoded polyline. The line will be antialiased and semitransparent. (Since 2.63)

Methods

Methods	Return Value	Description
`getVertexCount()`	`Number`	Returns the number of vertices in the polyline. (Since 2.46)
`getVertex(index)`	`GLatLng`	Returns the vertex with the given index in the polyline. (Since 2.46)

Events

Events	Arguments	Description
`remove`	*none*	This event is fired when the polyline is removed from the map, using `GMap2.removeOverlay()` or `GMap2.clearOverlays()`.

class GPolygon

This is very similar to a `GPolyline`, except that you can additionally specify a fill color and opacity.

Constructor

Constructor	Description
`GPolygon(points, strokeColor?, strokeWeight?, strokeOpacity?, fillColor?, fillOpacity?)`	Creates a polygon from an array of vertices. The `colors` are given as a string that contains the color in hexadecimal numeric HTML style, i.e. #RRGGBB. The `weight` is the width of the line in pixels. The `opacities` is given as a number between 0 and 1. The line will be antialiased and semitransparent. (Since 2.69)

Methods

Methods	Return Value	Description
`getVertexCount()`	`Number`	Returns the number of vertices in the polygon. (Since 2.69)
`getVertex(index)`	`GLatLng`	Returns the vertex with the given index in the polygon. (Since 2.69)

Events

Events	Arguments	Description
`remove`	*none*	This event is fired when the polygon is removed from the map, using `GMap2.removeOverlay()` or `GMap2.clearOverlays()`.

class GIcon

An icon specifies the images used to display a `GMarker` on the map. For browser compatibility reasons, specifying an icon is actually quite complex. Note that you can use the default Maps icon `G_DEFAULT_ICON` if you don't want to specify your own.

Constructor

Constructor	Description
`GIcon(copy?, image?)`	Creates a new icon object. If another icon is given in the optional `copy` argument, its properties are copied, otherwise they are left empty. The optional argument `image` sets the value of the `image` property.

Constants

Constants	Description
`G_DEFAULT_ICON`	The default icon used by markers.

Properties

Properties	Type	Description
`image`	`String`	The foreground image URL of the icon.
`shadow`	`String`	The shadow image URL of the icon.
`iconSize`	`GSize`	The pixel size of the foreground image of the icon.
`shadowSize`	`GSize`	The pixel size of the shadow image.
`iconAnchor`	`GPoint`	The pixel coordinate relative to the top left corner of the icon image at which this icon is anchored to the map.
`infoWindowAnchor`	`GPoint`	The pixel coordinate relative to the top left corner of the icon image at which the info window is anchored to this icon.
`printImage`	`String`	The URL of the foreground icon image used for printed maps. It must be the same size as the main icon image given by `image`.
`mozPrintImage`	`String`	The URL of the foreground icon image used for printed maps in Firefox/Mozilla. It must be the same size as the main icon image given by `image`.
`printShadow`	`String`	The URL of the shadow image used for printed maps. It should be a GIF image since most browsers cannot print PNG images.
`transparent`	`String`	The URL of a virtually transparent version of the foreground icon image used to capture click events in Internet Explorer. This image should be a 24-bit PNG version of the main icon image with 1% opacity, but the same shape and size as the main icon.
`imageMap`	`Array of Number`	An array of integers representing the x/y coordinates of the image map we should use to specify the clickable part of the icon image in browsers other than Internet Explorer.

class GPoint

A `GPoint` represents a point on the map by its pixel coordinates. Notice that in v2, it doesn't represent a point on the earth by its geographical coordinates anymore. Geographical coordinates are now represented by `GLatLng`.

In the map coordinate system, the `x` coordinate increases to the left, and the `y` coordinate increases downwards.

Notice that while the two parameters of a `GPoint` are accessible as properties `x` and `y`, it is better to never modify them, but to create a new object with different paramaters instead.

Constructor

Constructor	Description
`GPoint(x, y)`	Creates a `GPoint` object.

Properties

Properties	Type	Description
`x`	`Number`	`x` coordinate, increases to the left.
`y`	`Number`	`y` coordinate, increases downwards.

Methods

Methods	Return Value	Description
`equals(other)`	`Boolean`	Returns `true` iff the other point has equal coordinates.
`toString()`	`String`	Returns a string that contains the x and y coordinates, in this order, separated by a comma.

class GSize

A `GSize` is the size in pixels of a rectangular area of the map. The size object has two parameters, `width` and `height`. Width is a difference in the x-coordinate; height is a difference in the y-coordinate, of points.

Notice that while the two parameters of a `GSize` are accessible as properties `width` and `height`, it is better to never modify them, but to create a new object with different paramaters instead.

Constructor

Constructor	Description
`GSize(width, height)`	Creates a `GSize` object.

Properties

Properties	Type	Description
`width`	`Number`	The width parameter.
`height`	`Number`	The height parameter.

