



OpenWebGlobe SDK for WebGL Specification

VERSION 1.0 rev 19



Tuesday, 17. May 2011

Martin Christen
martin.christen@fhnw.ch

WARNING

This is a draft specification.

Certain things may still change - this draft is not fully compatible to future releases of the OpenWebGlobe SDK specification.

This document replaces all previous versions.

Table of Contents

1	Introduction	4
1.1	Basic Objects	4
1.2	Hello World	5
2	OpenWebGlobe Objects overview	6
3	OpenWebGlobe Functions	7
3.1	General Object functions	7
3.2	Context-Object	8
3.3	Setting Context-related Callback Functions	10
3.4	Scene Object	11
3.4.1	3D-Ellipsoid-WGS84 specific functions	12
3.4.2	3D-Flat-Cartesian	13
3.4.3	2D-Screen	13
3.5	Camera Object	15
3.6	Texture Object	16
3.7	Navigation Controller Object	17
3.8	World Object (Terrain / Globe)	19
3.9	POI Object	20
3.9.1	Image Layer	22
3.9.2	Elevation Layer	23
3.9.3	Waypoint Layer	24
3.9.4	POI Layer	25
3.9.5	Geometry Layer	26
3.9.6	Voxel Layer	27
3.10	Geometry	28
3.10.1	Geometry Object	28
3.10.2	Mesh Object	29
3.10.3	Point Object	29
3.10.4	Line Object	29
3.10.5	Material Object	29
3.10.6	Creating Default Objekts	29
3.11	Text Data	30
4	Misc Functions	33

1 Introduction

OpenWebGlobe SDK consists of **functions**, but is designed in an object oriented manner. The SDK is written in C++ and has bindings to other languages like Python, C# and VisualBasic or JavaScript (WebGL). A new application can be written in your favorite language.

The SDK consists of several objects. Each object contains a certain number of functions.

1.1 Basic Objects

Because OpenWebGlobe SDK consists of functions, all objects are referenced using an integer id.

Among the basic objects is the context object which is the combination of a render window and a graphics engine. A special mode allows windowless applications - which is for example useful for server applications or query engines.

The scene object describes the type of virtual globe visualisation. A virtual globe is usually represented as 3d-ellipsoid (WGS84), but it is also possible to create flat earth representations especially for local scenes or to create a 2D visualisation which is usually a 2d-map-type application.

Inside the scene there is a camera which controls what is visible. There can be several cameras in a scene, but only one is active at the same time. The camera has a navigation controller which allows navigation through the scene - for example with the mouse and keyboard as input. There is a default navigation controller available and a custom navigation can also be implemented.

The world object contains the globe in the format specified in the scene before (ellipsoid, flat, 2d). The world consists of different layers which can be added or removed during runtime. The most important layers are the image layer consisting of previously tiled ortho-photos and the elevation layer consisting of previously processed (tiled geometry) elevation data (DEM). Both layers allow streaming nearly unlimited data. The waypoint layer contains waypoints, for example previously tracked points with a GPS receiver. Furthermore the poi layer contains points of interest (POI), the geometry layer contains 3d-objects and the voxel layer contains preprocessed (tiled) point cloud data. All layers are streamable over the internet or also from a local area network or local hard drive.

1.2 Hello World

This example creates a virtual globe and adds image and elevation data to it.

```
<html lang="en">
<meta http-equiv="Content-Type" content="text/html; charset=utf-8" />
<script type="text/javascript" src="OpenWebGlobe.js"></script>

<script type="text/javascript">
function main()
{
    var context = ogCreateContextFromCanvas("canvas");
    var globe = ogCreateGlobe(context);
    ogAddImageLayer(globe, { url : ['http://www...'], layer: 'world500'});
    ogAddElevationLayer(globe, { url : ['http://www...'], layer: 'srtm'});
}
</script>
<body onload="main()">
    <div style="text-align: center">
        <canvas id="canvas" width="640" height="480"></canvas>
    </div>
</body>
```

