

Manual API Javascript CERCALIA

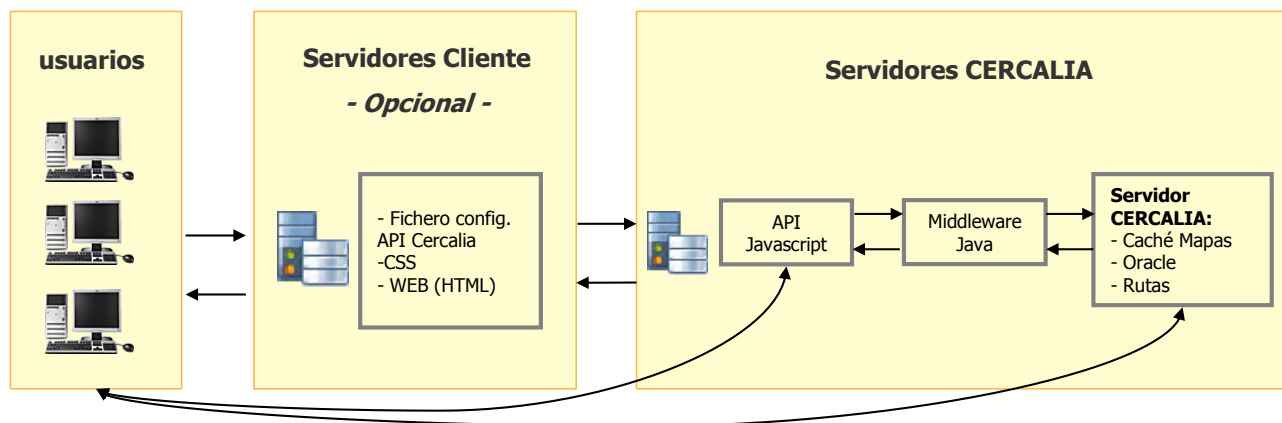
Empresa: Nexus Geografics SL
Fecha última modificación: 17/11/2010
Documento: manual_API_CERCALIAv3.1.doc

1 Contenido

1	Contenido	2
2	Arquitectura	4
2.1	Instalación.....	5
2.1.1	Con validación por dominio	5
2.1.2	Con validación por proxy	5
3	Integración y personalización de la interfaz	6
3.1	Ejemplo básico e integración.....	6
3.2	Personalización del estilo y las imágenes	7
3.3	El fichero de configuración	8
4	La clase <code>cercaliaClient</code>	10
4.1	Estructuras u objetos usados por los métodos de esta clase	10
4.1.1	Atributos de un error (<code>Error</code>)	10
4.1.2	Atributos de un candidato (<code>Candidate</code>)	11
4.1.3	Atributos de una geentidad (<code>Geoentity</code>)	11
4.1.4	Atributos de un punto de interés (<code>Poi</code>)	12
4.1.5	Atributos de proximidad (<code>Prox</code>)	12
4.1.6	Atributos de <code>arc</code> (<code>Arc</code>).....	12
4.1.7	Atributos de un objeto móvil (<code>Mo</code>).....	12
4.1.8	Atributos de un objeto de estilo marcador (<code>markerStyle</code>)	13
4.1.9	Atributos de un objeto de forma (<code>feature</code>).....	14
4.1.10	Atributos de un objeto de etapa de ruta (<code>Stage</code>)	14
4.1.11	Atributos de un objeto de subetapa de ruta(<code>Subtage</code>).....	15
4.2	Métodos para el tratamiento de marcadores.....	16
4.2.1	<code>createMarker(id,x,y,srs,style,id2)</code>	16
4.2.2	<code>createMarkers (obj,style)</code>	17
4.2.3	<code>moveMarker(id,x,y,srs,cont,contOver)</code>	18
4.2.4	<code>setMouseoverContent (id,contOver)</code>	19
4.2.5	<code>setMarkerIcon(id,url)</code>	19
4.2.6	<code>deleteMarkers([id1 [, id2 [...]])</code>	19
4.2.7	<code>centerToMarkers(changeScale [,id1 [, id2 [...]])</code>	20
4.2.8	<code>showMarkers(state [,id1 [, id2 [...]])</code>	20
4.2.9	<code>displayMarker(idMarker,visible)</code>	21
4.2.10	<code>showLabelMarkers (visible)</code>	21
4.2.11	<code>showLabel (visible,idMarker)</code>	21
4.3	Métodos para el tratamiento de formas.....	21
4.3.1	<code>createFeature(wkt,srs,style [, editstyle])</code>	21
4.3.2	<code>createFeature(gml,style [, editstyle])</code>	23
4.3.3	<code>showFeatures(visible [,feature1 [, feature2 [...]])</code>	23
4.3.4	<code>centerToFeatures(changeScale [,feature1 [, feature2 [...]])</code>	24
4.3.5	<code>deleteFeatures([feature1 [, feature2 [...]])</code>	24
4.3.6	<code>getFeaturesByState(created, modified, deleted, unchanged[,SRS])</code>	25
4.3.7	<code>getFeaturesByStateAsGML(created, modified, deleted, unchanged[,SRS])</code>	25
4.3.8	<code>getFeatures([feature1 [, feature2 [...]][,SRS])</code>	26
4.3.9	<code>getFeaturesAsGML([feature1 [, feature2 [...]][,SRS])</code>	26
4.3.10	<code>showLabelFeature (visible,id)</code>	26
4.4	Métodos tratamiento del mapa	26
4.4.1	<code>activateTool(tool)</code>	26
4.4.2	<code>activateMeasure (type, deactivateOnEnd)</code>	26
4.4.3	<code>deleteLastMeasurement ()</code>	27
4.4.4	<code>deleteAllMeasurements ()</code>	27
4.4.5	Evento clic en el mapa	27
4.4.6	<code>isCompleted()</code>	28
4.4.7	<code>showPois(visible,category[[,category],...])</code>	28
4.4.8	<code>setLanguage(lang)</code>	28
4.4.9	<code>getMap()</code>	28
4.4.10	<code>deleteRoute (resultCalculateRoute)</code>	29
4.4.11	<code>setParse< operation> (function)</code>	29
4.4.12	<code>showResult< operation> (obj)</code>	31

4.4.13	getExtent(srs)	31
4.4.14	getCenter(srs).....	31
4.4.15	setCenter(center,scale)	32
4.4.16	setCenter(center,srs,scale)	32
4.4.17	zoomToExtent(extent,srs)	33
4.4.18	getScale(closest)	33
4.4.19	getCurrentSRS().....	33
4.5	Métodos para obtener información.....	33
4.5.1	geocoding(search,returnFunc)	33
4.5.2	proximity(search,returnFunc).....	35
4.5.3	getDistance (p1,p2,weight,returnFunction)	37
4.5.4	geofencing (wktPol,point)	38
4.5.5	transformWKT (wkt,srs,srsdest)	38
4.5.6	getPoi (infxml, poicode [, poicode [...]], execute)	39
4.5.7	calculateRoute(stopList, routeParams, lineStyle, drawStopMarkers, execute)	39
4.5.8	createReport(xml,idElement,xsl_name)	42
4.5.9	createMapImage(routeId,lineStyle,routeParams, poicats ,drawFeatures,execute)	43
4.5.10	calculateMultiRoute(stopList, routeParams, execute)	44
4.5.11	transformCoordinates (objlist,srs).....	46
4.5.12	coordToPixel(obj).....	46
4.5.13	coordToPixel(obj,srs).....	46
5	Estilo del mapa	48
5.1	getStyle()	48
5.2	setStyle(name).....	48
5.3	createWMSStyle (name,url,params,options).....	48
6	Anexo: Geocodificación sin mapa	50
7	Anexo: Sistemas de coordenadas	52
8	Anexo: Botones customclick 1/2	53
8.1.1	getCoordFromPixel(pixel,srs).....	53
9	Anexo: Herramientas de edición	54
9.1	Personalización del menú	56
9.1.1	Función activateDrawControl(tool)	56
9.1.2	Función applyStyleControlDraw(type, style)	57
9.1.3	Función setRegularCircleDrawControl(isRegular)	57
9.1.4	Detectar creación formas	57
9.1.5	Ejemplo integrado en el código HTML	58
9.2	Herramienta SelectFeatureCustom	58
9.2.1	Función activateSelectFeatureCustom(callback)	58
9.2.2	Función deactivateSelectFeatureCustom(callback)	59
10	Anexo: Diferencias entre las versiones 1 y 2	60

2 Arquitectura



Los servidores de cercalia proporcionan una API Javascript que permite incorporar mapas y otras funcionalidades a sus páginas web de forma rápida y sencilla.

Para controlar el acceso a la API y a los servicios proporcionados por los servidores Cercalia se han implantado dos mecanismos de seguridad según las necesidades de nuestro distribuidor o cliente.

- **Validación por dominio**

Cada distribuidor tiene uno o más puntos de acceso. Cuando se accede a uno de ellos se comprueba que el acceso se haya hecho a través de una página servida por un dominio o servidor de nuestro distribuidor. Concretamente se comprueba que el referer de la petición pertenezca a un dominio o servidor de nuestro distribuidor. El distribuidor debe proporcionar una lista de dominios y puertos válidos.

Esta solución establece menos requerimientos en el servidor web del cliente. Usa tecnología **scriptag(dinamic scripting)** para evitar la validación de saltos de dominio de los navegadores. Esta tecnología tiene algunas limitaciones:

- El IE no es capaz de enviar mas de 256 bytes de información por lo que las peticiones mas largas no se envían correctamente.
- La sincronización entre llamadas es mas rudimentaria y en el IE6 en algunas ocasiones no funciona.

- **Validación por proxy**

En este caso el distribuidor debe instalarse un servlet proxy lo que requiere que el distribuidor posea un servidor de servlets. El servlet proxy recibe las peticiones de los clientes, añade el identificador de distribuidor y las reenvía balanceadas a los servidores Cercalia. Es importante proteger el proxy para que no se pueda acceder desde ubicaciones no válidas.

Esta solución es mucho mas robusta que la anterior. Los clientes acceden con peticiones AJAX al proxy y no tienen limitaciones en el tamaño de información. El proxy balancea las peticiones según la velocidad de respuesta de los servidores y tolera caídas de estos si las hubiera.

2.1 Instalación

2.1.1 Con validación por dominio

- Descomprimir el fichero suministrado **cercalian.zip** a una carpeta.
- Eliminar la subcarpeta WEB-INF.
- Editar el fichero **cercalia.js** y descomentar la sección de conexión por dominio. Modificar el valor de la variable contexto por el suministrado por Nexus.
- Publicar la carpeta en un servidor web.
- Informar a Nexus de la lista de dominios a los que sirve el servidor web.
- Probar la demo suministrada.

2.1.2 Con validación por proxy

- Descomprimir el fichero suministrado **cercalian.zip** a una carpeta.
- Añadir su clave de distribuidor (10 dígitos) en el fichero de configuración **WEB-INF/web.xml**.
- Editar el fichero **cercalia.js** y descomentar la sección de conexión con proxy.
- Publicar la carpeta en un servidor de aplicaciones.
- Probar la demo suministrada.

3 Integración y personalización de la interfaz

3.1 Ejemplo básico e integración

El siguiente código es un ejemplo básico de utilización de la API :

```
<html>
<head>
  <meta http-equiv=Content-Type content="text/html; charset=ISO-8859-1">
  <script type='text/javascript' src='cercalia.js'></script>
  <script>

    function execute() {
      f51 = cercalia.createFeature("POINT (-448003 4817761)",
        "EPSG:54004",{strokeWidth: 3, strokeColor: "#00ee00",
          fillColor: "#00ee00", fillOpacity: 0.2});
    }

    var properties;
    var cercalia;

    function createAll() {
      properties=new cercaliaProperties("config.xml");
      cercalia = new cercaliaClient(properties,execute,"map");
    }

  </script>
</head>
<body style="overflow: hidden;" onload="createAll();" >
  <div id="map" class="mapa"></div>
</body></html>
```

Antes de utilizar la API tenemos que cargarla con la siguiente instrucción:

```
<script type='text/javascript' src='cercalia.js'></script>
```

El constructor de **cercaliaClient** necesita el div donde colocar el mapa. El identificador del div no debe ser un nombre compuesto (por ejemplo: map_1 no es correcto, en cambio map1 si lo es). Vamos a crear un div dentro del tag body de la página:

```
<div id="map" class="map">
</div>
```

Cuando se llama el constructor **cercaliaClient** el div ya debe estar definido en la estructura de la página. Para asegurar una inicialización correcta debemos hacerla en el evento onload que se llama después de cargar y componer la página.

```
<body style="overflow: hidden;" onload="createCercalia();" >
```

La inicialización consiste básicamente en leer un fichero de propiedades y llamar al constructor de **cercaliaclient** pasándole por parámetro el nombre del div donde se debe colocar el mapa, un objeto con las propiedades y una función que se ejecutará cuando el mapa ya esté inicializado.

```
var cercalia;
```

```
function createCercalia() {
    var properties=new cercaliaProperties("config.xml");
    cercalia = new cercaliaclient(properties,execute,"map");
}
```

Una vez inicializado el mapa se llama a la función execute donde ya podemos llamar las funciones de la API que nos apetezcan:

```
function execute() {
    f51 = cercalia.createFeature("POINT (-448003 4817761)",
        "EPSG:54004",{strokeWidth: 3, strokeColor: "#00ee00", fillColor:
        "#00ee00",fillOpacity: 0.2});
}
```

3.2 Personalización del estilo y las imágenes

Después de cargar API podemos sobrecargar los estilos mediante la siguiente instrucción

```
<link rel="stylesheet" type="text/css" href="../../theme/otro/estil.css" />
```

En este archivo de estilo le podemos cambiar los nombres y/o las rutas de las imágenes de la interfaz:

- Imágenes de fondo de las pestañas en estado on y off.
- Imágenes de fondo de los títulos de los contenidos de los marcadores genéricos.
- Imágenes de cerrar, maximizar y minimizar los contenidos de los marcadores por defecto, los de la API y los simples de la API que solo tienen el botón cerrar.
- Imágenes de las opciones del menú.

Ejemplo: Cambiar la imagen de fondo del título de los marcadores.

Modificamos el fichero html para que cargue la hoja de estilos local:

```
<html>
<head>
  <meta http-equiv=Content-Type content="text/html; charset=ISO-8859-1">
  <script type='text/javascript' src='cercalia.js'></script>
  <link rel="stylesheet" type="text/css" href="../../theme/otro/estil.css"/>
</script>
```

En la hoja de estilos local buscamos las siguientes líneas:

```
.titolPopup
{
  color: #444444;
  background-image: url('img/bg_div_info.gif');
}
```

y las sustituimos por:

```
.titolPopup
{
  color: #444444;
  background-image: url('img/mifondo.gif ');
}
```

```
}
```

Copiamos la nueva imagen a la carpeta de imágenes.

