



Cómo crear mapas con el API de Google Maps e integrarlos en tu página web

Yaiza Temprado Rodríguez

La comodidad y flexibilidad de uso de los mapas de Google hacen que cada vez más gente quiera poder integrarlos en sus páginas web, poder personalizarlos e interactuar con ellos.



linux@software.com.pl

Gracias al API que Google ha puesto a disposición de sus usuarios y la gran cantidad de ejemplos que pueden encontrarse en la red, ya podemos crear los mapas que mejor se ajusten a nuestras necesidades y aprovecharnos de todo el potencial que Google nos ofrece. A través del siguiente artículo daremos los primeros pasos para la construcción de mapas que cubran las necesidades de la mayoría de los lectores, aunque no hay que olvidar que Google Maps da mucho, mucho más de sí.

Introducción

A principios de 2005 Google lanzó el servicio de Google Maps (<http://maps.google.com>), que consiste en una página web en la que se nos muestra un mapa, con mayor o menor detalle, de cualquier parte del mundo. Pero los mapas de Google son mucho más: sin duda lo que más sorprende de ellos es el nivel de interactividad que permiten; desde moverse por el mapa, hacer zoom, mostrar e introducir lugares y fotos de interés, hasta servicios más recientes, como mostrar el aspecto real de las calles a través de capturas de satélites, mostrar el tráfico (por ahora, sólo disponible en EEUU), o el más reciente en España Google Street, que nos permite movernos

literalmente por las calles de Madrid y Barcelona (aunque previsiblemente en los próximos meses veremos aumentar el número de ciudades disponibles).

Pero sin duda, lo que realmente supone una enorme diferencia respecto a los callejeros tradicionales es tener a nuestro alcance todo el potencial de Google Maps para ponerlo a nuestro servicio y crear nuestros propios mapas, tan sencillos o complejos como queramos, a través de la API pública y gratuita de este servicio. Esto nos permite crear mapas personalizados de todo tipo, desde mostrando la mejor ruta a nuestro negocio, marcando los puntos de interés de nuestro pueblo o superponiendo otro mapa para mostrar cómo quedaría un lugar con una nueva carretera. Las posibilidades son infinitas, y la flexibilidad inmensa. A lo largo de este artículo daremos los primeros pasos, explicaremos los principios básicos necesarios para crear nuestros propios mapas y algunas cosas interesantes que puedan ser de utilidad para sacar el máximo partido a este sistema.

Lenguaje de programación y software recomendado

Para crear mapas con Google Maps el único lenguaje de programación que nos va a hacer falta es Javascript. A pesar de



que en la página de Google Maps nos advierten de que es importante tener un conocimiento elevado de este lenguaje, la realidad es que hay tantos ejemplos disponibles en la red que unos pocos cambios normalmente serán suficientes para ajustarlos a nuestras necesidades. En cualquier caso, durante este artículo se irán dando algunas pautas necesarias sobre este lenguaje para aquellos lectores que hayan trabajado menos con Javascript, de forma que puedan comprender los ejemplos y adaptarlos a lo que están buscando.

Dado que hablamos de Javascript como lenguaje básico, el sistema operativo sobre el que vayan a correr los mapas será indiferente. Si que es altamente recomendable utilizar para ejecutar y depurar el código el navegador Firefox, ya que éste trae, en el menú *Herramientas*, una *Consola de Errores* donde podremos ir viendo los errores que cometamos con el Javascript. Además, si se desea depurar el código con un método más sofisticado, existe la posibilidad de instalar el plugin para Firefox Venkman (<http://www.mozilla.org/projects/venkman/>), que nos ofrece un debugger completo para ir depurando paso a paso nuestro código Javascript.

Respecto al editor de código, definitivamente no existe el editor perfecto. Sin embargo, una posibilidad es instalar el plugin para Firefox Extension Developer (<https://addons.mozilla.org/es-ES/firefox/addon/7434>), que nos permite editar nuestro código Javascript o el de cualquier página web que visitemos, guardar los ficheros modificados, etc.

Si nuestra intención es realizar mapas más complejos, por ejemplo en los que las coordenadas se lean de una base de datos o que manden información a un servidor, será necesario mezclar el código Javascript en páginas con un lenguaje de servidor, como puede ser PHP o JSP. Dado que su uso ya es para casos más específicos y complejos, en este artículo crearemos mapas que puedan gestionarse completamente a través de código Javascript, dejando fuera esa posibilidad.

