

Manuel API JavaScript CERCALIA

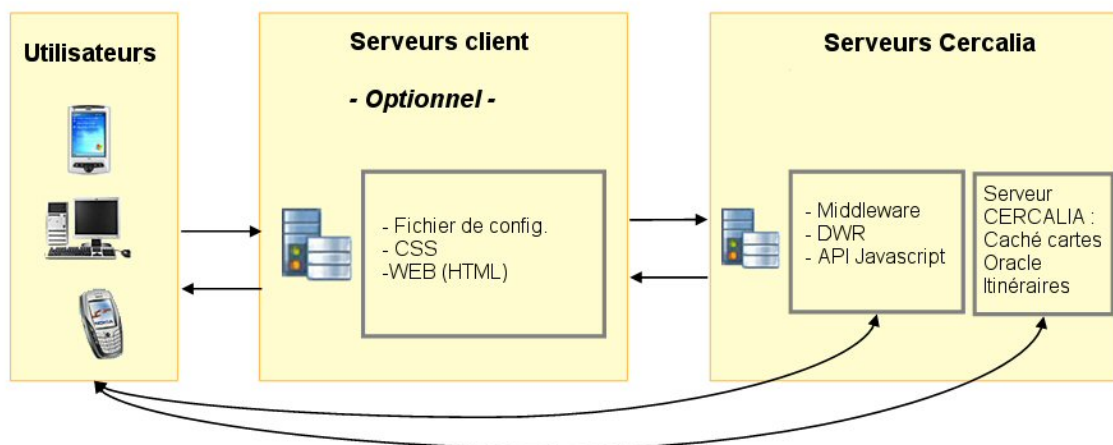
Entreprise : Nexus Geografics, S.L.
Dernière date de modification : 27/5/2010
Document : manuel_API_CERCALIAv2.9.5.doc

1 Table des matières

1	Table des matières.....	2
2	Architecture.....	4
2.1	Installation.....	5
2.1.1	Avec validation par domaine	5
2.1.2	Avec validation par proxy	5
3	Intégration et personnalisation de l'interface	6
3.1	Exemple introductif et intégration	6
3.2	Personnalisation du style et des images	7
3.3	Le fichier de configuration	8
4	La classe <code>cercaliaClient</code>	11
4.1	Structures ou objets utilisés par les méthodes de cette classe.....	11
4.1.1	Attributs d'une erreur (<code>Error</code>).....	11
4.1.2	Attributs d'un candidat (<code>Candidate</code>)	11
4.1.3	Attributs d'une entité géographique (<code>Geoentity</code>).....	11
4.1.4	Attributs d'un point d'intérêt (<code>POI</code>).....	12
4.1.5	Attributs de proximité (<code>Prox</code>).....	12
4.1.6	Attributs d'arc (<code>Arc</code>)	13
4.1.7	Attributs d'un objet mobile (<code>Mo</code>).....	13
4.1.8	Attributs d'un objet de style repère (<code>markerStyle</code>)	13
4.1.9	Attributs d'un objet de forme (<code>feature</code>).....	14
4.2	Méthodes pour le traitement des repères.....	14
4.2.1	<code>createMarker(id,x,y,srs,style,id2)</code>	14
4.2.2	<code>createMarkers (obj,style)</code>	15
4.2.3	<code>moveMarker(id,x,y,srs,cont, contOver)</code>	17
4.2.4	<code>setMouseoverContent (id,contOver)</code>	17
4.2.5	<code>setMarkerIcon(id,url)</code>	18
4.2.6	<code>deleteMarkers([id1 [, id2 [...]]])</code>	18
4.2.7	<code>centerToMarkers(changeScale [,id1 [, id2 [...]]])</code>	18
4.2.8	<code>showMarkers(state [,id1 [, id2 [...]]])</code>	19
4.2.9	<code>showLabelMarkers (visible)</code>	19
4.3	Méthodes pour le traitement des formes.....	19
4.3.1	<code>createFeature(wkt,srs,style [, editstyle])</code>	19
4.3.2	<code>createFeature(gml,style [, editstyle])</code>	21
4.3.3	<code>centerToFeatures(changeScale [,feature1 [, feature2 [...]]])</code>	21
4.3.4	<code>deleteFeatures([feature1 [, feature2 [...]]])</code>	22
4.3.5	<code>getFeaturesByState(created, modified, deleted, unchanged[,SRS])</code>	22
4.3.6	<code>getFeaturesByStateAsGML(created, modified, deleted, unchanged[,SRS])</code>	23
4.3.7	<code>getFeatures([feature1 [, feature2 [...]][,SRS])</code>	23
4.3.8	<code>getFeaturesAsGML([feature1 [, feature2 [...]][,SRS])</code>	23
4.4	Méthodes de traitement de la carte.....	24
4.4.1	<code>activateTool(tool)</code>	24
4.4.2	<code>isCompleted()</code>	24
4.4.3	<code>showPois(visible,category[[,category],...])</code>	24
4.4.4	<code>setLanguage(lang)</code>	24
4.4.5	<code>getMap()</code>	24
4.4.6	<code>deleteRoute (resultCalculateRoute)</code>	25
4.4.7	<code>setParse< operation> (function)</code>	25
4.4.8	<code>showResult< operation> (obj)</code>	26
4.4.9	<code>getExtent(srs)</code>	27
4.4.10	<code>getCenter(srs)</code>	27
4.4.11	<code>setCenter(center,scale)</code>	28
4.4.12	<code>setCenter(center,srs,scale)</code>	28
4.4.13	<code>zoomToExtent(extent,srs)</code>	29
4.4.14	<code>getScale(closest)</code>	29
4.4.15	<code>getCurrentSRS()</code>	29
4.5	Méthodes d'obtention d'informations.....	29
4.5.1	<code>geocoding(search,returnFunc)</code>	29
4.5.2	<code>proximity(search,returnFunc)</code>	31

4.5.3	getDistance (p1,p2,weight,returnFunction)	33
4.5.4	getPoi (infoxml, poicode [, poicode [...]], execute)	34
4.5.5	calculateRoute(stopList, routeParams, lineStyle, drawStopMarkers, execute)	35
4.5.6	createReport(xml,idElement,xsl_name)	38
4.5.7	createMapImage(routeId,lineStyle,routeParams, poicats, drawFeatures, execute)	38
4.5.8	calculateMultiRoute(stopList, routeParams, execute)	39
4.5.9	transformCoordinates (objlist,srs).....	41
4.5.10	coordToPixel(obj).....	42
4.5.11	coordToPixel(obj,srs).....	42
5	Style de la carte	43
5.1	getStyle()	43
5.2	setStyle(name).....	43
5.3	createWMSStyle (name,url,params,options).....	43
6	Annexe : géocodage sans carte	45
7	Annexe : systèmes de coordonnées	47
8	Annexe : boutons customclick 1/2.....	48
8.1.1	getCoordFromPixel(pixel,srs)	48
9	Annexe : outils d'édition.....	49
10	Annexe : différences entre les versions 1 et 2	52

2 Architecture



Les serveurs de Cercalia fournissent une API JavaScript qui permet d'intégrer des cartes et autres fonctionnalités dans vos sites web d'une manière simple et rapide.

Pour contrôler l'accès à l'API et aux services fournis par les serveurs Cercalia, deux mécanismes de sécurité ont été implantés selon les besoins de notre distributeur ou client.

- **Validation par domaine**

Chaque distributeur possède un ou plusieurs points d'accès. En accédant à l'un d'eux, il faut vérifier que l'accès s'est fait par le biais d'une page gérée par un domaine ou un serveur de notre distributeur. Plus précisément, il faut s'assurer que le référent de la requête appartient bien à un domaine ou un serveur de notre distributeur. Le distributeur doit fournir une liste de domaines et de ports valides.

Cette solution établit moins d'exigences dans le serveur web du client. La technologie **scriptag (dynamic scripting)** utilisée permet d'éviter la validation de sauts de domaine des navigateurs. Cette technologie a certaines limitations :

- L'IE n'est pas capable d'envoyer plus de 256 octets d'information, ce qui fait que les requêtes plus longues ne sont pas envoyées correctement.
- La synchronisation entre appels est plus rudimentaire et ne fonctionne pas toujours sous IE6.

- **Validation par proxy**

Dans ce cas, le distributeur doit installer un *servlet proxy* et, pour cela, il doit posséder un serveur de *servlets*. Le *servlet proxy* reçoit les requêtes des clients, ajoute l'identifiant de distributeur et les renvoie symétrisées aux serveurs Cercalia. Il est important de protéger le *proxy* afin que l'accès ne puisse se faire à partir d'emplacements invalides.

Cette solution est bien plus sûre que la précédente. Les clients peuvent accéder au *proxy* avec des requêtes AJAX et n'ont pas de limitations quant à la taille de l'information. Le proxy symétrise les requêtes selon la vitesse de réponse des serveurs et tolère les éventuels plantages de ces derniers.

2.1 Installation

2.1.1 Avec validation par domaine

- Décompresser le fichier fourni **cercalian.zip** dans un dossier.
- Supprimer le sous-dossier WEB-INF.
- Éditer le fichier **cercalia.js** et décommenter la section de connexion par domaine. Modifier la valeur de la variable du contexte en la remplaçant par celle fournie par Nexus.
- Publier le dossier dans un serveur web.
- Informer Nexus de la liste de domaines gérés par le serveur web.
- Essayer la démo fournie.

2.1.2 Avec validation par proxy

- Décompresser le fichier fourni **cercalian.zip** dans un dossier.
- Ajouter votre clé de distributeur (10 chiffres) dans le fichier de configuration **WEB-INF/web.xml**.
- Éditer le fichier **cercalia.js** et décommenter la section de connexion par proxy.
- Publier le dossier dans un serveur d'applications.
- Essayer la démo fournie.

3 Intégration et personnalisation de l'interface

3.1 Exemple introductif et intégration

Le code suivant est un exemple introductif d'utilisation de l'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>
```

Avant d'utiliser l'API, il faut la charger en suivant l'instruction suivante :

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

Le constructeur de **cercaliaClient** nécessite la *div* où placer la carte. L'identifiant de la *div* ne doit pas être un nom composé (par exemple : map_1 n'est pas correct ; en revanche, map1 est correct). On va créer une *div* dans le *tag body* de la page :

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

Lorsque le constructeur **cercaliaClient** est appelé, la *div* doit déjà être définie dans la structure de la page. Pour que le lancement se fasse correctement, il faut le faire dans l'évènement *onload* qui est appelé après le chargement et la composition de la page.

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

Le lancement consiste essentiellement à lire un fichier de propriétés et à appeler le constructeur de **cercaliaClient** en passant par paramètre le nom de la *div* où la carte doit être placée, un objet contenant les propriétés et une fonction qui s'exécutera après le lancement de la carte.