3.3 El fichero de configuración

Las propiedades de configuración del mapa se pueden cargar de un fichero en formato XML (por defecto fichero **config.xml**) con las etiquetas de los valores que se quieren personalizar. Ejemplo de fichero:

```
<?xml version="1.0" encoding="ISO-8859-1"?>
<config>
  <language>es</language>
  <panel_status>maximized</panel_status>
  <panel_pages>
    <search selected="true">true</search>
    <proximity>true</proximity>
    <route>true</route>
    <pois>true</pois>
  </panel_pages>

  <overview_map_status>minimized</overview_map_status>
  <overview_map_style>default</overview_map_style>

  <map_style>default</map_style>
  <show_mouse_position>false</show_mouse_position>

  <zoombar_status>hidden</zoombar_status>
  <pancontrol_status>visible</pancontrol_status>
  <map_center>410977,4896277</map_center>

  <xsl_name>rutas_ruta.xsl</xsl_name>

  <toolbar_status>maximized</toolbar_status>
  <toolbar_tools>
    <pan selected="true">true</pan>
    <zoomin>true</zoomin>
    <zoomout>true</zoomout>
    <draw>true</draw>
    <clean>true</clean>
    <measure>>false</measure>
    <infopoi>true</infopoi>
    <infoter>true</infoter>
    <pois>true</pois>
    <export>>false</export>
  </toolbar_tools>
</config>
```

En la siguiente tabla se listan las etiquetas que se pueden encontrar en el fichero de configuración y su significado:

client	Código de cliente que se concatena a la clave principal del distribuidor. No ponga su clave de distribuidor en este parámetro.
map_zoom_level	Nivel de escala inicial del mapa. 0 más próximo.
zoom_levels	Lista de niveles de zoom que deseamos que aparezcan disponibles o no en la barra de zoom del mapa. Para especificar si son niveles que deben aparecer o no usar el atributo take con valor true o false respectivamente. Ejemplo de utilización:

	<pre><zoom_levels take="false"> <zoom_level>1</zoom_level> <zoom_level>3</zoom_level> <zoom_level>5</zoom_level> <zoom_level>7</zoom_level> <zoom_level>9</zoom_level> <zoom_level>11</zoom_level> <zoom_level>13</zoom_level> </zoom_levels></pre> <p>En este caso no queremos que aparezcan disponibles los niveles de zoom indicados.</p>
map_center	Coordenadas del centro del mapa en Mercator EPSG:54004.
map_style	Estilo del mapa. Por defecto use " default ".
map_extent	Extensión máxima que se visualiza en el mapa. Formato: left, bottom, right, top
language	Código de idioma (nombre del fichero xml que contiene los mensajes) .
panel_status	Estado del panel lateral. Valores posibles: hidden : oculto. minimized : minimizado. maximized : expandido.
panel_style	Estilo de la barra lateral. Valores posibles: default rounded
panel_pages	Lista de páginas del panel lateral y si son visibles o no. Una de las páginas puede tener el atributo " selected " a " true " para indicar que inicialmente esta página estará seleccionada. Páginas posibles: geocoding : Página de búsqueda de candidatos. proximity : Página de resolución de proximidades. route : Página de cálculo de rutas pois : Página de control visualización puntos de interés.
zoombar_status	Estado de la barra de zoom. Valores posibles: hidden : oculta. minimized : corta. maximized : larga.
pancontrol_status	Estado del control de desplazamiento: hidden : oculto. visible : visible.
overview_map_status	Estado del mapa de localización. Valores posibles: hidden : oculto. minimized : minimizado. maximized : expandido.
overview_map_style	Estilo del mapa de localización. Por defecto use " default ".
toolbar_status	Estado de la barra de herramientas. hidden : oculta. minimized : mínima. maximized : expandida.
toolbar_tools	Determinación del orden y los botones que aparecen en la barra de herramientas y el que está inicialmente seleccionado. pan : Herramienta para desplazarse por el mapa.

	<p>zoomin: Herramienta para aproximarse a una región. zoomout : Herramienta para alejarse. draw: Herramienta para dibujar. (no implementado) clean: Elimina resultados búsquedas anteriores. measure: Cálculo de distancias (no implementado) infopoi: Información punto interés. infoter: Información geoentidad a partir de coordenada. pois: Abre el diálogo de selección de categorías de POIs. export: (no implementado) customclick1: botón para personalizar. (Vea Anexo) customclick2: botón para personalizar. (Vea Anexo)</p>
show_mouse_position	Si se especifica true muestra las coordenadas de la posición del ratón. Por defecto false .
xsl_name	Nombre y ruta de la plantilla xsl que utiliza por defecto la función createReport. Si no se especifica ruta, la plantilla debe encontrarse en la misma carpeta que el fichero de configuración.
lswitcher_status	Estado de la herramienta para escoger el tipo de mapa visible. Despliega una lista con las capas de mapa disponibles (capas WMS creadas con la función createWMSStyle , estilos definidos por el parámetro map_styles i los estilos disponibles para Cercalia). Indicar visible o hidden para hacerlo visible o invisible. <lswitcher_status>visible</lswitcher_status>
map_styles	Estilos disponibles en el herramienta para escoger el tipo de mapa visible (<i>OpenStreetMap</i> , <i>Fotos aéreas España (PNOA)</i>): <map_styles> <openstreetmap>true</openstreetmap> <pnoa>true</pnoa> </map_styles>

4 La clase cercaliaClient

4.1 Estructuras u objetos usados por los métodos de esta clase

La clase **cercaliaClient** es la principal de esta API. Los objetos de esta clase representan a un mapa y poseen todos los métodos de llamada a los servicios Cercalia. Los parámetros de salida de los métodos son estructuras compuestas de las siguientes clases:

4.1.1 Atributos de un error (Error)

Campo	Descripción
id	Código de error
type	Tipo de error
text	Descripción del error

4.1.2 Atributos de un candidato (Candidate)

Campo	Descripción
desc	Descripción del candidato
type	Tipo de candidato (st,ct,...,poi)
id	Identificador de geoentidad
name	Nombre de geoentidad
ge	Objeto tipo geoentity
pos	Número de orden (usado para ordenar los resultados)

4.1.3 Atributos de una geoentidad (Geoentity)

Campo	Descripción
type	Tipo de geoentidad : <ul style="list-style-type: none"> • ctry : País. • reg : Región. • subreg: Subregión. • mun: Municipio. • ct: Ciudad / población / localidad. • st: Calle • rd: Carretera • pcode: Código postal • poi: Punto de interés
countryId	Código país
country	Nombre país
regionId	Código región
region	Nombre región
subregionId	Código subregión
subregion	Nombre subregión
municipalityId	Código municipio
municipality	Nombre municipio
cityId	Código ciudad / población / localidad
city	Nombre ciudad / población / localidad
streetId	Código calle
street	Nombre calle
streetPrefix	Prefijo de calle
streetArticle	Artículo de calle
streetName	Nombre corto de calle
houseNumber	Número de la calle
postalCode	Código postal
roadId	Código de carretera
roadName	Nombre de carretera
km	Punto kilométrico de la carretera
poId	Código punto de interés
poiName	Nombre punto de interés
x	Coordenada x del centro de la geoentidad

y	Coordenada y del centro de la geoentidad
srs	Sistema de referencia de la coordenada
arc	Objeto de tipo Arc (sólo informado en proximidad)
prox	Objeto de tipo Prox (sólo informado en proximidad)
pos	Número de orden (usado para ordenar los resultados)

4.1.4 Atributos de un punto de interés (Poi)

Campo	Descripción
id	Código del punto de interés
name	Nombre del punto de interés
categoryId	Código de categoría
subcategoryId	Código de subcategoría
geometry	Tipo de geometría : P: Punto L: Línea Z: Polígono
info	Información sobre el punto de interés
infoxml	Información xml sobre el punto de interés en objeto string
x	Coordenada x del punto de interés
y	Coordenada y del punto de interés
srs	Sistema de referencia de la coordenada
ge	Geoentidad asociada al punto de interés
prox	Objeto de tipo Prox
pos	Número de orden (usado para ordenar los resultados)
wkt	Geometría del punto de interés en formato web known text o null. Este campo sólo lo informa calculateRoute

4.1.5 Atributos de proximidad (Prox)

Campo	Descripción
dist	Distancia euclidiana (línea recta) respecto el centro
routedist	Distancia en coche. (Sólo si se ha especificado weight en la llamada)
routetime	Distancia en tiempo. (Sólo si se ha especificado weight en la llamada)

4.1.6 Atributos de arc (Arc)

Campo	Descripción
frc	Tipo de tramo. Functional road class.
kmh	Velocidad del tramo.
zonename	Nombre la zona a la pertenece el tramo
zonetype	Tipo de zona

4.1.7 Atributos de un objeto móvil (Mo)

Campo	Descripción
id	Identificador

desc	Descripción (opcional)
status	Estado (opcional)
x	Coordenada X centro referente
y	Coordenada Y centro referente
srs	Sistema de referencia de las coordenadas (ver Anexo de códigos)
pos	Número de orden (usado para ordenar los resultados)

4.1.8 Atributos de un objeto de estilo marcador (*markerStyle*)

Campo	Descripción
icon	Nombre del fichero de imagen para el icono (ej. "ico2.gif")
width	Ancho de la imagen del icono en píxeles
height	Alto de la imagen del icono en píxeles
visible	booleano que indica si se desea mostrar la información del marcador
minimized	booleano que indica si sólo se desea mostrar el título del marcador
noHideOthers	booleano que nos indica si no hemos de minimizar todos los otros marcadores existentes cuando creamos el actual, en el caso que se pueda minimizar
mouseover_content	Formato HTML a mostrar en el evento mouseover del icono. Se cerrará solo en el evento mouseout o al hacer clic.
mouseover	Código javascript a evaluar en el evento mouseover del icono. Compatible con <code>mouseover_content</code> .
mouseout	Código javascript a evaluar en el evento mouseout del icono.
mouseclick	Código javascript a evaluar en el evento mouseclick del icono. Compatible con <code>content</code> y <code>title</code> , a continuación.
rightclick	Código javascript a evaluar en el evento mousedown del botón <u>derecho</u> del <i>mouse</i> . Se debe activar la propiedad <i>handleRightClicks</i> del objeto <i>cercaliaProperties</i> para que funcione: <pre>function createAll() { properties=new cercaliaProperties("cercalia/config.xml"); properties.handleRightClicks=true; cercalia= new cercaliaClient(.....); }</pre>
draggable	True o False, para indicar que el marcador se puede mover de posición haciendo clic derecho sobre éste y arrastrándolo sobre el mapa hasta que soltamos el botón derecho del Mouse. Se debe activar la propiedad <i>handleRightClicks</i> del objeto <i>cercaliaProperties</i> para evitar que aparezca el menú contextual del navegador sobre el mapa.
showNumber	booleano que indica si se desea mostrar un número en el icono del marcador
numberTextColor	String con el color del número del icono (ej. "#FFFFFF")
title	Formato del título del marcador
content	Formato del contenido
cssTitle	String con el estilo a aplicar al título
iconBorderColor	Color del borde del div que incluye el icono. Si no se indica color no aparece el borde.
iconBorderWidth	Ancho en píxeles del borde del div que incluye el icono (por defecto 0px)

label	Contenido del label a mostrar.
labelState	Visible/hidden según si se quiere mostrar el label desde el momento de la creación del marcador.
hidden	True para crear el marcador oculto en el mapa (Añadir también visible:false para el popup)
zIndex	Si indicamos valor zIndex el marcador aparece por encima de todos los marcadores que no tengan un valor asignado o que tengan un valor inferior (en el caso de que haya solapamiento).

4.1.9 Atributos de un objeto de forma (feature)

Campo	Descripción
<Geometría>	Forma geométrica, en formato según solicitud
id	Identificador interno de la forma
created	Boelano indicando que se trata de una forma creada por el usuario (*)
modified	Boelano indicando que se trata de una forma que el usuario ha modificado (*)
deleted	Boelano indicando que se trata de una forma que el usuario ha borrado (*)
style	Estilo con el que ha sido creada la forma.
styleSelect	Estilo con el que se visualiza la forma en el momento que se selecciona para su edición.

(*) si todos los boelanos son falso, la forma no ha cambiado respecto a la original

El campo **<Geometría>** se instancia como uno de los siguientes campos, según el método llamado para obtener las formas:

Campo	Descripción
wkt	Forma geométrica en "Well-known Text Representation"
gml	Forma geométrica en formato GML v.3

4.1.10 Atributos de un objeto de etapa de ruta (Stage)

Campo	Descripción
coste_v1	Coste peajes tipo vehículo 1
coste_v2	Coste peajes tipo vehículo 2
coste_v3	Coste peajes tipo vehículo 3
dest	Destino de la etapa.
orig	Origen de la etapa.
substages	Lista de subetapas de la etapa (objetos Substage).