Cómo crear un mapa básico

Antes de empezar a jugar con el código Javascript, necesitamos conseguir una clave de acceso al API de Google. Esta clave es gratuita y muy fácil de conseguir, pero es un paso imprescindible para tener acceso a las funciones del API. Para ello es necesario contar con una cuenta de correo de Google e introducir la URL de nuestro sitio web en la siguiente dirección: <http://code.google.com/intl/es/apis/maps/signup.html>

Es importante leer con cuidado las condiciones de uso que se especifican, ya que a pesar de que la API sea gratuita, tiene una

serie de restricciones, como mantener visibles los logotipos atributivos (no podemos tapar la imagen de Google en el mapa con otra imagen, por ejemplo) o el número máximo de geocalizaciones diarias al día (esto es, convertir direcciones postales en coordenadas, lo cual está limitado a 15.000 diarias). En general, las condiciones son bastante buenas y serán más que suficientes para cualquier sitio que estemos construyendo. No obstante, merece la pena echarles un vistazo. Una vez completado el formulario, obtendremos nuestra clave: una larga ristra de números y letras. Es importante no perderla, ya que tendremos que meterla en todos los mapas que construyamos.

Para crear nuestro primer mapa, creamos un documento *.html* con la estructura básica de una página web (head, body...) y copiamos dentro, por un lado el enlace a la clave que nos ha dado Google (no olvides sustituir el valor que sigue al parámetro *key* por tu propia clave):

```
<script src="http://maps.google.com/
maps?
file=api&v=2&key=MICLAVE&hl=es"
type="text/javascript">
</script>
```

Y por otro lado el código de nuestro primer mapa propiamente dicho (Listado 1).

Listado 1. El código de nuestro primer mapa

```
<script type="text/javascript">
  <![CDATA[
    // Comprobamos que el navegador es compatible
    if (GBrowserIsCompatible()) {
      // Creamos la variable "mapa" y asignamos los controles básicos var
      map = new GMap2(document.getElementById("map"));
      map.addControl(new GLargeMapControl());
      map.addControl(new GmapTypeControl());
      // Centramos el mapa y establecemos el nivel de zoom inicial.
      map.setCenter(new GLatLng(40.41, -3.70), 6);
    }
    // Si no lo es, mostramos un aviso
    else {
      alert("Este navegador no es compatible con Google Maps");
    }
  </script>
```

Listado 2. Creando marcadores

```
var point = new GLatLng(40.43, -4.78);
var marker = crearMarcador(point, 'Informacion sobre este punto');
map.addOverlay(marker);

function crearMarcador(point, html) {
  var marker = new GMarker(point);
  GEvent.addListener(marker, "click", function() {
    marker.openInfoWindowHtml(html);
  });
  return marker;
}
```

Listado 3. Definiendo un fichero XML

```
<markers>
  <marker lat="40.65" lng="-3.71" html="Info marcador 1" label="M1" />
  <marker lat="42.17" lng="-4.21" html="Info marcador 2" label="M2" />
  <marker lat="43.89" lng="-1.29" html="Info marcador 3" label="M3" />
</markers>
```




