Google Maps and Geolocation

Rev. Digitale 1.4 del 05/01/2021

Geolocation

HTML 5 offre supporto alle **geolocalizzazione**, cioè la localizzazione geografica di un utente attraverso le sue coordinate geografiche. Queste coordinate possono essere fornite in due modi:

- Nel caso di un dispositivo mobile da un sensore **GPS** (Global Positioning System)
- Nel caso di un PC possono essere fornite in modo approssimato dall'indirizzo IP della connessione internet.

Cenni sul sistema GPS

Il sistema GPS è un sistema di posizionamento e navigazione satellitare civile che, attraverso una rete satellitare dedicata di satelliti artificiali in orbita, fornisce ad un terminale mobile o ricevitore GPS informazioni sulle sue coordinate geografiche ed orario, in ogni condizione meteorologica, ovunque sulla Terra o nelle sue immediate vicinanze ove vi sia un contatto privo di ostacoli con almeno quattro satelliti del sistema. La localizzazione avviene tramite la trasmissione di un segnale radio da parte di ciascun satellite e l'elaborazione dei segnali ricevuti da parte del ricevitore. Il sistema GPS è gestito dal governo degli Stati Uniti d'America ed è liberamente accessibile da chiunque sia dotato di un ricevitore GPS. Il suo grado attuale di accuratezza è dell'ordine dei metri.

Le coordinate GPS sono espresse mediante la seguente coppia di numeri decimali:

1. **latitudine** (**numeri positivi = nord / sopra l'equatore**, **numeri negativi = sud / sotto l'equatore**)
2. **longitudine** rispetto a Greenwich (**numeri positivi = est**, **numeri negativi = ovest**)

entrambi con una precisione di 15 cifre decimali (double) di cui occorre usarne sempre **almeno 7**

Mappe Statiche di google (2D)

Per poter accedere alle google maps (statiche 2D, 3D, API) il link è sempre il seguente:

http://maps.googleapis.com/maps/api/

Le mappe statiche sono delle immagini fisse che google mette a disposizione del programmatore per poterle inserire nel proprio sito, ad esempio per indicare come raggiungere la sede aziendale.

La url completa per accedere alle mappe statiche 2D di google maps è la seguente:

```
https://maps.googleapis.com/maps/api/staticmap?nomeParametro=valore
```

Parametri

key scritto rigorosamente in **minuscolo**. Contiene la API key di identificazione

center definisce dove deve essere centrata la mappa (coordinate GPS del punto centrale). Le coordinate possono essere espresse come valori GPS oppure tramite il nome della località
Comune di Fossano : latitudine Nord +44.552304, longitudine Est +7.762786
Vallauri : 44.5557763, 7.7347183
Comune di Cavallermaggiore : 44.7077582, 7.6877771
Colosseo : 41.890617421405366, 12.493300437927246

zoom Definisce il valore di zoom tra 1 e 25 (i valori maggiori non provocano variazioni rispetto a 25) zoom = 1 mostra l'intero globo. Valori normalmente utilizzati : 16 / 17

size Dimensioni dell'area da visualizzare rispetto al CENTER (base x altezza, es 800 x 600).

Es: `http://maps.googleapis.com/maps/api/staticmap?center=piazza+navona,roma&zoom=15&size=640x480&key=**********$maptype=roadmap`

maptype Indica il tipo di mappa da visualizzare fra le seguenti: (scritte in **minuscolo**)

roadmap	mappa stradale (default)
satellite	mappa satellitare pura (oggi non più usata)
hybrid	mappa satellitare ibrida comprensiva anche delle informazioni stradali
terrain	mappa fisica del territorio (accessibile dal menù delle google maps / **Rilievo**)

format Formato dell'immagine tra GIF, JPG (default) e PNG

markers Consente di definire dei marcatori da posizionare sulla mappa per indicare la posizione del luogo ricercato. Per ogni marcatore occorre specificare le coordinate o il nome della località. Per ogni marcatore si può impostare uno o più stili separati dal pipe (|) e scritti secondo le regole dei css. Es: `markers=color:red|size:big|label:A|piazza+navona,roma`

Gli attributi devono essere scritti PRIMA delle coordinate. I possibili attributi sono:
color = colore del marker(accettato soltanto il nome alfanumerico)
size = dimensione fra big, mid, normal, small, tiny,
label = carattere alfanumerico da visualizzare all'interno del marker. E' accettato un **singolo carattere scritto in maiuscolo**. In tutti gli altri casi viene inserito automaticamente un puntino.