4.1.11 Atributos de un objeto de subetapa de ruta (Subtage)

Campo	Descripción				
name	Nombre descriptivo de la subetapa				
desc	Descripción de la subetapa				
angle	Angulo formado con la etapa anterior				
type	Tipo carretera de la subetapa *				
time	Tiempo para cruzar la subetapa				
dist	Distancia para cruzar la subetapa				
poilist	Lista de objetos <i>Poi</i> de la subetapa				
arealist	Lista de zonas de la subetapa				
	<table border="1"> <tr> <td>Ítem</td> <td> type: Tipo área: <ul style="list-style-type: none"> • U: Zona urbana • I: Polígono industrial • X: Ninguna de las anteriores (el nombre estará en blanco) </td> </tr> <tr> <td></td> <td>name: Nombre área</td> </tr> </table>	Ítem	type: Tipo área: <ul style="list-style-type: none"> • U: Zona urbana • I: Polígono industrial • X: Ninguna de las anteriores (el nombre estará en blanco) 		name: Nombre área
Ítem	type: Tipo área: <ul style="list-style-type: none"> • U: Zona urbana • I: Polígono industrial • X: Ninguna de las anteriores (el nombre estará en blanco) 				
	name: Nombre área				
municipalities	Lista de municipios por los que pasa la subetapa				
	<table border="1"> <tr> <td>Ítem</td> <td>id: Identificador</td> </tr> <tr> <td></td> <td>name: Nombre</td> </tr> </table>	Ítem	id: Identificador		name: Nombre
Ítem	id: Identificador				
	name: Nombre				
signal	Lista de señales que aparecen en el trayecto de la subetapa (carteles autovías, autopistas, ...)				
	<table border="1"> <tr> <td>Ítem</td> <td>type: Tipo de señal ^{*2}</td> </tr> <tr> <td></td> <td>value: Contenido de la señal</td> </tr> </table>	Ítem	type: Tipo de señal ^{*2}		value: Contenido de la señal
Ítem	type: Tipo de señal ^{*2}				
	value: Contenido de la señal				

* Valores posibles del atributo *type* (tipo de la carretera de la subetapa):

Valor	Descripción
A	Autopistas
G	Ferry
I	Carretera principal, con nombre
K	Carreteras secundarias, con nombre
J	Carreteras sin nombre
L	Calles con nombre
M	Calles secundarias, sin clasificación específica
R	Rotonda
S	Enlace
T	Vía de servicio

^{*2} Valores posibles del atributo *type* (tipo de señal):

Valor	Descripción
4G	Nombre salida
4E	Número salida
4I	Otros destinos
9D	Nombre lugar
6T	Nombre calle
RN	Código de carretera

4.2 Métodos para el tratamiento de marcadores

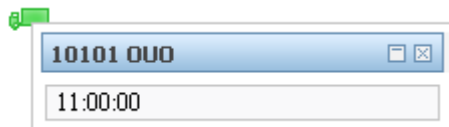
4.2.1 *createMarker(id,x,y,srs,style,id2)*

Crea un marcador según los parámetros que se le pasen:

- **id**: Identificador del marcador y del div que contendrá el contenido del marcador.
- **x,y**: coordenadas x e y donde se situará el marcador.
- **srs**: sistema de coordenadas utilizado (ver anexo sistemas de coordenadas) .
- **style**: Objeto de tipo `markerStyle` que define las características del marcador.
- **id2**: parámetro opcional que asocia el marcador actual con otro, y hace que se dibuje una flecha entre los dos.

Ejemplos de marcador:

Marcador maximizado:



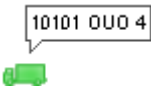
Marcador minimizado:



Marcador con contenido y título no visibles:



Marcador con *label* visible:



En este caso no asociamos el marcador a ningún otro, y por lo tanto, no pasamos el último parámetro.

Si se utiliza un `markerStyle` que no especifica **title** ni **cssTitle** en la etiqueta sólo habrá el contenido. El **content** tendrá que ser un html válido y el formato de visualización lo deberemos especificar nosotros mismos por `css` o mediante el atributo `style`. Si tampoco se especifica **content** se creará un marcador sin etiqueta.

Ejemplo de código:

```
var cont = '<font style="font: 12px black; padding: 5px;">11:00:00</font>' ;

var style={icon:"img/green_lorry.gif", width:22, height:22, visible:true,
  minimized:false, title:"10101OVO", cssTitle:"contTitolPopup", content:cont,
  label:" 10101OVO", labelState:'visible',noHideOthers:true};

cercalia.createMarker("10101 OVO 1", -3.808951, 43.458308, "EPSG:4326", style);
```

Ejemplo del contenido:

```

var cont='';

cont+='<div class="estilCSS">';
cont+=' interiorContenido';
cont+='</div>';

var cont2='';

cont2+='<div style="background: #FFFFFF; color: #000000; border-color: blue;">';
cont2+=' interiorContenido';
cont2+='</div>';

```

Donde en `interiorContenido` le podemos poner el código html que queramos: imágenes, tablas, etc. En el ejemplo anterior: `11:00:00`

4.2.2 createMarkers (obj,style)

Esta función crea los marcadores para los elementos contenidos en un objeto resultado de un geocoding o una proximidad. El parámetro **obj** tiene que tener tipo **resultGeocoding**, **resultProximity**, **resultGetPoi**, **Mo**, **Poi** o **Geoentity**. El objeto **style** es una lista de atributos pertenecientes a **markerStyle**.

En los atributos **icon**, **title** y **content** del **markerStyle** se pueden especificar campos del candidato, punto de interés o geoentidad de la que se crea el **marker** mediante el formato: `%{expr}%`, también se puede especificar un código de mensaje del fichero de mensajes con la sintaxis: `%[msg1]%`

Ejemplo de código para crear marcadores de puntos de interés :

```

function obtenerPois()
{
    cercalia.getPoi(1,"C57240362250410,C57240362276846",retorno);
}

function retorno(datos)
{
    var style={width:16, height:16, visible:true,
        minimized:false, noHideOthers:true,
        showNumber:false, content:null,
        cssTitle:"contTitolPopup"};

    var cont = '';

    if(datos.pois!=null && datos.pois.length>0)
    {
        cont+='<table>';
        cont+='<tr><td colspan="2"><b>{%pois.name}</b></td></tr>';
        cont+='<tr><td>[City]</td><td>{%pois.ge.city}</td></tr>';
        cont+='<tr><td>[PostalCode]</td><td>{%pois.ge.postalCode}</td></tr>';
        cont+='<tr><td>[Municipality]</td><td>{%pois.ge.municipality}</td></tr>';
        cont+='<tr><td>[Subregion]</td><td>{%pois.ge.subregion}</td></tr>';
        cont+='<tr><td>[Region]</td><td>{%pois.ge.region}</td></tr>';
        cont+='<tr><td>[Country]</td><td>{%pois.ge.country}</td></tr>';
        cont+='</table>';

        style.title="%{pois.name}";
        style.content=cont;
        style.numberTextColor="red";
        style.showNumerated=true;
        style.icon="img/point_prox.gif";
        cercalia.createMarkers(dades.pois, style);
    }
}

```

```

        cercalia.centerToMarkers(true);
    }
}

```

Ejemplo de código para crear el marcador del primer punto de interés:

```

function obtenerPois()
{
    cercalia.getPoi(1,"C57240362250410,C57240362276846",retorno);
}

function retorno(datos)
{
    var style={width:16, height:16, visible:true,
                minimized:false, noHideOthers:true,
                showNumber:false, content:null,
                cssTitle:"contTitolPopup"};

    var cont = '';

    if(datos.pois!=null && datos.pois.length>0)
    {
        cont+='<table>';
        cont+='<tr><td colspan="2"><b>{%poi.name}%</b></td></tr>';
        cont+='<tr><td>[City]</td><td>{%poi.ge.city}%</td></tr>';
        cont+='<tr><td>[PostalCode]</td><td>{%poi.ge.postalCode}%</td></tr>';
        cont+='<tr><td>[Municipality]</td><td>{%poi.ge.municipality}%</td></tr>';
        cont+='<tr><td>[Subregion]</td><td>{%poi.ge.subregion}%</td></tr>';
        cont+='<tr><td>[Region]</td><td>{%poi.ge.region}%</td></tr>';
        cont+='<tr><td>[Country]</td><td>{%poi.ge.country}%</td></tr>';
        cont+='</table>';

        style.title="%{poi.name}%";
        style.content=cont;
        style.numberTextColor="red";
        style.showNumerated=false;
        style.icon="img/point_prox.gif";
        cercalia.createMarkers(dades.pois[0],style);
        cercalia.centerToMarkers(true);
    }
}

```

4.2.3 moveMarker(id,x,y,srs,cont,contOver)

Mover el marcador con el identificador **id** a las nuevas coordenadas.

- **id**: identificador del marcador y del div que contendrá el contenido del marcador.
- **x,y**: coordenadas x e y donde se situará el marcador.
- **srs**: sistema de coordenadas utilizado.
- **cont**: Contenido html válido para refrescar el popup del marcador. (Parámetro opcional).
- **contOver**: Contenido html válido para refrescar el popup del marcador que aparece en el evento mouseover. (Parámetro opcional)

Ejemplo 1:

```
cercalia.moveMarker("10101 OUO 1",43.458425,-3.80895,"EPSG:4326");
```

Ejemplo 2:

Cambia el contenido del popup del marcador indicado por parámetro.

```
var cont = '<font style="font: 12px black; padding:5px;">'+lon+', '+lat+'</font>';
cercalia.moveMarker("10101 OUO 1",43.458425,-3.80895,"EPSG:4326",cont);
```

Ejemplo 3:

Cambia el contenido del popup que aparece en el evento **mouseover** sin modificar el contenido del popup que aparece al hacer clic sobre el marcador.

```
var contOver = '<font style="font:12px black;padding:5px;">'+lon+', '+lat+'</font>';
cercalia.moveMarker("10101 OUO 1",43.458425,-3.80895,"EPSG:4326",null,contOver);
```

4.2.4 setMouseoverContent (id,contOver)

Cambia el contenido del popup que aparece en el evento **mouseover**. Sólo funciona con los marcadores para los que se ha indicado la propiedad *mouseover_content*.

- **id**: identificador del marcador.
- **contOver**: Contenido html válido para refrescar el popup del marcador que aparece en el evento mouseover.

```
var contOver = '<font style="font:12px black;padding:5px;">'+lon+', '+lat+'</font>';
cercalia.setMouseoverContent("10101 OUO 1",contOver);
```

4.2.5 setMarkerIcon(id,url)

Permite modificar el icono del marcador indicado por el parámetro *id*. La imagen por la que se substituirá la actual se indica a través del parámetro *url*. Ambos parámetros son obligatorios y debe existir un marcador con el identificador indicado.

Se puede combinar, por poner un ejemplo, con la utilidad del atributo *mouseclick* del parámetro *style* de la función *createMarker*. Así podemos modificar el estado del icono cuando el usuario hace clic sobre éste.

Ejemplo 1:

```
cercalia.setMarkerIcon("10101 OUO 1","img/red_car.gif");
```

4.2.6 deleteMarkers([id1 [, id2 [...]]])

Elimina los marcadores los cuales pasan su identificador por parámetro. Si no se pasa ningún identificador, se eliminarán todos los marcadores del mapa.

Ejemplos:

Eliminar todos los marcadores:

```
cercalia.deleteMarkers();
```

Eliminar un marcador:

```
cercalia.deleteMarkers("10101 OUO 1");
```

Eliminar tres marcadores:

```
cercalia.deleteMarkers("10101 OUO 1","10101 OUO 2","10101 OUO 3");
```

4.2.7 *centerToMarkers(changeScale [,id1 [, id2 [...]]])*

Centra el mapa respecto los marcadores los cuales se pasa su identificador por parámetro. Si no se pasa ningún identificador se centrará en todos los marcadores del mapa.

- **changeScale**: booleano que nos indica si se ha de cambiar la escala para que se visualicen todos los marcadores afectados.

Ejemplos:

Centrar el mapa respecto a todos los marcadores cambiando la escala de manera que se visualicen todos:

```
cercalia.centerToMarkers(true);
```

Centrar el mapa en un marcador sin cambiar la escala:

```
cercalia.centerToMarkers(false,"10101 OUO 1");
```

Centrar el mapa respecto a tres marcadores cambiando la escala:

```
cercalia.centerToMarkers(true,"10101 OUO 1","10101 OUO 2","10101 OUO 3");
```

4.2.8 *showMarkers(state [,id1 [, id2 [...]]])*

Cambiar el estado de visibilidad de los marcadores al estado pasado por el parámetro **state**. Este cambio afecta a todos los marcadores que se pase su identificador por parámetro y en caso de no pasar ningún identificador, afectará a todos los marcadores del mapa.

- **state**: estado en que se cambiarán los marcadores afectados. Éstos pueden ser tres:
 - **closed**: cerrarán los marcadores.
 - **minimized**: minimizarán los marcadores.
 - **maximized**: maximizarán los marcadores.

Ejemplos:

Maximizar todos los marcadores del mapa:

```
cercalia.showMarkers("maximized");
```

Minimizar un marcador:

```
cercalia.showMarkers("minimize","10101 OUO 1");
```

Ocultar la información asociada de tres marcadores:

```
cercalia.showMarkers("closed","10101 OOU 1","10101 OOU 2","10101 OOU 3");
```

4.2.9 *displayMarker(idMarker,visible)*

Muestra u oculta el marcador con un identificador igual a idMarker.

Ejemplos:

Mostrar marcador:

```
cercalia.displayMarker("marker02020",true);
```

Ocultar marcador:

```
cercalia.displayMarker("marker02020",false);
```

4.2.10 *showLabelMarkers (visible)*

Muestra u oculta los elementos *label* de los marcadores.

Ejemplo:

Esconder etiquetas:

```
cercalia.showLabelMarkers(false);
```

4.2.11 *showLabel (visible,idMarker)*

Muestra u oculta los elementos *label* del marcador indicado.

4.3 Métodos para el tratamiento de formas

4.3.1 *createFeature(wkt,srs,style [, editstyle])*

Crea una o más formas a partir de su definición geométrica en WKT, el sistema de referencia y la definición de estilo. Devuelve una lista de identificadores de las formas creadas.

- **wkt**: formas geométricas
- **srs**: sistema de coordenadas usado*
- **style**: estilo de representación
- **editstyle**: estilo de representación en edición

* Valores posibles sistema coordenadas (parámetro srs): ver anexo

El formato de WKT (“Well-known Text Representation for Geometry”) es un estándar para el intercambio de geometrías definido por [Open Geospatial Consortium](#). La siguiente tabla muestra ejemplos de este formato:

Geometry Type	Text Literal Representation	Comment
Point	'POINT (10 10)'	a Point
LineString	'LINESTRING (10 10, 20 20, 30 40)'	a LineString with 3 points
Polygon	'POLYGON ((10 10, 10 20, 20 20, 20 15, 10 10))'	a Polygon with 1 exteriorRing and 0 interiorRings
Multipoint	'MULTIPOINT (10 10, 20 20)'	a MultiPoint with 2 points
MultiLineString	'MULTILINESTRING ((10 10, 20 20), (15 15, 30 15))'	a MultiLineString with 2 linestrings
MultiPolygon	'MULTIPOLYGON (((10 10, 10 20, 20 20, 20 15, 10 10)), ((60 60, 70 70, 80 60, 60 60)))'	a MultiPolygon with 2 polygons
GeomCollection	'GEOMETRYCOLLECTION (POINT (10 10), POINT (30 30), LINESTRING (15 15, 20 20))'	a GeometryCollection consisting of 2 Point values and a LineString value

Para mayor información consulte el documento [OpenGIS® Implementation Specification for Geographic information - Simple feature access - Part 1: Common architecture](#).