Methods

Methods	Return Value	Description
`equals(other)`	`Boolean`	Returns `true` iff the other size has exactly equal components.
`toString()`	`String`	Returns a string that contains the wdith and height parameter, in this order, separated by a comma.

class GBounds

`GBounds` is a rectangular area of the map in pixel coordinates. Notice that a rectangle in geographical coordinates is represented by a `GLatLngBounds` object.

Constructor

Constructor	Description
`GBounds(points)`	Constructs a rectangle that contains all the given `points`.

Properties

Properties	Type	Description
`minX`	`Number`	The x coordinate of the left edge of the rectangle.
`minY`	`Number`	The y coordinate of the top edge of the rectangle.
`maxX`	`Number`	The x coordinate of the right edge of the rectangle.
`maxY`	`Number`	The y coordinate of the bottom edge of the rectangle.

Methods

Methods	Return Value	Description
`toString()`	`String`	Returns a string that contains the coordinates of the upper left and the lower right corner points of the box, in this order, separated by comma, surrounded by parentheses.
`min()`	`GPoint`	The point at the upper left corner of the box.
`max()`	`GPoint`	The point at the lower right corner of the box.
`containsBounds(other)`	`Boolean`	Returns `true` iff the other box is entirely contained in this box.
`extend(point)`	*none*	Enlarges this box so that the point is also contained in this box.
`intersection(other)`	`GBounds`	Returns the box by which this box overlaps the other box. If there is no overlap, returns an empty box.

class GLatLng

`GLatLng` is a point in geographical coordinates longitude and latitude.

Notice that although usual map projections associate longitude with the x-coordinate of the map, and latitude with the y-coordinate, the latitude cooridnate is always written first, followed by the longitude, as it is custom in cartography.

Notice also that you cannot modify the coordinates of a `GLatLng`. If you want to compute another point, you have to create a new one.

Constructor

Constructor	Description
`GLatLng(lat, lng, unbounded?)`	Notice the ordering of latitude and longitude. If the `unbounded` flag is `true`, then the numbers will be used as passed, oterwise latitude will be clamped to lie between -90 degrees and +90 degrees, and longitude will be wrapped to lie between -180 degrees and +180 degrees.

Methods

Methods	Return Value	Description

`lat()`	`Number`	Returns the latitude coordinate in degrees, as a number between -90 and +90. If the `unbounded` flag was set in the constructor, this coordinate can be outside this interval.
`lng()`	`Number`	Returns the longitude coordinate in degrees, as a number between -180 and +180. If the `unbounded` flag was set in the constructor, this coordinate can be outside this interval.
`latRadians()`	`Number`	Returns the latitude coordinate in radians, as a number between -PI/2 and +PI/2. If the `unbounded` flag was set in the constructor, this coordinate can be outside this interval.
`lngRadians()`	`Number`	Returns the longitude coordinate in radians, as a number between -PI and +PI. If the `unbounded` flag was set in the constructor, this coordinate can be outside this interval.
`equals(other)`	`Boolean`	Returns `true` iff the other size has equal components, within certain roundoff margins.
`distanceFrom(other)`	`Number`	Returns the distance, in meters, from this point to the given point. The earth is approximated as a sphere, hence the distance could be off by as much as 0.3%.
`toUrlValue()`	`String`	Returns a string that represents this point that is suitable for use as a URL paramater value. It contains the latitude and the longitide in degrees to 6 decimal digits, in this order, separated by a comma, without whitespace.

Properties

These properties exist for backwards compatibility with v1 event handler functions only. They should not be used.

These properties mirror the return values of the `lng()` and `lat()` accessor methods and they allow a `GLatLng` to appear in places where a `GPoint` is expected by a v1 client. This is necessary where GLatLng appears in event details (i.e. in arguments of event handler functions). In contrast to method wrappers, it is impossible in the current infrastructure to create event wrappers.

Properties	Type	Description
`x`	`Number`	Deprecated.
`y`	`Number`	Deprecated.

class GLatLngBounds

A `GLatLngBounds` instance represents a rectangle in geographical coordinates, including one that crosses the 180 degrees meridian.

Constructor

Constructor	Description
`GLatLngBounds(sw?, ne?)`	Constructs a rectangle from the points at its south-west and north-east corners.

Methods

Methods	Return Value	Description
`equals(other)`	`Boolean`	Returns `true` iff all parameters in this rectangle are equal to the parameters of the other, within a certain roundoff margin.
`contains(latlng)`	`Boolean`	Returns `true` iff the geographical coordinates of the point lie within this rectangle.
`intersects(other)`	`Boolean`	What the name says.
`containsBounds(other)`	`Boolean`	What the name says.