Mappe Statiche 3D (Street View)

La url per accedere alle mappe statiche 3D (**street view**) di google maps è la seguente:

`http://maps.googleapis.com/maps/api/streetview?nomeParametro=valore`

Parametri
Sono più o meno gli stessi del caso precedente. Non esistono ***maptype*** e ***center***, quest'ultimo sostituito da **location.** Sono invece disponibili i seguenti parametri aggiuntivi:

heading Angolazione dell'obiettivo sull'asse orizzontale fra 0 e 360° che indicano entrambi il nord. 90° indica est, 180° indica sud, 270 indica ovest (rotazione antioraria).

pitch angolazione dell'obiettivo verso l'alto (numeri positivi) oppure verso il basso (numeri negativi). espressa in gradi. 0°=orizzontale (max 30°)

fov zoom del'obiettivo (distanza focale) tra 20 e 200. Valori piccoli indicano che siamo **vicini** al soggetto da fotografare, e quindi l'immagine risulta maggiormente ingrandita.
Valore Typico: **100**.

zoom sullo streetView non è riconosciuto

format come 2D

markers come 2D

Se si utilizza il nome alfanumerico invece delle coordinate LAT/LNG occorre modificare il valore heading

Collegamento dinamico alle google maps

Il collegamento diretto alle google maps può essere eseguito tramite un **iframe** con il seguente **src** :

`src="https://www.google.it/maps/embed?nomeParametro=valore`

con una lunga lista di parametri che possono essere copiati direttamente dalla pagina delle google maps(**Menù / Condividi e incorpora mappa / incorpora mappa**).
Dal 2014 i vari parametri sono stati accorpati in un unico parametro **pb** di tipo stringa e suddivisi da un !

Per eseguire questo tipo di collegamento **NON serve** la API Key

Le API v3 per la gestione delle Google Maps

Data di rilascio ufficiale v1: **13 novembre 2013**
Ultima versione (2018): **3.32 del 13 febbraio 2018**

Sono un insieme di API che consentono la gestione delle google maps '**da codice'** con diverse possibilità di personalizzazione. Hanno un peso molto ridotto (35k) sono compatibili con tutti i dispositivi mobile. Per poter utilizzare all'interno di una pagina HTML occorre inserire il seguente link:

```
<script
      src="https://maps.googleapis.com/maps/api/js?key=**************">
</script>
```

Gli oggetti Map, LatLng, Point

Tutte le classi sono contenute all'interno del namespace **google.maps**

google.maps.Map

Oggetto fondamentale. Consente di creare una mappa associata ad un tag **DIV** indicato e avente le opzioni indicate. P**er il tag DIV occorre definire width e height altrimenti il default è 0 e l'oggetto Map non carica nulla.**

```
var map = new google.maps.Map(div, opzioni);
```

oggetto Point

Si tratta di un semplice json contenente i due campi **lat** e **lng**.

```
var posizione = {lat:44.7077582, lng:7.6877771}
```

google.maps.LatLng

Si tratta di un oggetto simile all'oggetto Point precedente. All'interno di Point **lat** e **lng**.sono semplici campi, mentre l'oggetto LatLng dispone dei seguenti metodi:

- un **costruttore** che accetta come parametri due numeri float indicanti le coordinate GPS ma che NON accetta come parametri stringhe alfanumeriche
- **.lat()** restituisce la latitudine del punto corrente
- **.lng()** restituisce la longitudine del punto corrente
- **.toString()** restituisce una stringa contenente le due coordinale
- **.toJSON()** Restituisce n json di tipo JSON (che deve essere eventualmente serializzato)

Dunque si tratta di un oggetto più completo del precedente, che può essere facilmente serializzato con un semplice .toString(). Esempio:

```
var posizione = new google.maps.LatLng(44.7077582,  7.6877771);
alert (posizione.toString())
```