El parámetro **style** es una lista de propiedades con su valor:

Propiedad	Descripción	Valor por defecto
fillColor	Color de relleno	"#ee9900"
fillOpacity	Opacidad del relleno	0.4
strokeColor	Color de la línea o contorno	"#ee9900"
strokeOpacity	Opacidad de la línea o contorno	1
strokeWidth	Ancho de la línea o contorno	1
pointRadius	Tamaño del punto en píxeles	0
label	Contenido html de un label informativo a mostrar	null
labelState	Visibilidad del label (visible/hidden)	null

Ejemplo para dibujar una poli línea:

```
f25 = cercalia.createFeature("LINESTRING(-2.4027508398930078 41.190563340655544,-2.402553210530501 41.188147539419674,-2.398420960223551 41.18071635184718,-2.398340111847981 41.179012827498276)","EPSG:4326",{strokeWidth: 5,strokeColor: "#FF0000"});
```

Ejemplo para dibujar un polígono:

```
f31 = cercalia.createFeature("POLYGON(-448003 4817761,-706393 4682834,-667531 4466724,-448003 4817761)","EPSG:54004",{strokeWidth: 3,strokeColor: "#00ee00",fillColor: "#00ee00",fillOpacity: 0.2});
```

Ejemplo para dibujar un rectángulo:

```
f41 = cercalia.createFeature("POLYGON(-758765 4940710,-756487 4940710,-756487 4941341,-758765 4941341,-758765 4940710)","EPSG:54004",{strokeWidth: 3,strokeColor: "#00ee00",fillColor: "#00ee00",fillOpacity: 0.2});
```

Ejemplo para dibujar un círculo:

```
f51 = cercalia.createFeature("POINT (-448003 4817761)", "EPSG:54004", {strokeWidth: 3,strokeColor: "#00ee00", fillColor: "#00ee00",fillOpacity: 0.2});
```

De forma análoga, el **parámetro editstyle** contiene el estilo de las formas cuando el usuario pase a editarlas.

4.3.2 *createFeature(gml,style [, editstyle])*

Crea una o más formas a partir de su definición geométrica en GML 3.1.1 (Geography Markup Language) en forma de cadena (*String*) y la definición de estilo. Devuelve una lista de identificadores de las formas creadas.

- **gml**: formas geométricas
- **style**: estilo de representación
- **editstyle**: estilo de representación en edición

El formato GML debe contener valor para el campo *srsName*. Si no es así se supondrá que la coordenadas están representadas en el sistema de coordenadas del mapa.

Para más información sobre el estándar GML ver la especificación proporcionada por OpenGIS (<http://www.opengeospatial.org/standards/gml>).

4.3.3 *showFeatures(visible [,feature1 [, feature2 [...]]])*

Muestra u oculta en el mapa las formas que se pasan en uno o más parámetros. Las formas han sido creadas previamente con la función *createFeature* o las *herramientas de edición*. Si no se indica ninguna se muestran u ocultan todas las formas creadas en el mapa.

Para crear las formas no visibles con la función *createFeature*, simplemente se debe añadir la propiedad '**display:none**' en el parámetro **style** de la función:

```
var style={strokeWidth: 2,strokeColor: "yellow", pointRadius:10, externalGraphic:"img/car.gif", display:"none"};  
  
f42= cercalia.createFeature("POINT(-412246 4898974, -412164 4898979)", "EPSG:54004", style);
```

En los casos que se tenga que crear un número importante de figuras en el mapa pero no sea necesario que todas sean visibles en el mismo instante, se pueden crear ocultas todas al principio i posteriormente ir haciendo visibles las que sea necesario.

Ejemplos:

Para hacer visible una figura:

```
cercalia.showFeatures(true, f42);
```

Para ocultar todas las figuras:

```
cercalia.showFeatures(false);
```

4.3.4 *centerToFeatures(changeScale [, feature1 [, feature2 [...]]])*

Centra el mapa a las formas que se pasan en uno o más parámetros. Si no se indica ninguna forma se centra a todas las formas presentes en el mapa.

- **changeScale:** booleano que nos indica si se ha de cambiar la escala para que se visualicen todas las formas afectadas.

Ejemplos:

Centrar el mapa respecto a todas las formas cambiando la escala de manera que se visualicen todas:

```
cercalia.centerToFeatures(true);
```


Centrar el mapa en una forma sin cambiar la escala:

```
cercalia.centerToFeatures(false, f25);
```

Centrar el mapa respecto a dos formas cambiando la escala:

```
cercalia.centerToFeatures(true, f25, f31);
```

4.3.5 *deleteFeatures([feature1 [, feature2 [...]]])*

Elimina las formas especificadas en uno o más parámetros. Si no se indica ninguna forma se borran todas. En el caso de que no se indiquen parámetros, también se borran los elementos de la lista de formas que habían sido eliminadas con las utilidades de la herramienta “draw”  (ver anexo “Herramientas de edición”). A partir de ese momento no se podrán consultar las formas con estado ‘deleted’ con el método *getFeaturesByState*.

Ejemplos:

Eliminar todas las formas:

```
cercalia.deleteFeatures();
```

Eliminar una forma:

```
cercalia.deleteFeatures(f25);
```

Eliminar dos formas:

```
cercalia.deleteFeatures(f25, f31);
```

4.3.6 *getFeaturesByState(created, modified, deleted, unchanged[,SRS])*

Obtiene las formas que cumplen los estados indicados: permite recuperar y guardar en una base de datos propia la geometría de las formas creadas mediante el módulo de digitalización de la API (ven Anexo 8).

Como parámetro opcional se puede indicar el sistema de coordenadas en que queremos obtener las geometrías. Si no se indica se utilizará el correspondiente al mapa de fondo.

Estados:

- **created**: booleano para obtener las formas nuevas que haya creado el usuario.
- **modified**: booleano para obtener las formas el usuario ha modificado.
- **deleted**: booleano para obtener las formas el usuario ha borrado.
- **unchanged**: booleano para obtener las formas el usuario no ha cambiado.


Se puede hacer cualquier combinación de estados. Sin embargo, todos los estados al **false** no devolverán respuesta.

Las formas se devuelven en un objeto **resultFeatures**, con la siguiente definición:

Campo	Descripción
features	Lista de objetos de tipo Feature

Las feature tendrán el atributo **wkt** para contener la definición de la geometría.

Ejemplos:

Obtiene las formas que han sido creadas en el mapa (formas nuevas) mediante la herramienta “draw”  (ver anexo “Herramientas de edición”).

```
var resfeatures=cercalia.getFeaturesByState(true,false,false,false);  
var numFeatsNew = resfeatures.features.length;
```

Obtiene las formas que han sido creadas mediante la función createFeature y que posteriormente han sido modificadas mediante las utilidades de la herramienta “draw”.

```
var resfeatures=cercalia.getFeaturesByState(false,true,false,false);
```

4.3.7 *getFeaturesByStateAsGML(created, modified, deleted, unchanged[,SRS])*


Obtiene las formas que cumplen los estados indicados, de forma análoga a la función anterior.

Las formas se devuelven en un objeto **resultFeatures**. Las feature tendrán el atributo **gml** para contener la definición de la geometría.

4.3.8 *getFeatures([feature1 [, feature2 [...]]],SRS)*

Obtiene las formas especificadas en uno o más parámetros. Si no se indica ninguna forma se devolverán todas. Permite recuperar y guardar en una base de datos propia la geometría de las formas creadas mediante el módulo de digitalización de la API (vea el Anexo 8).

Las formas se devuelven en un objeto **resultFeatures**. Las feature tendrán el atributo **wkt** para contener la definición de la geometría.

Sólo se devolverán formas eliminadas con la herramienta “draw”  si han sido creadas con la función *createFeature*.

4.3.9 *getFeaturesAsGML([feature1 [, feature2 [...]]],SRS)*

Obtiene las formas especificadas en uno o más parámetros, de forma análoga a la función anterior.

Las formas se devuelven en un objeto **resultFeatures**. Las feature tendrán el atributo **gml** para contener la definición de la geometría.

4.3.10 *showLabelFeature (visible,id)*

Muestra u oculta los elementos *label* de la forma indicada.

Ejemplo:

```
var f1= cercalia.createFeature(.....);  
cercalia.showLabelFeature(true,f1);
```

4.4 Métodos tratamiento del mapa

4.4.1 *activateTool(tool)*

Selecciona o activa la herramienta que se indica por parámetro (*pan, zoomin, zoomout, draw, clean, measure, infopoi, infoter, pois, export, customclick1, customclick2*). La herramienta debe estar disponible en la barra de herramientas o no se realiza ninguna acción.

```
cercalia.activateTool("customclick1");
```

4.4.2 *activateMeasure (type, deactivateOnEnd)*

Activa el control para medir distancias que se indica por el parámetro **type** (**road, line o polygon**). El parámetro **deactivateOnEnd** a *true* indica que el control se

desactiva una vez realizada la acción y para volver a realizar una medición se debe volver a activar el control. Con valor *false* se pueden realizar tantas mediciones como se desee hasta que se ejecute alguna instrucción que desactive el control. Si se indica valor *false* se pueden desactivar los controles mediante la instrucción *cercalia.activateTool('pan')* que vuelve a activar la funcionalidad de desplazar el mapa.

Ejemplos:

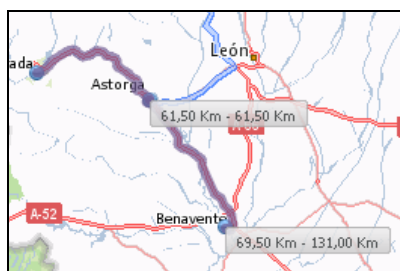
```
var deactivateOnEnd=true;
function activateMeasureRoad(deactivateOnEnd) {
    cercalia.activateMeasure('road',deactivateOnEnd);
}

function activateMeasureLine(deactivateOnEnd) {
    cercalia.activateMeasure('line',deactivateOnEnd);
}

function activateMeasureArea(deactivateOnEnd) {
    cercalia.activateMeasure('polygon',deactivateOnEnd);
}
```

4.4.3 deleteLastMeasurement ()

Elimina la última medición realizada (desde el primer clic hasta el doble clic que la finaliza).



4.4.4 deleteAllMeasurements ()

Elimina todas las mediciones realizadas del mapa.

4.4.5 Evento clic en el mapa

Se puede programar el evento clic para el mapa para ejecutar la función que se desee. Se puede hacer tanto con el clic izquierdo del *mouse* como con el derecho. Esta funcionalidad permanece activa mientras estén seleccionadas las herramientas de mover el mapa o de hacer zoom.

- **Clic izquierdo:** Sólo tenemos que indicar qué función queremos que se ejecute al detectar el clic en el mapa.

```
properties=new cercaliaProperties("cercalia/config.xml");
```

```
properties.functionDefaultClick=defaultCustomClickMapa;
```

```
function defaultCustomClickMapa(e) {  
    //Podemos obtener la coordenada a través del evento  
    var lonlat = cercalia.getCoordFromPixel(e.xy, "EPSG:54004");  
}
```

- **Clic derecho**: Sólo tenemos que habilitar la captura de eventos con el clic derecho e indicar qué función queremos que se ejecute.

```
properties=new cercaliaProperties("cercalia/config.xml");  
properties.handleRightClicks=true;  
properties.functionRightClick= defaultRightClickMapa;
```

```
function defaultRightClickMapa(e) {  
    //Podemos obtener la coordenada a través del evento  
    var lonlat = cercalia.getCoordFromPixel(e.xy, "EPSG:54004");  
}
```

4.4.6 *isCompleted()*

Devuelve booleano true si el objeto cercalia está listo para usar.

4.4.7 *showPois(visible,category[[,category],...])*

Activa o desactiva la visualización de los puntos de interés de las categorías especificadas. Si no se indican categorías se visualizarán o se desactivará la visualización de todas las categorías existentes según indique el parámetro *visible*.

Ejemplo para visualizar los puntos de interés de las categorías C001 y C003:

```
cercalia.showPois(true,'C001','C003');
```

Ejemplo para desactivar la visualización de los puntos de interés de las categorías C001 y C003:

```
cercalia.showPois(false,'C001','C003');
```

4.4.8 *setLanguage(lang)*

Modifica el idioma del cliente. El parámetro *lang* indica a que idioma queremos cambiar. Si no existe un archivo .xml para el idioma indicado aparecerá un mensaje de error.

Ejemplo para cambiar el idioma actual a inglés:

```
cercalia.setLanguage('EN');
```

4.4.9 *getMap()*

Devuelve un objeto OpenLayers.Map. Consulte la API de documentación de OpenLayers para ver las posibilidades de este objeto (<http://www.openlayers.org>).

Ejemplo :

```
var map = cercalia.getMap();  
map.events.register("zoomend", map, actualizarPosicion);
```

4.4.10 deleteRoute (resultCalculateRoute)

Elimina la representación gráfica (marcadores y formas) de la ruta que se le pasa por parámetro.

4.4.11 setParse< operation > (function)

Estos métodos permiten la personalización de la respuesta recibida por el usuario cuando activa acciones de la barra lateral o botones.

Los nombres de las funciones a llamar para intercalar el código, y los respectivos parámetros que deberán tener las nuevas funciones son:

setParsePanelSearchGeo (function)

Permite personalizar las acciones que se realizan después de buscar los candidatos con el panel lateral de dirección o código postal. El parámetro *function* debe indicar una función que recibe un parámetro de tipo *resultGeocoding*. Esta función puede llamar a `showResultPanelSearchGeo(resultGeocoding)` para mostrar los resultados.

setParsePanelSearchCoord (function)

Permite personalizar las acciones que se realizan después de buscar una coordenada en el panel lateral. El parámetro *function* debe indicar una función que recibe un parámetro de tipo *resultProximity*. Esta función puede llamar a `showResultPanelSearchCoord(resultProximity)` para mostrar los resultados.

setParsePanelProximity (function)

Permite personalizar las acciones que se realizan después de buscar una proximidad con el panel lateral. El parámetro *function* debe indicar una función que recibe un parámetro de tipo *resultProximity*. Esta función puede llamar a `showResultPanelProximity(resultProximity)` para mostrar los resultados.

setParseToolInfoPOI (function)

Permite personalizar las acciones que se realizan después de buscar la información del POI seleccionado. El parámetro *function* debe indicar una función que recibe un parámetro de tipo *resultProximity*. Esta función puede llamar a `showToolInfoPOI(resultProximity)` para mostrar los resultados.

setParseToolInfoTer (function)

Permite personalizar las acciones que se realizan después de obtener la información territorial. El parámetro *function* debe indicar una función que recibe un parámetro de tipo *resultProximity*. Esta función puede llamar a *showToolInfoTer(resultProximity)* para mostrar los resultados.