```
var cercalia;
```

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

Une fois la carte lancée et la fonction *execute* appelée, on peut maintenant appeler les fonctions de l'API souhaitées :

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

3.2 Personnalisation du style et des images

Après le chargement de l'API, on peut surcharger les styles à l'aide de l'instruction suivante :

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

Dans ce fichier de style, on peut changer les noms ou les chemins des images de l'interface :

- Images de fond des onglets à l'état *on* et *off*.
- Images de fond des titres des contenus des repères génériques.
- Images fermer, agrandir et réduire les contenus des repères par défaut, ceux de l'API et les repères simples de l'API qui n'ont que le bouton fermer.
- Images des options du menu.

Exemple : changer l'image de fond du titre des repères.

On modifie le fichier *html* afin qu'il charge la feuille de styles locale :

```
<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>
```

Dans la feuille de styles locale, on recherche les lignes suivantes :

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

et les remplaçons par :

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

On copie la nouvelle image dans le dossier d'images.

3.3 Le fichier de configuration

Les propriétés de configuration de la carte peuvent être chargées à partir d'un fichier au format XML (par défaut, fichier **config.xml**) avec les étiquettes des valeurs à personnaliser. Exemple de fichier :

```
<?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>
```

Le tableau suivant présente une liste des étiquettes qui peuvent figurer dans le fichier de configuration ainsi que leur signification :

client	Code client concaténé à la clé principale du distributeur. Ne pas mettre la clé de distributeur dans ce paramètre.
map_zoom_level	Niveau d'échelle initial de la carte. 0 plus proche.
zoom_levels	Liste des niveaux de zoom que l'on souhaite voir apparaître dans la barre de zoom de la carte. Pour préciser si ce sont des niveaux devant apparaître ou pas, utiliser l'attribut take avec les valeurs de true ou false respectivement. Exemple d'utilisation :

	<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>Dans ce cas, on ne souhaite pas que les niveaux de zoom marqués apparaissent comme disponibles.</p>
map_center	Coordonnées du centre de la carte sous Mercator EPSG:54004.
map_style	Style de la carte. Utiliser « default » par défaut.
map_extent	Extension maximale affichée sur la carte. Format : <i>left, bottom, right, top</i>
language	Code de langue (nom du fichier XML qui contient les messages).
panel_status	État du panneau latéral. Valeurs possibles : hidden : caché. minimized : réduit. maximized : agrandi.
panel_style	Style de la barre latérale. Valeurs possibles : default rounded
panel_pages	Liste des pages du panneau latéral et si elles sont visibles ou non. L'une des pages peut avoir l'attribut « selected » à « true » pour indiquer que cette page sera initialement sélectionnée. Pages possibles : geocoding : page de recherche de candidats. proximity : page de résolution de proximités. route : page de calcul d'itinéraires. pois : page de contrôle de l'affichage des points d'intérêt.
zoombar_status	État de la barre de zoom. Valeurs possibles : hidden : cachée. minimized : courte. maximized : longue.
pancontrol_status	État de la commande de déplacement : hidden : caché. visible : visible.
overview_map_status	État de la carte de localisation : Valeurs possibles : hidden : cachée. minimized : réduite. maximized : agrandie.
overview_map_style	Style de la carte de localisation. Utiliser « default » par défaut.
toolbar_status	État de la barre d'outils. hidden : cachée. minimized : réduite. maximized : agrandie.
toolbar_tools	Définir l'ordre et les boutons qui apparaissent sur la barre d'outils et le bouton initialement

	<p>sélectionné.</p> <p>pan : outil servant à se déplacer sur la carte.</p> <p>zoomin : outil servant à se rapprocher d'une région.</p> <p>zoomout : outil servant à s'éloigner.</p> <p>draw : outil servant à dessiner (non implanté).</p> <p>clean : supprime les résultats des recherches précédentes.</p> <p>measure : calcul des distances (non implanté).</p> <p>infopoi : information sur les points d'intérêt.</p> <p>infoter : information concernant une entité géographique obtenue d'une coordonnée.</p> <p>pois : ouvre la boîte de dialogue de sélection des catégories de POI.</p> <p>export : (non implanté).</p> <p>customclick1 : bouton de personnalisation (voir Annexe).</p> <p>customclick2 : bouton de personnalisation (voir Annexe).</p>
show_mouse_position	Affiche les coordonnées de la position de la souris si true est spécifié. False par défaut.
xsl_name	Nom et chemin du modèle XSL utilisant par défaut la fonction <code>createReport</code> . Si le chemin n'est pas spécifié, le modèle doit se trouver dans le même dossier que le fichier de configuration.
lswitcher_status	<p>État de l'outil de choix du type de carte visible. Déroule un liste des couches disponibles de la carte (couches WMS créées par la fonction createWMSStyle, styles définis par le paramètre map_styles et styles disponibles pour Cercalia). Préciser visible ou hidden pour le rendre visible ou invisible.</p> <p><code><lswitcher_status>visible</lswitcher_status></code></p>
map_styles	<p>Styles disponibles dans l'outil de choix du type de carte visible (<i>OpenStreetMap, Fotos aéreas España (PNOA)</i>):</p> <pre><map_styles> <openstreetmap>true</openstreetmap> <pnoa>true</pnoa> </map_styles></pre>

4 La classe cercaliaClient

4.1 Structures ou objets utilisés par les méthodes de cette classe

La classe **cercaliaClient** est la principale de cette API. Les objets de cette classe représentent une carte et possèdent toutes les méthodes d'appel aux services Cercalia. Les paramètres de sortie des méthodes sont des structures composées des classes suivantes :

4.1.1 Attributs d'une erreur (Error)

Zone	Description
id	Code d'erreur
type	Type d'erreur
text	Description de l'erreur

4.1.2 Attributs d'un candidat (Candidate)

Zone	Description
desc	Description du candidat
type	Type de candidat (st, ct ... poi)
id	Identifiant d'entité géographique
name	Nom d'entité géographique
ge	Objet type d' entité géographique
pos	Numéro d'ordre (utilisé pour ordonner les résultats)

4.1.3 Attributs d'une entité géographique (Geoentity)

Zone	Description
type	Type d'entité géographique : <ul style="list-style-type: none">• ctry : pays.• reg : région.• subreg : sous-région.• mun : commune.• ct : ville/localité.• st : rue.• rd : route.• pcode : code postal.• poi : point d'intérêt.
countryId	Code de pays
country	Nom de pays
regionId	Code de région
region	Nom de région
subregionId	Code de sous-région
subregion	Nom de sous-région

municipalityId	Code de commune
municipality	Nom de commune
cityId	Code de ville/localité
city	Nom de ville/localité
streetId	Code de rue
street	Nom de rue
streetPrefix	Préfixe de rue
streetArticle	Article de rue
streetName	Nom court de rue
houseNumber	Numéro de la rue
postalCode	Code postal
roadId	Code de route
roadName	Nom de route
km	Point kilométrique de la route
poId	Code du point d'intérêt
poiName	Nom du point d'intérêt
x	Coordonnée x du centre de l'entité géographique
y	Coordonnée y du centre de l'entité géographique
srs	Système de référence de la coordonnée
arc	Objet de type Arc (informé seulement dans proximité)
prox	Objet de type Prox (informé seulement dans proximité)
pos	Numéro d'ordre (utilisé pour ordonner les résultats)

4.1.4 Attributs d'un point d'intérêt (POI)

Zone	Description
id	Code du point d'intérêt
name	Nom du point d'intérêt
categoryId	Code de catégorie
subcategoryId	Code de sous-catégorie
geometry	Type de géométrie : P : point L : ligne Z : polygone
info	Information sur le point d'intérêt
infoxml	Information XML sur le point d'intérêt dans un objet <i>string</i>
x	Coordonnée x du point d'intérêt
y	Coordonnée y du point d'intérêt
srs	Système de référence de la coordonnée
ge	Entité géographique associée au point d'intérêt
prox	Objet de type Prox
pos	Numéro d'ordre (utilisé pour ordonner les résultats)
wkt	Géométrie du point d'intérêt au format <i>web known text</i> ou <i>null</i> . Cette zone n'est renseignée que par calculateRoute

4.1.5 Attributs de proximité (Prox)

Zone	Description
dist	Distance euclidienne (ligne droite) par rapport au centre
routedist	Distance en voiture (seulement si le <i>weight</i> a été spécifié lors de l'appel)
routetime	Distance en temps (seulement si le <i>weight</i> a été spécifié lors de

	l'appel)
--	----------

4.1.6 Attributs d'arc (Arc)

Zone	Description
frc	Type de tronçon. Classe de route fonctionnelle
kmh	Vitesse du tronçon
zonename	Nom de la zone à laquelle appartient le tronçon
zonetype	Type de zone

4.1.7 Attributs d'un objet mobile (Mo)

Zone	Description
id	Identifiant
desc	Description (optionnelle)
status	État (optionnel)
x	Coordonnée X centre référent
y	Coordonnée Y centre référent
srs	Système de référence des coordonnées (voir annexe des « Codes »)
pos	Numéro d'ordre (utilisé pour ordonner les résultats)

4.1.8 Attributs d'un objet de style repère (markerStyle)

Zone	Description
icon	Nom du fichier d'image pour l'icône (ex. « ico2.gif »).
width	Largeur de l'image de l'icône en pixels.
height	Hauteur de l'image de l'icône en pixels.
visible	Booléen indiquant si l'on souhaite afficher l'information du repère.
minimized	Booléen indiquant si l'on souhaite afficher le titre du repère.
noHideOthers	Booléen indiquant s'il n'est pas nécessaire de réduire tous les autres repères existants lorsque l'on crée le repère courant, dans le cas où il puisse être réduit.
mouseover_content	Format HTML à afficher dans l'évènement <i>mouseover</i> de l'icône. Il se fermera seulement pendant l'évènement <i>mouseout</i> ou en cliquant.
mouseover	Code JavaScript à évaluer dans l'évènement <i>mouseover</i> de l'icône. Pris en charge par le <i>mouseover_content</i> .
mouseclick	Code JavaScript à évaluer dans l'évènement <i>mouseclick</i> de l'icône. Pris en charge par <i>content</i> et ensuite par <i>title</i> .
showNumber	Booléen indiquant si l'on souhaite afficher un numéro dans l'icône du repère.
numberTextColor	<i>String</i> avec la couleur du numéro de l'icône (ex. "#FFFFFF").
title	Format du titre du repère.
content	Format du contenu.
cssTitle	<i>String</i> avec le style à appliquer au titre.
iconBorderColor	Couleur du bord de la <i>div</i> incluant l'icône. Si une couleur n'est pas indiquée, le bord n'apparaît pas.
iconBorderWidth	Largeur en pixels du bord de la <i>div</i> incluant l'icône (par défaut

	Opx).
label	Contenu du label.
labelState	Visible/hidden selon si on veut afficher le label du moment de création du repère.

4.1.9 Attributs d'un objet de forme (feature)

Zone	Description
<Géométrie>	Forme géométrique, au format selon la demande.
id	Identifiant interne de la forme.
created	Booléen indiquant qu'il s'agit d'une forme créée par l'utilisateur (*).
modified	Booléen indiquant qu'il s'agit d'une forme que l'utilisateur a modifiée (*).
deleted	Booléen indiquant qu'il s'agit d'une forme que l'utilisateur a effacée (*).
style	Style utilisé pour créer la forme.
styleSelect	Style que présente la forme au moment où elle est sélectionnée en vue de son édition.

(*) si tous les booléens sont faux, la forme n'a pas changé par rapport à l'original

La zone **<Géométrie>** est instanciée comme l'une des zones suivantes, selon la méthode appelée pour obtenir les formes :

Zone	Description
wkt	Forme géométrique en « Well-known Text Representation »
gml	Forme géométrique au format GML v.3

4.2 Méthodes pour le traitement des repères

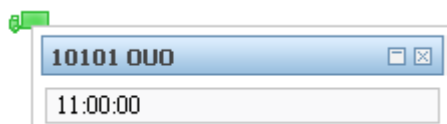
4.2.1 createMarker(id,x,y,srs,style,id2)

Cette méthode crée un repère selon les paramètres passés :

- **id** : Identifiant du repère et de la *div* qui contiendra le contenu du repère.
- **x,y** : coordonnées x et y où doit être situé le repère.
- **srs** : système de coordonnées utilisé (voir annexe « Systèmes de coordonnées »).
- **style** : Objet de type markerStyle qui définit les caractéristiques du repère.
- **id2** : paramètre optionnel qui associe le repère courant à un autre, et fait qu'une flèche soit dessinée entre les deux.

Exemples de repère :

Repère agrandi :



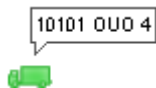
Repère réduit :



Repère avec contenu et titre non visibles :



Repère avec le *label* visible:



Dans ce cas, il ne faut associer le repère à aucun autre et, par conséquent, le dernier paramètre ne doit pas être passé.

Si un **markerStyle** est utilisé sans spécifier **title** ni **cssTitle** dans l'étiquette, seul le contenu sera présent. Le **content** doit être un HTML valide et le format de visualisation doit être spécifié par l'utilisateur lui-même pos css ou à l'aide de l'attribut style. Si le **content** n'est pas spécifié non plus, il faudra créer un repère sans étiquette.

Exemple de code :