Opzioni dell'oggetto Map

E' un json definito all'interno di **google.maps.MapOptions** interface. I campi più importanti sono:

center centro mappa. Accetta come parametro sia un oggetto **LatLng** sia un semplice json **Point**.
zoom E' lo stesso delle Mappe Statiche. Definisce un valore di zoom tra 1 (globo) e 25 (max zoom)
mapTypeId Accetta un oggetto di tipo `google.maps.MapTypeId` i cui possibili valori sono gli stessi delel mappe statiche però scritti in maiuscolo: ROADMAP (default), SATELLITE, HYBRUD e TERRAIN.

Esempio :

```
window.onload = function(){
      var div = document.getElementById("wrapper");
      var posizione = new google.maps.LatLng(44.7077582,7.6877771);
      var mapOptions = {
           center:posizione,
           zoom:16,
           mapTypeId: google.maps.MapTypeId.HYBRID  };
      var mappa = new google.maps.Map(div, mapOptions);
}
```

I valori delle option possono essere letti/modificati anche in un 2° tempo tramite i seguenti metodi:

```
mappa.setCenter(currentPoint)
mappa.setZoom(13)
mappa.setMapTypeId(google.maps.MapTypeId.TERRAIN)
mappa.setCenter(marcatore.getPosition())
```

`mappa.panTo(currentPoint)` è simile a **`setCenter()`** ma esegue lo spostamento verso la nuova posizione tramite una piccola animazione (smoothly, liscia)

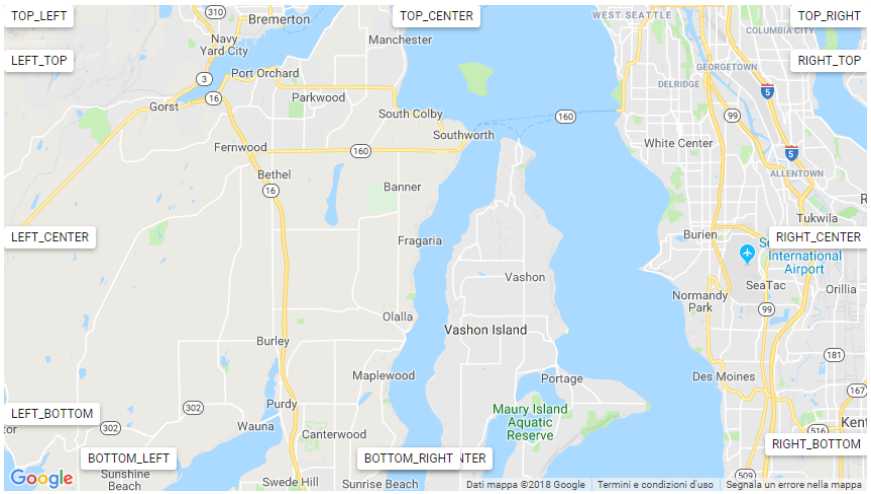
Personalizzazione dei pulsanti di controllo

All'interno delle opzioni precedenti è possibile aggiungere alcune opzioni che consentono il posizionamento personalizzato dei pulsanti di controllo della mappa.

```
// disabilito tutti i pulsanti di controllo
disableDefaultUI:true,

// Pulsanti di zoom + -
zoomControl: true,
zoomControlOptions: {
   // style: deprecata,
   position: google.maps.ControlPosition.RIGHT_CENTER
},

// Omino StreetView
streetViewControl: true,
streetViewControlOptions: {
   position: google.maps.ControlPosition.RIGHT_CENTER
},

// Pulsanti switch Mappa/Satellite (il chek Rilievo visualizza il TERRAIN)
mapTypeControl: true,
mapTypeControlOptions: {
   style: google.maps.MapTypeControlStyle.HORIZONTAL_BAR, // default
   style: google.maps.MapTypeControlStyle.DROPDOWN_MENU,  // verticale
   position: google.maps.ControlPosition.TOP_LEFT,
},

// Pulsante FullScreen
fullscreenControl: true,
fullscreenOptions: {
   // Non ha opzioni, utilizza una posizione fissa in alto a destra
},
```

```
// Visualizza nella riga di stato in basso a destra un fattore di scala
// per default è disabilitato
scaleControl: true,
scaleControlOptions: {
   // Non ha opzioni
}
```