Ejemplo: Modificar la salida de candidatos para que salga en el mapa un marcador para cada candidato.

```
function functionSetParsePanelGeo(data)
{
    if(data.candidates && data.candidates.length>1)
    {
        var cont="";
        cont+='<table style="font-size: 11px;font-family: Tahoma;color: #666666;
width:100%; height:100%; padding:4px; ">';
        cont+='<tr><td style="border-bottom: 1px solid rgb(134, 134, 134);
padding-top:5px; padding-bottom:5px;" colspan="2"><b>{candidates.name}</b></td></tr>';
        cont+='<tr><td style=" padding-top:5px; padding-bottom:5px;"
><b>[City]</b></td><td>{candidates.ge.city}</td></tr>';
        cont+='<tr><td style="padding-bottom:5px;"
><b>[PostalCode]</b></td><td>{candidates.ge.postalCode}</td></tr>';
        cont+='<tr><td style="padding-bottom:5px;"
><b>[Municipality]</b></td><td>{candidates.ge.municipality}</td></tr>';
        cont+='<tr><td style="padding-bottom:5px;"
><b>[Subregion]</b></td><td>{candidates.ge.subregion}</td></tr>';
        cont+='<tr><td style="padding-bottom:5px;"
><b>[Region]</b></td><td>{candidates.ge.region}</td></tr>';
        cont+='<tr><td style="padding-bottom:5px;"
><b>[Country]</b></td><td>{candidates.ge.country}</td></tr>';
        cont+='</table>';
        var style={ icon:"img/xinxeta.gif",
            width:16,
            height:16,
            visible:false,
            minimized:false,
            title:"{candidates.name}",
            noHideOthers:true,
            showNumber:false,
            content:cont,cssTitle:""};
        style.cssTitle="contTitolPopup";
        cercalia.createMarkers (data.candidates,style);
        cercalia.centerToMarkers (true);
    }
    else cercalia.showResultPanelSearchGeo (data);
}

cercalia.setParsePanelSearchGeo (functionSetParsePanelGeo);
```

Ejemplo: Modificar la salida de información de infoposición para que nos indique la coordenada en formato geográficas.

```
function infoposicion(data)
{
    cercalia.transformCoordinates (data.geoentities,"EPSG:4326");
    cercalia.showToolInfoTer (data);
}

cercalia.setParseToolInfoTer (infoposicion);
```

4.4.12 *showResult< operation> (obj)*

Si con las funciones solo desea manipular los datos y quiere que se termine con la presentación por defecto. Puede llamar a los siguientes métodos:

showPanelSearchGeo (resultGeocoding)

Muestra el resultado de una búsqueda de candidatos en el panel lateral (dirección o código postal).

showPanelSearchCoord (resultProximity)

Muestra el resultado de una búsqueda por coordenada en el panel lateral.

showPanelProximity (resultProximity)

Muestra en el panel lateral el resultado de una proximidad.

showToolInfoPOI (resultProximity)

Muestra en el mapa la información del punto de interés más cercano.

showToolInfoTer (resultProximity)

Muestra en el mapa la información de la geoentidad más cercana.

4.4.13 *getExtent(srs)*

Devuelve un objeto `OpenLayers.Bounds` con las coordenadas de la extensión del mapa que se está visualizando. Las coordenadas estarán en el sistema indicado por el parámetro obligatorio `srs`.

Ejemplo: Obtener la extensión del mapa con el sistema de coordenadas actual:

```
function obtenerExtension()
{
    var srs = cercalia.getCurrentSRS();
    var ext = cercalia.getExtent(srs);

    alert(ext.left+","+ext.bottom+","+ext.right+","+ext.top);
}
```

4.4.14 *getCenter(srs)*

Devuelve un objeto `OpenLayers.LonLat` con las coordenadas del centro del mapa que se está visualizando. Las coordenadas estarán en el sistema indicado por el parámetro obligatorio `srs`.

Ejemplo: Obtener el centro del mapa con el sistema de coordenadas actual:

```
function obtenerCentro()
{
    var srs = cercalia.getCurrentSRS();
    var center=cercalia.getCenter(srs);
    alert(center.lon+", "+center.lat);
}
```

4.4.15 *setCenter(center, scale)*

Cambia el centro del mapa que se está visualizando en ese momento. El parámetro `center` tendrá el siguiente formato `{x: ..., y:..., srs: ...}` donde `x`, `y` son coordenadas y `srs` el sistema de coordenadas. Si se indica el parámetro `scale` se adaptará la visualización del mapa al nivel de escala¹ del sistema de `cercalia`. Si no estamos trabajando con la cartografía `Cercalia` se adaptará al nivel de escala `cercalia` más próximo al actual.

Ejemplo: Actualizar el centro del mapa con coordenadas del sistema `EPSG:4326` e indicando un nivel de escala `cercalia`:

```
function actualizaCentro()
{
    var point={x:"2.820709",y:"41.976419",srs:"EPSG:4326"};
    cercalia.setCenter(point,5);
}
```

4.4.16 *setCenter(center, srs, scale)*

Cambia el centro del mapa al indicado. El parámetro `center` debe tener tipo `OpenLayers.LonLat`. Y las coordenadas estar en el sistema de referencia indicado por `srs`. Si se indica el parámetro `scale` se adaptará la visualización del mapa al nivel de escala² del sistema de `cercalia`. Si no estamos trabajando con la cartografía `Cercalia` se adaptará al nivel de escala `cercalia` más próximo al actual.

Ejemplo: Actualizar el centro del mapa con coordenadas del sistema `EPSG`:

```
function actualizaCentro()
```

¹ En el sistema `cercalia` hay 14 niveles de escalas (de 0 a 13). El nivel 0 corresponde a la escala más pequeña y el nivel 13 a la mayor escala disponible en el sistema.

² En el sistema `cercalia` hay 14 niveles de escalas (de 0 a 13). El nivel 0 corresponde a la escala más pequeña y el nivel 13 a la mayor escala disponible en el sistema.

```
{  
  
    var lonlat = new OpenLayers.LonLat("2.820709", "41.976419");  
    cercalia.setCenter(lonlat, "EPSG:4326");  
  
}
```

4.4.17 **zoomToExtent(extent,srs)**

Adapta el zoom del mapa a la extensión indicada. El parámetro *srs* indica el sistema de coordenadas en el que se expresa la extensión. La extensión se indica con el parámetro *extent* y puede ser un objeto `OpenLayers.Bounds` o bien una lista con cuatro atributos: {x1:.....,y1:.....,x2:.....,y2:.....}

Ejemplo:

```
var ext=new OpenLayers.Bounds(-22000000,-13191574,21000000,18500000);  
cercalia.zoomToExtent(ext, "EPSG:54004");  
  
ó  
  
var ext={x1:"-22000000", y1:"-13191574", x2:"21000000", y2:"18500000"};  
cercalia.zoomToExtent(ext, "EPSG:54004");
```

4.4.18 **getScale(closest)**

Devuelve un numérico que indica el nivel de escala³ actual del mapa. Si *closest* es cierto devuelve el nivel de escala cercalia actual y si trabajamos con otra cartografía nos devuelve el nivel de cercalia más próximo al del mapa que estamos utilizando. Si *closest* es falso devuelve siempre el nivel de escala de cercalia y nulo si trabajamos con otra cartografía. El método `getStyle` nos indica la cartografía que estamos usando.

4.4.19 **getCurrentSRS()**

Devuelve un string indicando el sistema de coordenadas actual del mapa.

4.5 Métodos para obtener información

4.5.1 **geocoding(search,returnFunc)**

Realiza una petición de geoconversión según los parámetros del objeto *search*. Una vez resuelta se ejecuta la función *returnFunc* pasándole por parámetro un objeto tipo `resultGeocoding`.

El objeto *search* puede contener los siguientes campos:

³ En el sistema cercalia hay 14 niveles de escalas (de 0 a 13). El nivel 0 corresponde a la escala más pequeña y el nivel 13 a la mayor escala disponible en el sistema.

Campo	Descripción	Par. servicio Cercalia
name	Filtro nombre sin especificar nivel	locn
countryId	Filtro código país	ctryc
country	Filtro nombre país	ctryn
regionId	Filtro código región	regc
region	Filtro nombre región	regn
subregionId	Filtro código subregión	subregc
subregion	Filtro nombre subregión	subregn
subOrRegion	Filtro nombre de la región o subregión	rsn
municipalityId	Filtro código municipio	munc
municipality	Filtro nombre municipio	munnc
cityId	Filtro código ciudad / población / localidad	ctc
city	Filtro nombre ciudad / población / localidad	ctn
postalCode	Filtro código postal. Necesario especificar código país	pcode
streetId	Filtro código calle	stc
street	Filtro nombre calle	stn
street2	Filtro nombre segunda calle para calcular intersección	istn
street2Id	Filtro código segunda calle para calcular intersección	istc
houseNumber	Filtro número de calle	stnum
roadName	Filtro nombre de carretera	rdn
roadId	Filtro código de carretera	rdc
km	Filtro punto kilométrico para carretera	km
fullSearch	Resolver todos los parámetros de candidatos	fullsearch
numCand	Número de candidatos a devolver por página	numcand
posCand	Posición del primer candidato a devolver, empezando por 0	poscand
num	Número máximo de resultados	
poicat	Filtro lista de categorías de puntos de interés	poicat
poild	Filtro código de punto de interés	poic
poiName	Filtro nombre de punto de interés	poiname

Los parámetros de intersección son incompatibles con el número de calle, y sólo pueden usarse cuando existe especificación de calle. Los parámetros para especificar calle y carretera son incompatibles entre si.

Si hay mas de un candidato la coordenada devuelta puede no ser la del elemento buscado. Por ejemplo si se busca un portal (calle + numero) y se nos devuelven candidatos de país, ciudad, calle, etc... las coordenadas de estos candidatos se corresponden al país, ciudad o calle respectivamente y no al del portal.

Los parámetros se pasan de esta forma:

```
var search={ countryId:null, city:null };
search.countryId = "ESP"; // Código del país
search.city = "Madrid"; // Nombre de localidad
cercalia.geocoding(search, execute);

function execute(data)
{
    // tratar información
}
```

Los datos de la información de **resultGeocoding** son los siguientes:

Respuesta	Campo	Descripción
<i>error</i>	error	Objeto de tipo Error (si no hay error contiene null)
<i>candidatos</i>	candidates	Lista de objetos de tipo Candidate
	pos	Posición del primer candidato devuelto
	numCand	Número de candidatos por página
	num	Número máximo de resultados deseados indicados en la consulta.
	total	Número total de candidatos de la búsqueda

4.5.2 *proximity(search,returnFunc)*

Realiza una petición de proximidad según los parámetros del objeto *search*. Para realizar una proximidad es necesario especificar parámetros de un referente (el centro de la búsqueda) y parámetros que especifiquen cuáles son los referidos (los buscados cerca del centro).

Para especificar el referente se usan los mismos parámetros que en un geocoding:

Campo	Descripción	Par. servicio Cercalia
name	Filtro nombre sin especificar nivel	locn
countryId	Filtro código país	ctryc
country	Filtro nombre país	ctryn
regionId	Filtro código región	regc
region	Filtro nombre región	regn
subregionId	Filtro código subregión	subregc
subregion	Filtro nombre subregión	subregn
subOrRegion	Filtro nombre de la región o subregión	rsn
municipalityId	Filtro código municipio	munc
municipality	Filtro nombre municipio	mun
cityId	Filtro código ciudad / población / localidad	ctc
city	Filtro nombre ciudad / población / localidad	ctn
postalCode	Filtro código postal. Necesario especificar código país	pcode
streetId	Filtro código calle	stc
street	Filtro nombre calle	stn
street2	Filtro nombre segunda calle para calcular intersección	istn
street2Id	Filtro código segunda calle para calcular intersección	istc
houseNumber	Filtro número de calle	stnum
roadName	Filtro nombre de carretera	rdn
roadId	Filtro código de carretera	rdc
km	Filtro punto kilométrico para carretera	km
fullSearch	Resolver todos los parámetros de candidatos	fullsearch
numCand	Número de candidatos a devolver	numcand
posCand	Posición del primer candidato a devolver, empezando por 0	poscand
num	Número máximo de resultados deseado	

una geometría:

Campo	Descripción
wkt	Geometría referente
srs	Sistema de referencia de las coordenadas (ver Anexo de códigos)

Si se utiliza como referente una geometría la proximidad sólo se puede calcular para niveles de **geoentidad** o **poi**. Por lo tanto, para indicar el tipo referido, en este caso, siempre se utilizará el campo **rqge** o **rqpoicats** respectivamente.

o bien se especifica un centro de coordenada :

Campo	Descripción
x	Coordenada X centro referente
y	Coordenada Y centro referente
srs	Sistema de referencia de las coordenadas (ver Anexo de códigos)

Para especificar los referidos se puede especificar uno de los siguientes tres tipos: geoentidad, puntos de interés o objetos móviles.

En los tres casos se puede especificar un radio de búsqueda (en distancia euclidiana) y un número máximo de resultados.

En el caso de puntos de interés se puede definir el parámetro **weight** para calcular la distancia y tiempo en coche y ordenar el resultado según el coste mínimo de tiempo (**time**), distancia (**distance**) o dinero (**money**). Por defecto se calcula el coste de ir del referente al referido si se prefiere al revés se debe especificar el parámetro **inverse** a **true**.

Tipo Referido	Campo	Descripción
<i>geoentidad</i>	rqge	Nivel de geoentidad (adr,st,ct,mun,subreg,reg,ctry)
<i>puntos interés</i>	rqpoicats	Lista categorías de puntos de interés separadas por comas
	infxml	Numérico que indica la información xml del punto de interés a devolver.
	weight	Opcional. Coste utilizado para ordenar los resultados. Valores posibles time, distance, money
	inverse	Opcional. Puede tener los valores true o false. Cuando es true el coste es de los referidos al referente.
<i>objetos móviles</i>	mos	Lista de objetos móviles de tipo Mo
<i>comunes</i>	num	Número máximo de resultados
	rad	Radio máximo de búsqueda (distancia euclidiana)

Los parámetros se pasan de la siguiente forma:

```
var search={countryId:null, city:null, num:0, rad:0, rqpoicats:null };
search.countryId="ESP"; // Código del país
search.city="Madrid"; // Nombre de localidad
search.num=5; // cinco candidatos
search.rad=1000; // radio de 1000 metros
search.rqpoicats="C001"; // Gasolineras
cercalia.proximity(search,execute);

function execute(data)
{
    // tratar información
}
```

Si el resultado de la búsqueda no tuviera candidatos o más de un candidato se llama a la función **returnFunc** indicada con la misma estructura que un **resultGeocoding**.