`extend(latlng)`	*none*	Enlarges this rectangle such that it contains the given point. In longitude direction, it is enlarged in the smaller of the two possible ways. If both are equal, it is enlarged at the eastern boundary.
`getSouthWest()`	`GLatLng`	Returns the point at the south-west corner of the rectangle.
`getNorthEast()`	`GLatLng`	Returns the point at the north-east corner of the rectangle.
`toSpan()`	`GLatLng`	Returns a `GLatLng` whose cordinates represent the size of this rectangle.
`isFullLat()`	`Boolean`	Returns `true` if this rectangle extends from the south pole to the north pole.
`isFullLng()`	`Boolean`	Returns `true` if this rectangle extends fully around the earth in the longitude direction.
`isEmpty()`	`Boolean`	Returns true if this rectangle is empty.
`getCenter()`	`GLatLng`	Returns the point at the center of the rectangle. (Since 2.52)

interface GControl

This interface is implemented by all controls. You can implement it in order to provide a custom control for the map. Controls are added to the map using the `GMap2.addControl()` method.

In contrast to overlays, which are positioned relative to the map, controls are positioned relative to the map view, i.e. they don't move when the map moves.

Constructor

Constructor	Description
`GControl(printable?, selectable?)`	Creates the prototype instance for a new control class. Flag `printable` indicates that the control should be visible in the print output of the map. Flag `selectable` indicates that the control will contain text that should be selectable.

Methods

These methods will be called by the map when the control is added to the map using `GMap2.addControl()`. Thus, these methods will not be called by you, but they will be implemented by you.

Methods	Return Value	Description
`printable()`	`Boolean`	Returns to the map if the control should be printable.
`selectable()`	`Boolean`	Returns to the map if the control contains selectable text.
`initialize(map)`	`Node`	Will be called by the map so the control can initialize itself. The control will use the method `GMap2.getContainer()` to get hold of the DOM element that contains the map, and add itself to it. It returns the added element.
`getDefaultPosition()`	`GControlPosition`	Returns to the map the position in the map view at which the control appears by default. This will be overridden by the second argument to `GMap2.addControl()`.

class GControlPosition

This class described the position of a control in the map view. It consists of a corner relative to where the control is postioned, and an offset that determines this position. It can be passed as optional argument `position` to the method `GMap2.addControl()`, and it is returned from method `GControl.getDefaultPosition()`.

Constructor

Constructor	Description
`GControlPosition(anchor, offset)`	Creates a specification for a control position.

enum GControlAnchor

Constants

Constants	Description
`G_ANCHOR_TOP_RIGHT`	The control will be anchored in the top right corner of the map.
`G_ANCHOR_TOP_LEFT`	The control will be anchored in the top left corner of the map.
`G_ANCHOR_BOTTOM_RIGHT`	The control will be anchored in the bottom right corner of the map.
`G_ANCHOR_BOTTOM_LEFT`	The control will be anchored in the bottom left corner of the map.

class GControl

These implementations of `interface GControl` are available.

Constructor

Constructor	Description
`GSmallMapControl()`	Creates a control with buttons to pan in four directions, and zoom in and zoom out.
`GLargeMapControl()`	Creates a control with buttons to pan in four directions, and zoom in and zoom out, and a zoom slider.
`GSmallZoomControl()`	Creates a control with buttons to zoom in and zoom out.
`GScaleControl()`	Creates a control that displays the map scale.
`GMapTypeControl()`	Creates a control with buttons to switch between map types.

class GMapType

Google provides some predefined map types - this class is used to define custom ones. In order to show them on the map, use the `GMap2` constructor, or the `GMap2.addMapType()` method. See also `GTileLayerOverlay` to add to (rather than entirely replace) the map's tile layers.

This class can also be subclassed. Constructor arguments can be omitted if instantiated as a prototype. A subclass constructor must invoke the `GMapType` constructor using `call()`.

Constructor

Constructor	Description
`GMapType(layers, projection, name, opts?)`	Constructs a map type with the given tile layers, projection, name, and optional parameters.

Methods

These methods are mostly called by the map that this maptype is passed to, but some methods may also be called from outside the map, e.g. `getBoundsZoomLevel()`.