Figura 1. Primer mapa en la vista híbrida

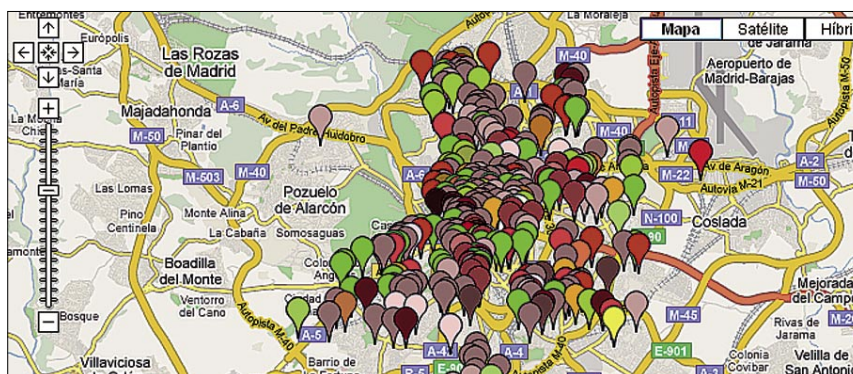


Figura 2. Mapa con marcadores personalizados

Al abrir el documento con un navegador, veremos que con este código hemos creado un mapa completo, centrado en el punto que nosotros queramos (tal y como se especifica en los comentarios del código), con el nivel de zoom que hayamos decidido y la escala en la parte izquierda con la que el usuario puede controlar el nivel de zoom. El mapa ya es plenamente funcional, y podemos moverlo y cambiarlo de tamaño exactamente igual que los mapas que aparecen en Google Maps, además de mostrar las tres vistas que ofrece:

- La normal con los nombres de las calles,
- La de satélite con el aspecto real de las calles vistas desde arriba,
- Una versión mixta entre ambas.

Entre estas tres vistas podemos cambiar con unos botones que se han creado en la parte superior del mapa. Sin embargo, aún hay muchas cosas que podemos añadir...

Cómo crear marcadores

Los marcadores son esos iconos en forma de gota invertida que señalan una posición. Esta posición puede estar marcada, o bien por un texto (por ejemplo, Callao, 12, Madrid) o por unas coordenadas (por ejemplo, latitud=41.32, longitud=-3.71). Vamos prime-

ro a ver el caso más sencillo en el que tenemos el punto exacto dado en sus coordenadas. Para ello, tendremos que introducir un poco más de código en nuestra página web.

Marcadores básicos

Por un lado, el siguiente código muestra cómo se crea una variable *point* (que es la clase que maneja coordenadas, y el tipo de dato que vamos a poder situar directamente en el mapa), y después creamos el marcador propiamente

dicho a través de la función `crearMarcador` (a continuación), el cual recibe dos parámetros: el punto que queremos pintar y lo que queremos que se muestre en el globo que aparece cuando se pincha sobre el marcador. Esta información es código html puro y duro, con lo que podremos introducir en él cualquier elemento de html como una imagen, una tabla, etc. El último paso es añadir el marcador al mapa mediante la función `addOverlay`, que añade elementos o capas sobre el mapa básico (en este caso, el marcador). Será esta función la que usaremos para añadir otros elementos al mapa, como rutas, imágenes superpuestas, etc (Listado 2).

Marcadores desde un fichero KML

Esta forma de definir los puntos está bien si no tenemos demasiados. ¿Pero qué pasa si tenemos 50 puntos? ¿Vamos a definir cada uno de ellos directamente dentro del código? Obviamente no. Para ello, lo mejor es definir un fichero XML con los campos necesarios para definir cada marcador. Un ejemplo sería este presentado en el Listado 3.

De esta forma tenemos aislada la información de los puntos de interés de nuestro código fuente, quedando este último mucho más limpio y escalable. Para leer ahora los marcadores que hemos definido en el XML, lo único que debemos hacer es lo que presenta el Listado 4.

Marcadores personalizados

Por defecto, todos los marcadores que se crean tienen exactamente el mismo aspecto: forma de gota invertida, de color rojo claro y con un punto negro en medio. Sin embargo, es muy probable que queramos cambiar este aspecto básico de los iconos por imágenes que caractericen mejor los puntos señalados. Para ello, primero definiremos un objeto derivado del tipo *GIcon*



Figura 3. Cálculo de una ruta mostrando las indicaciones para llegar de un punto a otro



(la imagen en sí) con las dimensiones que mejor se ajusten a nuestra imagen (Listado 5).

Una vez definido el nuevo tipo de icono, creamos las variables de ese tipo con la imagen que queremos que se muestre (tanto de base como de sombra):

```
var miIcono = new GIcon(baseIcon,
    "imagenes/icono.png", null,
    "imagenes/iconoSombra.png");
```

Por último, definiremos una nueva función para crear estos marcadores personalizados (o sobrescribiremos la que ya teníamos), cuya mayor diferencia es que, a la hora de crear el objeto GMarker, llamamos a un constructor con

dos parámetros: el punto y nuestro nuevo icono (o, en lo que a nosotros nos afecta, la función pasará de recibir como parámetros el punto y la información a mostrar, a recibir estos dos más el icono personalizado) (Listado 6).

Por lo demás, el código permanece igual. Sólo habrá que cambiar las llamadas a `crearMarcador` por `crearMarcadorPersonalizado` que aparecen en el código anterior.

