Wtyczka JMaps Framework jQuery zapewnia proste, ale potężne API dla usług Google Maps.

- **Funkcje:**

- Dane geograficzne i odwrotny adres na świecie za pomocą API Google geokodowania
- Szukaj dowolne miejsca.
- Dodawanie i usuwanie znaczników
- Dodawanie i usuwanie wielokątów i polilinii
- Dodawanie i usuwanie warstwy graficznej na mapie
- Dodawanie i usuwanie warstw Google AdSense
- Dodawanie i usuwanie warstw ruchu
- Uzyskanie informacji z powrotem, takich jak centrum mapy, rozmiar mapy, mapy, itp.

I wiele innych.

1-r64

{codecitation class='brush: js'}/** * @classDescription The Mapifies variable is the main class object for jMaps */ var Mapifies; if (!Mapifies) Mapifies = {}; /** * The main object that holds the maps */ Mapifies.MapObjects = {}; /** * Creates a new map on the passed element with the defined options. Creates a global object that contains the map. * @method * @namespace Mapifies.MapObjects * @id Mapifies.MapObjects.Set * @alias Mapifies.MapObjects.Set * @param {jQuery} element The element that contains the map. * @param {Object} options An object that contains the options. * @return {Object} The object that contains the map. */ Mapifies.MapObjects.Set = function (element, options) { var

mapName = jQuery(element).attr('id'); var thisMap = new GMap2(element); Mapifies.MapObjects[mapName] = thisMap; Mapifies.MapObjects[mapName].Options = options; return Mapifies.MapObjects[mapName]; }; /** * Adds additional objects and functions to an existing MapObject * @method * @namespace Mapifies.MapObjects * @id Mapifies.MapObjects.Append * @alias Mapifies.MapObjects.Append * @param {jQuery} element The element that contains the map * @param {Object} description The name of the object to create * @param {Object} appending The object or function to append */ Mapifies.MapObjects.Append = function (element, description, appending) { var mapName = jQuery(element).attr('id'); Mapifies.MapObjects[mapName][description] = appending; }; /** * Returns the current map object for the passed element * @method * @namespace Mapifies.MapObjects * @id Mapifies.MapObjects.Get * @alias Mapifies.MapObjects.Get * @param {jQuery} element The element that contains the map. * @return {Object} Mapifies The Mapifies object that contains the map. */ Mapifies.MapObjects.Get = function (element) { return Mapifies.MapObjects[jQuery(element).attr('id')]; }; /** * The main function to initialise the map * @method * @namespace Mapifies * @id Mapifies.Initialise * @alias Mapifies.Initialise * @param {jQuery} element The element to initialise the map on. * @param {Object} options The object that contains the options. * @param {Object} callback The callback function to pass out after initialising the map. * @return {Function} callback The callback option with the map object and options. */ Mapifies.Initialise = function (element, options, callback) { /** * Default options for Initialise * @method * @namespace Mapifies.Initialise * @id Mapifies.Initialise.defaults * @alias Mapifies.Initialise.defaults * @param {String} language The locale language for the map * @param {String} mapType The type of map to create. Options are 'map' (default), 'sat' and 'hybrid'. * @param {Object} mapCenter An array that contains the Lat/Lng coordinates of the map center. * @param {Number} mapZoom The initial zoom level of the map. * @param {String} mapControl The option for the map control. The options are 'small' (default), 'large' or 'none' * @param {Boolean} mapEnableType Defines if the buttons for map type are shown. Default false. * @param {Boolean} mapEnableOverview Defines if the map overview is shown. Default false. * @param {Boolean} mapEnableDragging Defines if the map is draggable or not. Default true. * @param {Boolean} mapEnableInfoWindows Defines if info windows are shown on the map or not. Default true. * @param {Boolean} mapEnableDoubleClickZoom Defines if double clicking zooms the map. Default false. * @param {Boolean} mapEnableSmoothZoom Defines if smooth scrolling is enabled. Default false. * @param {Boolean} mapEnableGoogleBar Defines if the google map search tool is enabled. Default false. * param {Boolean} mapEnableScaleControl Defines if the scale bar is shown. Default false. * @param {Boolean} mapShowjMapsIcon Defines if the jMaps icon is shown. Default true. * @param {Boolean} debugMode Defines if the map object created is returned to the Firebug console. Default false. * @return {Object} The options for SearchAddress */ function defaults() { return { // Initial type of map to display 'language': 'en', // Options: "map", "sat", "hybrid" 'mapType': 'map', // Initial map center 'mapCenter': [55.958858,-3.162302], // Initial zoom level 'mapZoom': 12, // Initial map control size // Options: "large", "small", "none" 'mapControl': 'small', // Initialise type of map control 'mapEnableType': false, // Initialise small map overview 'mapEnableOverview': false, // Enable map dragging when left button held down 'mapEnableDragging': true, // Enable map info windows 'mapEnableInfoWindows': true, // Enable double click zooming 'mapEnableDoubleClickZoom': false, // Enable zooming