Le varie costanti di posizionamento sono dettagliate nella seguente mappa:



L'oggetto Marker

Consente di associare dei Marker alla Mappa

```
var marcatore1 = new google.maps.Marker({
   map:mappa,
   position:posizione,
   title:"Martin's Guest House",    // tooltip
   animation: google.maps.Animation.DROP,
   zIndex: 3,       // in caso di marcatori vicini/sovrapposti
   draggable:true, // rende il marcatore 'trascinabile'
   label:"A",
   icon:icon1,
});
```

- Il campo **icon** consente di specificare una icona personalizzata.
- **icon** e **label**, sono di solito alternativi, o si utilizza uno oppure si utilizza l'altro (altrimenti si sovrappongono)
- Se non si specificano nè **icon** nè **label**, viene utilizzata una **icona di default** a forma di palloncino con colore di sfondo rosso e pallino nero centrale.
  Questa icona non sembra modificabile, se non per sostituzione con un'altra icona personalizzata.
- Il campo **icon** può puntare direttamente ad una immagine tramite l'url (***simple icon***) oppure può puntare ad un oggetto icon all'interno del quale si possono meglio specificare le opzioni dell'icona (***complex icon***)

Invece di definire le varie proprietà in modo statico al momento dell'istanza del Marker, è anche possibile leggerle / modificarle in un secondo tempo tramite i metodi corrispondenti:

```
marcatore1.setPosition(newPosition)
marcatore1.getPosition()
marcatore1.setMap(null);    // per eliminare il marcatore dalla mappa
```

Esempio di icona personalizzata:

```
var icon1 = {
   url: 'icone/hotfoodcheckpoint.png',
   scaledSize: new google.maps.Size(16, 19),
   anchor: new google.maps.Point(30, 40),
   origin: new google.maps.Point(0,0),
}
```

scaledSize rappresenta la dimensione con cui visualizzare l'icona. Affinchè l'icona non venga deformata occorre verificare le dimensioni dell'icona (ad esempio con Paint) ed applicare delle dimensioni proporzionali. Se si omette, l'icona viene visualizzata con le sue dimensioni reali.
anchor rappresenta un offset rispetto alla posizione definita da position. Per spostare leggermente il marcatore rispetto alla posizione di visualizzazione. Nell'esempio 30px verso sinistra e 40px verso l'alto
origin consente di specificare un offset di visualizzazione dell'icona all'interno del marcatore. Consente di traslare le icone quando ho più icone memorizzate tutte nello stesso file. Default 0,0

Dove trovare le icone

mapicons.mapsmarker.com/
Presenta una infinità di icone suddivise in gruppi e scaricabili per gruppi.
Per ogni icona sono poi presenti 8 diverse versioni.

map-icons.com, con disponibilità di CDN. Consente di comporre l'icona su una delle 7 forme e di personalizzare i colori tramite css. Ha però uno standard suo con proprietà abbastanza diverse rispetto allo standard dei Marker di google.

L'oggetto `InfoWindow`

Spesso abbinata ai marcatori. In corrispondenza del click sul marcatore, consente di aprire una finestra con informazioni aggiuntive sul punto indicato dal marcatore.

```
var infoWindow = new google.maps.InfoWindow({
   content:
      `<div id="infoWindow">
         <h2> ITIS Vallauri
            <img src="img/vallauri.jpg" align="top">
         </h2>
         <p>indirizzo: Via San Michele 68, Fossano</p>
         <p>coordinate GPS: ${position.toString()} </p>
      </div>`,
      width:150
});

marcatore1.addListener('click', function() {
      finestra.open(mappa, marcatore1);
});
```

Il primo parametro di open rappresenta l'oggetto genitore all'interno del quale aprire la finestra.
Il secondo parametro di open rappresenta la posizione dove aprire la finestra, posizione che di solito (ma NON necessariamente) coincide con il marker sul quale si è fatto click.