```
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:"contTitotlPopup", content:cont,  
  noHideOthers:true};  
  
cercalia.createMarker("10101 OVO 1", -3.808951, 43.458308, "EPSG:4326", style);
```

Exemple de contenu :

```
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>';
```

Où dans `interieurContenu` , on peut mettre le code HTML souhaité : images, tableaux, etc. Dans l'exemple précédent : `'11:00:00'`

4.2.2 *createMarkers (obj,style)*

Cette fonction crée les repères pour les éléments contenus dans un objet résultant d'un géocodage ou d'une proximité. Le paramètre **obj** doit avoir un type **resultGeocoding**, **resultProximity**, **resultGetPoi**, **Mo**, **Poi** ou **Geoentity**. L'objet **style** est une liste d'attributs appartenant à **markerStyle**.

Dans les attributs **icon**, **title** et **content** du `markerStyle`, les zones du candidat, du point d'intérêt ou de l'entité géographique à partir de laquelle est créé le repère peuvent être spécifiés à l'aide du format : `%{expr}%`. Un code de message du fichier de messages peut aussi être spécifié en utilisant la syntaxe : `%[msg1]%`

Exemple de code permettant de créer des repères de points d'intérêt :

```
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);
    }
}
```

Exemple de code permettant de créer le repère du premier point d'intérêt :

```
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)*

Déplacer le repère avec l'identifiant **id** vers les nouvelles coordonnées.

- **id** : Identifiant du repère et de la *div* qui contiendra le contenu du repère.
- **x,y** : coordonnées x et y où doit être situé le repère.
- **srs** : système de coordonnées utilisé.
- **cont** : contenu HTML valide pour actualiser le *popup* du repère (paramètre optionnel).
- **contOver**: Contenu html valable pour rafraîchir le popup du repère dans l'évènement *mouseover*. (Paramètre optionnel)

Exemple 1 :

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

Exemple 2 :

Cette méthode modifie le contenu du *popup* du repère indiqué par paramètre.

```

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

```

4.2.4 *setMouseoverContent (id,contOver)*

Il change le contenu du popup qui figure dans l'évènement **mouseover**. Il fonctionne seulement avec les repères pour lesquels la propriété *mouseover_content* est indiquée.

- **id**: identifiant du repère.
- **contOver**: Contenu html valable pour rafraîchir le popup du repère qui figure dans l'évènement *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)*

Cette méthode permet de modifier l'icône du repère indiqué par le paramètre *id*. L'image qui remplacera l'image courante est indiquée par l'intermédiaire du paramètre *url*. Les deux paramètres sont obligatoires et il faut qu'il existe un repère avec l'identifiant indiqué.

À titre d'exemple, elle peut être combinée avec l'utilité de l'attribut *mouseclick* du paramètre *style* de la fonction *createMarker*. Cela permet de modifier l'état de l'icône lorsque l'utilisateur clique sur celle-ci.

Exemple 1 :

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

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

Cette méthode supprime les repères et leur identifiant est passé par paramètre. Si aucun identifiant n'est passé, tous les repères seront supprimés de la carte.

Exemples :

Supprimer tous les repères :

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

Supprimer un repère :

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

Supprimer trois repères :

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

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

Cette méthode centre la carte par rapport aux repères et leur identifiant est passé par paramètre. Si aucun identifiant n'est passé, il sera centré sur tous les repères de la carte.

- **changeScale** : booléen indiquant si l'échelle doit être modifiée afin de pouvoir afficher tous les repères concernés.

Exemples :

Centrer la carte par rapport à tous les repères en modifiant l'échelle de manière qu'ils soient tous affichés :

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

Centrer la carte par rapport à un repère sans modifier l'échelle :

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

Centrer la carte par rapport à trois repères en modifiant l'échelle :

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

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

Cette méthode bascule l'état de visibilité des repères vers l'état passé par le paramètre **state**. Cette modification affecte tous les repères dont l'identifiant est passé par paramètre et, dans le cas où aucun identifiant ne serait passé, cela aura une incidence sur tous les repères de la carte.

- **state** : état dans lequel les repères concernés seront modifiés. Il peut y avoir trois états :
 - **closed** : permet de fermer les repères.
 - **minimized** : permet de réduire les repères.
 - **maximized** : permet d'agrandir les repères.

Exemples :

Agrandir tous les repères de la carte :

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

Réduire un repère :

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

Cacher l'information associée de trois repères :

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

4.2.9 *showLabelMarkers (visible)*

Il affiche ou cache les éléments *label* des repères.

Exemple:

Cacher les labels:

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

4.3 Méthodes pour le traitement des formes

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

Cette méthode crée une ou plusieurs formes à partir de leur définition géométrique en WKT, du système de référence et de la définition de style. Elle retourne une liste d'identifiants des formes créées.

- **wkt** : formes géométriques
- **srs** : système de coordonnées utilisé*
- **style** : style de représentation
- **editstyle** : style de représentation en édition

* Valeurs possibles du système de coordonnées (paramètre srs) : voir annexe

Le format de WKT (« Well-known Text Representation for Geometry ») est un standard permettant l'échange de géométries définie par Open Geospatial Consortium. Le tableau suivant montre des exemples de ce format :

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

Pour de plus amples informations, consulter le document OpenGIS® Implementation Specification for Geographic information - Simple feature access - Part 1: Common architecture.

Le paramètre style est une liste de propriétés avec leur valeur :

Propriété	Description	Valeur par défaut
fillColor	Couleur de remplissage	"#ee9900"
fillOpacity	Opacité du remplissage	0.4
strokeColor	Couleur de la ligne ou du contour	"#ee9900"
strokeOpacity	Opacité de la ligne ou du contour	1
strokeWidth	Largeur de la ligne ou du contour	1
pointRadius	Taille du point en pixels	0

Exemple de dessin d'une polyligne :

```
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"});
```

Exemple de dessin d'un polygone :

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

Exemple de dessin d'un rectangle :

```
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});
```

Exemple de dessin d'un cercle :

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

De manière analogue, quand l'utilisateur édite les formes, le paramètre **editstyle** contient le style des formes.

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

Cette méthode crée une ou plusieurs formes à partir de leur définition géométrique en GML 3.1.1. (Geography Markup Language) sous forme de chaîne (*String*) et de la définition de style. Elle retourne une liste d'identifiants des formes créées.

- **gml** : formes géométriques
- **style** : style de représentation
- **editstyle** : style de représentation en édition

Le format GML doit contenir une valeur pour la zone *srsName*. S'il n'en est pas ainsi, les coordonnées sont censées être représentées dans le système de coordonnées de la carte.

Pour de plus amples informations sur le standard GML, voir la spécification fournie par OpenGIS (<http://www.opengeospatial.org/standards/gml>).

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

Cette méthode centre la carte par rapport aux formes qui sont passées par un ou plusieurs paramètres. Si aucune forme n'est indiquée, le centrage est appliqué à toutes les formes présentes sur la carte.

- **changeScale** : booléen indiquant s'il faut modifier l'échelle pour que toutes les formes concernées soient affichées.