with scroll wheel 'mapEnableScrollZoom': false, // Enable smooth zoom 'mapEnableSmoothZoom': false, // Enable Google Bar 'mapEnableGoogleBar': false, // Enables scale bar 'mapEnableScaleControl': false, // Enable the Mapifies icon 'mapShowjMapsIcon': true, //Debug Mode 'debugMode': false }; }; options = jQuery.extend(defaults(), options); if (GBrowserIsCompatible()) { var thisMap = Mapifies.MapObjects.Set(element, options); var mapType = Mapifies.GetMapType(options.mapType); thisMap.setCenter(new GLatLng(options.mapCenter[0], options.mapCenter[1]), options.mapZoom, mapType); if (options.mapShowjMapsIcon) { Mapifies.AddScreenOverlay(element, { 'imageUrl':'http://hg.digitalspaghetti.me.uk/jmaps/raw-ile/3228fade0b3c/docs/images/jmaps-mapicon.png', 'screenXY':[70,10], 'overlayXY':[0,0], 'size':[42,25] }); } // Attach a controller to the map view // Will attach a large or small. If any other value passed (i.e. "none") it is ignored switch (options.mapControl) { case "small": thisMap.addControl(new GSmallMapControl()); break; case "large": thisMap.addControl(new GLargeMapControl()); break; }; // Type of map Control (Map,Sat,Hyb) if (options.mapEnableType) thisMap.addControl(new GMapTypeControl()); // Off by default // Show the small overview map if (options.mapEnableOverview) thisMap.addControl(new GOverviewMapControl());// Off by default // GMap2 Functions (in order of the docs for clarity) // Enable a mouse-dragable map if (!options.mapEnableDragging) thisMap.disableDragging(); // On by default // Enable Info Windows if (!options.mapEnableInfoWindows) thisMap.disableInfoWindow(); // On by default // Enable double click zoom on the map if (options.mapEnableDoubleClickZoom) thisMap.enableDoubleClickZoom(); // On by default // Enable scrollwheel on the map if (options.mapEnableScrollZoom) thisMap.enableScrollWheelZoom(); //Off by default // Enable smooth zooming if (options.mapEnableSmoothZoom) thisMap.enableContinuousZoom(); // Off by default // Enable Google Bar if (options.mapEnableGoogleBar) thisMap.enableGoogleBar(); //Off by default // Enables Scale bar if (options.mapEnableScaleControl) thisMap.addControl(new GScaleControl()); if (options.debugMode) console.log(Mapifies); if (typeof callback == 'function') return callback(thisMap, element, options); } else { jQuery(element).text('Your browser does not support Google Maps.'); return false; } return; }; /** * A function to move a map to a passed position * @method * @namespace Mapifies * @id Mapifies.MoveTo * @alias Mapifies.MoveTo * @param {jQuery} element The element to initialise the map on. * @param {Object} options The object that contains the options. * @param {Object} callback The callback function to pass out after initialising the map. * @return {Function} callback The callback option with the map object and options or true. */ Mapifies.MoveTo = function (element, options, callback) { /** * Default options for MoveTo * @method * @namespace Mapifies * @id Mapifies.MoveTo * @alias Mapifies.MoveTo * @param {String} centerMethod The element to initialise the map on. * @param {String} mapType The type of map to create. Options are 'map' (default), 'sat' and 'hybrid'. * @param {Object} mapCenter An array that contains the Lat/Lng coordinates of the map center. * @param {Number} mapZoom The initial zoom level of the map. * @return {Function} callback The callback option with the point object and options or true. */ function defaults() { return { 'centerMethod': 'normal', 'mapType': null, 'mapCenter': [], 'mapZoom': null }; }; var thisMap = Mapifies.MapObjects.Get(element); options = jQuery.extend(defaults(), options); if

(options.mapType) var mapType = Mapifies.GetMapType(options.mapType); var point = new GLatLng(options.mapCenter[0], options.mapCenter[1]); switch (options.centerMethod) { case 'normal': thisMap.setCenter(point, options.mapZoom, mapType); break; case 'pan': thisMap.panTo(point); break; } if (typeof callback == 'function') return callback(point, options); }; /** * Save your current position on the map * @method * @namespace Mapifies * @id Mapifies.SavePosition * @alias Mapifies.SavePosition * @param {jQuery} element The element to initialise the map on. * @param {Object} options The object that contains the options. * @param {Object} callback The callback function to pass out after initialising the map. * @return {Function} callback The callback option with the map object and options or true. */ Mapifies.SavePosition = function(element, options, callback) { var thisMap = Mapifies.MapObjects.Get(element); thisMap.savePosition(); if (typeof callback == 'function') return callback(thisMap); }; /** * Goto a previously saved position * @method * @namespace Mapifies * @id Mapifies.GotoSavedPosition * @alias Mapifies.GotoSavedPosition * @param {jQuery} element The element to initialise the map on. * @param {Object} options The object that contains the options. * @param {Function} callback The callback function to pass out after initialising the map. * @return {Function} callback The callback option with the map object and options or true. */ Mapifies.GotoSavedPosition = function (element, options, callback) { var thisMap = Mapifies.MapObjects.Get(element); thisMap.returnToSavedPosition(); if (typeof callback == 'function') return callback(thisMap); }; /** * Create a keyboard handler to handle keyboard navigation * @method * @namespace Mapifies * @id Mapifies.CreateKeyboardHandler * @alias Mapifies.CreateKeyboardHandler * @param {jQuery} element The element to initialise the map on. * @param {Object} options The object that contains the options. * @param {Object} callback The callback function to pass out after initialising the map. * @return {Function} callback The callback option with the keyboard handler. */ Mapifies.CreateKeyboardHandler = function(element, options, callback) { var thisMap = Mapifies.MapObjects.Get(element); var keyboardHandler = new GKeyboardHandler(thisMap); if (typeof callback == 'function') return callback(keyboardHandler); }; /** * The SearchAddress function takes a map, options and callback function. The options can contain either an address string, to which a point is returned - or reverse geocoding a GLatLng, where an address is returned * @method * @namespace Mapifies * @id Mapifies.SearchAddress * @param {jQuery} element The jQuery object containing the map element. * @param {Object} options An object of options * @param {Function} callback The callback function that returns the result * @return {Function} Returns a passed callback function or true if no callback specified */ Mapifies.SearchAddress = function(element, options, callback) { /** * Default options for SearchAddress * @method * @namespace Mapifies.SearchAddress * @id Mapifies.SearchAddress.defaults * @alias Mapifies.SearchAddress.defaults * @param {String} query The Address or GLatLng to query in the geocoder * @param {String} returnType The type of value you want to return from Google. This is mapped to the function names available, the options are 'getLatLng' which returns coordinates, and 'getLocations' which returns points. * @param {GGeoCache} cache The GGeoCache to store the results in if required * @param {String} countryCode The country code to localise results * @return {Object} The options for SearchAddress */ function defaults() { return { // Address to search for 'query': null, // Return Type 'returnType': 'getLatLng', // Optional Cache to store Geocode Data (not implemented yet) 'cache': undefined, // Country code for localisation (not implemented yet) 'countryCode': 'uk' }; }; var thisMap =