Marcadores Drag & Drop

Pero los marcadores no tienen por qué permanecer fijos en un punto. Podemos definirlos de tal manera que el usuario pueda arrastrar el marcador hasta colocarlo en una nueva posición. Una vez situado en el punto que estábamos buscando,

podemos obtener sus nuevas coordenadas y actualizarlas, o bien en nuestro código, o bien en el fichero XML. Esta actualización podría hacerse de forma automática tirando un poco más del hilo del código Javascript; pero eso queda fuera del propósito de este artículo, por lo que lo dejaremos para otra ocasión.

En cualquier caso, las funciones que debemos introducir en nuestro código para poder arrastrar los marcadores de un punto a otro son las que muestra el Listado 7.

Como podemos ver, por un lado tenemos que la definición del marcador ha cambiado ligeramente, ya que en esta ocasión, además de crearlo dando la posición en la que debe estar, introducimos el modificador `draggable`, que nos permite arrastrarlo. Ésta es la diferencia más importante, ya que las funciones que reciben los eventos de inicio y fin del arrastrado lo único que hacen es ocultar y mostrar la ventana de información respectivamente.

Para que este cambio de posición sea más útil, vamos a modificar ligeramente la función que se invoca cuando soltamos el marcador en su nueva posición, de tal manera que la ventana de información, en vez de mostrar un mensaje estático muestre las coordenadas del nuevo punto (Listado 8).

Creación de rutas

La creación de rutas puede ser desde algo tan sencillo como unir dos puntos del mapa a través de una línea recta (que a priori puede no tener mucha utilidad) hasta la creación de rutas reales utilizando la información de Google Maps y, lo que es más importante en el caso de las rutas en coche, las restricciones de las calles (dirección prohibida, sentido único, etc). Primero veremos el caso básico de creación de líneas y luego nos meteremos con el cálculo de rutas *transitables*.

Creación de una línea recta entre dos o más puntos

La creación de líneas rectas entre dos o más puntos se llama *Polyline*, y su creación es muy sencilla, ya que se basa en puntos exactamente iguales que los utilizados para la creación de marcadores. Para pintar la ruta lo único que necesitamos son las coordenadas de los puntos que van a formar la ruta. Suponiendo que queremos pintar una ruta formada por tres puntos (aunque este número no está limitado), la función que pinta el Polyline será así como la presenta el Listado 9.

Creación de rutas reales para recorrer a pie o en coche

Pero sin duda, mucho más interesante que trazar líneas rectas entre varios puntos, es obtener

Listado 4. Leyendo los marcadores

```
var gmarkers = [];
// Leemos del fichero coordenadas.xml
GDownloadUrl("coordenadas.xml", function(doc) {
    var xmlDoc = GXml.parse(doc);
    var markers = xmlDoc.documentElement.getElementsByTagName("marker");
    for (var i = 0; i < markers.length; i++) {
        // Sacamos los campos del xml para cada marcador
        var lat = parseFloat(markers[i].getAttribute("lat"));
        var lng = parseFloat(markers[i].getAttribute("lng"));
        var point = new GLatLng(lat, lng);
        var html = markers[i].getAttribute("html");
        var label = markers[i].getAttribute("label");
        // Creamos el marcador
        var marker = crearMarcador(point, label, html);
        // Y lo añadimos al mapa
        map.addOverlay(marker);
    }
    // Añadimos la barra lateral
    document.getElementById("side_bar").innerHTML = side_bar_html;
});
```

Listado 5. Creando marcadores personalizados

```
var iconoNuevo = new GIcon();
// Las dimensiones de la imagen sombra del icono
iconoNuevo.shadowSize=new Gsize(56,32);
// Las dimensiones del icono
iconoNuevo.iconAnchor=new Gpoint(16,32);
//Las dimensiones de la ventana de información
iconoNuevo.infoWindowAnchor=new Gpoint(16,0);
```

Listado 6. Definiendo una nueva función para crear marcadores personalizados

```
function crearMarcadorPersonalizado(point,html,miIcono) {
    var marker = new GMarker(point,icon);
    GEvent.addListener(marker, "click", function() {
        marker.openInfoWindowHtml(html);
    });
    return marker;
}
```




una ruta funcional entre ellos. Y sin embargo es incluso más sencillo que en el caso anterior, aunque hay que analizar con cuidado lo que significa cada línea (Listado 10).