Methods	Return Value	Description
`getSpanZoomLevel(center, span, viewSize)`	`Number`	Returns to the map the zoom level at which the map section defined by center and span fits into the map view of the given size in pixels.
`getBoundsZoomLevel(bounds, viewSize)`	*none*	Returns to the map the zoom level at which the map section defined by bounds fits into the map view of the given size in pixels.
`getName(opt_short)`	`String`	Returns to the map the name of the map type to be used as the button label in the `GMapTypeControl`.
`getProjection()`	`GProjection`	Returns to the map the projection of this map type.
`getTileSize()`	`Number`	Returns to the map the map tile size in pixels of this map type. The tiles are assumed to be quadratic. All tile layers have the same tile size.
`getTileLayers()`	`Array of GTileLayer`	Returns to the map the array of tile layers.
`getMinimumResolution(latlng?)`	`Number`	Returns to the map the lowest zoom level at which this map type is defined.
`getMaximumResolution(latlng?)`	`Number`	Returns to the map the highest zoom level at which this map type is defined.
`getTextColor()`	`String`	Returns to the map the color that is best used for text that is overlaid on the map. Used for the color of the text of the copyright message displayed by the copyroght control.
`getLinkColor()`	`String`	Returns to the map the color that is best used for a hyperlink that is overlaid on the map. Used for the color of the link to the terms of use displayed by the copyroght control.
`getErrorMessage()`	`String`	Returns to the map the error message that is displayed in areas or on zoom level where this map type doesn't have map tiles.
`getCopyrights(bounds, zoom)`	`Array of String`	Returns to the map the copyright messages appropriate for the region described by `bounds` at the given zoom level. This is used by the `GCopyrightControl` to display the copyright message on the map.
`getUrlArg()`	`String`	Returns to the map a value that is used as a URL paramater value to identify this map type in permalinks to the current map view. This is currently only used by the maps application.

Constants

Constants	Description
`G_NORMAL_MAP`	This is the normal street map type.
`G_SATELLITE_MAP`	This map type shows Google Earth satellite images.
`G_HYBRID_MAP`	This map type shows transparent street maps over Google Earth satellite images.

`G_DEFAULT_MAP_TYPES`	An array of all three predefined map types described above.

Events

Events	Arguments	Description
`newcopyright`	`copyright`	This event is fired when a new copyright is added to the copyright collection of one of the tile layers contained in this map type.

class GMapTypeOptions

Instances of this class are used as the `opts?` argument to the `GMapType` constructor. There is no constructor for this class. Instead, this class is instantiated as a javascript object literal.

Properties

Properties	Type	Description
`shortName`	`String`	Sets the short name of the map type that is returned from `GMapType.getName(true)`. Default is the same as the `name`.
`urlArg`	`String`	Sets the url argument of the map type that is returned from `GMapType.getUrlArg()`. Default is the empty string.
`maxResolution`	`Number`	Sets the maximum zoom level of this map type, returned by `GMapType.getMaximumResolution()`. Default is the maximum of all tile layers.
`minResolution`	`Number`	Sets the minimum zoom level of this map type, returned by `GMapType.getMinimumResolution()`. Default is the minimum of all tile layers.
`tileSize`	`Number`	Set the tile size returned by `GMapType.getTileSize()`. Default is `256`.
`textColor`	`String`	Sets the text color returned by `GMapType.getTextColor()`. Default is `"black"`.
`linkColor`	`String`	Sets the text color returned by `GMapType.getLinkColor()`. Default is `"#7777cc"`.
`errorMessage`	`String`	Sets the error message returned by `GMapType.getErrorMessage()`. Default is the empty string.
`alt`	`String`	Sets the alternative text to the map type returned by `GMapType.getAlt()`. Default is the empty string. (Since 2.64)

interface GTileLayer

You implement this interface in order to provide custom map tile layers, either through `GMapType` or `GTileLayerOverlay`. Your implementation of this interface should use an instance of `GTileLayer` as a prototype, because this implements the copyright handling for you.

Constructor

Constructor	Description
`GTileLayer(copyrights, minResolution, maxResolution)`	Constructor arguments can be omitted if instantiated as a prototype. A Subclass constructor must invoke this constructor using `call()`.

Methods

These methods are called by the map and the map type to which this tile layer is passed. You will need to implement the methods marked abstract when you

implement a custom tile layer.

Methods	Return Value	Description
`minResolution()`	`Number`	Returns to the map type the lowest zoom level of this tile layer.
`maxResolution()`	`Number`	Returns to the map type the highest zoom level of this tile layer.
`getTileUrl(tile, zoom)`	`String`	Abstract. Returns to the map the URL of the map tile with the tile indices given by the `x` and `y` properties of the `GPoint`, at the given zoom level.
`isPng()`	`Boolean`	Abstract. Returns to the map whether the tiles are in PNG image format and hence can be transparent. Otherwise GIF is assumed.
`getOpacity()`	`Number`	Abstract. Returns to the map the opacity with which to display this tile layer. 1.0 is opaque, 0.0 is transparent.

Events

Events	Arguments	Description
`newcopyright`	`copyright`	This event is fired when a new copyright is added to the copyright collection of this tile layer.

class GTileLayerOverlay

A `GTileLayerOverlay` augments the map with a `GTileLayer`. It implements the `GOverlay` interface and thus is added to the map using the `GMap2.addOverlay()` method. The `GTileLayer` is presented on top of the existing map imagery - to replace the imagery instead, put the `GTileLayer` inside a custom `GMapType`.