Los datos de retorno **resultProximity** son los siguientes:

Respuesta	Campo	Descripción
<i>error</i>	error	Objeto tipo error (si no hay error contiene null)
<i>candidatos</i>	candidates	Lista de candidatos de tipo candidate (sólo si hay más de un candidato)
	pos	Posición del primer candidato devuelto
	num	Número de resultados máximo deseado indicado en la consulta.
	numCand	Número de candidatos por página.
	total	Número total de candidatos de la búsqueda
<i>proximidad</i>	type	Tipo de proximidad pedida {adr, st, ct, mun, subreg, reg, ctry, poi}
	x	Coordenada x del referente (centro de búsqueda).
	y	Coordenada y del referente (centro de búsqueda).
	srs	Sistema de referencia usado para indicar el centro de búsqueda.
<i>proximidad POIS</i>	pois	Lista de objetos poi (Sólo en proximidad resuelta de pois)
<i>proximidad geoentidades</i>	location	Geoentidad donde se encuentra el referente
	geentities	Lista de objetos tipo geoentity (Sólo en proximidad resuelta de geoentidades)
<i>proximidad objetos móviles</i>	mos	Lista de objetos tipo mo (Sólo en proximidad resuelta de objetos móviles)

4.5.3 *getDistance (p1,p2,weight,returnFunction)*

Devuelve la distancia calculada desde el punto p1 al punto p2. Si no se indica valor para el parámetro *weight* se calcula la distancia en línea recta. Si se indica valor se calcula la distancia en coche según coste mínimo de tiempo (*weight='time'*), distancia (*weight='distance'*) o dinero (*weight='money'*).

Parámetros:

- **p1**: una coordenada: {**x**:<x> ,**y**:<y>, **srs**:<srs>}
- **p2**: una coordenada: {**x**:<x> ,**y**:<y>, **srs**:<srs>}
- **weight**: {String} Valores 'time', 'distance' o 'money'. Parámetro opcional.
- **returnFunction**: {function} Función *callback* para recoger el resultado.

Respuesta:

Respuesta	Campo	Descripción
<i>error</i>	error	Objeto tipo error (si no hay error contiene null)
<i>dist</i>		Distancia en línea recta entre los dos puntos.
<i>routedist</i>		Distancia calculada según el parámetro "weight". Si no se ha indicado valor para este parámetro, tendrá valor null.
<i>routetime</i>		Tiempo (h:m:s) calculado según el parámetro "weight". Si no se ha indicado valor para este parámetro, tendrá valor null.

Ejemplo:

```
function obtenerDistanciaDosPuntos()
{
    var p1={x:2.822084,y:41.980320,srs:'EPSG:4326'};
    var p2={x:2.169359,y:41.389963,srs:'EPSG:4326'};
    var weight='time';
    cercalia.getDistance(p1,p2 ,weight,returnGetDistance);
}

function returnGetDistance(data)
{
    if(!data.error)alert(data.dist+"\n"+data.routedist+"\n"+data.routetime);
}
```

4.5.4 geofencing (wktPol,point)

Indica si el polígono representado por la cadena en formato WKT contiene el punto indicado. Devuelve TRUE si el polígono contiene el punto indicado. FALSE en caso contrario.

Parámetros:

- **wktPoi:** {String} WKT que representa un polígono. Se supone que las coordenadas de la cadena WKT están el sistema de coordenadas de la cartografía que se está visualizando. Para transformar un WKT de un sistema de coordenadas a otro utilizar la función **transformWKT**.
- **point:** una coordenada: {x:<x> ,y:<y> ,srs:<srs>}

```
var wkt="POLYGON((220304.49101464302 5068070.470213028,222685.7397287687
5071377.760093757,226919.07077610327 5069657.969355778,227448.23715702008
5065821.513094131,226919.07077610327 5061985.056832484,222024.2817526227
5065689.221498902,220304.49101464302 5068070.470213028))";

var point={x:'2.01051045423568',y:'41.560930505782565',srs:'EPSG:4326'};

var contains=cercalia.geofencing(wkt, point);
```

4.5.5 transformWKT (wkt,srs,srsdest)

Transforma una cadena WKT representada en el sistema de coordenadas indicado en el parámetro **srs** a otra en el sistema de coordenadas indicado en el parámetro **srsdest**.

```
cercalia.transformWKT(wkt, "EPSG:54004", "EPSG:4326");
```

4.5.6 *getPoi (infoxml, poicode [, poicode [...]], execute)*

Lanza una petición para obtener la información de los POIS indicados mediante sus correspondientes códigos. Infoxml es un numérico para especificar el tipo de información xml a devolver. Por defecto utilice el 0. Una vez resuelta la información se llama a la función execute con un parámetro de tipo **resultGetPoi** que contiene el resultado de la operación.

```
cercalia.getPoi("C57240362250385", 0, execute);

function execute(getPoiResult)
{
    if (getPoiResult.error) {
        alert("Error:" + getPoiResult.error.id)
    } else {
        for (i=0; i< getPoiResult.pois.length; i++) {
            alert (getPoiResult.pois[i].name);
        }
    }
}
```

Formato de un **resultGetPoi**:

Respuesta	Campo	Descripción
<i>error</i>	error	Objeto tipo error (si no hay error contiene null)
<i>lista de POIS</i>	pois	Lista de objetos poi

4.5.7 *calculateRoute(stopList, routeParams,LineStyle, drawStopMarkers, execute)*

Calcula la ruta óptima entre una lista de paradas.

El parámetro **stopList** especifica la lista de paradas. El primer objeto de la lista especifica el origen y el último el destino. Los objetos intermedios especifican las paradas. Cada elemento de la lista puede especificarse mediante:

- un objeto **poi**.
- un objeto **geoentity**.
- un objeto **mo**.
- una coordenada: **{x:<x> ,y:<y>, srs:<srs>}**
- un código de geoentidad. Ej.: **{cityId: ESP17240300056757 }**
- un código de punto de interés: **{poild: C17240332252905}**

Los tres primeros pueden ser objetos resultado de una geocodificación o proximidad.

El parámetro routeparams especifica la lista de opciones para el cálculo de la ruta:

Campo	Descripción	
weight	Coste utilizado para el cálculo de la ruta. (Obligatorio).	
	Valor	
	Idioma	
	distance	La ruta se calcula minimizando la distancia
	time	La ruta se calcula minimizando el tiempo
money	La ruta evita pasar por peajes	
lang	Lenguaje para el report de la ruta. (Por defecto "es").	
	Valor	
	Idioma	
	ca	Catalán
	de	Alemán
	en	Inglés
	es	Español
	fr	Francés
	it	Italiano
nl	Holandés	
pt	Portugués	
mindist	Distancia mínima recorrida en metros para generar un salto entre subetapas. (Por defecto 1000) Cuanto más grande sea esta distancia más compacto será el texto del report y menos saltos de subetapa aparecerán.	
report	Indica si la respuesta contiene un report o sólo las distancias recorridas. true – Devuelve xml con la descripción de la ruta (por defecto). false – Devuelve xml sólo con las distancias y tiempos recorridos.	
reorder	Indica si se deben reordenar o no las paradas al calcular el camino óptimo. true – Las paradas se reordenan para encontrar el camino óptimo. false – Las paradas no se reordenan (por defecto).	
getpoicats	String con las categorías separadas por coma. Obtiene información de los puntos de interés de las categorías seleccionadas que se encuentran asociados a los arcos por donde discurre la ruta.	
poiwkt	Booleano que indica si se debe informar de la geometría (wkt) de los puntos de interés. Por defecto false .	
poitolerance	Valor real que indica el grado de simplificación de las geometrías de los puntos de interés. (Por defecto 100.0 metros)	
infoxml	Si se especifica con un valor numérico diferente a 0 devuelve la información XML asociada a los puntos de interés. Por defecto 0.	
toll	Parámetro para especificar si se debe devolver el coste total de los peajes de las carreteras por donde pasa la ruta calculada. true – Se devuelve la información. false – No se devuelve la información. Peajes únicamente disponibles para España, Portugal y Andorra.	
intoll	Lista de enteros o entero. Cada entero controla si la parada correspondiente puede o no encontrarse en un tramo de peaje.	

	<p>Si se especifica -1 la parada siempre se encontrará fuera de un tramo de peaje. Si se especifica un número positivo se permitirá que la parada esté en un tramo de peaje siempre y cuando no esté a más metros del número indicado.</p> <p>Si se terminan los enteros el último se usa para el resto de paradas. Por ejemplo si especificamos intoll=[-1] todas las paradas estarán fuera de tramos de peaje.</p>
net	Especifica la red para el cálculo de rutas. Sólo se debe especificar en casos especiales.
xml	Booleano que indica si se desea que se informe el xml report de la ruta. Por defecto false .
tolerance	Valor real que indica el grado de simplificación de las geometría de la ruta. (Por defecto 100.0 metros).

El parámetro **lineStyle** especifica las propiedades de la línea de ruta. Si este parámetro es **null** no se dibujará la ruta.

Propiedad	Descripción	Valor por defecto
strokeColor	Color de la línea o contorno	"#ee9900"
strokeOpacity	Opacidad de la línea o contorno	1
strokeWidth	Ancho de la línea o contorno	1

El parámetro **drawParadeMarkers** es un booleano que especifica si se debe crear un *marker* para el origen, paradas y destino de la ruta.

El parámetro **execute** define la función que se llamará una vez resuelta la ruta con un parámetro de tipo **resultCalculateRoute**.

Formato de un **resultCalculateRoute**:

Respuesta	Campo	Descripción
<i>error</i>	error	Objeto tipo error (si no hay error contiene null)
<i>ruta</i>	id	Identificador de ruta
	distance	Distancia total en metros de la ruta.
	time	Tiempo total en formato hh:mm:ss.
	weight	Peso por el que se ha minimizado la ruta.
	lang	Lenguaje del report (null si no hay report).
	mindist	Distancia mínima entre subetapas (null si no hay report)
	coste_v1	Coste peajes tipo vehículo 1 (null si no hay report)
	coste_v2	Coste peajes tipo vehículo 2 (null si no hay report)
	coste_v3	Coste peajes tipo vehículo 3 (null si no hay report)
	stoplist	Lista de paradas de tipo poi , geoentity o mo , ordenadas según la ruta.
	stoproute	Posiciones de stoplist ordenadas teniendo en cuenta la posición de la parada en la ruta (Puede cambiar si reorder=true).
	pois	Lista de objetos de tipo poi , por donde pasa la ruta.
	xml	Documento xml con el report de la ruta o null si no se ha pedido.
stages	Lista de objetos de tipo stage (null si no hay report).	

Para mostrar los elementos de **stoplist** según el orden indicado por **stoproute**:

```
for(i=0;i<data.stoproute.length;i++)
{
    .....
    parada = data.stoplist[parseInt(data.stoproute[i])];
    .....
}
```

Ejemplo: Calcular una ruta y generar el reporte.

```
function calcularRuta ()
{
    var stopList=new Array();
    stopList[0]={ctc:'ESP17240300056757'};
    stopList[1]={ctc:'ESP17240332252624'};
    stopList[2]={ctc:'ESP47240300010809'};

    var routeParams={weight:'time',tolerance:5,mindist:'0',xml:true,
        infoxml:1,fullsearch:false,lang:'es',poiwkt:false,reorder:false};
    var lineStyle={strokeWidth: 3, strokeColor: "#045FB4", fillOpacity: 0.2};
    var drawParadaMarkers=true;
    cercalia.calculateRoute(stopList,routeParams,lineStyle,
        drawParadaMarkers, mostrarReporte);
}

function mostrarReporte(data)
{
    if(data.error) {
        alert(data.error.errorText);
    } else {
        cercalia.createReport(data.xml,'div_ruta');
    }
}
```

4.5.8 createReport(xml,idElement,xsl_name)

Aplica la plantilla xsl_name (si no se especifica utiliza la definida en el config.xml) al xml que se le pasa por parámetro (resultado de la función **calculateRoute**) y el resultado se le añade al elemento con Id idElement del Html.

Ejemplo: Calcular una ruta y generar el reporte.

```
function calcularRuta ()
{
    var stopList=new Array();
    stopList[0]={ctc:'ESP17240300056757'};
    stopList[1]={ctc:'ESP17240332252624'};
    stopList[2]={ctc:'ESP47240300010809'};

    var routeParams={weight:'time',tolerance:5,mindist:'0',xml:true,
        infoxml:1,fullsearch:false,lang:'es',poiwkt:false,reorder:false};
    var lineStyle={strokeWidth: 3, strokeColor: "#045FB4", fillOpacity: 0.2};
    var drawParadaMarkers=true;
    cercalia.calculateRoute(stopList,routeParams,lineStyle,
        drawParadaMarkers, mostrarReporte);
}

function mostrarReporte(data)
```

```

{
  if(data.error) {
    alert(data.error.errorText);
  } else {
    cercalia.createReport(data.xml, 'div_ruta');
  }
}

```

4.5.9 *createMapImage(routeld,lineStyle,routeParams, poicats ,drawFeatures,execute)*

Retorna la url donde se encuentra una imagen del mapa con la extensión actual. El mapa aparece centrado respecto a la ruta indicada por el parámetro *routeld*. El color (en formato hexadecimal) y ancho de la línea de la ruta serán los indicados en el parámetro *lineStyle(*)*.

En el parámetro *routeParams* deben aparecer indicados el atributo “weight” y el atributo “tolerance” con los que deseamos obtener la ruta.

El parámetro *drawFeatures* a “true” indica si la imagen debe contener la figuras y/o iconos dibujados en el mapa en la extensión que se está visualizando.

El parámetro *execute* corresponde a la función que se ejecutará una vez se obtenga la imagen. Dicha función recibe como parámetro la url de la imagen resultante.

Si se recibe un valor de *routeld* nulo se calcula la imagen del mapa con la extensión actual y sin ruta. Los parámetros *lineStyle* y *routeParams* serán omitidos.

Si se da valor al parámetro *poicats* (cadena de categorías de POIs separadas por comas), en la imagen del mapa aparecerán los puntos de interés correspondientes a las categorías indicadas. Éste parámetro se puede omitir.

(*)El servidor de mapas aplica una capa de transparencia por lo que se recomienda aumentar la intensidad del color. También puede ser recomendable indicar un ancho de línea superior a 4 píxeles.

Ver definiciones de *lineStyle* y *routeParams* en la descripción de la función *calculateRoute*.

Ejemplo: Calcular una ruta y abrir una nueva ventana con la imagen generada.

```

function calcularRuta ()
{
  var stopList=new Array();
  stopList[0]={ctc:'ESP17240300056757'};
  stopList[1]={ctc:'ESP17240332252624'};
  stopList[2]={ctc:'ESP47240300010809'};

  var routeParams={weight:'time',tolerance:5,mindist:'0',xml:true,
    infoxml:1,fullsearch:false,lang:'es',poiwkt:false,reorder:false};
  var lineStyle={strokeWidth: 3, strokeColor: "#045FB4", fillOpacity: 0.2};
  var drawParadaMarkers=true;
  cercalia.calculateRoute(stopList,routeParams,lineStyle,
    drawParadaMarkers, mostrarImagen);
}

function mostrarImagen(data)
{
  var routeParams={weight:'time',tolerance:5 };
  var lineStyle={strokeWidth: 5, strokeColor: "#045FB4"};

```

```

        cercalia.createMapImage(data.ruta.id,lineStyle,routeParams,null,false,
                               verRuta);
    }

    function verRuta(url)
    {
        var w =window.open( url,
                            "Imprimir",
                            "width=800,height=620,scrollbars=yes,resizable=yes,
                            toolbar=no,location=no,directories=no,titlebar=yes,
                            status=no,menubar=no");
    }

```

4.5.10 *calculateMultiRoute(stopList, routeParams, execute)*

Calcula las rutas óptimas especificadas por el parámetro `stopList` y devuelve los distintos costes (distancia, tiempo, dinero).