Mapifies.MapObjects.Get(element); options = jQuery.extend(defaults(), options); // Check to see if the Geocoder already exists in the object // or create a temporary locally scoped one. if (typeof thisMap.Geocoder === 'undefined') { if (typeof options.cache === 'undefined') { var geoCoder = new GClientGeocoder(); } else { var geoCoder = new GClientGeocoder(cache); } Mapifies.MapObjects.Append(element, 'Geocoder', geoCoder); // We need to get the map object again, now we have attached the geocoder thisMap = Mapifies.MapObjects.Get(element); } thisMap.Geocoder[options.returnType](options.query, function(result){ if (typeof callback === 'function') { return callback(result, options); } }); return; }; /** * The SearchDirections function allows you to search for directions between two or more points and return it to a map and a directions panel * @method * @namespace Mapifies * @id Mapifies.SearchDirections * @param {jQuery} element The jQuery object containing the map element. * @param {Object} options An object of options * @param {Function} callback The callback function that returns the result * @return {Function} Returns a passed callback function or true if no callback specified */ Mapifies.SearchDirections = function(element, options, callback) { /** * Default options for SearchDirections * @method * @namespace Mapifies.SearchDirections * @id Mapifies.SearchDirections.defaults * @alias Mapifies.SearchDirections.defaults * @param {String} query The directions query to parse. Must contain one 'from:' and one 'to:' query, but can contain multiple 'to:' queries. * @param {String} panel The ID of the panel that the directions will be sent to. * @param {String} local The local for the directions. * @param {String} travelMode Allows you to specify the travel mode, either 'driving' or 'walking'. Driving is the default. * @param {Boolean} avoidHighways Allows you to avoid Highways/Motorway's on trips. Please note this may not always be possible depending on the route. * @param {Boolean} getPolyline Decides if the returned result will draw a polyline on the map on the journey. Default is True. * @param {Boolean} getSteps Decides if the textual directions are returned to the directions panel. * @param {Boolean} preserveViewport Decides if the map will zoom and center in on the directions results. * @param {Boolean} clearLastSearch Clears the last direction search if you do not want to have multiple points. * @return {Object} The options for SearchDirections */ function defaults() { return { // From address 'query': null, // Optional panel to show text directions 'panel': null, //The locale to use for the directions result. 'locale': 'en_GB', //The mode of travel, such as driving (default) or walking 'travelMode': 'driving', // Option to avoid highways 'avoidHighways': false, // Get polyline 'getPolyline': true, // Get directions 'getSteps': true, // Preserve Viewport 'preserveViewport' : false, // clear last search 'clearLastSearch' : false }; }; var thisMap = Mapifies.MapObjects.Get(element); options = jQuery.extend(defaults(), options); var queryOptions = { 'locale': options.locale, 'travelMode': options.travelMode, 'avoidHighways': options.avoidHighways, 'getPolyline': options.getPolyline, 'getSteps': options.getSteps, 'preserveViewport' : options.preserveViewport }; var panel = $(options.panel).get(0); if (typeof thisMap.Directions === 'undefined') { Mapifies.MapObjects.Append(element, 'Directions', new GDirections(thisMap, panel)); } GEvent.addListener(thisMap.Directions, "load", onLoad); GEvent.addListener(thisMap.Directions, "error", onError); if (options.clearLastSearch) { thisMap.Directions.clear(); } thisMap.Directions.load(options.query, queryOptions); function onLoad() { if (typeof callback == 'function') return callback(thisMap.Directions, options); } function onError() { if (typeof callback == 'function') return callback(thisMap.Directions, options); } return; }; /** * Create an adsense ads manager for the map. The Adsense manager will parse your page

and show adverts on the map that relate to this. Requires your adsense publisher id and channel * @method * @namespace Mapifies * @id Mapifies.CreateAdsManager * @param {jQuery} element The jQuery object containing the map element. * @param {Object} options An object of options * @param {Function} callback The callback function that returns the result * @return {Function} Returns a passed callback function or true if no callback specified */ Mapifies.CreateAdsManager = function(element, options, callback) { /** * Default options for CreateAdsManager * @method * @namespace Mapifies.CreateAdsManager * @id Mapifies.CreateAdsManager.defaults * @alias Mapifies.CreateAdsManager.defaults * @param {String} publisherId Your Adsense publisher ID * @param {Number} maxAdsOnMap The maximum number of ads to show on the map at one time * @param {Number} channel The AdSense channel this belongs to * @param {Number} minZoomLevel The minimum zoom level to begin showing ads at * @return {Object} The options for CreateAdsManager */ function defaults() { return { 'publisherId':'', 'maxAdsOnMap':3, 'channel':0, 'minZoomLevel':6 } }; var thisMap = Mapifies.MapObjects.Get(element); options = jQuery.extend(defaults(), options); var adsOptions = { 'maxAdsOnMap':options.maxAdsOnMap, 'channel':options.channel, 'minZoomLevel':options.minZoomLevel } if (typeof thisMap.AdsManager == 'undefined') { Mapifies.MapObjects.Append(element, 'AdsManager', new GAdsManager(thisMap, options.publisherId, adsOptions)); } if (typeof callback == 'function') return callback(thisMap.AdsManager, options); }; /** * This function allows you to pass a GeoXML or KML feed to a Google map. * @method * @namespace Mapifies * @id Mapifies.AddFeed * @alias Mapifies.AddFeed * @param {jQuery} element The element to initialise the map on. * @param {Object} options The object that contains the options. * @param {Fucntion} callback The callback function to pass out after initialising the map. * @return {Function} callback The callback option with the feed object and options. */ Mapifies.AddFeed = function(element, options, callback) { /** * Default options for AddFeed * @method * @namespace Mapifies.AddFeed * @id Mapifies.AddFeed.defaults * @alias Mapifies.AddFeed.defaults * @param {String} feedUrl The URL of the GeoXML or KML feed. * @param {Object} mapCenter An array with a lat/lng position to center the map on * @return {Object} The options for AddFeed */ function defaults() { return { // URL of the feed to pass (required) 'feedUrl': null, // Position to center the map on (optional) 'mapCenter': [] }; }; var thisMap = Mapifies.MapObjects.Get(element); options = jQuery.extend(defaults(), options); // Load feed var feed = new GGeoXml(options.feedUrl); // Add as overlay thisMap.addOverlay(feed); // If the user has passed the optional mapCenter, // then center the map on that point if (options.mapCenter[0] && options.mapCenter[1]) thisMap.setCenter(new GLatLng(options.mapCenter[0], options.mapCenter[1])); if (typeof callback == 'function') return callback(feed, options); return; }; /** * This function allows you to remove a GeoXML or KML feed from a Google map. * @method * @namespace Mapifies * @id Mapifies.RemoveFeed * @alias Mapifies.RemoveFeed * @param {jQuery} element The element to initialise the map on. * @param {GGeoXML} feed The feed to remove from the map * @param {Function} callback The callback function to pass out after initialising the map. * @return {Function} callback The callback option with the feed object and options. */ Mapifies.RemoveFeed = function (element, feed, callback) { var thisMap = Mapifies.MapObjects.Get(element); thisMap.removeOverlay(feed); if (typeof callback == 'function') return callback(feed); return; }; /** * This function allows you to add a ground overlay to a map * @method * @namespace Mapifies * @id Mapifies.AddGroundOverlay *