Constructor

Constructor	Description
`GTileLayerOverlay(tileLayer)`	Creates a `GOverlay` that wraps the `tileLayer`. (Since 2.61)

Methods

Methods	Return Value	Description
`()`	*none*	
`hide()`	*none*	Hides this overlay so it is not visible, but maintains its position in the stack of overlays. (Since 2.71)
`show()`	*none*	Shows a previously hidden TileLayerOverlay. (Since 2.71)

class GCopyrightCollection

You use this class to manage copyright messages displayed on maps of custom map type. If you don't implement custom map types, then you don't need to use this class. A copyright collection contains information about which copyright to display for which region on the map at which zoom level. This is very important for map types that display heterogenous data such as the satellite map type.

Constructor

Constructor	Description
`GCopyrightCollection(prefix?)`	Copyright messages produced from this copyright collection will have the common prefix given as the argument. Example: "Imagery (C) 2006"

Methods

Methods	Return Value	Description
`()`	*none*	
`addCopyright(copyright)`	*none*	Adds a copyright information object to the collection.
`getCopyrights(bounds, zoom)`	`Array of String`	Returns all copyright strings that are pertinent for the given map region at the given zoom level. Example: ["Google", "Keyhole"]
`getCopyrightNotice(bounds, zoom)`	`String`	Returns the prefix and all relevant copyright strings that are pertinent for the given map region at the given zoom level. Example: "Imagery (C) 2006 Google, Keyhole"

Events

Events	Arguments	Description
`newcopyright`	`copyright`	This event is fired when a new copyright was added to this copyright collection.

class GCopyright

A copyright object contains information about which copyright message applies to a region of the map given by a rectangle, at a given zoom level. You need this object only if you implement custom map types or tile layers.

Constructor

Constructor	Description
`GCopyright(id, bounds, minZoom, text)`	Creates a copyright information object with the given properties.

Properties

Properties	Type	Description
`id`	`Number`	A unique identifier for this copyright information.
`minZoom`	`Number`	The lowest zoom level at which this information applies.
`bounds`	`GLatLngBounds`	The region to which this information applies.
`text`	`String`	The text of the copyright message.

interface GProjection

This is the interface for map projections. A map projection instance is passed to the constructor of `GMapType`. This interface is implemented by the `class GMercatorProjection`, which is used by all predefined map types. You can implement this interface if you want to define map types with different map projections.

Methods

These methods are called by the map. You need to implement them.

Methods	Return Value	Description
`fromLatLngToPixel(latlng, zoom)`	`GPoint`	Returns the map coordinates in pixels for the point at the given geographical coordinates, and the given zoom level.
`fromPixelToLatLng(pixel, zoom, unbounded?)`	*none*	Returns the geographical coordinates for the point at the given map coordinates in pixels, and the given zoom level. Flag `unbounded` causes the geographical longitude coordinate not to wrap when beyond the -180 or 180 degrees meridian.
`tileCheckRange(tile, zoom, tilesize)`	*none*	Returns to the map if the tile index is in a valid range for the map type. Otherwise the map will display an empty tile. It also may modify the `tile` index to point to another instance of the same tile in the case that the map contains more than one copy of the earth, and hence the same tile at different tile coordinates.
`getWrapWidth(zoom)`	*none*	Returns to the map the periodicity in x-direction, i.e. the number of pixels after which the map repeats itself because it wrapped once round the earth. By default, returns `Infinity`, i.e. the map will not repeat itself. This is used by the map to compute the placement of overlays on map views that contain more than one copy of the earth (this usually happens only at low zoom levels). (Since 2.46)

class GMercatorProjection

This implementation of the `GProjection` interface for the mercator projection is used by all predefined map types.

Constructor

Constructor	Description
`GMercatorProjection(zoomlevels)`	Creates a mercator projection for the given number of zoom levels.

Methods

Methods	Return Value	Description
`fromLatLngToPixel(latlng, zoom)`	`GPoint`	See `GProjection`.
`fromPixelToLatLng(pixel, zoom, unbounded?)`	`GLatLng`	See `GProjection`.
`checkTileRange(tile, zoom, tilesize)`	*none*	See `GProjection`.
`getWrapWidth(zoom)`	*none*	See `GProjection`. Mercator projection is periodic in longitude direction, therefore this returns the width of the map of the entire Earth in pixels at the given zoom level. (Since 2.46)

namespace GEvent

This namespace contains functions that you use to register event handlers, both for custom events and for DOM events, and to fire custom events. All the events defined by this API are custom events that are internally fired by `GEvent.triggerEvent()`.