Por un lado vemos que en este caso hemos introducido directamente direcciones postales. Dado que estamos dando una ruta de un punto

a otro, parece tener más sentido este caso que no dar directamente latitudes y longitudes. No obstante, si en algún caso es necesario disponer de esa opción, el API de Google Maps también la contempla.

Por otro lado vemos que sacamos un elemento del código html (en este caso llamado

indicaciones) y lo utilizamos como parte de la llamada al constructor de GDirections. Ésto se debe a que es en este elemento de la página (típicamente un DIV) donde aparecerán las indicaciones a las que Google nos tiene acostumbrados, del estilo de *gire a la derecha en la tercera rotonda o siga recto hasta la salida 36*. No obstante, si sólo quisiéramos mostrar la ruta dibujada sobre el mapa al estilo de un Polyline y sin las indicaciones exactas, lo único que tendríamos que hacer es eliminar la referencia a la caja de texto, de la manera presentada en el Listado 11.

Afortunadamente, el método `load()` de GDirections admite ambas posibilidades.

Superposición de imágenes

La primera posibilidad que nos viene a la cabeza tras saber manejar distintos tipos de iconos, crear Polylines de colores y demás personalizaciones es poder introducir una leyenda en el mapa, de forma que la persona que lo está viendo entienda qué significa cada una de las cosas que hemos introducido en el mapa. También puede interesarnos superponer el logotipo de nuestra empresa o producto sobre el mapa de Google, o posicionar sobre una dirección dada una foto de nuestro negocio. Claramente tratamos aquí con dos tipos de imágenes: por un lado unas que queremos que permanezcan fijas en una posición y sin sufrir modificaciones dependientes del zoom (como puede ser la leyenda o el logo), y otras que tienen que moverse con el mapa ya que están ancladas a unas coordenadas determinadas. Veamos cómo podemos definir cada una de ellas.

Superposición de imágenes fijas

El código se basa en cosas que ya hemos visto en ejemplos anteriores. Poco a poco vamos viendo una tónica general en la forma de los constructos de Google Maps (Listado 12):

- Por un lado creamos la capa que vamos a colocar encima del mapa e indicamos dónde se encuentra la imagen a superponer.
- El siguiente parámetro es la posición dentro de la imagen (o fuera de ella) desde la que queremos contar para posicionar la imagen. Es decir, si los valores para este parámetro son (0, 0) estaremos diciendo que la posición de la pantalla que indiquemos con el siguiente parámetro es donde tiene que estar la esquina inferior izquierda.
- Definimos el punto en el que queremos situar la imagen, sabiendo que esta distancia se cuenta desde la esquina inferior izquierda del mapa y que por defecto se mide en píxeles (igual que en el punto anterior).
- Definimos las dimensiones de la imagen. Podemos definir las como fracciones del ma-

Listado 7. Introduciendo las funciones para poder arrastrar los marcadores de un punto a otro

```
var punto = new GLatLng(41.4419, -4.1419);
var marker = new GMarker(punto, {draggable: true});
GEvent.addListener(marker, "dragstart", function() {
    map.closeInfoWindow();
});
GEvent.addListener(marker, "dragend", function() {
    marker.openInfoWindowHtml("Nueva posición del marcador");
});
```

Listado 8. Modificando la función para que muestre las coordenadas del nuevo punto

```
GEvent.addListener(marker, "dragend", function(overlay, latlng) {
    var info = "Lat.: " + latlng.lat() + "<br/>Long.: " + latlng.lng();
    marker.openInfoWindowHtml(info);
});
```

Listado 9. Creando una línea recta

```
function crearRuta(puntos) {
    // Con esta linea creamos el polyline y
    // cambiamos el color y el ancho
    map.addOverlay(new GPolyline(puntos, '#000000', '5'));
}
var puntos = [];
puntos[0] = new GLatLng(40.87, -3.75);
puntos[1] = new GLatLng(41.21, -2.97);
puntos[2] = new GLatLng(39.82, -3.94);
// Llamamos a la función para que se pinte la ruta
crearRuta(puntos);
```