Puede utilizarse para calcular la distancia y tiempo entre *n* puntos de origen, varios puntos de paso, y *n* destinos. De esta forma, por ejemplo podemos resolver preguntas como cuál es el vehículo de una flota propia que tardará menos en ir a recoger una mercancía y llevarla a su destinación, así como obtener el tiempo y la distancia calculados.

El parámetro **stopList** especifica la lista de paradas. El primer objeto de la lista especifica el origen y el último el destino. Los objetos intermedios especifican las paradas. Cada elemento de la lista puede especificarse mediante:

- un objeto **poi**.
- un objeto **geoentity**.
- un objeto **mo**.
- una lista de objetos **mo**.
- una coordenada: **{x:<x> ,y:<y> ,srs:<srs>}**
- una lista de coordenadas: **[[{x:<x> ,y:<y> ,srs:<srs>},{x:<x> ,y:<y> ,srs:<srs>}]]**
- un código de geoentidad. Ej.: **{cityId: ESP17240300056757 }**
- un código de punto de interés: **{poild: C17240332252905}**

Los tres primeros pueden ser objetos resultado de una geocodificación o proximidad.

El parámetro **routeParams** especifica la lista de opciones para el cálculo de la ruta:

Campo	Descripción
-------	-------------

weight	Lista de costes utilizados para el cálculo de la ruta. El primero de ellos especifica el coste a minimizar del origen a la primera parada, el segundo especifica el coste a minimizar de la primera parada a la segunda parada y así sucesivamente. Si no hay suficientes valores el último valor se repite para el resto de paradas. También se puede especifica un solo valor y este se aplicará a todas las etapas.	
	Valor	Idioma
	distance	La ruta se calcula minimizando la distancia
	time	La ruta se calcula minimizando el tiempo
	money	La ruta evita pasar por peajes
intoll	Lista de enteros o entero. Cada entero controla si la parada correspondiente puede o no encontrarse en un tramo de peaje. Si se especifica -1 la parada siempre se encontrará fuera de un tramo de peaje. Si se especifica un número positivo se permitirá que la parada esté en un tramo de peaje siempre y cuando no esté a más metros del número indicado. Si se terminan los enteros el último se usa para el resto de paradas. Por ejemplo si especificamos intoll=[-1] todas las paradas estarán fuera de tramos de peaje.	
net	Especifica la red para el cálculo de rutas. Sólo se debe especificar en casos especiales.	
xml	Booleano que indica si se desea el xml de respuesta. Por defecto false .	

El parámetro **execute** define la función que se llamará una vez resuelta la ruta con un parámetro de tipo **resultCalculateRouteCosts**.

Formato de un **resultCalculateRouteCosts**:

Respuesta	Campo	Descripción
<i>error</i>	error	Objeto tipo error (si no hay error contiene null)
<i>rutas</i>	routes	Lista de objetos route
	xml	Documento xml con el report de la ruta o null si no se ha pedido.

Formato de un **route**:

Campo	Descripción
id	Identificador de ruta
distance	Distancia total en metros de la ruta.
time	Tiempo total en formato hh:mm:ss.
weight	Lista de pesos o peso por el que se ha minimizado la ruta.
stoplist	Lista de paradas de tipo poi , geoentity o mo ordenadas según la ruta.

Ejemplo:

```

function calcularCostesPosiblesRutas()
{
    var start={x:'241628',y:'5041216',srs:'EPSG:54004'};
    var destinos_posibles=new Array();

    destinos_posibles[0]={x:'2.8260998838408864',y:'41.98895955699918',srs:'EPSG:
4326', id:'destino1'};
    destinos_posibles[1]={x:'-97724',y:'5081698',id:'destino2'};

    var stopList = new Array();

    stopList[0] = { x: start.x, y: start.y, srs: start.srs };
    stopList[1] = destinos_posibles;

    var routeParams = { weight: 'distance', tolerance: 5, mindist: '0', xml:
true, infoxml: 1, fullsearch: false, lang: 'es', poiwkt: false, reorder:
false };

    cercalia.calculateMultiRoute(stopList, routeParams, retornoCalcularRuta);
}

function retornoCalcularRuta(data)
{
    var i=0;
    for(i=0;i<data.routes.length;i++)
        alert("Distancia ruta "+(i+1)+"-"+data.routes[i].distance);
}

```

4.5.11 *transformCoordinates (objlist,srs)*

Esta función transforma las coordenadas contenidas en un objeto o lista de objetos de tipo **candidate**, **poi** o **geoentity** al sistema de referencia indicado

```
cercalia.transformCoordinates(resultGeocoding.candidates,"EPSG:4326");
```

4.5.12 *coordToPixel(obj)*

Devuelve el píxel de la imagen que corresponde a las coordenadas indicadas. Se deben indicar los tres atributos del parámetro (srs, x e y).

Parámetro:

obj - {Object} Objeto con la estructura {x:...,y:...,srs:...}

Ejemplo:

```

var obj={x:-3.808951, y:43.458308, srs:"EPSG:4326"};
var pixel=cercalia.coordToPixel(obj);

```

4.5.13 *coordToPixel(obj,srs)*

Devuelve el píxel de la imagen que corresponde a las coordenadas indicadas. Es obligatorio indicar el sistema de coordenadas utilizado. De lo contrario no se obtendrá resultado.

Parameters:

obj - {Object} Objecto del tipo OpenLayers.LonLat (ver: <http://www.openlayers.org>)

srs - {String} Sistema de coordenadas utilizado

Ejemplo:

```
var obj= new OpenLayers.LonLat(-3.808951, 43.458308);  
var pixel=cercalia.coordToPixel(obj,"EPSG:4326");
```

5 Estilo del mapa

Los servidores Cercalia proporcionan distintos estilos de cartografía base para los mapas. Consulte con Nexus Geografics para resolver sus necesidades.

5.1 `getStyle()`

Devuelve el nombre del estilo de cartografía de fondo.

5.2 `setStyle(name)`

Aplica el estilo indicado para la cartografía de fondo.

- Aplicar estilo por defecto de *Cercalia*: `setStyle("default");`
- Aplicar estilo por defecto de *OpenStreetMap*: `setStyle("openstreetmap");`

5.3 `createWMSStyle (name,url,params,options)`

Crema un nuevo estilo de cartografía de fondo con los datos de un servidor WMS.

Parámetro	Descripción
name	Nombre del nuevo estilo de cartografía.
url	URL del servidor WMS
params	Objeto con pares clave/valor con los parámetros necesarios para realizar una consulta GETMAP al servidor WMS. Si no se especifican algunos tienen asociado un valor por defecto: service: "WMS" version: "1.1.1" request: "GetMap" styles: "" exceptions: "application/vnd.ogc.se_inimage" format: "image/jpeg" Son obligatorios los siguientes: srs layers
options	Tabla hash con opciones extra para configurar la capa. opacity: Número real entre 0.0 y 1.0 que especifica la opacidad de la capa. Por defecto: opaca (1.0) encodeBBOX: true para que se codifique el valor parámetro BBOX (el estándar WMS advierte que no se debe codificar). Por defecto: false. gutter: Número de píxeles alrededor de la imagen que se ignoran. Por defecto 0. scales: Lista de escalas válidas en orden descendiente para el

	<p>servidor WMS. Por defecto :null (se permite cualquier escala)</p> <p>resolutions: Lista de resoluciones (unidades de mapa por píxel) en orden descendiente. Si no se especifica se calculará en función de la extensión, máxima resolución, máxima escala, etc ...</p> <p>maxResolution: Real que indica la resolución máxima. Por defecto 360 deg / 256 px.</p> <p>minResolution: Real que indica la resolución mínima.</p> <p>numZoomLevels: Entero que indica el número de niveles de zoom.</p> <p>minscale: Real que indica el mínimo denominador de escala.</p> <p>maxscale: Real que indica el máximo denominador de escala.</p> <p>displayOutsideMaxExtent: Booleano que indica si se debe pedir el mapa aunque se esté completamente fuera de la extensión. Por defecto: false</p> <p>maxExtent: Array con cuatro valores que representan la extensión máxima de la cartografía. El centro del mapa nunca estará fuera de este rectángulo. Además si displayOutsideMaxExtent está activado no se pedirán mapas fuera de esta extensión.</p> <p>minExtent: Array con cuatro valores que representan la extensión mínima de la cartografía, en la proyección del servidor WMS.</p> <p>displayName: Nombre que aparecerá en el desplegable para escoger el tipo de mapa visible.</p> <p>Es necesario indicar alguna de las opciones de escala, preferiblemente <i>scales</i>.</p>
--	---

Ejemplo para obtener la Orto 50m del ICC :

```

var params = {'layers':'mtc50m,', 'format':'image/jpeg',
              'exceptions':'application/vnd.ogc.se_xml', 'srs':"EPSG:23031"};
var options = {'opacity':1};
cercalia.createWMSStyle("wmsICC",
    "http://shagrat.icc.es/lizardtech/iserv/ows?VERSION=1.1.0&service=WMS",
    params,options);
cercalia.setStyle("wmsICC");

```

6 Anexo: Geocodificación sin mapa

Existe la posibilidad de realizar geocodificaciones sin tener que inicializar el mapa.

Para ello se tiene que cargar el fichero 'cercaliageo.js' en la página que queramos usar la funcionalidad con la siguiente instrucción:

```
<script type='text/javascript' src='cercaliageo.js'></script>
```

AVISO: La página con esta referencia sólo podrá usar la función de geocoding de la API. El resto de funciones devolverán error.

Antes de usar el geocoding debemos inicializar la API. La inicialización consiste en leer un fichero de propiedades y llamar al constructor de **cercaliacient** pasándole por parámetro el objeto con las propiedades.

```
properties=new cercaliaProperties("cercalia/config.xml");
cercalia = new cercaliaClient(properties);
```

En este punto ya podemos llamar la función de geocoding:

```
var search={poicat:'C008',city:'Barcelona',countryId:'ESP',poiName:'Citroën',num:10,rad:2000};
clone(search,ultimaGeo);
cercalia.geocoding(search,retornoGeocoding);
```

Ejemplo:

```
<script type='text/javascript' src='cercaliageo.js'></script>
```

```
function createAll()
{
```

```
    properties=new cercaliaProperties("cercalia/config.xml");
    cercalia = new cercaliaClient(properties);
    geocodingPois();
}
```

```
function geocodingPois()
{
```

```
    var search={poicat:'C008',city:'Barcelona',countryId:'ESP',poiName:'Citroën',num:10,rad:2000};
    clone(search,ultimaGeo);
    cercalia.geocoding(search, returnGeocoding);
}
```

```
function returnGeocoding(dades)
{
```

```
    var cont='';
    document.getElementById('emergent').style.top='-2000';
```

```
    if(dades.error)alert("Error: "+dades.error.errorText);
```

```
    else
    {
```

```
        if(dades.candidates!=null && dades.candidates.length>=1)
```

```
        {
```

```
            var i=0;
            var body_candidates=document.getElementById('body_candidates');
            var tr=null;
            var td=null;
            var a=null;
```


7 Anexo: Sistemas de coordenadas

Valores posibles sistema coordenadas (parámetro srs):

Valor	Sistema de coordenadas
EPSG:4326	WGS 84/ LonLat
EPSG:23030	ED50 /UTM zone 30N
EPSG:23031	ED50 /UTM zone 31N
EPSG:23032	ED50 /UTM zone 32N
EPSG:23033	ED50 /UTM zone 33N
EPSG:32627	WGS 84 / UTM zone 27N
EPSG:32628	WGS 84 / UTM zone 28N
EPSG:32629	WGS 84 / UTM zone 29N
EPSG:32630	WGS 84 / UTM zone 30N
EPSG:32631	WGS 84 / UTM zone 31N
EPSG:32632	WGS 84 / UTM zone 32N
EPSG:32633	WGS 84 / UTM zone 33N
EPSG:32634	WGS 84 / UTM zone 34N
EPSG:32635	WGS 84 / UTM zone 35N
EPSG:32636	WGS 84 / UTM zone 36N
EPSG:32637	WGS 84 / UTM zone 37N
EPSG:54004	WGS 84 / World Mercator
EPSG:102014	Europe Lambert Conformal Conic
EPSG:900913	Google

8 Anexo: Botones customclick 1/2

customClick<n> son botones de la barra de herramientas pensados para que cuando estén seleccionados y se pincha un punto del mapa se realice una acción personalizada.

Para su correcto funcionamiento se debe definir en el fichero de configuración y en el mismo contexto donde se declara el objeto cercalia se debe declarar la función menuCustomClick{n}, que se ejecutará cuando se pinche el mapa. Esta función recibirá un objeto de tipo OpenLayers.Event que entre otra información contiene el píxel del mapa donde se ha pinchado.

Consulte la API de documentación de OpenLayers para ver la especificaciones concretas de los objetos OpenLayers.Event (<http://www.openlayers.org>)

8.1.1 getCoordFromPixel(pixel,srs)

Esta función está diseñada para llamarse dentro de la función menuCustomclick<n>. Devuelve un objeto OpenLayers.LonLat que contiene las coordenadas en el sistema de referencia indicado por el parámetro **srs** del píxel indicado en el parámetro **píxel**. El parámetro píxel debe ser un parámetro de tipo OpenLayers.Pixel. Vea ejemplos de uso en la sección anterior.