Exemples :

Centrer la carte par rapport à toutes les formes en modifiant l'échelle de manière qu'elles soient toutes affichées :

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


Centrer la carte par rapport à une forme sans modifier l'échelle :

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

Centrer la carte par rapport à trois formes en modifiant l'échelle :

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

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

Cette méthode supprime les formes spécifiées dans un ou plusieurs paramètres. Si aucune forme n'est indiquée, elles sont toutes effacées. Si aucun paramètre n'est indiqué, les éléments de la liste de formes qui avaient été supprimées en utilisant les fonctionnalités de l'outil « draw »  (voir annexe « Outils d'édition ») sont également effacés. À partir de ce moment-là, les formes à l'état 'deleted' ne pourront pas être consultées avec la méthode *getFeaturesByState*.

Exemples :

Supprimer toutes les formes :

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

Supprimer une forme :

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

Supprimer deux formes :

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

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

Cette méthode obtient les formes qui répondent aux états indiqués : elle permet de récupérer et d'enregistrer dans une base de données propre la géométrie des formes créées à l'aide du module de numérisation de l'API (voir Annexe 8).

Le système de coordonnées dans lequel l'on souhaite obtenir les géométries peut être indiqué comme paramètre optionnel. S'il n'est pas indiqué, il faudra utiliser la carte de fond correspondante.

États :

- **created** : booléen pour obtenir les formes nouvelles que l'utilisateur aura créées.
- **modified** : booléen pour obtenir les formes que l'utilisateur a modifiées.
- **deleted** : booléen pour obtenir les formes que l'utilisateur a effacées.
- **unchanged** : booléen pour obtenir les formes que l'utilisateur n'a pas modifiées.


Il est possible de faire n'importe quelle combinaison d'états. Cependant, les états à **false** ne retourneront pas tous une réponse.

Les formes sont retournées dans un objet **resultFeatures**, avec la définition suivante :

Zone	Description
features	Liste d'objets de type Feature

Les *features* doivent avoir l'attribut **wkt** pour contenir la définition de la géométrie.

Exemples :

Cette méthode obtient les formes qui ont été créées sur la carte (formes nouvelles) à l'aide de l'outil « draw »  (voir annexe « Outils d'édition »).

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

Cette méthode obtient les formes qui ont été créées à l'aide de la fonction *createFeature*, puis modifiées en utilisant les fonctionnalités de l'outil « draw ».

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

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


Cette méthode obtient les formes qui répondent aux états indiqués de manière analogue à la fonction précédente.

Les formes sont retournées dans un objet **resultFeatures**. Les *features* doivent avoir l'attribut **gml** pour contenir la définition de la géométrie.

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

Cette méthode obtient les formes spécifiées dans un ou plusieurs paramètres. Si aucune forme n'est indiquée, elles seront toutes retournées. Elle permet de récupérer et d'enregistrer dans une base de données propre la géométrie des formes créées à l'aide du module de numérisation de l'API (voir Annexe 8).

Les formes sont retournées dans un objet **resultFeatures**. Les *features* doivent avoir l'attribut **wkt** pour contenir la définition de la géométrie.

Seules les formes supprimées à l'aide de l'outil « draw »  seront retournées si elles ont été créées à l'aide de la fonction *createFeature*.

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

Cette méthode obtient les formes spécifiées dans un ou plusieurs paramètres, de manière analogue à la fonction précédente.

Les formes sont retournées dans un objet **resultFeatures**. Les *features* doivent avoir l'attribut **gml** pour contenir la définition de la géométrie.

4.4 Méthodes de traitement de la carte

4.4.1 *activateTool(tool)*

Cette méthode sélectionne ou active l'outil indiqué par paramètre (*pan*, *zoomin*, *zoomout*, *draw*, *clean*, *measure*, *infopoi*, *infoter*, *pois*, *export*, *customclick1*, *customclick2*). L'outil doit être disponible dans la barre d'outils sinon aucune action ne se réalise.

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

4.4.2 *isCompleted()*

Cette méthode retourne le booléen *true* si l'objet *cercalia* est prêt à employer.

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

Cette méthode active ou désactive l'affichage des points d'intérêt des catégories spécifiées. Si aucune catégorie n'est indiquée, l'affichage de toutes les catégories existantes sera activé ou désactivé selon les indications du paramètre *visible*.

Exemple pour afficher les points d'intérêt des catégories C001 et C003 :

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

Exemple pour désactiver l'affichage des points d'intérêt des catégories C001 et C003 :

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

4.4.4 *setLanguage(lang)*

Cette méthode change la langue du client. Le paramètre *lang* indique la langue que l'on souhaite changer. Lorsqu'il n'existe pas de fichier .xml pour la langue indiquée, un message d'erreur s'affiche.

Exemple pour passer de la langue en cours à l'anglais :

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

4.4.5 *getMap()*

Cette méthode retourne un objet *OpenLayers.Map*. Pour voir les possibilités que présente cet objet, consulter l'API de documentation d'*OpenLayers* (<http://www.openlayers.org>).

Exemple :

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

4.4.6 *deleteRoute (resultCalculateRoute)*

Cette méthode supprime la représentation graphique (repères et formes) de l'itinéraire qui est passé par paramètre.

4.4.7 *setParse< operation > (function)*

Ces méthodes permettent de personnaliser la réponse reçue par l'utilisateur lorsqu'il active des actions de la barre latérale ou des boutons.

Les noms des fonctions à appeler pour intercaler le code et les paramètres respectifs que doivent avoir les nouvelles fonctions sont :

setParsePanelSearchGeo (function)

Permet de personnaliser les actions réalisées après la recherche de candidats à l'aide du panneau latéral d'adresse ou de code postal. Le paramètre *function* doit indiquer une fonction qui reçoit un paramètre de type *resultGeocoding*. Cette fonction peut appeler `showResultPanelSearchGeo(resultGeocoding)` pour afficher les résultats.

setParsePanelSearchCoord (function)

Permet de personnaliser les actions réalisées après la recherche d'une coordonnée dans le panneau latéral. Le paramètre *function* doit indiquer une fonction qui reçoit un paramètre de type *resultProximity*. Cette fonction peut appeler `showResultPanelSearchCoord(resultProximity)` pour afficher les résultats.

setParsePanelProximity (function)

Permet de personnaliser les actions réalisées après la recherche d'une proximité à l'aide du panneau latéral. Le paramètre *function* doit indiquer une fonction qui reçoit un paramètre de type *resultProximity*. Cette fonction peut appeler `showResultPanelProximity(resultProximity)` pour afficher les résultats.

setParseToolInfoPOI (function)

Permet de personnaliser les actions réalisées après la recherche de l'information du POI sélectionné. Le paramètre *function* doit indiquer une fonction qui reçoit un paramètre de type *resultProximity*. Cette fonction peut appeler `showToolInfoPOI(resultProximity)` pour afficher les résultats.

setParseToolInfoTer (function)

Permet de personnaliser les actions réalisées après obtention de l'information territoriale. Le paramètre *function* doit indiquer une fonction qui reçoit un paramètre

de type *resultProximity*. Cette fonction peut appeler `showToolInfoTer(resultProximity)` pour afficher les résultats.

Exemple : modifier la sortie de candidats afin qu'un repère pour chaque candidat apparaisse sur la carte.

```
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);
```

Exemple : Modifier la sortie de données d'infoposition afin que la coordonnée soit indiquée au format géographique.

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

cercalia.setParseToolInfoTer(infoposicion);
```

4.4.8 *showResult*< operation > (obj)

Si, en utilisant les fonctions, l'utilisateur souhaite uniquement manipuler les données et s'il veut que l'opération se termine par la présentation par défaut, il peut appeler les méthodes suivantes :

showPanelSearchGeo (resultGeocoding)

Affiche le résultat d'une recherche de candidats dans le panneau latéral (adresse ou de code postal).

showPanelSearchCoord (resultProximity)

Affiche le résultat d'une recherche par coordonnée dans le panneau latéral.

showPanelProximity (resultProximity)

Affiche le résultat d'une proximité dans le panneau latéral.

showToolInfoPOI (resultProximity)

Affiche sur la carte les données concernant le point d'intérêt le plus proche.

showToolInfoTer (resultProximity)

Affiche sur la carte les données concernant l'entité géographique la plus proche.

4.4.9 getExtent(srs)

Cette méthode retourne un objet `OpenLayers.Bounds` avec les coordonnées de l'extension de la carte en cours d'affichage. Les coordonnées se trouveront dans le système indiqué par le paramètre obligatoire *srs*.

Exemple : obtenir l'extension de la carte avec le système de coordonnées courant :

```
function obtenirExtension()  
{  
    var srs = cercalia.getCurrentSRS();  
    var ext = cercalia.getExtent(srs);  
    alert(ext.left+" "+ext.bottom+" "+ext.right+" "+ext.top);  
}
```

4.4.10 getCenter(srs)

Retourne un objet `OpenLayers.LonLat` avec les coordonnées du centre de la carte en cours d'affichage. Les coordonnées seront dans le système indiqué par le paramètre obligatoire *srs*.

Exemple : Obtenir le centre de la carte avec le système de coordonnées courant :

```
function obtenerCentro()
{
    var srs = cercalia.getCurrentSRS();

    var center=cercalia.getCenter(srs);

    alert(center.lon+", "+center.lat);
}
```

4.4.11 *setCenter(center, scale)*

Cette méthode change le centre de la carte actuellement en cours d'affichage. Le paramètre *center* aura le format suivant {x: ..., y:..., srs: ...}, où x, y sont des coordonnées et srs le système de coordonnées. Si l'on indique le paramètre *scale*, l'affichage de la carte sera adapté au niveau d'échelle¹ du système de cercalia. Si l'on ne travaille pas avec la cartographie Cercalia, l'affichage sera adapté au niveau d'échelle cercalia le plus proche du niveau en cours.

Exemple : actualiser le centre de la carte avec des coordonnées du système EPSG:4326 et indiquer un niveau d'échelle cercalia :

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

4.4.12 *setCenter(center, srs, scale)*

Cette méthode fait basculer le centre de la carte vers celui indiqué. Le paramètre *center* doit avoir le type OpenLayers.LonLat. Et les coordonnées doivent être dans le système de référence indiqué par *srs*. Si l'on indique le paramètre *scale*, l'affichage de la carte sera adapté au niveau d'échelle² du système de cercalia. Si l'on ne travaille pas avec la cartographie Cercalia, l'affichage sera adapté au niveau d'échelle cercalia le plus proche du niveau en cours.

Exemple : actualiser le centre de la carte avec des coordonnées du système EPSG :

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

¹ Dans le système cercalia, il y a 14 niveaux d'échelle (de 0 à 13). Le niveau 0 correspond à l'échelle la plus petite et le niveau 13 à la plus grande échelle disponible dans le système.