Listado 10. Creando una ruta real

```
var panelIndicaciones;
var direcciones;
function mostrarRuta(origen, destino) {
    panelIndicaciones = document.getElementById("indicaciones");
    direcciones = new GDirections(map, panelIndicaciones);
    direcciones.load("from: " + origen + " to:" + destino);
}
mostrarRuta('Galileo, 40, Madrid', 'Barceló, 3, Madrid');
```

Listado 11. Eliminando la referencia a la caja de texto

```
var direcciones;
function mostrarRuta(origen, destino) {
    direcciones = new GDirections(map);
    direcciones.load("from: " + origen + " to:" + destino);
}
mostrarRuta('Galileo, 40, Madrid', 'Barceló, 3, Madrid');
```



Think smart **ESET** **Smart Security**

**Un nuevo concepto en protección
inteligente para su PC**

Seguramente ya estarás confiando en una suite de seguridad. Hay muchas de ellas, pero sólo ESET ofrece una solución unificada completamente diferente.

Puede pensar.

Gracias a su tecnología ThreatSense® tiene la habilidad de anticiparse a peligros potenciales, sin ralentizar su sistema operativo y protegiendo proactivamente su ordenador.

Es inteligente.

Sea también proactivo y pruebe su versión de evaluación gratuita de 30 días en www.eset.es

COMPONENTES INTEGRADOS:

ESET NOD32 **Antivirus**
ESET NOD32 **Antispyware**
ESET **Personal Firewall**
ESET **Antispam**



c/Martínez Valls 56,bajos - 46870 Ontinyent (Valencia)
ventas@protegerse.com - Teléfono 902.33.48.33
<http://www.eset.es>



pa o como número de píxeles. En este caso hemos usado la medida de fracción (esta forma de calcular las posiciones también puede aplicarse al resto de parámetros).

- Por último, como siempre, añadimos el objeto leyenda al mapa como una capa más.

A pesar de lo fácil que es introducir imágenes en cualquier punto del mapa, no hay que olvidar que una de las condiciones de uso de la API gratuita de Google Maps es dejar a la vista el logo y demás información de autoría que se muestra por defecto. Lo contrario constituiría una violación de la licencia, por lo que debemos ser cuidadosos en este aspecto.

Superposición de imágenes ancladas a unas coordenadas

Para superponer una imagen que se encuentre posicionada de forma fija, y que además modifique su tamaño de acuerdo con el nivel de zoom que estemos usando, el código es ligeramente diferente (Listado 13).

De esta forma, tenemos la imagen centrada entre dos puntos, lo que servirá como referencia a la hora de dejar posicionada la imagen aunque el usuario cambie de posición, y para mantener el ratio de tamaño cuando se produce un zoom, ya que la imagen también cambiará de tamaño proporcionalmente para ajustarse al nuevo mapa.

El formato KML

Si alguno de los lectores ha intentado hacer algo con Google Maps se habrá dado cuenta de que en muchas ocasiones se menciona el formato KML. KML está basado en XML y sirve para mostrar datos geográficos en un navegador de la Tierra, como Google Maps o Google Earth. KML es un formato que inicialmente se definió para Google Earth, por lo que la parte que aplica en Google Maps es solamente un subconjunto de este estándar con el que podemos definir marcadores, Polyline, rutas, estilos, etc.

A continuación (Listado 14) se muestra un ejemplo de la estructura de un fichero KML en el que se ha definido un estilo personalizado para un Polyline, así como la definición del Polyline en sí utilizando este estilo. Toda la información de definición de aspecto gráfico se encuentra entre etiquetas `<Style></Style>`, dentro de las cuales se define otra etiqueta indicando para qué tipo de elemento estamos definiendo la apariencia; puede ser un icono (`<IconStyle>`) o, como en este caso, un Polyline (`<LineStyle>` para el grosor de la línea y `<PolyStyle>` para el color). En la segunda parte del fichero, donde se definen los puntos que conforman el Polyline, es importante fijarse en un par de detalles: primero, la etiqueta

`<styleUrl>`, que es la que contiene el estilo definido anteriormente que se va a aplicar (precedido del símbolo '#'); segundo, recordar que ahora nos estamos moviendo en 3 dimensiones,

por lo que las coordenadas están formadas por 3 valores: latitud, longitud y altura.