L'oggetto `Geocoder`

L'oggeto Geocoder consente di convertire un indirizzo alfanumerico ("Fossano Via San Michele 68") in un json corrispondente contenente tutte le informazioni relative all'indirizzo indicato, da cui poter estrarre le coordinate Lat e Lng. **L'oggetto è pensato per indirizzi esistenti ed univoci**. Per indirizzi incompleti esistono altri oggetti che consentono di utilizzare dei filtri per arrivare all'indirizzo esatto

Il metodo **`geocoder.geocode()`** si aspetta come parametro il seguente json:

```
{
    address: string,
    location: LatLng,
    placeId: string,
    bounds: LatLngBounds,
    componentRestrictions: GeocoderComponentRestrictions,
    region: string
}
```

I primi tre parametri sono alternativi fra loro. E' possibile passare al metodo `geocode()` una stringa alfanumerica relativa alla località da ricercare, oppure un corrispondente oggetto **LatLng**, oppure il corrispondente **placeId** (ogni luogo delle google maps è identificato da un apposito placeId univoco). Gli altri parametri sono facoltativi e consentono comunque di restringere il campo di ricerca.

Il metodo **`geocoder.geocode()`** in tutti i casi restituisce, tramite una funzione di callback, un oggetto **GeocoderResult** contenente TUTTE le informazioni relative alla località ricercata e strutturato nel modo seguente:

```
{
 "formatted_address" : "Via S. Michele, 68, 12045 Fossano CN, Italia",
 "geometry" : {"location" : {"lat":44.555452,  "lng":7.736542999999983},    // Point
 "types" : ["street_address"] ,
"address_components":    [
      {"long_name":"68",  "short_name":"68", "types":["street_number"]},
      {"long_name":"Via San Michele", "short_name":"Via S. Michele", "types":["route"]},
      {"long_name":"Fossano",  "short_name":"Fossano", "types":["locality","political"]},
      {"long_name":"Fossano", "short_name":"Fossano","types":["administrative_area_level_3","political"]},
      {"long_name":"Provincia di Cuneo", "short_name":"CN", "types":["administrative_area_level_2","political"]},
      {"long_name":"Piemonte", "short_name":"Piemonte", "types":["administrative_area_level_1","political"]},
      {"long_name":"Italia", "short_name":"IT", "types":["country","political"]},
      {"long_name":"12045", "short_name":"12045", "types":["postal_code"]}
 ],
 "location_type":"ROOFTOP",
 "place_id":"ChIJa0Nw_KtZzRIRXqu7iJC19CA",
 "viewport": {  "south":44.5541030197085,
                "west":7.735194019708501,
                "north":44.55680098029149,
                "east":7.737891980291465  }
}
```

Esempio

```
var address = "Fossano Via San Michele 68";
var geocoder = new google.maps.Geocoder();
```

```
geocoder.geocode( {'address': address}, function(results, status) {
     if (status == google.maps.GeocoderStatus.OK)
          disegnaMappa(results[0]);
     else
          alert("Stringa immessa non valida");
});
```

Il metodo .geocode() inietta alla funzione di callback due parametri:
status = google.maps.GeocoderStatus.OK / NOK
results che è un **vettore enumerativo di oggetti GeocoderResult** di lunghezza 1, cioè contenente un unico GeocoderResult con tutte le informazioni sulla località ricercata.

La funzione disegna Mappa() potrà istanziare un oggetto **Map** e disegnare una Google Map centrata nel punto **results[0].geometry.location**

La visualizzazione di una route

Il calcolo di una route è basato sull'utilizzo di 2 oggetti fondamentali;

google.maps.DirectionsService

Il metodo **.route()** valuta tutti i possibili percorsi percorribili tra un punto di partenza ed un punto di arrivo. Restituisce alla funzione di callback un oggetto **routes** contenente un vettore enumerativo di routes, contenente cioè l'insieme di TUTTE le routes individuate, di cui la prima è quella preferibile. Ogni routes a sua volta può essere suddivisa in più tratte sequenziali detti **legs**.

google.maps.DirectionsRenderer
Il metodo **.setDirections()** visualizza sulla mappa associata la prima delle routes restituite dal metodo route dell'oggetto precedente