```
}
```

4.4.13 *zoomToExtent(extent,srs)*

Cette méthode adapte le zoom de la carte à l'extension indiquée. Le paramètre *srs* indique le système de coordonnées dans lequel l'extension est exprimée. L'extension est indiquée en utilisant le paramètre *extent* et peut être un objet `OpenLayers.Bounds` ou bien une liste de quatre attributs : `{x1:.....,y1:.....,x2:.....,y2:.....}`

Exemple :

```
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");
```

4.4.14 *getScale(closest)*

Cette méthode retourne une valeur numérique qui indique le niveau d'échelle³ courant de la carte. Si *closest* est vrai, elle retourne le niveau d'échelle *cercalia* courant. Si l'on travaille avec une autre cartographie, elle retourne le niveau de *cercalia* le plus proche de celui de la carte que l'on est en train d'utiliser. Si *closest* est faux, elle retourne toujours le niveau d'échelle de *cercalia*. Si l'on travaille avec une autre cartographie, elle retourne une valeur nulle. La méthode *getStyle* indique la cartographie que l'on est en train d'utiliser.

4.4.15 *getCurrentSRS()*

Cette méthode retourne un *string* indiquant le système de coordonnées courant de la carte.

4.5 Méthodes d'obtention d'informations

4.5.1 *geocoding(search,returnFunc)*

Cette méthode effectue une requête de géoconversion selon les paramètres de l'objet *search*. Une fois résolue, la fonction *returnFunc* est exécutée en passant par paramètre un objet de type `resultGeocoding`.

L'objet *search* peut contenir les zones de texte suivantes :

³ Dans le système *cercalia*, il y a 14 niveaux d'échelle (de 0 à 13). Le niveau 0 correspond à l'échelle la plus petite et le niveau 13 à la plus grande échelle disponible dans le système.

Zone	Description	Par. service Cercalia
name	Filtre nom sans spécifier le niveau	locn
countryId	Filtre code pays	ctryc
country	Filtre nom pays	ctryn
regionId	Filtre code région	regc
region	Filtre nom région	regn
subregionId	Filtre code sous-région	subregc
subregion	Filtre nom sous-région	subregn
subOrRegion	Filtre nom région ou sous-région	rsn
municipalityId	Filtre code commune	munc
municipality	Filtre nom commune	munnc
cityId	Filtre code ville/localité	ctc
city	Filtre nom ville/localité	ctn
postalCode	Filtre code postal. Le code pays doit être spécifié	pcode
streetId	Filtre code rue	stc
street	Filtre nom rue	stn
street2	Filtre nom deuxième rue pour calculer intersection	istn
street2Id	Filtre code deuxième rue pour calculer intersection	istc
houseNumber	Filtre numéro de rue	stnum
roadName	Filtre nom de route	rdn
roadId	Filtre code de route	rdc
km	Filtre point kilométrique pour route	km
fullSearch	Résoudre tous les paramètres de candidats	fullsearch
numCand	Nombre de candidats à retourner par page	numcand
posCand	Position du premier candidat à retourner, en commençant par 0	poscand
num	Nombre maximal de résultats	
poicat	Filtre liste de catégories de points d'intérêt	poicat
poild	Filtre code de point d'intérêt	poic
poiName	Filtre nom de point d'intérêt	poiname

Les paramètres d'intersection sont incompatibles avec le numéro de rue et ne peuvent être utilisés que lorsqu'il existe une spécification de rue. Les paramètres permettant de spécifier rue et route sont incompatibles entre eux.

S'il existe plus d'un candidat, la coordonnée retournée peut ne pas être celle de l'élément recherché. Par exemple, si en recherchant une entrée (rue + numéro), les données retournées sont des candidats de pays, de ville, de rue, etc., les coordonnées de ces candidats correspondent, respectivement, au pays, à la ville ou à la rue et non à celle de l'entrée.

Les paramètres sont passés de cette manière :

```

var search={ countryId:null, city:null };
search.countryId = "FR"; // Code du pays
search.city = "Paris"; // Nom de localité
cercalia.geocoding(search, execute);

function execute(data)
{
    // traiter l'information
}

```

Les données de l'information de **resultGeocoding** sont les suivantes :

Réponse	Zone	Description
<i>erreur</i>	error	Objet de type Error (contient <i>null</i> s'il n'y a pas d'erreur)
<i>candidats</i>	candidates	Liste d'objets de type Candidate
	pos	Position du premier candidat retourné
	numCand	Nombre de candidats par page
	num	Nombre maximal de résultats souhaités indiqués dans la requête.
	total	Nombre total de candidats de la recherche

4.5.2 proximity(search,returnFunc)

Cette méthode effectue une requête de proximité selon les paramètres de l'objet *search*. Pour effectuer une proximité, il faut spécifier les paramètres d'un référent (le centre de la recherche) et les paramètres spécifiant quels sont les référés (ceux qui sont recherchés près du centre).

Pour spécifier le référent, on utilise les mêmes paramètres que dans un *geocoding* :

Zone	Description	Par. service Cercalia
name	Filtre nom sans spécifier niveau	locn
countryId	Filtre code pays	ctryc
country	Filtre nom pays	ctryn
regionId	Filtre code région	regc
region	Filtre nom région	regn
subregionId	Filtre code sous-région	subregc
subregion	Filtre nom sous-région	subregn
subOrRegion	Filtre nom de la région ou sous-région	rsn
municipalityId	Filtre code commune	munc
municipality	Filtre nom commune	mun
cityId	Filtre code ville/localité	ctc
city	Filtre nom ville/localité	ctn
postalCode	Filtre code postal. Le code pays doit être spécifié	pcode
streetId	Filtre code rue	stc
street	Filtre nom rue	stn
street2	Filtre nom deuxième rue pour calculer intersection	istn
street2Id	Filtre code deuxième rue pour calculer intersection	istc
houseNumber	Filtre numéro de rue	stnum
roadName	Filtre nom de route	rdn
roadId	Filtre code de route	rdc
km	Filtre point kilométrique pour route	km
fullSearch	Résoudre tous les paramètres de candidats	fullsearch
numCand	Nombre de candidats à retourner	numcand
posCand	Position du premier candidat à retourner, en commençant par 0	poscand
num	Nombre maximal de résultats souhaité	

Une géométrie :

Champ	Description
-------	-------------

wkt	Géométrie de référence
srs	Système de référence des coordonnées (voir Annexe des codes)

Si l'on utilise comme référent une géométrie, la proximité peut seulement être calculée par des niveaux de géoentité. Par conséquent, pour indiquer le type référé, dans ce cas, l'on utilisera toujours le champ **rqge**.

ou bien on spécifie un centre de coordonnée :

Zone	Description
x	Coordonnée X centre référent
y	Coordonnée Y centre référent
srs	Système de référence des coordonnées (voir annexe des « Codes »)

Pour spécifier les référés, on peut spécifier l'un des trois types suivants : entité géographique, points d'intérêt ou objets mobiles.

Dans les trois cas, on peut spécifier un rayon de recherche (en distance euclidienne) et un nombre maximal de résultats.

Dans le cas des points d'intérêt, on peut définir le paramètre **weight** pour calculer la distance et le temps en voiture et ordonner le résultat en fonction du coût minimal en temps (**time**), distance (**distance**) ou argent (**money**). Par défaut, le programme calcule le coût que représente le passage du référent au référé ; si l'on préfère faire l'inverse, il faut spécifier le paramètre **inverse** à **true**.

Type référé	de	Zone	Description
<i>entité géographique</i>		rqge	Niveau d'entité géographique (adr,st,ct,mun,subreg,reg,ctry)
<i>points d'intérêt</i>		rqpoicats	Liste de catégories de points d'intérêt séparées par des virgules
		infoxml	Valeur numérique indiquant l'information XML du point d'intérêt à retourner
		weight	Optionnel. Coût utilisé pour ordonner les résultats. Valeurs possibles <i>time</i> , <i>distance</i> , <i>money</i>
		inverse	Optionnel. Peut avoir les valeurs <i>true</i> ou <i>false</i> . Quand la valeur est <i>true</i> , le coût représente le passage des référés au référent
<i>objets mobiles</i>		mos	Liste d'objets mobiles de type Mo
<i>objets communs</i>		num	Nombre maximal de résultats
		rad	Rayon maximal de recherche (distance euclidienne)

Les paramètres sont passés de cette manière :

```
var search={countryId:null, city:null, num:0, rad:0, rqpoicats:null };
search.countryId="FR"; // Code du pays
search.city="Paris"; // Nom de localité
search.num=5; // cinq candidats
search.rad=1000; // rayon de 1 000 m
search.rqpoicats="C001"; // Stations-services
cercalia.proximity(search,execute);

function execute(data)
{
    // traiter l'information
}
```

}

Si le résultat de la recherche n'a pas de candidats ou plus d'un candidat, la fonction **returnFunc** indiquée avec la même structure qu'un **resultGeocoding** est appelée.

Les données de retour **resultProximity** sont les suivantes :

Réponse	Zone	Description
<i>erreur</i>	error	Objet type error (contient <i>null</i> s'il n'y a pas d'erreur)
<i>candidats</i>	candidates	Liste de candidats de type candidate (seulement s'il y a plus d'un candidat)
	pos	Position du premier candidat retourné
	num	Nombre maximal de résultats souhaité indiqué dans la requête
	numCand	Nombre de candidats par page
	total	Nombre total de candidats de la recherche
<i>proximité</i>	type	Type de proximité demandée {adr, st, ct, mun, subreg, reg, ctry, poi}
	x	Coordonnée x du référent (centre de recherche).
	y	Coordonnée y du référent (centre de recherche).
	srs	Système de référence utilisé pour indiquer le centre de recherche.
<i>proximité POI</i>	pois	Liste d'objets poi (seulement dans une proximité résolue de POI)
<i>proximité entités géographiques</i>	location	Entité géographique où se trouve le référent
	geoentities	Liste d'objets type geoentity (seulement dans une proximité résolue d'entités géographiques)
<i>proximité objets mobiles</i>	mos	Liste d'objets type mo (seulement dans une proximité résolue d'objets mobiles)

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

Cette méthode rend la distance calculée du point p1 au point p2. Si aucune valeur est indiquée pour le paramètre *weight* elle calcule la distance en ligne droite. Si la valeur est indiquée la méthode calcule la distance en voiture selon le coût minimum de temps (weight='time'), distance (weight='distance') ou argent (weight='money').

Paramètres:

- **p1**: une coordonnée: {x:<x> ,y:<y>, srs:<srs>}
- **p2**: une coordonnée: {x:<x> ,y:<y>, srs:<srs>}
- **weight**: {String} Valeurs 'time', 'distance' ou 'money'. Paramètre optionnel.
- **returnFunction**: {function} Fonction *callback* pour ramasser le résultat.

Réponse:

Réponse	Champ	Description
<i>erreur</i>	erreur	Objet type erreur (si aucune erreur est détectée contient null)
<i>Dist</i>		Distance en ligne droite entre les deux points.
<i>routedist</i>		Distance calculée selon le paramètre "weight". Si aucune valeur est indiquée pour ce paramètre, la valeur sera null.
<i>routetime</i>		Temps (h:m:s) calculé selon le paramètre "weight". Si aucune valeur est indiquée pour ce paramètre, la valeur sera null.

Exemple:

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

Cette méthode lance une requête pour obtenir les données des POI indiqués au moyen de leur code correspondant. Infoxml est une valeur numérique permettant de spécifier le type d'information XML à retourner. Par défaut, utiliser le 0. Une fois les données obtenues, la fonction **execute** est appelée à l'aide d'un paramètre de type **resultGetPoi** contenant le résultat de l'opération.