@alias Mapifies.AddGroundOverlay * @param {jQuery} element The element to initialise the map on. * @param {Object} options The object that contains the options. * @param {Function} callback The callback function to pass out after initialising the map. * @return {Function} callback The callback option with the feed object and options. */ Mapifies.AddGroundOverlay = function(element, options, callback) { /** * Default options for AddGroundOverlay * @method * @namespace Mapifies.AddGroundOverlay * @id Mapifies.AddGroundOverlay.defaults * @alias Mapifies.AddGroundOverlay.defaults * @param {Object} overlaySouthWestBounds The coordinates of the South West bounds of the image * @param {Object} overlayNorthEastBounds The coordinates of the North East bounds of the image * @param {String} overlayImage The URL of the image to be loaded * @return {Object} The options for AddGroundOverlay */ function defaults() { return { // South West Boundry 'overlaySouthWestBounds': undefined, // North East Boundry 'overlayNorthEastBounds': undefined, // Image 'overlayImage': undefined }; }; var thisMap = Mapifies.MapObjects.Get(element); options = jQuery.extend(defaults(), options); var boundries = new GLatLngBounds(new GLatLng(options.overlaySouthWestBounds[0], options.overlaySouthWestBounds[1]), new GLatLng(options.overlayNorthEastBounds[0], options.overlayNorthEastBounds[1])); groundOverlay = new GGroundOverlay(options.overlayImage, boundries); thisMap.addOverlay(groundOverlay); if (typeof callback == 'function') return callback(groundOverlay, options); return; }; /** * This function removes an existing ground overlay * @method * @namespace Mapifies * @id Mapifies.RemoveGroundOverlay * @alias Mapifies.RemoveGroundOverlay * @param {jQuery} element The element to initialise the map on. * @param {GGroundOverlay} groundOverlay The ground overlay to remove. * @param {Function} callback The callback function to pass out after initialising the map. * @return {Function} callback The callback option with the feed object and options. */ Mapifies.RemoveGroundOverlay = function (element, groundOverlay, callback) { var thisMap = Mapifies.MapObjects.Get(element); thisMap.removeOverlay(groundOverlay); if (typeof callback === 'function') return callback(groundOverlay); return; }; /** * This function allows you to add markers to the map with several options * @method * @namespace Mapifies * @id Mapifies.AddMarker * @alias Mapifies.AddMarker * @param {jQuery} element The element to initialise the map on. * @param {Object} options The object that contains the options. * @param {Function} callback The callback function to pass out after initialising the map. * @return {Function} callback The callback option with the marker object and options. */ Mapifies.AddMarker = function (element, options, callback) { /** * Default options for AddGroundOverlay * @method * @namespace Mapifies.AddGroundOverlay * @id Mapifies.AddGroundOverlay.defaults * @alias Mapifies.AddGroundOverlay.defaults * @param {Object} pointLatLng The Lat/Lng coordinates of the marker. * @param {String} pointHTML The HTML to appear in the markers info window. * @param {String} pointOpenHTMLEvent The javascript event type to open the marker info window. Default is 'click'. * @param {Boolean} pointIsDraggable Defines if the point is draggable by the end user. Default false. * @param {Boolean} pointIsRemovable Defines if the point can be removed by the user. Default false. * @param {Boolean} pointRemoveEvent The event type to remove a marker. Default 'dblclick'. * @param {Number} pointMinZoom The minimum zoom level to display the marker if using a marker manager. * @param {Number} pointMaxZoom The maximum zoom level to display the marker if using a marker manager. * @param {GIcon} pointIcon A GIcon to display instead of the standard marker graphic. * @param {Boolean} centerMap Automatically center the map