Nell'esempio seguente si suppone che partenza e arrivo siano due json **Point**.
In realtà sono ammesse **anche** stringhe alfanumeriche convertite automaticamente in JSON Point.

```
function visualizzaPercorso(partenza, arrivo) {
    var directionsRenderer = new google.maps.DirectionsRenderer();
    var directionsService  = new google.maps.DirectionsService();

    var percorso = {
       origin: partenza,
       destination: arrivo,
       travelMode: google.maps.TravelMode.DRIVING
    };

    directionsService.route(percorso, function(routes, status) {
        if (status == google.maps.DirectionsStatus.OK) {
            var mappa =  new google.maps.Map(div, mapOptions);
            directionsRenderer.setMap(mappa);
            directionsRenderer.setDirections(routes);
           // Pannello contenente tutte le indicazioni sul percorso
            directionsRenderer.setPanel(document.getElementById("panel"));
        }
    });
}
```

Nota
Il metodo **`.route()`** impiega un certo tempo ad eseguire l'elaborazione, per cui se l'oggetto **Map** viene costruito prima di lanciare il metodo route(), la mappa viene visualizzata per un istante per poi essere sovrascritta dalla nuova mappa con il percorso. Per cui è meglio istanziare l'oggetto **Map**() all'interno del metodo route().

Distanza fra due punti e Tempo di percorrenza

Per calcolare la distanza fra due punti esistono due possibili alternative.

1) Distanza stradale sulla base del percorso calcolato dal metodo **`directionsService.route()`**, che restituisce tutte le possibile **`routes`** tra due punti. Si prende in considerazione la prima route (routes[0]) e, al suo interno, la prima tratta (legs[0]), supponendo che la route sia costituita da una unica tratta. Per calcolare la distanza fra due punti lungo una route costituta da più tratte è sufficiente sommare tra loro le distanze delle singole tratte

```
directionsService.route(percorso, function(routes, status) {
    if (status == google.maps.DirectionsStatus.OK) {
        var mappa =  new google.maps.Map(div, mapOptions);
        directionsRender.setMap(mappa);
        directionsRender.setDirections(routes);
        directionsRender.setPanel(document.getElementById("panel"));

        // distanza fra i due punti
        alert(routes.routes[0].legs[0].distance.text);
        // tempo di percorrenza
        alert(routes.routes[0].legs[0].duration.text)
    }
});
```

2) Distanza in linea d'aria (great-circle distance). calcolabile tramite il seguente metodo statico:

```
var distance = google.maps.geometry.spherical.computeDistanceBetween
(partenza, arrivo);
```

Occorre però aggiungere la seguente libreria nel link alle google.maps:
`https://maps.googleapis.com/maps/api/js?libraries=geometry&key=.....`

I parametri devono necessariamente essere dei Point GPS.
Non sono accettate stringhe alfanumeriche

Lettura della posizione corrente

HTML5 mette a disposizione un nuovo oggetto **navigator.geolocation** che, tramite i metodi **.getCurrentPosition()** e **.watchPosition()** consente di ottenere le coordinate GPS della posizione corrente. Nel rispetto della normativa sulla privacy, nel momento in cui una applicazione fa uso dell'oggetto **navigator.geolocation,** l'applicazione deve richiedere il consenso all'utente.

- Nel caso dei browser la richiesta di consenso viene visualizzata in automatico in corrispondenza dell'utilizzo dell'oggetto **navigator.geolocation.** Nel caso in cui l'utente neghi il consenso viene automaticamente richiamata la procedura di errore con codice di errore 1.
- Nel caso delle applicazioni android, è l'applicazione che chiede il consenso all'atto dell'installazione. Se l'utente nega il consenso quella parte di applicazione non potrà essere eseguita.

sensor=true/false

Agli inizi, in corrispondenza del link di accesso alle google API, oltre alla google Key l'utente doveva dichiarare anche un secondo parametro **sensor** in cui l'utente dichiarava se la propria applicazione avrebbe fatto uso del sensore di posizione o meno. Oggi questo parametro non è più obbligatorio e non incide sul funzionamento, anche se spesso negli esempi in rete è ancora presente.