```
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);
        }
    }
}
```

Format d'un **resultGetPoi** :

Réponse	Zone	Description
<i>erreur</i>	error	Objet type error (contient <i>null</i> s'il n'y a pas d'erreur)
<i>liste de POI</i>	pois	

4.5.5 *calculateRoute(stopList, routeParams, lineStyle, drawStopMarkers, execute)*

Cette méthode calcule l'itinéraire optimal parmi une liste d'arrêts.

Le paramètre **stopList** spécifie la liste d'arrêts. Le premier objet de la liste spécifie l'origine et le dernier la destination. Les objets intermédiaires spécifient les arrêts. Chaque élément de la liste peut être spécifié au moyen :

- d'un objet **poi**,
- d'un objet **geoentity**,
- d'un objet **mo**,
- d'une coordonnée : {x:<x> ,y:<y>, srs:<srs>},
- d'un code d'entité géographique, Ex. : {cityId: **ESP17240300056757** },
- d'un code de point d'intérêt : {poild: **C17240332252905**}.

Les trois premiers peuvent être des objets qui sont le résultat d'un géocodage ou d'une proximité.

Le paramètre *routeParams* spécifie la liste d'options pour le calcul de l'itinéraire :

Zone	Description	
weight	Coût utilisé pour le calcul de l'itinéraire (obligatoire).	
	Valeur	Langue
	distance	L'itinéraire se calcule en réduisant la distance
	time	L'itinéraire se calcule en réduisant le temps
	money	L'itinéraire évite de passer par des péages
lang	Langue du rapport de l'itinéraire (par défaut « fr »).	
	Valeur	Langue
	ca	Catalan
	de	Allemand
	en	Anglais
	es	Espagnol
	fr	Français
	it	Italien
	nl	Hollandais
pt	Portugais	
mindist	Distance minimale parcourue en mètres pour générer un saut entre sous-étapes (par défaut 1 000). Plus cette distance est grande plus le texte du rapport est compact et moins de sauts de sous-étape s'affichent.	
report	Indique si la réponse contient un rapport ou seulement les distances parcourues. true – retourne XML avec la description de l'itinéraire (par défaut). false – retourne XML seulement avec les distances et les temps parcourus.	
reorder	Indique si les arrêts doivent être réordonnés ou non au moment de calculer le chemin optimal.	

	true – les arrêts sont réordonnés afin de trouver le chemin optimal. false – les arrêts ne sont pas réordonnés (par défaut).
getpoicats	<i>String</i> avec les catégories séparées par une virgule. Obtient de l'information sur les points d'intérêt des catégories sélectionnées se trouvant associés aux arcs par lesquels passe l'itinéraire.
poiwkt	Booléen indiquant s'il faut signaler la géométrie (wkt) des points d'intérêt. Par défaut, false .
poitolerance	Valeur réelle indiquant le degré de simplification des géométries des points d'intérêt (par défaut 100 m).
infoxml	S'il est spécifié avec une valeur numérique différente de 0, il retourne l'information XML associée aux points d'intérêt. Par défaut, 0.
toll	Paramètre permettant de spécifier s'il faut retourner le coût total des péages des routes par où passe l'itinéraire calculé. true – l'information est retournée. false – l'information n'est pas retournée. Péages disponibles uniquement pour l'Espagne, le Portugal et l'Andorre.
intoll	Liste des entiers ou entier. Chaque entier vérifie si l'arrêt correspondant peut ou non se trouver sur un tronçon à péage. Si -1 est spécifié, l'arrêt se trouvera toujours hors d'un tronçon à péage. Si l'on spécifie un nombre positif, l'arrêt pourra se trouver sur un tronçon à péage à condition qu'il ne soit pas à plus de mètres que le nombre indiqué. Si les entiers sont terminés, le dernier est utilisé pour le reste des arrêts. Par exemple, si l'on spécifie intoll=[-1], tous les arrêts seront en dehors des tronçons à péage.
net	Spécifie le réseau pour le calcul des itinéraires. Ne doit être spécifié que dans les cas particuliers.
xml	Booléen indiquant si l'on souhaite renseigner le rapport XML de l'itinéraire. Par défaut, false .
tolerance	Valeur réelle indiquant le degré de simplification des géométries de l'itinéraire (par défaut 100 m).

Le paramètre **lineStyle** spécifie les propriétés de la ligne d'itinéraire. Si ce paramètre est **null**, l'itinéraire ne sera pas dessiné.

Propriété	Description	Valeur par défaut
strokeColor	Couleur de la ligne ou du contour	"#ee9900"
strokeOpacity	Opacité de la ligne ou du contour	1
strokeWidth	Largeur de la ligne ou du contour	1

Le paramètre **drawParadeMarkers** est un booléen qui spécifie s'il faut créer un *repère* pour l'origine, les arrêts et la destination de l'itinéraire.

Le paramètre **execute** définit la fonction qui sera appelée une fois l'itinéraire résolu avec un paramètre de type **resultCalculateRoute**.

Format d'un **resultCalculateRoute** :

Réponse	Zone	Description
<i>erreur</i>	error	Objet type error (contient <i>null</i> s'il n'y a pas d'erreur)
<i>itinéraire</i>	id	Identifiant d'entité géographique
	distance	Distance totale en mètres de l'itinéraire.
	time	Temps total au format hh:mm:ss.
	weight	Coût de réduction de l'itinéraire.
	lang	Langue du rapport (<i>null</i> , s'il n'y a pas de rapport).
	mindist	Distance minimale entre sous-étapes (<i>null</i> , s'il n'y a pas de rapport).
	coste_v1	Coût des péages véhicule de type 1 (<i>null</i> , s'il n'y a pas de rapport).
	coste_v2	Coût des péages véhicule de type 2 (<i>null</i> , s'il n'y a pas de rapport).
	coste_v3	Coût des péages véhicule de type 3 (<i>null</i> , s'il n'y a pas de rapport).
	stoplist	Liste des arrêts de type poi , geoentity ou mo , ordonnés selon l'itinéraire.
	stoproute	Positions de stoplist ordonnées compte tenu de la position de l'arrêt sur l'itinéraire (peut changer si <i>reorder=true</i>).
	pois	Liste d'objets de type poi par où passe l'itinéraire.
xml	Document XML avec le rapport de l'itinéraire ou <i>null</i> si le rapport n'a pas été demandé.	

Pour afficher les éléments de **stoplist** dans l'ordre précisé par **stoproute** :

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

Exemple : calculer un itinéraire et générer le rapport.

```
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');
    }
}
```

4.5.6 *createReport(xml,idElement,xsl_name)*

Cette méthode applique le modèle *xsl_name* (s'il n'est pas spécifié, cette méthode utilise le modèle défini dans le *config.xml*) au XML qui est passé par paramètre (résultat de la fonction *calculateRoute*) et le résultat est ajouté à l'élément avec *idElement* de HTML.

Exemple : calculer un itinéraire et générer le rapport.

```
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');
  }
}
```

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

Cette méthode retourne l'URL où se trouve une image de la carte avec l'extension courante. La carte apparaît centrée par rapport à l'itinéraire indiqué par le paramètre *routeld*. La couleur (au format hexadécimal) et la largeur de la ligne de l'itinéraire seront ceux indiqués dans le paramètre *lineStyle*(*).

Dans le paramètre *routeParams*, l'attribut « *weight* » et l'attribut « *tolerance* » que l'on souhaite utiliser pour obtenir l'itinéraire doivent être indiqués.

Le paramètre "*true*" de *drawFeatures* indique si l'image doit inclure les figures dessinées sur la carte dans l'extension que l'on visualise.

Le paramètre *execute* définit la fonction qui sera exécutée une fois l'image obtenue. Cette fonction reçoit comme paramètre l'URL de l'image résultante.

Si la valeur de *routeld* reçue est nulle, l'image de la carte est calculée avec l'extension courante et sans itinéraire. Les paramètres *lineStyle* et *routeParams* seront omis.

Si l'on donne une valeur au paramètre *poicats* (chaîne de catégories de POI séparées par des virgules), les points d'intérêt correspondant aux catégories indiquées seront affichés sur l'image de la carte. Ce paramètre peut être omis.

(*) Le serveur de cartes applique un calque de transparence, il est donc recommandé d'augmenter l'intensité de la couleur. Il est également recommandable d'indiquer une largeur de lignes supérieure à quatre pixels.

Voir définitions de *lineStyle* et de *routeParams* dans la description de la fonction *calculateRoute*.

Exemple : calculer un itinéraire et ouvrir une nouvelle fenêtre avec l'image générée.