Static Methods

Static Methods	Return Value	Description
`addListener(source, event, handler)`	`GEventListener`	Registers an event handler for a custom event on the source object. Returns a handle that can be used to eventually deregister the handler. The event handler will be called with `this` set to the source object.
`addDomListener(source, event, handler)`	`GEventListener`	Registers an event handler for a DOM event on the source object. The source object must be a DOM Node. Returns a handle that can be used to eventually deregister the handler. The event handler will be called with `this` set to the source object. This function uses the DOM methods for the current browser to register the event handler.
`removeListener(handle)`	*none*	Removes a handler that was installed using `addListener()` or `addDomListener()`.
`clearListeners(source, event)`	*none*	Removes all handlers on the given object for the given event that were installed using `addListener()` or `addDomListener()`.
`clearInstanceListeners(source)`	*none*	Removes all handlers on the given object for all events that were installed using `addListener()` or `addDomListener()`.
`trigger(source, event, ...)`	*none*	Fires a custom event on the source object. All arguments after `event` are passed as arguments to the event handler functions.
`bind(source, event, object, method)`	*none*	Registers an invocation of the method on the given object as the event handler for a custom event on the source object. Returns a handle that can be used to eventually deregister the handler.
`bindDom(source, event, object, method)`	*none*	Registers an invocation of the method on the given object as the event handler for a custom event on the source object. Returns a handle that can be used to eventually deregister the handler.
`callback(object, method)`	*none*	Returns a closure that calls `method` on `object`.
`callbackArgs(object, method, ...)`	*none*	Returns a closure that calls `method` on `object` with the given arguments.

Events

Events	Arguments	Description
`clearlisteners`	`event?`	This event is fired on object when `clearListeners()` or `clearInstanceListeners()` is called on that object. Of course, the event is fired before the functions are executed.

class GEventListener

This class is opaque. It has no methods and no constructor. Its instances are returned from `GEvent.addListener()` or `GEvent.addDomListener()` and are eventually passed back to `GEvent.removeListener()`.

namespace GXmlHttp

This namespace provides a factory method to create `XmlHttpRequest` instances in a browser independent way.

Static Methods

Static Methods	Return Value	Description
`create()`	`GXmlHttp`	Factory to create a new instance of `XmlHttpRequest`.

namespace GXml

This namespace provides static methods to handle XML documents and document fragments.

Static Methods

Static Methods	Return Value	Description
`parse(xmltext)`	`Node`	Parses the given string as XML text and returns a DOM representation. If the browser doesn't support XML parsing natively, this returns the DOM node of an empty `DIV` element.
`value(xmlnode)`	`String`	Returns the text value (i.e., only the plain text content) of the XML document fragment given in DOM representation.

class GXslt

This class provides methods to apply XSLT to XML in a browser-independent way.

Static Methods

Static Methods	Return Value	Description
`create(xsltnode)`	`GXslt`	Creates a `GXslt` instance from the XSLT stylesheet given as DOM representation.
`transformToHtml(xmlnode, htmlnode)`	`Boolean`	Uses the XSLT stylesheet given in the constructor of this `GXslt` instance to transform the XML document given as DOM representation in `xmlnode`. Appends the resulting HTML document fragment to the given `htmlnode`. This only works if the browser natively supports XSL transformations, in which case it will return `true`. Otherwise, this function will do nothing and return `false`.

namespace GLog

This namespace contains some static methods that help you to debug web applications. When you use one of the `write*()` methods for the first time, a floating window opens on the page and displays the written messages.

Static Methods

Static Methods	Return Value	Description
`write(message, color?)`	*none*	Writes the message as plain text into the log window. HTML markup characters will be escaped so that they are visible as characters.

`writeUrl(url)`	*none*	Writes a link to the given URL into the log window.
`writeHtml(html)`	*none*	Writes text as HTML in the log window.

class GDraggableObject

This class makes a DOM element draggable. The static members for changing the drag cursors affect all subsequently created draggable objects, such as the map, zoom control slider, and overview map rectangles. The per-instance members affect only their particular instance. For example, before creating the map, you can call `GDraggableObject.setDraggableCursor('default')` and `GDraggableObject.setDraggingCursor('move')` to get the pre-API 2.56 style. Alternatively, the Map constructor can take options to set its DraggableObject's cursor style. See the W3C CSS specification for allowable cursor values.

Constructor

Constructor	Description
`GDraggableObject(src, opts?)`	Sets up event handlers so that the source element can be dragged. Left and top optionally position the element, and the optional container serves as a bounding box. (Since 2.59)

Static Methods

Static Methods	Return Value	Description
`setDraggableCursor(cursor)`	*none*	Sets the draggable cursor for subsequently created draggable objects. (Since 2.59)
`setDraggingCursor(cursor)`	*none*	Sets the dragging cursor for subsequently created draggable objects. (Since 2.59)

Methods

Methods	Return Value	Description
`setDraggableCursor(cursor)`	*none*	Sets the cursor when the mouse is over this draggable objects. (Since 2.59)
`setDraggingCursor(cursor)`	*none*	Sets the cursor when the mouse is held down, dragging this draggable object. (Since 2.59)

This class represents optional arguments to the `GDraggableObject` constructor. It has no constructor, but is instantiated as object literal.