I metodi dell'oggetto navigator.geolocation

.getCurrentPosition(onSuccess, onError, options) restituisce la posizione corrente. In caso di successo viene richiamata la funzione indicata come primo parametro, in caso di errore viene richiamata la funzione indicata come secondo parametro.

.watchPosition(onSuccess, onError, options) Lancia la lettura su un thread separato e legge con continuità fino a quando il thread non viene terminato. La frequenza delle letture dipende dal sensore e non è configurabile dall'utente. Se si desidera una lettura a cadenze regolari l'alternativa migliore è quella di utilizzare getCurrentPosition() all'interno di un setTimeout(). watchPosition restituisce un **watchID** identificativo del thread di lettura .

.clearWatch(watchId) consente di arrestare il thread associato al `watchId` indicato

Il terzo parametro **options** contiene i seguenti tre campi:

```
var options = {
     enableHighAccuracy: true/false
     timeout: 5000,
     maximumAge: 0,
     frequency : 10000          // deprecata (ignorata)
}
```

enableHighAccuracy I sensori possono restituire le posizioni GPS in modo più approssimativo (7 cifre, Based GPS feature) oppure in modo più accurato (Built-in GPS feature) però con un maggior consumo di risorse (energia). Questa proprietà consente di impostare quale modalità utilizzare. Il default è false e va benissimo nella maggior parte dei casi, a meno che non servano altissimi livelli precisione. C'è però ancora un piccolo problema. Alcuni SO dispongono soltanto di una delle due modalità per cui se la richiesta usa una opzione diversa vanno in errore. La documentazione consiglia di utilizzare false e, in caso di errore, riprovare con true.

timeout (ms) indica il tempo max di attesa della risposta. Se la risposta non arriva entro questo tempo viene richiamata la procedura onError. Il valore di default è infinito, nel qual caso il metodo getCurrentPosition non ritorna nulla.

maximumAge (ms) indica al sensore un tempo max di cache dei valori restituiti nell'ultima lettura effettiva. Il default è 0 e significa che si chiede al sensore di restituire sempre valori effettivi.

frequency nel caso di watchPosition doveva indicare il tempo tra una lettura e l'altra in assenza di variazione delle posizione. E' però stata dichiarata obsoleta nelle API v3 e non più supportata.

function onSuccess(position){ }

riceve come parametro un oggetto **position** contenente i seguenti parametri:

coords.latitude	double -> decimal degrees
coords.longitude	double -> decimal degrees
coords.altitude	double or null -> meters (solo per sensori. Nel caso di indirizzo IP è null)
coords.accuracy	double -> meters (es: ± 100.000 metri nel caso dell'indirizzo IP)
coords.altitudeAccuracy	meters -> (es: ± 100 metri)
coords.speed	double or null -> meters/second velocità con cui il dispositivo si sta muovendo
timestamp	long -> timestamp unix espresso in msec a partire dal 1/1/1970

function onError(err){ }

riceve come parametro un oggetto **err** contenente i seguenti parametri:

err.code codice dell'errore
err.message messaggio dell'errore

Possibili codici di errore :

- PERMISSION_DENIED (1) if the browser denies you access to their location.
- POSITION_UNAVAILABLE (2) if the network is down or the positioning satellites can't be contacted.
- TIMEOUT (3) if the network is up but it takes too long to calculate the user's position.

Esempio di utilizzo da smartphone

```
function onSuccess(position) {
    var currentPoint = new google.maps.LatLng(position.coords.latitude,
                                              position.coords.longitude);
    if(mapID == null){  // il primo giro
        var wrapper = document.getElementById("wrapper");
        mapID = new google.maps.Map(wrapper, mapOptions);
        var mapOptions = {
            center:currentPoint,
            zoom: 15,
            mapTypeId: google.maps.MapTypeId.HYBRID
        };
        // opzioni.center.lat = position.coords.latitude;
        // opzioni.center.lng = position.coords.longitude;

        markerID = new google.maps.Marker({
           position: currentPoint,
           title: "Questa è la tua posizione!",   // tooltip
           map: mapID
      });
    }
    // Sposto il marcatore alla nuova posizione
    markerID.setPosition(currentPoint);
}

function onError(err) {
    alert("ERRORE:" + err.code + " - " + err.message); }
```