on the new marker. Default false. * @param {String} centerMoveMethod The method in which to move to the marker. Options are 'normal' (default) and 'pan' * @return {Object} The options for AddGroundOverlay */ function defaults() { var values = { 'pointLatLng': undefined, 'pointHTML': undefined, 'pointOpenHTMLEvent': 'click', 'pointIsDraggable': false, 'pointIsRemovable': false, 'pointRemoveEvent': 'dblclick', 'pointMinZoom': 4, 'pointMaxZoom': 17, 'pointIcon': undefined, 'centerMap': false, 'centerMoveMethod':'normal' }; return values; }; var thisMap = Mapifies.MapObjects.Get(element); options = jQuery.extend({}, defaults(), options); var markerOptions = {} if (typeof options.pointIcon == 'object') jQuery.extend(markerOptions, {'icon': options.pointIcon}); if (options.pointIsDraggable) jQuery.extend(markerOptions, {'draggable': options.pointIsDraggable}); if (options.centerMap) { switch (options.centerMoveMethod) { case 'normal': thisMap.setCenter(new GLatLng(options.pointLatLng[0],options.pointLatLng[1])); break; case 'pan': thisMap.panTo(new GLatLng(options.pointLatLng[0],options.pointLatLng[1])); break; } } // Create marker, optional parameter to make it draggable var marker = new GMarker(new GLatLng(options.pointLatLng[0],options.pointLatLng[1]), markerOptions); // If it has HTML to pass in, add an event listner for a click if(options.pointHTML) GEvent.addListener(marker, options.pointOpenHTMLEvent, function(){ marker.openInfoWindowHtml(options.pointHTML, {maxContent: options.pointMaxContent, maxTitle: options.pointMaxTitle}); }); // If it is removable, add dblclick event if(options.pointIsRemovable) GEvent.addListener(marker, options.pointRemoveEvent, function(){ thisMap.removeOverlay(marker); }); // If the marker manager exists, add it if(thisMap.MarkerManager) { thisMap.MarkerManager.addMarker(marker, options.pointMinZoom, options.pointMaxZoom); } else { // Direct rendering to map thisMap.addOverlay(marker); } if (typeof callback == 'function') return callback(marker, options); return; }; /** * This function allows you to remove markers from the map * @method * @namespace Mapifies * @id Mapifies.RemoveMarker * @alias Mapifies.RemoveMarker * @param {jQuery} element The element to initialise the map on. * @param {GMarker} options The marker to be removed * @param {Function} callback The callback function to pass out after initialising the map. * @return {Function} callback The callback option with the marker object. */ Mapifies.RemoveMarker = function (element, marker, callback) { var thisMap = Mapifies.MapObjects.Get(element); thisMap.removeOverlay(marker); if (typeof callback === 'function') return callback(marker); return; }; /** * This function allows you to create a marker manager to store and manage any markers created on the map. Google recommends not using this marker manager and instead using the open source one. * @method * @deprecated * @namespace Mapifies * @id Mapifies.CreateMarkerManager * @alias Mapifies.CreateMarkerManager * @param {jQuery} element The element to initialise the map on. * @param {GMarker} options The marker to be removed * @param {Function} callback The callback function to pass out after initialising the map. * @return {Function} callback The callback option with the marker object and options. */ Mapifies.CreateMarkerManager = function(element, options, callback) { /** * Default options for CreateMarkerManager * @method * @namespace Mapifies.CreateMarkerManager * @id Mapifies.CreateMarkerManager.defaults * @alias Mapifies.CreateMarkerManager.defaults * @param {Number} borderPadding Specifies, in pixels, the extra padding outside the map's current viewport monitored by a manager. Markers that fall within this padding are added to the map, even if they are not fully visible. * @param {Number} maxZoom The maximum zoom

level to show markers at * @param {Boolean} trackMarkers Indicates whether or not a marker manager should track markers' movements. * @return {Object} The options for CreateMarkerManager */ function defaults() { return { // Border Padding in pixels 'borderPadding': 100, // Max zoom level 'maxZoom': 17, // Track markers 'trackMarkers': false } } var thisMap = Mapifies.MapObjects.Get(element); options = jQuery.extend(defaults(), options); var markerManager = new GMarkerManager(thisMap, options); Mapifies.MapObjects.Append(element, 'MarkerManager',markerManager); // Return the callback if (typeof callback == 'function') return callback(markerManager, options); }; /** * This function allows you to add a polygon to a map using GLatLng points * @method * @namespace Mapifies * @id Mapifies.AddPolygon * @alias Mapifies.AddPolygon * @param {jQuery} element The element to initialise the map on. * @param {Object} options The object that contains the options. * @param {Function} callback The callback function to pass out after initialising the map. * @return {Function} callback The callback option with the polygon object, polygon options and options. */ Mapifies.AddPolygon = function(element, options, callback) { /** * Default options for AddPolygon * @method * @namespace Mapifies.AddPolygon * @id Mapifies.AddPolygon.defaults * @alias Mapifies.AddPolygon.defaults * @param {Object} polygonPoints An array of Lat/Lng points that make up the vertexes of the polygon. * @param {String} polygonStrokeColor The stroke colour for the polygon. * @param {Number} polygonStrokeWeight The thickness of the polygon line. * @param {Number} polygonStrokeOpacity A value from 0 to 1 of for the line opacity. * @param {String} polygonFillColor The colour of the fill area for the polygon. * @param {Number} polygonFillOpacity The value from 0 to 1 for the polygon fill opacity. * @param {Object} mapCenter An array containing the LatLng point to center on. * @param {Boolean} polygonClickable Defines if the polygon is clickable or not. Default true. * @return {Object} The options for AddPolygon */ function defaults() { return { // An array of GLatLng objects 'polygonPoints': [], // The outer stroke colour 'polygonStrokeColor': "#000000", // Stroke thickness 'polygonStrokeWeight': 5, // Stroke Opacity 'polygonStrokeOpacity': 1, // Fill colour 'polygonFillColor': "#ff0000", // Fill opacity 'polygonFillOpacity': 1, // Optional center map 'mapCenter': undefined, // Is polygon clickable? 'polygonClickable': true } } var thisMap = Mapifies.MapObjects.Get(element); options = jQuery.extend(defaults(), options); var polygonOptions = {}; if (!options.polygonClickable) polygonOptions = jQuery.extend(polygonOptions, {clickable: false}); if(typeof options.mapCenter !== 'undefined' && options.mapCenter[0] && options.mapCenter[1]) thisMap.setCenter(new GLatLng(options.mapCenter[0], options.mapCenter[1])); var allPoints = []; jQuery.each(options.polygonPoints, function(i, point) { allPoints.push(new GLatLng(point[0],point[1])); }); var polygon = new GPolygon(allPoints, options.polygonStrokeColor, options.polygonStrokeWeight, options.polygonStrokeOpacity, options.polygonFillColor, options.polygonFillOpacity, polygonOptions); thisMap.addOverlay(polygon); if (typeof callback == 'function') return callback(polygon, polygonOptions, options); return; } /** * This function allows you to remove a polygon from the map * @method * @namespace Mapifies * @id Mapifies.RemovePolygon * @alias Mapifies.RemovePolygon * @param {jQuery} element The element to initialise the map on. * @param {GPolygon} polygon The polygon to be removed * @param {Function} callback The callback function to pass out after initialising the map. * @return {Function} callback The callback option with the polygon. */ Mapifies.RemovePolygon = function (element, polygon,