Properties

Properties	Type	Description
`left`	`Number`	The left starting position of the object. (Since 2.59)
`top`	`Number`	The top starting position of the object. (Since 2.59)
`container`	`Node`	A DOM element that will act as a bounding box for the draggable object (Since 2.59)
`draggableCursor`	`String`	The cursor to show on mousover. (Since 2.59)
`draggingCursor`	`String`	The cursor to show while dragging. (Since 2.59)

enum GGeoStatusCode

Numeric equivalents for each symbolic constant are specified in parentheses.

Constants

Constants	Description
`G_GEO_SUCCESS (200)`	No errors occurred; the address was successfully parsed and its geocode has been returned. (Since 2.55)
`G_GEO_SERVER_ERROR (500)`	A geocoding request could not be successfully processed, yet the exact reason for the failure is not known. (Since 2.55)
`G_GEO_MISSING_ADDRESS (601)`	The HTTP `q` parameter was either missing or had no value. (Since 2.55)
`G_GEO_UNKNOWN_ADDRESS (602)`	No corresponding geographic location could be found for the specified address. This may be due to the fact that the address is relatively new, or it may be incorrect. (Since 2.55)
`G_UNAVAILABLE_ADDRESS (603)`	The geocode for the given address cannot be returned due to legal or contractual reasons. (Since 2.55)
`G_GEO_BAD_KEY (610)`	The given key is either invalid or does not match the domain for which it was given. (Since 2.55)

enum GGeoAddressAccuracy

There are no symbolic constants defined for this enumeration.

Constants

Constants	Description
`0`	Unknown location. (Since 2.59)
`1`	Country level accuracy. (Since 2.59)
`2`	Region (state, province, prefecture, etc.) level accuracy. (Since 2.59)
`3`	Sub-region (county, municipality, etc.) level accuracy. (Since 2.59)
`4`	Town (city, village) level accuracy. (Since 2.59)
`5`	Post code (zip code) level accuracy. (Since 2.59)
`6`	Street level accuracy. (Since 2.59)
`7`	Intersection level accuracy. (Since 2.59)
`8`	Address level accuracy. (Since 2.59)

class GClientGeocoder

This class is used to communicate directly with Google servers to obtain geocodes for user specified addresses. In addition, a geocoder maintains its own cache of addresses, which allows repeated queries to be answered without a round trip to the server.

Constructor

Constructor	Description
`GClientGeocoder(cache?)`	Creates a new instance of a geocoder that talks directly to Google servers. The optional cache parameter allows one to specify a custom client-side cache of known addresses. If none is specified, a `GFactualGeocodeCache` is used. (Since 2.55)

Methods

Methods	Return Value	Description
`getLatLng(address, callback)`	*none*	Sends a request to Google servers to geocode the specified address. If the address was successfully located, the user-specified callback function is invoked with a `GLatLng` point. Otherwise, the callback function is given a `null` point. In case of ambiguous addresses, only the point for the best match is passed to the callback function. (Since 2.55)
`getLocations(address, callback)`	*none*	Sends a request to Google servers to geocode the specified address. A reply that contains status code, and if successful, one or more `Placemark` objects, is passed to the user-specified callback function. Unlike the `GClientGeocoder.getLatLng` method, the callback function may determine the reasons for failure by examining the code value of the `Status` field. (Since 2.55)
`getCache()`	`GGeocodeCache`	Returns currently used geocode cache, or `null`, if no client-side caching is performed. (Since 2.55)
`setCache(cache)`	*none*	Sets a new client-side caching. If this method is invoked with cache set to `null`, client-side caching is disabled. Setting a new cache discards previously stored addresses. (Since 2.55)
`reset()`	*none*	Resets the geocoder. In particular this method calls the `GGeocodeCache.reset` method on the client-side cache, if one is used by this geocoder. (Since 2.55)

class GGeocodeCache

This class maintains a map from addresses to known locations. While this class is fully funtional, it is intended as a base class from which more sophisticated caches are derived.

Constructor

Constructor	Description
`GGeocodeCache()`	Creates a new cache for storing a map from adresses to locations. The constructor immediately calls the `GGeocodeCache.reset` method. (Since 2.55)

Methods

Methods	Return Value	Description
`get(address)`	`Object`	Returns the reply which was stored under the given `address`. If no reply was ever stored for the given address, this method returns `null`. (Since 2.55)
`isCachable(reply)`	`Boolean`	Returns whether or not the given reply should be cached. By default very rudimentary checks are performed on the `reply` object. In particular, this class makes sure that the object is not `null` and that it has the `name` field . This method may be overridden by extending classes to provide more precise conditions on the `reply` object. (Since 2.55)
`put(address, reply)`	*none*	Stores the given `reply` under the given `address`. This method calls the `GGeocodeCache.isCachable` method to verify that the `reply` may be cached. If it gets a go-ahead, it caches the `reply` under the address normalized with the help of the `GGeocodeCache.toCanoninical` method. (Since 2.55)
`reset()`	*none*	Purges all replies from the cache. After this method returns, the cache is empty. (Since 2.55)

`toCanonical(address)`	`String`	Returns what is considered a canonical version of the address. It converts the `address` parameter to lower case, replaces comas with spaces and replaces multiple spaces with one space. (Since 2.55)

class GFactualGeocodeCache

This class refines the basic `GGeocodeCache` class by placing stricter conditions on cached replies. It only caches replies which are very unlikely to change within a short period of time.