Ejemplo: Lanzar un mensaje con las coordenadas geográficas del punto pinchado:

```
function menuCustomClick1(e) {
    alert("coord EPSG:4326 "
        +cercalia.getCoordFromPixel(e.xy,"EPSG:4326"));
}
```

Ejemplo: Lanzar un mensaje con las coordenadas Mercator del punto pinchado:

```
function menuCustomClick1(e) {
    alert("coord EPSG:54004 "
        +cercalia.getCoordFromPixel(e.xy,"EPSG:54004"));
}
```

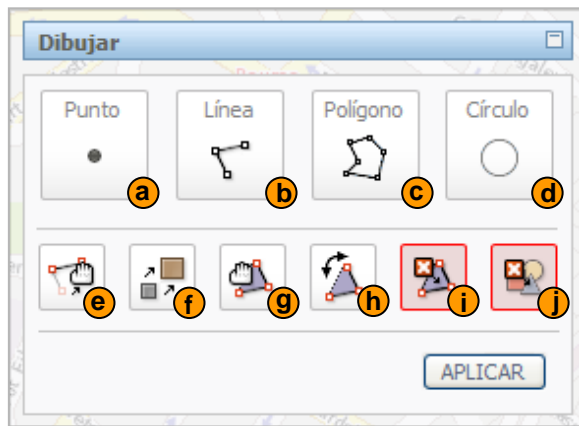
9 Anexo: Herramientas de edición

En este anexo se explica el funcionamiento de la herramienta  (“draw”) así como su interacción con los métodos para el tratamiento de formas.

En general, esta herramienta permite dibujar formas sobre el mapa con un estilo personalizado (colores, estilo línea, ...). Una vez creadas estas formas se puede consultar su información a través de los métodos *getFeatures* y *getFeaturesByState* descritos en este manual.

Por otro lado, también permite editar y por tanto, modificar (editar vértices, rotar, mover y eliminar) las formas creadas en el mapa a partir de la función *createFeature*.

Veamos a partir de la imagen siguiente el aspecto del menú de utilidades que se despliega al activar la herramienta:

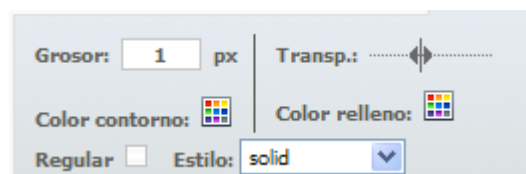


- a) Dibujar puntos
- b) Dibujar líneas
- c) Dibujar polígonos
- d) Dibujar círculos
- e) Editar vértices
- f) Escalar forma
- g) Mover forma
- h) Rotar forma
- i) Eliminar forma
- j) Eliminar todas las formas

El funcionamiento es bastante intuitivo. Cada vez que se hace clic en uno de los botones se activa su utilidad. Por lo tanto, cuando el mapa reciba eventos desde el *mouse* se ejecutará una acción u otra según el botón que haya activo.

En el caso de la opción “j” la utilidad no queda activa sino que simplemente se procede a eliminar todas las formas creadas en el mapa (con la herramienta y / o con la función *createFeature*).

Al activarse una de las utilidades *a*, *b*, *c* ó *d*, se muestran opciones de personalización del estilo de las formas. Por ejemplo en el caso *d*:



Para que se apliquen estas opciones, a la hora de dibujar las figuras en el mapa, se debe hacer clic en el botón “APLICAR” una vez seleccionada la configuración deseada.

a) Dibujar puntos

Esta utilidad permite dibujar puntos de un tamaño y color personalizados.

b) Dibujar líneas

Esta utilidad permite dibujar líneas de un grosor, color y estilo (continua, discontinua de puntos, ...) personalizados.

c) Dibujar polígonos

Esta utilidad permite dibujar polígonos con color y grosor de línea, color de fondo, transparencia y estilo de línea personalizados.

d) Dibujar círculos

Esta utilidad permite dibujar círculos regulares e irregulares con color y grosor de línea, color de fondo, transparencia y estilo de línea personalizados.

e) Editar vértices

Esta utilidad permite editar los vértices de nuestras figuras para poder modificar su forma.

Si la figura ha sido creada mediante la función *createFeature* su estado pasa a ser 'modified'.

f) Escalar forma

Esta utilidad permite modificar el tamaño de nuestras figuras.

Si la figura ha sido creada mediante la función *createFeature* su estado pasa a ser 'modified'.

g) Mover forma

Esta utilidad permite desplazar nuestras figuras en el mapa.

Si la figura ha sido creada mediante la función *createFeature* su estado pasa a ser 'modified'.

h) Rotar forma

Esta utilidad permite rotar nuestras figuras.

Si la figura ha sido creada mediante la función *createFeature* su estado pasa a ser 'modified'.

i) Eliminar forma

Esta utilidad permite eliminar una o varias formas seleccionadas, tanto las creadas con las utilidades *a, b, c* y *d* como las resultantes de la función *createFeature*. Muestra un mensaje de confirmación antes de realizar la acción.

Las figuras eliminadas que habían sido creadas mediante la función *createFeature*, se guardan en una estructura interna y su información geométrica puede ser recuperada mediante el método *getFeatureByState* con el parámetro 'deleted' a cierto y los demás a falso.

j) Eliminar todas las formas

Esta utilidad permite eliminar todas las figuras presentes en el mapa (las creadas con las utilidades *a, b, c* y *d* y las resultantes de la función *createFeature*). Muestra un mensaje de confirmación antes de realizar la acción.

Las figuras eliminadas que habían sido creadas mediante la función *createFeature*, se guardan en una estructura interna y su información geométrica puede ser recuperada mediante el método *getFeatureByState* con el parámetro *'deleted'* a cierto y los demás a falso.

Si deseamos eliminar las figuras de manera permanente debemos utilizar la función *deleteFeatures* explicada en el manual.

9.1 Personalización del menú

Si no usamos menú de herramientas o queremos hacer un menú personalizado podemos activar las herramientas mediante código *Javascript*. A continuación se explican las funciones que nos permiten utilizar los controles para crear, modificar i eliminar formas en el mapa sin utilizar el menú.

9.1.1 Función *activateDrawControl(tool)*

Esta función permite activar el control de dibujo que se indica por parámetro. Los valores que admite como parámetro son:

- ***point***: Activa la herramienta para dibujar puntos en el mapa.
- ***line***: Activa la herramienta para dibujar líneas en el mapa.
- ***polygon***: Activa la herramienta para dibujar polígonos en el mapa.
- ***circle***: Activa la herramienta para dibujar círculos en el mapa.
- ***modify***: Activa la herramienta para editar los vértices de las figuras que seleccionemos posteriormente.
- ***resize***: Activa la herramienta para modificar el tamaño de las figuras que seleccionemos posteriormente.
- ***drag***: Activa la herramienta para modificar la posición de las figuras que seleccionemos posteriormente.
- ***rotate***: Activa la herramienta para rotar la posición de las figuras que seleccionemos posteriormente.
- ***deleteFeature***: Activa la herramienta para eliminar las figuras que seleccionemos posteriormente. Pide confirmación antes de eliminarlas.
- ***deleteAll***: Ejecuta la acción de eliminar todas las figuras existentes en el mapa. Pide confirmación antes de eliminarlas.

Para desactivar cualquier control usar la instrucción *activateTool('pan')*.

A continuación veremos dos funciones que sirven para modificar el estilo de las figuras que se dibujarán posteriormente.

9.1.2 Función *applyStyleControlDraw(type, style)*

Ésta función permite definir el estilo que tendrán las formas que se dibujen posteriormente. Se puede definir un estilo en general indicando *type='all'* o definir un estilo para cada tipo de figura:

- *point*
- *line*
- *circle*
- *polygon*

Ejemplos:

Definir estilo para sólo para los polígonos:

```
cercalia.applyStyleControlDraw('polygon',{strokeColor:'red',strokeWidth:3,fillColor:'red',strokeDashstyle:'dot'})
```

Definir estilo para cualquier tipo de figura:

```
cercalia.applyStyleControlDraw('all',{strokeColor:'red',strokeWidth:3,fillColor:'red',strokeDashstyle:'dot'})
```

9.1.3 Función *setRegularCircleDrawControl(isRegular)*

Ésta función permite definir el tipo de círculos que se dibujarán posteriormente cuando la herramienta de dibujo de círculos esté activa. Si queremos dibujar círculos regulares se debe indicar *isRegular=true*. Indicar *isRegular=false* para dibujar círculos no regulares.

9.1.4 Detectar creación formas

Podemos definir una función que se ejecutará cuando el usuario crea alguna forma a partir de las herramientas de edición. Esta función recibe un objeto *Feature* con las características del que acabamos de crear (id, wkt, style,...). Para definir qué función queremos ejecutar cada vez que se cree una forma simplemente hay que definir la propiedad ***drawToolsFeatureAdded*** del objeto *cercaliaProperties*.

Ejemplo:

```
function createAll(){
  properties=new cercaliaProperties("cercalia/config.xml");
  properties.drawToolsFeatureAdded=nuevaFormaDibujada;
  cercalia = new cercaliaClient(properties,execute,'demo');
}
function nuevaFormaDibujada(feature)
{
  //Su código aquí
}
```

9.1.5 Ejemplo integrado en el código HTML

```
<p>Herramientas dibujo</p>
<input type="button" onclick="javascript:cercalia.activateDrawControl('point');"
value="POINT" style="width:90px;" /><br/>
<input type="button" onclick="javascript:cercalia.activateDrawControl('line');"
value="LINE" style="width:90px;" /><br/>
<input type="button" onclick="javascript:cercalia.activateDrawControl('polygon');"
value="POLYGON" style="width:90px;" /><br/>
<input type="button" onclick="javascript:cercalia.activateDrawControl('circle');"
value="CIRCLE" style="width:90px;" /><input type="checkbox" value="S" id="regular"
name="regular" onchange="cercalia.setRegularCircleDrawControl(this.checked)"
/>&nbsp;&nbsp;&nbsp;Regular<br/>
<input type="button"
onclick="javascript:cercalia.applyStyleControlDraw('polygon',{strokeColor:'red',strokeWidth:3,fillColor:'red',strokeDashstyle:'dot'})" value="ESTILO POL."
style="width:90px" /> Definir estilo polígonos<br />

<p>Herramientas edición</p>
<input type="button" onclick="javascript:cercalia.activateDrawControl('modify');"
value="MODIFY" style="width:90px;" /><br/>
<input type="button" onclick="javascript:cercalia.activateDrawControl('resize');"
value="RESIZE" style="width:90px;" /><br/>
<input type="button" onclick="javascript:cercalia.activateDrawControl('drag');"
value="DRAG" style="width:90px;" /><br/>
<input type="button" onclick="javascript:cercalia.activateDrawControl('rotate');"
value="ROTATE" style="width:90px;" /><br/>
<input type="button"
onclick="javascript:cercalia.activateDrawControl('deleteFeature');" value="DELETE"
style="width:90px;" /><br/>
<input type="button"
onclick="javascript:cercalia.activateDrawControl('deleteAll');" value="DELETE ALL"
style="width:90px;" /><br/>
```

9.2 Herramienta SelectFeatureCustom

Esta herramienta sirve para detectar formas creadas en el mapa (mediante herramientas de edición o a partir de la función `createFeature`). Una vez seleccionada la figura se ejecuta la función que el programador ha definido. Esta función recibe como parámetro el objeto *Feature* que representa la figura seleccionada.

9.2.1 Función `activateSelectFeatureCustom(callback)`

Esta función sirve para activar la herramienta. Mientras esta herramienta está activa se pueden seleccionar las formas visibles en el mapa con el clic izquierdo o derecho del *mouse*.

Ejemplo:

```
(...)
cercalia.activateSelectFeatureCustom(selectCallback);
(...)

function selectCallback(f)
{
    //Su código aquí
}
```

9.2.2 Función `deactivateSelectFeatureCustom(callback)`

Esta función sirve para desactivar la herramienta de seleccionar formas en el mapa.

Ejemplo:

```
cercalia.deactivateSelectFeatureCustom();
```

Ejemplo 2:

```
function selectFeaturesCallback(feature)
{
    //Su código aquí
    (.....)

    cercalia.deactivateSelectFeatureCustom(); // ----> Una vez finalizado el código
                                              podemos desactivar la herramienta
}
```

10 Anexo: Diferencias entre las versiones 1 y 2

En este anexo se resumen los cambios realizados respecto la primera versión de la API y se explican algunos pasos para poder migrar los proyectos realizados con la primera versión.

- Se ha incorporado tolerancia a servidores caídos o inaccesibles. Ahora el script de inicialización detecta si el servidor al que se conecta está caído y lo prueba con otro.
- Se ha añadido la posibilidad de que el cliente se comunice a un servlet proxy. El proxy contiene la clave de distribuidor por lo que si se utiliza esta opción no hace falta el control de acceso por dominio en el servidor Cercalia.
- Se ha simplificado el código inicialización del objeto Cercalia. Ya no hace falta definir los div's **barralat**, **foramapa**, **mapa** la API los construirá automáticamente dentro del div que se le pase por parámetro al constructor.
- Se ha modificado el formato del fichero de configuración XML para que tenga identificadores más claros y se han añadido nuevas opciones. La siguiente tabla muestra las equivalencias entre parámetros de la primera versión y la nueva. Vea el capítulo donde se explica el formato del fichero de configuración para ver los valores válidos de cada parámetro.

Versión Anterior (1.x)	Versión actual (3.x)
client	client
initial_zoom	map_zoom_level
menu_options	panel_pages
buscar	geocoding
proximidad	proximity
rutas	route
menu_initial	panel_pages
barralat_visible	panel_status
barralat_maxim	panel_status
zoombar_visible	zoombar_status
overview_map_maxim	overview_map_status
panel_options	toolbar_tools
language	language
cart_layer	map_style
zoombar_small	zoombar_status
overview_map_visible	overview_map_status
extent	map_extent

- Se ha sustituido la función **gotoPoiCode** por la función **getPoi**. Esta función devuelve todos los datos de uno o más puntos de interés pero a diferencia de la anterior, no crea los marcadores ni centra el mapa a ellos.
- Se ha sustituido la función **pois** por la función **showPois**. Esta función a diferencia de la anterior se le pasan sólo las categorías de puntos de interés

que deseamos cambiar de estado. Las categorías que no se especifiquen mantendrán su estado anterior.

- Se ha añadido la función **activateTool** para poder seleccionar por código la herramienta activa.
- Se ha añadido el método **transformCoordinates** que transforma las coordenadas de una lista de candidatos, puntos de interés o geentidades al sistema de coordenadas que se especifique.
- Se ha añadido el método **createMarkers** que permite crear los marcadores de una respuesta de geocoding, proximidad, etc.
- Se ha modificado el método **moveMarker** añadiendo un parámetro opcional para cambiar el contenido del *popup* del marcador de forma dinámica. Ahora la función **moveMarker** nos puede servir para refrescar el estado de un marcador de forma periódica y dinámica sin que se produzcan pérdidas de memoria del navegador.
- Los métodos que aparecen en la API actual permiten todas las funcionalidades de la versión anterior. Las funciones que no aparecen en el manual han sido eliminadas o han dejado de funcionar para el cliente nuevo. Por ejemplo la función *proximity* indicando coordenadas, se puede utilizar en lugar de *inverseGeocoding*.