callback) { var thisMap = Mapifies.MapObjects.Get(element); thisMap.removeOverlay(polygon); if (typeof callback === 'function') return callback(polygon); return; }; /** * This function allows you to add a polyline to a map using GLatLng points * @method * @namespace Mapifies * @id Mapifies.AddPolyline * @alias Mapifies.AddPolyline * @param {jQuery} element The element to initialise the map on. * @param {Object} options The object that contains the options. * @param {Function} callback The callback function to pass out after initialising the map. * @return {Function} callback The callback option with the polygon object, polygon options and options. */ Mapifies.AddPolyline = function (element, options, callback) { /** * Default options for AddPolyline * @method * @namespace Mapifies.AddPolyline * @id Mapifies.AddPolygon.defaults * @alias Mapifies.AddPolygon.defaults * @param {Object} polylinePoints An array of Lat/Lng points that make up the vertexes of the polyline. * @param {String} polylineStrokeColor The stroke colour for the polyline. * @param {Number} polylineStrokeWidth The thickness of the polyline line. * @param {Number} polylineStrokeOpacity A value from 0 to 1 of for the line opacity. * @param {Object} mapCenter An array containing the LatLng point to center on. * @param {Boolean} polylineGeodesic Defines if the line follows the curve of the earth. Default false. * @param {Boolean} polylineClickable Defines if the polygon is clickable or not. Default true. * @return {Object} The options for AddPolyline */ function defaults() { return { // An array of GLatLng objects 'polylinePoints': [], // Colour of the line 'polylineStrokeColor': "#ff0000", // Width of the line 'polylineStrokeWidth': 10, // Opacity of the line 'polylineStrokeOpacity': 1, // Optional center map 'mapCenter': [], // Is line Geodesic (i.e. bends to the curve of the earth)? 'polylineGeodesic': false, // Is line clickable? 'polylineClickable': true }; }; var thisMap = Mapifies.MapObjects.Get(element); options = jQuery.extend(defaults(), options); var polyLineOptions = {}; if (options.polylineGeodesic) jQuery.extend(polyLineOptions, {geodesic: true}); if(!options.polylineClickable) jQuery.extend(polyLineOptions, {clickable: false}); if (options.mapCenter[0] && options.mapCenter[1]) thisMap.setCenter(new GLatLng(options.mapCenter[0], options.mapCenter[1])); var allPoints = []; jQuery.each(options.polylinePoints, function(i, point) { allPoints.push(new GLatLng(point[0],point[1])); }); var polyline = new GPolyline(allPoints, options.polylineStrokeColor, options.polylineStrokeWidth, options.polylineStrokeOpacity, polyLineOptions); thisMap.addOverlay(polyline); if (typeof callback == 'function') return callback(polyline, polyLineOptions, options); return; } /** * This function allows you to remove a polyline from the map * @method * @namespace Mapifies * @id Mapifies.RemovePolyline * @alias Mapifies.RemovePolyline * @param {jQuery} element The element to initialise the map on. * @param {GPolyline} polyline The polyline to be removed * @param {Function} callback The callback function to pass out after initialising the map. * @return {Function} callback The callback option with the polyline. */ Mapifies.RemovePolyline = function (element, polyline, callback) { var thisMap = Mapifies.MapObjects.Get(element); thisMap.removeOverlay(polyline); if (typeof callback === 'function') return callback(polyline); return; }; /** * This function allows you to add a screen overlay to a map. * @method * @namespace Mapifies * @id Mapifies.AddScreenOverlay * @alias Mapifies.AddScreenOverlay * @param {jQuery} element The element to initialise the map on. * @param {Object} options The object that contains the options. * @param {Function} callback The callback function to pass out after initialising the map. * @return {Function} callback The callback option with the screen overlay and options. */ Mapifies.AddScreenOverlay = function(element, options, callback) { /** * Default options for