Constructor

Constructor	Description
`GFactualGeocodeCache()`	Creates a new cache that stores only replies it considers factual. (Since 2.55)

Methods

Methods	Return Value	Description
`isCachable(reply)`	`Boolean`	Overrides the default implementation of this method to perform a more thorough check of the status code. Only a reply with `Status.code` set to `G_GEO_SUCCESS`, or known to be invalid, is considered cachable. Replies that timed out or resulted in a generic server error are not cached. (Since 2.55)

class GMarkerManager

This class is used to manage visibility of hundreds of markers on a map, based on the map's current viewport and zoom level.

Constructor

Constructor	Description
`GMarkerManager(map, opts?)`	Creates a new marker manager that controlls visibility of markers for the specified map. (Since 2.67)

Methods

Methods	Return Value	Description
`addMarkers(markers, minZoom, maxZoom?)`	*none*	Adds a batch of markers to this marker manager. The markers are not added to the map, until the `refresh()` method is called. Once placed on a map, the markers are shown if they fall within the map's current viewport and the map's zoom level is greater than or equal to the specified `minZoom`. If the `maxZoom` was given, the markers are automatically removed if the map's zoom is greater than the one specified. (Since 2.67)

`addMarker(marker, minZoom, maxZoom?)`	*none*	Adds a single marker to a collection of markers controlled by this manager. If the marker's location falls within the map's current viewport and the map's zoom level is within the specified zoom level rage, the marker is immediately added to the map. Similar to the `addMarkers` method, the `minZoom` and the optional `maxZoom` parameters specify the range of zoom levels at which the marker is shown. (Since 2.67)
`refresh()`	*none*	Forces the manager to update markers shown on the map. This method must be called if markers were added using the `addMarkers` method. (Since 2.67)
`getMarkerCount(zoom)`	`Number`	Returns the total number of markers potentially visible at the given zoom level. This may include markers at lower zoom levels. (Since 2.67)

Events

Events	Arguments	Description
`changed`	`bounds, markerCount`	This event is fired when markers managed by a manager have been added to or removed from the map. The event handler function should be prepared to accept two arguments. The fist one is the rectangle definining the bounds of the visible grid. The second one carries the number of markers currently shown on the map.

class GMarkerManagerOptions

This class represents optional arguments to the `GMarkerManager` constructor. It has no constructor, but is instantiated as object literal.

Properties

Properties	Type	Description
`borderPadding`	`Number`	Specifies, in pixels, the extra padding outside the map's current viewport monitored by a manager. Markers that fall within this padding are added to the map, even if they are not fully visible. (Since 2.67)
`maxZoom`	`Number`	Sets the maximum zoom level monitored by a marker manager. If not given, the manager assumes the maximum map zoom level. This value is also used when markers are added to the manager without the optional `maxZoom` parameter. (Since 2.67)
`trackMarkers`	`Boolean`	Indicates whether or not a marker manager should track markers' movements. If you wish to move managed markers using the `setPoint` method, this option should be set to `true`. The default value is `false`. (Since 2.67)

function GDownloadUrl

This function provides a convenient way to asynchronously retrieve a resource identified by a URL. Notice that, since the XmlHttpRequest object is used to execute the request, it is subject to the same-origin restriction of cross-site scripting, i.e. the URL must refer to the same server as the URL of the current document that executes this code. Therefore, it is usually redundant to use an absolute URL for the `url` argument, and it is better to use an absolute or relative path only.

Function

Function	Return Value	Description

`GDownloadUrl(url, onload)`	*none*	Retrieves the resource from then given URL and calls the `onload` function with the text of the document as first argument, and the HTTP response status code as the second. This is subject to cross site scripting restrictions. Uses the underlying `XmlHttpRequest` implementation of the browser.

function GBrowserIsCompatible

This function decides whether the maps API can be used in the current browser.

Function

Function	Return Value	Description
`GBrowserIsCompatible()`	`Boolean`	Returns `true` if the current browser supports the maps API library.

function GUnload

You can call this function to cause the map API to cleanup internal data structures to release memory. This helps you to work around various browser bugs that cause memory leaks in web applications. You should call this function in the `unload` event handler of your page. After this function was called, the map objects that you've created in this page will be dysfunctional.

Function

Function	Return Value	Description
`GUnload()`	*none*	Dismantles all registered event handlers in order to prevent memory leaks. Should be called as a handler for the `unload` event.