Es muy importante saber que la estructura del fichero es bastante fija, y sobre todo que

Listado 12. Superponiendo imágenes fijas

```
var leyenda = new GScreenOverlay('images/leyenda.jpg',
    new GScreenPoint(0, 0, 'fraction', 'fraction'),
    new GScreenPoint(10, 5),
    new GScreenSize(0.2, 0.2, 'fraction', 'fraction'));
map.addOverlay(leyenda);
```

Listado 13. Superponiendo imágenes ancladas

```
//Definimos los puntos que representan las esquinas suroeste y noreste
var so = new GLatLng(38.67,-3.21);
var ne = new GLatLng(38.68,-3.20);
//Creamos el objeto que limitará la imagen a esos puntos
var limites = new GLatLngBounds(so, ne);
//Creamos el objeto de la imagen anclada con la imagen y los límites
var imagen = new GGroundOverlay("imagen.jpg", limites);
//La añadimos como una capa más al mapa
map.addOverlay(imagen);
```

Listado 14. Un ejemplo de la estructura de un fichero KML


```
<?xml version="1.0" encoding="UTF-8"?>
<kml xmlns="http://earth.google.com/kml/2.2">
  <Document>
    <Style id="polyAzul">
      <LineStyle>
        <width>1.5</width>
      </LineStyle>
      <PolyStyle>
        <color>7dff0000</color>
      </PolyStyle>
    </Style>
    <Placemark>
      <name>Edificio 41</name>
      <styleUrl>#polyAzul</styleUrl>
      <Polygon>
        <extrude>1</extrude>
        <altitudeMode>relativeToGround</altitudeMode>
        <outerBoundaryIs>
          <LinearRing>
            <coordinates>
              -122.0858169768481,37.42231408832346,0
              -122.085852582875,37.42230337469744,0
              -122.0858799945639,37.42225686138789,0
              -122.0858860101409,37.422311076138,0
              -122.0858069157288,37.42220250173855,0
            </coordinates>
          </LinearRing>
        </outerBoundaryIs>
      </Polygon>
    </Placemark>
  </Document>
</kml>
```




Sobre la autora

Yaiza Temprado Rodríguez (ytr@moviquity.com) es Ingeniero Superior en Informática y actualmente realiza su doctorado en la Universidad Carlos III de Madrid. Además de trabajar para Telefónica I+D como experta en Data Mining, imparte clases de Aprendizaje Automático en la Universidad Europea de Madrid. Es además administradora de la página www.chicaslinux.org y su bitácora personal puede encontrarse en www.losmundosdeyaiza.com

dad con que están pensadas las clases y su flexibilidad, hacen que poco tiempo después de haber empezado a crear nuestros propios mapas podamos deducir fácilmente cómo se va a programar algo sin apenas mirar la documentación.

Con este artículo se ha pretendido dar una visión general sobre las necesidades más frecuentes que puede encontrarse un programador a la hora de crear uno de estos mapas, aunque existen muchísimas más posibilidades, desde superponer otros mapas sobre los ya creados, colorear regiones del mapa o, incluso, que al calcular una ruta entre dos puntos aparezca un icono de un coche recorriendo la ruta para que podamos ir la siguiendo. Las posibilidades son enormes, máxime cuando Google no se ha quedado de brazos cruzados y cada pocos meses nos enteramos de nuevos servicios que van apareciendo, como el reciente Google Mars (con mapas de la superficie de Marte) o Google Street (que permite que visitemos las calles desde el punto de vista de un peatón). Las posibilidades a día de hoy ya son enormes, pero seguro que el tiempo nos ofrece aún más posibilidades de diversión con los nuevos servicios con los que, seguro, Google seguirá sorprediéndonos. 