AddScreenOverlay * @method * @namespace Mapifies.AddScreenOverlay * @id Mapifies.AddScreenOverlay.defaults * @alias Mapifies.AddScreenOverlay.defaults * @param {String} imageUrl The URL of the image to load. * @param {Object} screenXY The X/Y position in the viewport to place the image. * @param {Object} overlayXY The overlay X/Y position in the viewport. * @param {Object} size The size of the image, which is converted to a GSize. * @return {Object} The options for AddScreenOverlay */ function defaults() { return { 'imageUrl':'', 'screenXY':[], 'overlayXY':[], 'size':[] }; }; var thisMap = Mapifies.MapObjects.Get(element); options = jQuery.extend(defaults(), options); var overlay = new GScreenOverlay(options.imageUrl, new GScreenPoint(options.screenXY[0],options.screenXY[1]), new GScreenPoint(options.overlayXY[0],options.overlayXY[1]), new GScreenSize(options.size[0],options.size[1])); thisMap.addOverlay(overlay); if (typeof callback == 'function') return callback(overlay, options); }; /** * This function allows you to remove a screen overlay from the map * @method * @namespace Mapifies * @id Mapifies.RemoveScreenOverlay * @alias Mapifies.RemoveScreenOverlay * @param {jQuery} element The element to initialise the map on. * @param {GScreenOverlay} overlay The overlay to be removed * @param {Function} callback The callback function to pass out after initialising the map. * @return {Function} callback The callback option with the overlay. */ Mapifies.RemoveScreenOverlay = function (element, overlay, callback) { var thisMap = Mapifies.MapObjects.Get(element); thisMap.removeOverlay(overlay); if (typeof callback === 'function') return callback(overlay); return; }; /** * This function allows you to add a Google Streetview * @method * @namespace Mapifies * @id Mapifies.CreateStreetviewPanorama * @alias Mapifies.CreateStreetviewPanorama * @param {jQuery} element The element to initialise the map on. * @param {Object} options The object that contains the options. * @param {Function} callback The callback function to pass out after initialising the map. * @return {Function} callback The callback option with the street view. */ Mapifies.CreateStreetviewPanorama = function(element, options, callback) { /** * Default options for CreateStreetviewPanorama * @method * @namespace Mapifies.CreateStreetviewPanorama * @id Mapifies.CreateStreetviewPanorama.defaults * @alias Mapifies.CreateStreetviewPanorama.defaults * @param {String} overideContainer A ID of a div to put the street view into, otherwise it will default to the map. * @param {Object} latlng The starting Lat/Lng of the streetview - this is required. * @param {Object} pov The point of view to initialse the map on. This is 3 values, X/Y/Z * @return {Object} The options for CreateStreetviewPanorama */ function defaults() { return { 'overideContainer':'', 'latlng':[40.75271883902363, -73.98262023925781], 'pov': []} }; var thisMap = Mapifies.MapObjects.Get(element); options = jQuery.extend(defaults(), options); // Create Street View Overlay var container = null; if (options.overideContainer !== '') { container = jQuery(options.overideContainer).get(0); } else { container = jQuery(element).get(0); } var viewOptions = {}; if (options.pov.length > 0) { jQuery.extend(viewOptions, {'pov':new GPov(options.latlng[0],options.latlng[1],options.latlng[2])}); } if (options.latlng.length > 0) { jQuery.extend(viewOptions, {'latlng':new GLatLng(options.latlng[0],options.latlng[1])}); } var overlay = new GStreetviewPanorama(container, viewOptions); if (typeof callback == 'function') return callback(overlay, options); return; }; /** * This function allows you to remove a street view from the map * @method * @namespace Mapifies * @id Mapifies.RemoveStreetviewPanorama * @alias Mapifies.RemoveStreetviewPanorama * @param {jQuery} element The element to initialise the map on. * @param {GStreetView} view

The view to be removed * @param {Function} callback The callback function to pass out after initialising the map. * @return {Function} callback The callback option with the view. */ Mapifies.RemoveStreetviewPanorama = function (element, view, callback) { var thisMap = Mapifies.MapObjects.Get(element); view.remove(); if (typeof callback == 'function') return callback(view); return; }; /** * This function allows you to add a Google Traffic Layer * @method * @namespace Mapifies * @id Mapifies.AddTrafficInfo * @alias Mapifies.AddTrafficInfo * @param {jQuery} element The element to initialise the map on. * @param {Object} options The object that contains the options. * @param {Function} callback The callback function to pass out after initialising the map. * @return {Function} callback The callback option with the traffic layer. */ Mapifies.AddTrafficInfo = function(element, options, callback) { /** * Default options for AddTrafficInfo * @method * @namespace Mapifies.AddTrafficInfo * @id Mapifies.AddTrafficInfo.defaults * @alias Mapifies.AddTrafficInfo.defaults * @param {Object} mapCenter The Lat/Lng to center the map on * @return {Object} The options for AddTrafficInfo */ function defaults() { return { // Center the map on this point (optional) 'mapCenter': [] }; }; var thisMap = Mapifies.MapObjects.Get(element); options = jQuery.extend(defaults(), options); var trafficOverlay = new GTrafficOverlay; // Add overlay thisMap.addOverlay(trafficOverlay); // If the user has passed the optional mapCenter, // then center the map on that point if (options.mapCenter[0] && options.mapCenter[1]) { thisMap.setCenter(new GLatLng(options.mapCenter[0], options.mapCenter[1])); } if (typeof callback == 'function') return callback(trafficOverlay, options); }; /** * This function allows you to remove a traffic layer from the map * @method * @namespace Mapifies * @id Mapifies.RemoveTrafficInfo * @alias Mapifies.RemoveTrafficInfo * @param {jQuery} element The element to initialise the map on. * @param {GTrafficOverlay} trafficOverlay The traffic overlay to be removed * @param {Function} callback The callback function to pass out after initialising the map. * @return {Function} callback The callback option with the traffic overlay. */ Mapifies.RemoveTrafficInfo = function (element, trafficOverlay, callback) { var thisMap = Mapifies.MapObjects.Get(element); thisMap.removeOverlay(trafficOverlay); if (typeof callback === 'function') return callback(trafficOverlay); return; }; /** * A helper method that allows you to pass the status code of a search and get back a friendly oject * @method * @namespace Mapifies * @id Mapifies.SearchCode * @param {Number} code The status code of the query * @return {Object} Returns a friendly object that contains the 'code', a 'success' boolean and a helpful 'message'. */ Mapifies.SearchCode = function (code) { switch (code) { case G_GEO_SUCCESS: return {'code':G_GEO_SUCCESS,'success':true,'message':'Success'}; case G_GEO_UNKNOWN_ADDRESS: return {'code' : G_GEO_UNKNOWN_ADDRESS, 'success' : false, 'message' : 'No corresponding geographic location could be found for one of the specified addresses. This may be due to the fact that the address is relatively new, or it may be incorrect'}; break; case G_GEO_SERVER_ERROR: return {'code' : G_GEO_UNKNOWN_ADDRESS, 'success' : false, 'message' : 'A geocoding or directions request could not be successfully processed, yet the exact reason for the failure is not known.'}; break; case G_GEO_MISSING_QUERY: return {'code' : G_GEO_UNKNOWN_ADDRESS, 'success' : false, 'message' : 'The HTTP q parameter was either missing or had no value. For geocoder requests, this means that an empty address was specified as input. For directions requests, this means that no query was specified in the input.'}; break; case G_GEO_BAD_KEY: return {'code' :