```
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,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" );
}
```

4.5.8 *calculateMultiRoute(stopList, routeParams, execute)*

Cette méthode calcule les itinéraires optimaux spécifiés par le paramètre *stopList* et retourne les divers coûts (distance, temps, argent).

Elle peut être utilisée pour calculer la distance et le temps entre *n* points d'origine, plusieurs points de passage et *n* destinations. Cela permet, par exemple, de répondre à des questions comme la suivante : Quel véhicule d'une flotte propre mettra le moins de temps pour aller recueillir une marchandise et la transporter jusqu'à sa destination ? Et aussi d'obtenir le temps et la distance calculés.

Le paramètre **stopList** spécifie la liste d'arrêts. Le premier objet de la liste spécifie l'origine et le dernier la destination. Les objets intermédiaires spécifient les arrêts. Chaque élément de la liste peut être spécifié au moyen :

- d'un objet **poi**,
- d'un objet **geoentity**,
- d'un objet **mo**,
- d'une liste d'objets **mo**,
- d'une coordonnée : **{x:<x> ,y:<y>, srs:<srs>}**,
- d'une liste de coordonnées : **[{x:<x> ,y:<y>, srs:<srs>},{x:<x> ,y:<y>, srs:<srs>}]**,
- d'un code d'entité géographique. Ex. : **{cityId: ESP17240300056757 }**,
- d'un code de point d'intérêt : **{poild: C17240332252905}**.

Les trois premiers peuvent être des objets qui sont le résultat d'un géocodage ou d'une proximité.

Le paramètre **routeparams** spécifie la liste d'options pour le calcul de l'itinéraire :

Zone	Description	
weight	Liste de coûts utilisés pour le calcul de l'itinéraire. Le premier d'entre eux spécifie le coût à réduire de l'origine au premier arrêt ; le deuxième spécifie le coût à réduire du premier arrêt au deuxième arrêt et ainsi de suite. S'il n'y a pas assez de valeurs, la dernière valeur est répétée pour le reste des arrêts. On peut aussi spécifier une seule valeur et l'appliquer à toutes les étapes.	
	Valeur	
	Langue	
	distance	L'itinéraire se calcule en réduisant la distance
	time	L'itinéraire se calcule en réduisant le temps
	money	L'itinéraire évite de passer par des péages
intoll	Liste des entiers ou entier. Chaque entier vérifie si l'arrêt correspondant peut ou non se trouver sur un tronçon à péage. Si -1 est spécifié, l'arrêt se trouvera toujours hors d'un tronçon à péage. Si l'on spécifie un nombre positif, l'arrêt pourra se trouver sur un tronçon à péage à condition qu'il ne soit pas à plus de mètres que le nombre indiqué. Si les entiers sont terminés, le dernier est utilisé pour le reste des arrêts. Par exemple, si l'on spécifie intoll=[-1], tous les arrêts seront en dehors des tronçons à péage.	
net	Spécifie le réseau pour le calcul des itinéraires. Ne doit être spécifié que dans des cas particuliers.	
xml	Booléen indiquant si l'on souhaite obtenir le XML de réponse. Par défaut, false .	

Le paramètre **execute** définit la fonction qui sera appelée une fois l'itinéraire résolu avec un paramètre de type **resultCalculateRouteCosts**.

Format d'un **resultCalculateRouteCosts** :

Réponse	Zone	Description
<i>erreur</i>	error	Objet type error (contient <i>null</i> s'il n'y a pas d'erreur)
<i>itinéraires</i>	routes	Liste d'objets itinéraire
	xml	Document XML avec le rapport de l'itinéraire ou <i>null</i> si le rapport n'a pas été demandé.

Format d'un **itinéraire** :

Zone	Description
id :	Identifiant d'entité géographique
distance	Distance totale en mètres de l'itinéraire.
time	Temps total au format hh:mm:ss.
weight	Liste de coûts ou coût de réduction l'itinéraire.
stoplist	Liste des arrêts de type poi , geoentity ou mo ordonnés selon l'itinéraire.

Exemple:

```
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);
}
```

4.5.9 transformCoordinates (objlist,srs)

Cette fonction transforme les coordonnées contenues dans un objet ou une liste d'objets de type **candidate**, **poi** ou **geoentity** en coordonnées du système de référence indiqué.

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

4.5.10 *coordToPixel(obj)*

Cette méthode retourne le pixel de l'image correspondant aux coordonnées indiquées. Il faut indiquer les trois attributs du paramètre (srs, x et y).

Paramètre :

obj - {Object} Objet ayant la structure {x:...,y:...,srs:...}

Exemple :

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

4.5.11 *coordToPixel(obj,srs)*

Cette méthode retourne le pixel de l'image correspondant aux coordonnées indiquées. Il est obligatoire d'indiquer le système de coordonnées utilisé, sinon aucun résultat ne sera obtenu.

Paramètres :

obj - {Object} Objet du type OpenLayers.LonLat (voir : <http://www.openlayers.org>)

srs - {String} Système de coordonnées utilisé

Exemple :

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

5 Style de la carte

Les serveurs Cercalia fournissent différents styles de cartographie de fond pour les cartes. Consultez Nexus Geografics pour obtenir des solutions à vos besoins.

5.1 getStyle()

Cette méthode retourne le nom du style de cartographie de fond.

5.2 setStyle(name)

Cette méthode applique le style indiqué pour la cartographie du fond.

- Le style par défaut de *Cercalia*: `setStyle("default");`
- Le style par défaut d'*OpenStreetMap*: `setStyle("openstreetmap");`

5.3 createWMSStyle (name,url,params,options)

Cette méthode crée un nouveau style de cartographie de fond avec les données d'un serveur WMS.

Paramètre	Description
name	Nom du nouveau style de cartographie.
url	URL du serveur WMS.
params	Objet contenant les paires clé/valeur avec les paramètres nécessaires pour effectuer une requête GETMAP au serveur WMS. Si certains paramètres ne sont pas spécifiés, une valeur par défaut leur est attribuée : service: "WMS" version: "1.1.1" request: "GetMap" styles: "" exceptions: "application/vnd.ogc.se_inimage" format: "image/jpeg" Les paramètres suivants sont obligatoires : srs layers
options	Tableau <i>hash</i> contenant des options extra permettant de configurer le calque. <i>opacity</i> : Nombre réel entre 0.0 et 1.0 spécifiant l'opacité du calque. Par défaut : opaque (1.0). <i>encodeBBOX</i> : <i>true</i> pour que la valeur paramètre BBOX soit codée (le standard WMS avertit qu'elle ne doit pas être codée). Par défaut :

	<p><i>false</i>.</p> <p><i>gutter</i> : nombre de pixels ignorés autour de l'image. Par défaut, 0.</p> <p><i>scales</i> : liste d'échelles valides dans l'ordre descendant pour le serveur WMS. Par défaut, <i>null</i> (toutes les échelles sont autorisées).</p> <p><i>resolutions</i> : liste de résolutions (unités de carte par pixel) dans l'ordre descendant. Si aucune résolution n'est spécifiée, elle sera calculée en fonction de l'extension, de la résolution maximale, de l'échelle maximale, etc.</p> <p><i>maxResolution</i> : <i>real</i> indiquant la résolution maximale. Par défaut, 360 degrés/256 pixels.</p> <p><i>minResolution</i> : <i>real</i> indiquant la résolution minimale.</p> <p><i>numZoomLevels</i> : entier indiquant le nombre de niveaux de zoom.</p> <p><i>minscale</i> : <i>real</i> indiquant le plus petit dénominateur d'échelle.</p> <p><i>maxscale</i> : <i>real</i> indiquant le plus grand dénominateur d'échelle.</p> <p><i>displayOutsideMaxExtent</i> : booléen indiquant s'il faut demander la carte même si l'on est complètement en dehors de l'extension. Par défaut : <i>false</i>.</p> <p><i>maxExtent</i> : <i>array</i> contenant quatre valeurs qui représentent l'extension maximale de la cartographie. Le centre de la carte ne doit en aucun cas se trouver hors de ce rectangle. Par ailleurs, si <i>displayOutsideMaxExtent</i> est activé, on ne demandera pas de cartes hors de cette extension.</p> <p><i>minExtent</i> : <i>array</i> contenant quatre valeurs qui représentent l'extension minimale de la cartographie, dans la projection du serveur WMS.</p> <p>displayName: Nom apparaissant dans le dépliant de choix du type de carte visible.</p> <p>Il est nécessaire d'indiquer l'une des options d'échelle ; de préférence, des <i>scales</i>.</p>
--	--

Exemple d'obtention de l'Orto 50 m de l'ICC (Institut cartographique de Catalogne) :

```
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 Annexe : géocodage sans carte

Il est possible d'effectuer des géocodages sans avoir à lancer la carte.

Pour cela, il faut télécharger le fichier 'cercaliageo.js' sur la page où l'on veut utiliser la fonction contenant l'instruction suivante :

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

ATTENTION ! La page contenant cette référence peut uniquement utiliser la fonction de géocodage de l'API. Le reste des fonctions donnera erreur.

Avant d'utiliser le géocodage, il faut lancer l'API. Le lancement consiste à lire un fichier de propriétés et à appeler le constructeur de **cercaliaclient** en passant par paramètre l'objet contenant les propriétés.

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

À ce stade, on peut appeler la fonction de géocodage :

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

Exemple :

```
<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;
```

```

        var num=dades.num;
        if(!num || num>dades.total)num=dades.total;

while(body_candidates.firstChild)body_candidates.removeChild(body_candidates.firstChild);

        var total = dades.total;
        if(dades.num && dades.num<total) total = dades.num;
        var ncand = parseInt(dades.numCand);
        var poscand = dades.pos;

        if(total>10)//Paginaremos resultado
        {
            var tr_paginar=document.getElementById('paginador');
            var td2 = document.createElement('TD');
            var cont_td = "";
            while(tr_paginar.firstChild)tr_paginar.removeChild(tr_paginar.firstChild);
            td2 = document.createElement('TD');
            for(i=0;i<total;i=i+ncand)
            {
                if(poscand==i)
                {
                    cont_td += "&nbsp;|&nbsp;";
                    decoration:none;"><b>"+"+(i/ncand)+1)+"</b></a>";
                }
                else
                {
                    cont_td += "&nbsp;|&nbsp;";
                    href="javascript:buscarCandPagina(""+i+"",this);\" style="text-decoration:underline;" >"+"+(i/ncand)+1)+"</a>";
                }
                if(i==(total-1)) cont_td += "&nbsp;|&nbsp;";
            }

            td2.innerHTML=cont_td;
            tr_paginar.appendChild(td2);

        }

        for(i=0;i< dades.candidates.length;i++)
        {
            tr=document.createElement('TR');
            td=document.createElement('TD');
            a=document.createElement('A');
            a.appendChild(document.createTextNode(dades.candidates[i].desc));
            a.href="#";
            a.className="opDemo";
            td.appendChild(a);
            tr.appendChild(td);
            body_candidates.appendChild(tr);
        }

        document.getElementById('emergent').style.top='50%';

    }
    else
    {
        alert("No se han encontrado datos.");
    }
}

function buscarCandPagina(poscand)
{
    var search={};
    clone(ultimaGeo,search);
    search.posCand=poscand;
    cercalia.geocoding(search,returnFerGeocoding);
}

```

7 Annexe : systèmes de coordonnées

Valeurs possibles du système de coordonnées (paramètre srs) :

Valeur	Système de coordonnées
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 Annexe : boutons customclick 1/2

customClick<n> sont des boutons de la barre d'outils conçus pour réaliser une action personnalisée en les sélectionnant et en cliquant sur un point de la carte.