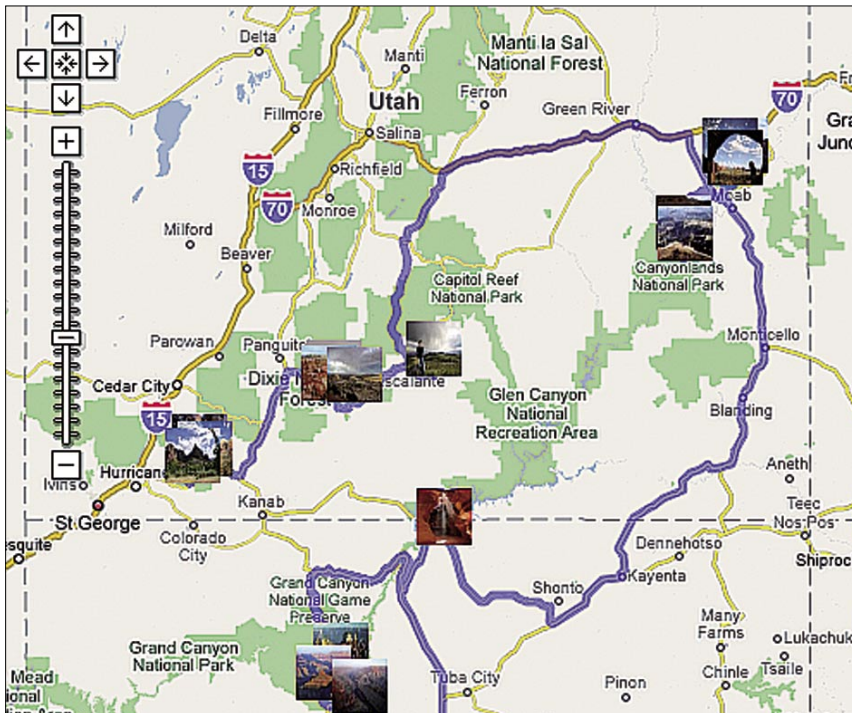


Figura 4. Cálculo de una ruta anclando imágenes relevantes en los puntos de interés

distingue entre minúsculas y mayúsculas en las etiquetas, lo que puede ser motivo de un porcentaje muy alto de problemas con este formato. En el siguiente ejemplo se ve la estructura y los campos necesarios para describir un marcador en formato KML. De nuevo se repite, igual que en el caso anterior, la característica de que cada marcador tiene 3 coordenadas, incluyendo la altura (Listado 15).

Una vez que hemos definido el fichero, tendremos que guardarlo con extensión .kml o .kmz, que permitirá que tanto Google Maps como Google Earth sean capaces de interpretarlo. Una vez finalizado el fichero ya podemos mandarlo a Google Maps como un parámetro más de la URL, de esta forma: <http://maps.google.com/maps?q=http://www.ejemploscongooglemaps.com/ejemplo.kml>

¡Ojo! Para poder pintar nuestro fichero KML sobre Google Maps el fichero tiene que estar disponible online. Es decir, por ejemplo la ruta <http://localhost/ejemplo1.kml> no es válida. Otra cosa a tener en cuenta y que puede hacer que resulte confuso el uso de KML es que, una vez integrado un fichero .kml en un mapa, si cambiamos el fichero el mapa permanece cacheado durante un tiempo, haciendo invisibles los cambios que hayamos realizado justo antes.

El lenguaje KML resulta bastante sencillo de utilizar, aunque a veces su gran potencia hace que resulte complejo para aquellos que están empezando a utilizarlo. Por ejemplo, el último ejemplo mostrado en el que se ha definido un punto de una forma muy sencilla, no es más que una forma de las diversas que permite para

definirlo. Aunque ya queda fuera de este tutorial, su potencia permite que pudiéramos repetir casi cada uno de los elementos que se han ido explicando en apartados anteriores (rutas, inserción de imágenes, marcadores personalizados...). Para más información sobre la sintaxis y los distintos usos del formato KML se puede consultar el siguiente enlace: http://code.google.com/intl/es-ES/apis/kml/documentation/kml_tut.html#basic_kml

Conclusiones

Como hemos podido ver, la potencia del API de Google Maps que tenemos a nuestra disposición nos permite hacer desde mapas muy sencillos hasta cualquier cosa que se nos ocurra, todo ello con el mínimo de código y sin tenernos que complicar la vida más allá del lenguaje Javascript. Además, la uniformi-

Listado 15. La estructura y los campos necesarios para describir un marcador en formato KML

```
<?xml version="1.0" encoding="UTF-8"?>
<kml xmlns="http://earth.google.com/kml/2.2">
  <Document>
    <Placemark>
      <name>Ejemplo de marcador</name>
      <description>Esto seria un marcador</description>
      <Point>
        <coordinates>102.595626,14.996729</coordinates>
      </Point>
    </Placemark>
  </Document>
</kml>
```