G_GEO_UNKNOWN_ADDRESS, 'success' : false, 'message' : 'The given key is either invalid or does not match the domain for which it was given.'}; break; case G_GEO_BAD_REQUEST: return {'code' : G_GEO_UNKNOWN_ADDRESS, 'success' : false, 'message' : 'A directions request could not be successfully parsed.'}; break; default: return { 'code': null, 'success': false, 'message': 'An unknown error occurred.' }; break; }; } /** * An internal function to get the google maptype constant * @method * @namespace Mapifies * @id Mapifies.GetMapType * @alias Mapifies.GetMapType * @param {String} mapType The string of the map type. * @return {String} mapType The Google constant for a maptype. */ Mapifies.GetMapType = function (mapType) { // Lets set our map type based on the options switch(mapType) { case 'map': // Normal Map mapType = G_NORMAL_MAP; break; case 'sat': // Satallite Imagery mapType = G_SATELLITE_MAP; break; case 'hybrid': //Hybrid Map mapType = G_HYBRID_MAP; break; }; return mapType; }; /** * An internal function to get the google travel mode constant * @method * @namespace Mapifies * @id Mapifies.GetTravelMode * @alias Mapifies.GetTravelMode * @param {String} travelMode The string of the travel mode. * @return {String} travelMode The Google constant for a travel mode. */ Mapifies.GetTravelMode = function (travelMode) { switch(travelMode) { case 'driving': travelMode = G_TRAVEL_MODE_DRIVING; break; case 'walking': travelMode = G_TRAVEL_MODE_WALKING; break; }; return travelMode; }; /** * A helper function to create a google GIcon * @method * @namespace Mapifies * @id Mapifies.createIcon * @alias Mapifies.createIcon * @param {Object} options The options to create the icon * @return {GIcon} A GIcon object */ Mapifies.createIcon = function (options) { /** * Default options for createIcon * @method * @namespace Mapifies.createIcon * @id Mapifies.createIcon.defaults * @alias Mapifies.createIcon.defaults * @param {String} iconImage The foreground image URL of the icon. * @param {String} iconShadow The shadow image URL of the icon. * @param {GSize} iconSize The pixel size of the foreground image of the icon. * @param {GSize} iconShadowSize The pixel size of the shadow image. * @param {GPoint} iconAnchor The pixel coordinate relative to the top left corner of the icon image at which this icon is anchored to the map. * @param {GPoint} iconInfoWindowAnchor The pixel coordinate relative to the top left corner of the icon image at which the info window is anchored to this icon. * @param {String} iconPrintImage The URL of the foreground icon image used for printed maps. It must be the same size as the main icon image given by image. * @param {String} iconMozPrintImage The URL of the foreground icon image used for printed maps in Firefox/Mozilla. It must be the same size as the main icon image given by image. * @param {String} iconPrintShadow The URL of the shadow image used for printed maps. It should be a GIF image since most browsers cannot print PNG images. * @param {String} iconTransparent The URL of a virtually transparent version of the foreground icon image used to capture click events in Internet Explorer. This image should be a 24-bit PNG version of the main icon image with 1% opacity, but the same shape and size as the main icon. * @return {Object} The options for createIcon */ function defaults() { return { 'iconImage': undefined, 'iconShadow': undefined, 'iconSize': undefined, 'iconShadowSize': undefined, 'iconAnchor': undefined, 'iconInfoWindowAnchor': undefined, 'iconPrintImage': undefined, 'iconMozPrintImage': undefined, 'iconPrintShadow': undefined, 'iconTransparent': undefined }; }; options = jQuery.extend(defaults(), options); var icon = new GIcon(G_DEFAULT_ICON); if(options.iconImage) icon.image = options.iconImage; if(options.iconShadow) icon.shadow = options.iconShadow; if(options.iconSize)

icon.iconSize = options.iconSize; if(options.iconShadowSize) icon.shadowSize = options.iconShadowSize; if(options.iconAnchor) icon.iconAnchor = options.iconAnchor; if(options.iconInfoWindowAnchor) icon.infoWindowAnchor = options.iconInfoWindowAnchor; return icon; }; /** * A helper function to get the map center as a GLatLng * @method * @namespace Mapifies * @id Mapifies.getCenter * @alias Mapifies.getCenter * @param {jQuery} element The element that contains the map. * @return {GLatLng} A object containing the center of the map */ Mapifies.getCenter = function (element) { var thisMap = Mapifies.MapObjects.Get(element); return thisMap.getCenter(); }; /** * A helper function to get the bounds of the map * @method * @namespace Mapifies * @id Mapifies.getBounds * @alias Mapifies.getBounds * @param {jQuery} element The element that contains the map. * @return {GSize} The bounds of the map */ Mapifies.getBounds = function (element){ var thisMap = Mapifies.MapObjects.Get(element); return thisMap.getBounds(); };var Mapifies; if (!Mapifies) Mapifies = {}; (function($){ $.fn.jmap = function(method, options, callback) { return this.each(function(){ if (method == 'init' && typeof options == 'undefined') { new Mapifies.Initialise(this, {}, null); } else if (method == 'init' && typeof options == 'object') { new Mapifies.Initialise(this, options, callback); } else if (method == 'init' && typeof options == 'function') { new Mapifies.Initialise(this, {}, options); } else if (typeof method == 'object' || method == null) { new Mapifies.Initialise(this, method, options); } else { try { new Mapifies[method](this, options, callback); } catch(err) { throw Error('Mapifies Function Does Not Exist'); } } }); } })(jQuery); {/codecitation}

Aleksander Dombrowski