Pour assurer un bon fonctionnement de ces boutons, il faut définir la fonction menuCustomClick{n} dans le fichier de configuration et dans le même contexte où l'objet cercalia est déclaré. En cliquant sur la carte, la fonction s'exécute. Cette fonction recevra un objet de type OpenLayers.Event qui, entre autres données, contient le pixel de la carte où l'on a cliqué.

Pour voir les spécifications précises des objets OpenLayers.Event, consulter l'API de documentation d'OpenLayers (<http://www.openlayers.org>).

8.1.1 getCoordFromPixel(pixel,srs)

Cette fonction est conçue pour être appelée dans la fonction menuCustomclick<n>. Elle retourne un objet OpenLayers.LonLat contenant les coordonnées dans le système de référence indiqué par le paramètre **srs** du pixel indiqué dans le paramètre **pixel**. Le paramètre pixel doit être un paramètre de type OpenLayers.Pixel. Voir exemples d'utilisation dans la section précédente.


Exemple : lancer un message contenant les coordonnées géographiques du point cliqué :

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

Exemple : lancer un message contenant les coordonnées Mercator du point cliqué :

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

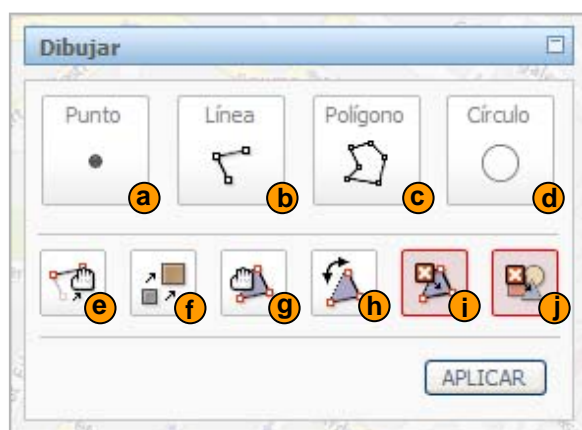
9 Annexe : outils d'édition

La présente annexe explique le fonctionnement de l'outil  (« draw ») ainsi que son interaction avec les méthodes de traitement des formes.

Cet outil permet généralement de dessiner des formes sur la carte en utilisant un style personnalisé (couleurs, style de ligne...). Une fois ces formes créées, on peut consulter les données s'y rapportant par l'intermédiaire des méthodes *getFeatures* et *getFeaturesByState* décrites dans ce manuel.

Il permet en outre d'éditer et donc de modifier (éditer des sommets, faire pivoter, déplacer et supprimer) les formes créées sur la carte à partir de la fonction *createFeature*.

Sur l'image suivante, on va voir l'aspect que présente le menu de fonctionnalités une fois déroulé en activant cet outil :

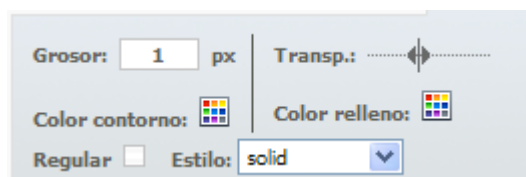


- a) Dessiner des points
- b) Dessiner des lignes
- c) Dessiner des polygones
- d) Dessiner des cercles
- e) Éditer des sommets
- f) Mettre une forme à l'échelle
- g) Déplacer forme
- h) Faire pivoter une forme
- i) Supprimer une forme
- j) Supprimer toute les formes

Le fonctionnement est assez intuitif. Chaque fois que l'on clique sur l'un des boutons, la fonction de ce bouton s'active. Par conséquent, lorsque des évènements sont transmis à la carte à partir de la souris, une action spécifique s'exécute en fonction du bouton activé.

Dans le cas de l'option « j », la fonctionnalité ne reste pas active, toutes les formes créées sur la carte sont tout simplement supprimées (à l'aide de l'outil ou de la fonction *createFeature*).

En activant l'une des fonctionnalités *a*, *b*, *c* ou *d*, des options de personnalisation du style des formes s'affichent. Par exemple, dans le cas *d* :



Pour que ces options s'appliquent, au moment de dessiner les figures sur la carte et après avoir sélectionné le paramétrage souhaité, il faut cliquer sur le bouton « APPLIQUER ».

a) Dessiner des points

Cette fonctionnalité permet de dessiner des points d'une taille et d'une couleur personnalisées.

b) Dessiner des lignes

Cette fonctionnalité permet de dessiner des lignes d'une épaisseur, d'une couleur et d'un style (continu, en pointillé...) personnalisés.

c) Dessiner des polygones

Cette fonctionnalité permet de dessiner des polygones ayant une couleur et une épaisseur de ligne, une couleur de fond, une transparence et un style de ligne personnalisés.

d) Dessiner des cercles

Cette fonctionnalité permet de dessiner des cercles réguliers et irréguliers ayant une couleur et une épaisseur de ligne, une couleur de fond, une transparence et un style de ligne personnalisés.

e) Éditer des sommets

Cette fonctionnalité permet d'éditer les sommets de nos figures afin d'être en mesure d'en modifier la forme.

Si la figure a été créée à l'aide de la fonction *createFeature*, elle passe à l'état 'modified'.

f) Mettre une forme à l'échelle

Cette fonctionnalité permet de modifier la taille de nos figures.

Si la figure a été créée à l'aide de la fonction *createFeature*, elle passe à l'état 'modified'.

g) Déplacer forme

Cette fonctionnalité permet de déplacer nos figures sur la carte.

Si la figure a été créée à l'aide de la fonction *createFeature*, elle passe à l'état 'modified'.

h) Faire pivoter une forme

Cette fonctionnalité permet de faire pivoter nos figures.

Si la figure a été créée à l'aide de la fonction *createFeature*, elle passe à l'état 'modified'.

i) Supprimer une forme

Cette fonctionnalité permet de supprimer une ou plusieurs formes sélectionnées, aussi bien celles créées avec les fonctionnalités *a,b,c* et *d* que celles résultant de la fonction *createFeature*. Elle affiche un message de confirmation avant de réaliser l'action.

Les figures supprimées qui avaient été créées à l'aide de la fonction *createFeature* sont enregistrées dans une structure interne. Leurs données géométriques peuvent alors être récupérées à l'aide de la méthode *getFeatureByState* avec le paramètre 'deleted' à vrai et les autres à faux.

j) Supprimer toute les formes

Cette fonctionnalité permet de supprimer toutes les figures présentes sur la carte (celles créées avec les fonctionnalités *a,b,c* et *d* et celles résultant de la fonction *createFeature*). Elle affiche un message de confirmation avant de réaliser l'action.

Les figures supprimées qui avaient été créées à l'aide de la fonction *createFeature* sont enregistrées dans une structure interne. Leurs données géométriques peuvent alors être récupérées à l'aide de la méthode *getFeatureByState* avec le paramètre 'deleted' à vrai et les autres à faux.

Pour supprimer les figures de manière permanente, il faut utiliser la fonction *deleteFeatures* expliquée dans le manuel.

10 Annexe : différences entre les versions 1 et 2

Cette annexe présente un résumé des modifications effectuées par rapport à la première version de l'API. Certaines étapes sont en outre expliquées afin de pouvoir migrer les projets réalisés avec la première version.

- La tolérance aux serveurs plantés ou inaccessibles a été incorporée. À présent, le script de lancement détecte si le serveur auquel il se connecte est planté et fait un essai avec un autre.
- La possibilité pour le client de communiquer avec un *servlet proxy* a été ajoutée. Le proxy contient la clé de distributeur, ce qui fait que le contrôle d'accès par domaine au serveur Cercalia n'est pas nécessaire si cette option est utilisée.
- Le code de lancement de l'objet Cercalia a été simplifié. Il n'est plus nécessaire de définir les div **barralat**, **foramapa**, **mapa**. L'API les construira automatiquement dans la *div* passée par paramètre au constructeur.
- Le format du fichier de paramétrage XML a été modifié afin qu'il ait des identifiants plus clairs et de nouvelles options ont été ajoutées. Le tableau suivant montre les équivalences entre les paramètres de la première version et la nouvelle. Se reporter au chapitre dans lequel est expliqué le format du fichier de paramétrage afin de voir les valeurs valides de chaque paramètre.

Version antérieure (1.x)	Version en cours (2.0)
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

- La fonction **gotoPoiCode** a été remplacée par la fonction **getPoi**. Cette fonction retourne toutes les données d'un point d'intérêt ou de plusieurs, mais, contrairement à la précédente, elle ne crée pas les repères ni ne centre la carte par rapport à ceux-ci.

- La fonction **pois** a été remplacée par la fonction **showPois**. Contrairement à la fonction précédente, seules les catégories de points d'intérêt que l'on souhaite changer d'état sont passées. Les catégories qui n'ont pas été spécifiées conserveront leur état antérieur.
- La fonction **activateTool** a été ajoutée afin de pouvoir sélectionner par code l'outil activé.
- La méthode **transformCoordinates** qui transforme les coordonnées d'une liste de candidats, de points d'intérêt ou d'entités géographiques en coordonnées du système spécifié a été ajoutée.
- La méthode **createMarkers** qui permet de créer les repères d'une réponse de géocodage, de proximité, etc., a été ajoutée.
- La méthode **moveMarker** a été modifiée en ajoutant un paramètre optionnel afin de changer le contenu du *popup* du repère de manière dynamique. À présent, la fonction **moveMarker** peut servir à actualiser régulièrement et de manière dynamique l'état d'un repère sans pertes de mémoire dans le navigateur.
- Les méthodes qui apparaissent dans l'API actuelle permettent toutes les fonctionnalités de la version antérieure. Les fonctions qui n'apparaissent pas dans le manuel ont été supprimées ou bien ont cessé de fonctionner pour les nouveaux clients. Par exemple, la fonction *proximity* indiquant des coordonnées peut être utilisée à la place de la fonction *inverseGeocoding*.