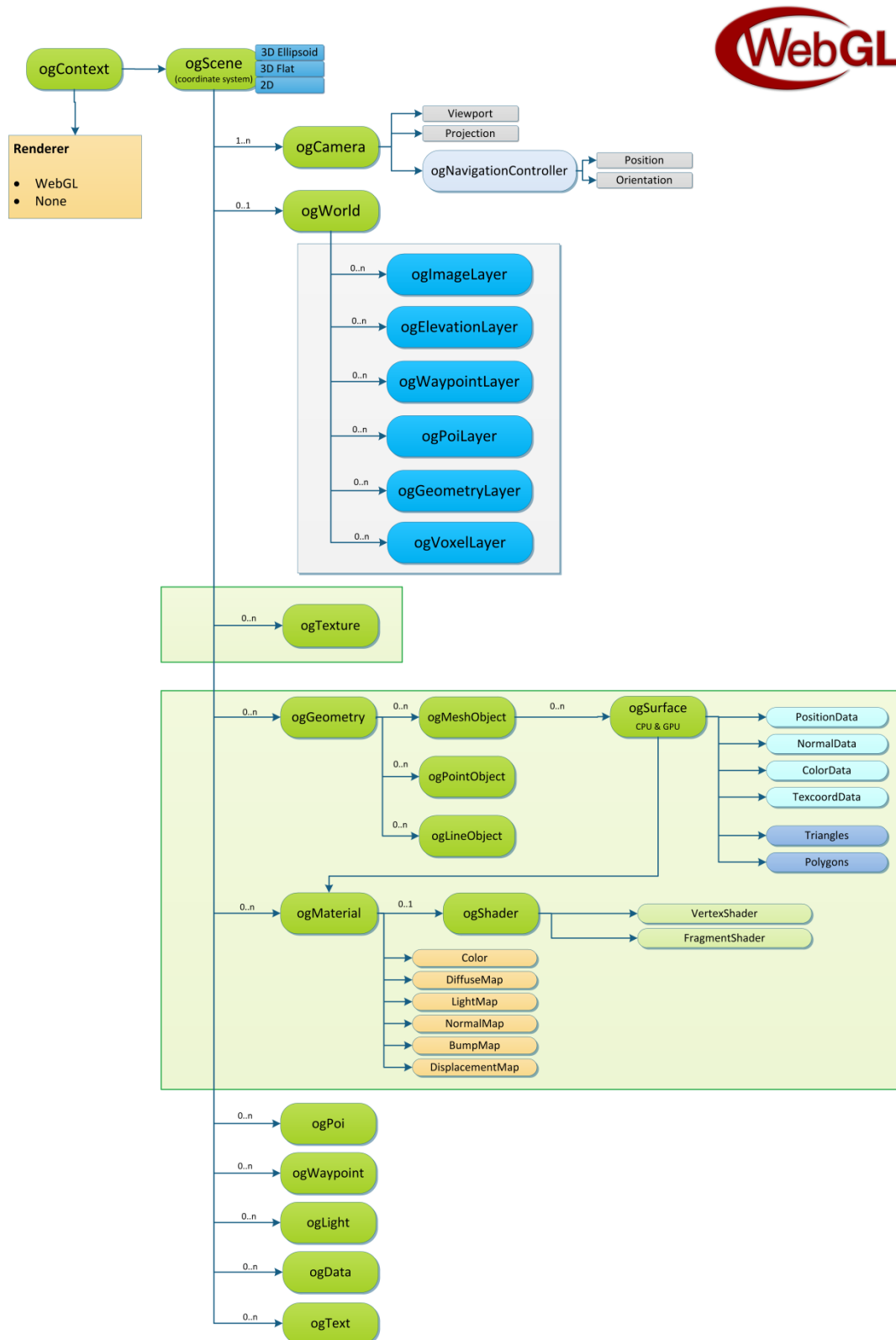
The function **ogCreateContextFromCanvas** is a simple function to setup the render environment. There is also a more complex function **ogCreateContext** which provides more options. The return value of these functions are an id to a context object. This id is required for other functions. It is also possible to use several contexts at the same time.

The second function **ogCreateGlobe** directly creates a WGS84 scene containing a camera and a world object. This is a convenience function as there are also functions available to create scene objects, camera objects and world objects manually.

The next functions add image (**ogAddImageLayer**) and elevation (**ogAddElevationLayer**) layers to the scene. These functions also return an id to the objects but they are not used for this demo.

2 OpenWebGlobe Objects overview

The OpenWebGlobe SDK for WebGL has the following object hierarchy:



t

3 OpenWebGlobe Functions

3.1 General Object functions

Object Operations	
<code>type = ogGetObjectType(int object);</code>	Returns the type of the object, which can be one of the following: OG_OBJECT_CONTEXT, OG_OBJECT_SCENE, OG_OBJECT_WORLD, OG_OBJECT_IMAGELAYER, OG_OBJECT_ELEVATIONLAYER, OG_OBJECT_WAYPOINTLAYER, OG_OBJECT_POILAYER, OG_OBJECT_GEOMETRYLAYER, OG_OBJECT_VOXELLAYER, OG_OBJECT_IMAGE, OG_OBJECT_TEXTURE, OG_OBJECT_PIXELBUFFER, OG_OBJECT_GEOMETRY, OG_OBJECT_MESH, OG_OBJECT_SURFACE, OG_OBJECT_CAMERA, OG_OBJECT_TEXT, OG_OBJECT_BINARYDATA, OG_OBJECT_LIGHT, OG_OBJECT_POI
<code>num = ogGetNumObjects();</code>	Returns the number of all objects. Because OpenWebGlobe is a DLL it is possible several applications use it at the same time. Therefore this function only gives back the number of objects the caller thread has.
<code>object_id = ogGetObjectAt(index);</code>	Returns the object at the specified index. The index can be in the range 0..ogGetNumObjects()-1. If an invalid index is given, -1 is returned.
<code>object_id = ogGetParentObject(object_id);</code>	Returns the parent object (for example if you call this on a world object you get the scene. If there is no parent object available, this function returns -1.
<code>ogSetObjectName(object_id, name);</code>	Set name of the object. This is a convenience function, at it does the same as <code>ogSetStringProperty(object, "name", "ObjectName");</code>
<code>ogGetObjectByName(name);</code>	Retrieve an OpenWebGlobe object by its name. Please note that names are not unique, so the first found will be returned. The OpenWebGlobe developer is responsible to give objects unique names.
Functions for asynchronous objects	
<code>status = ogGetObjectStatus(int object);</code>	Returns the state of the object. This function should be used for asynchronous objects. Return value: 0: Object is ready (OG_OBJECT_READY) 1: Object is being downloaded (OG_OBJECT_BUSY) 2: Object can't be downloaded (OG_OBJECT_FAILED) Synchronous objects always have the value OG_OBJECT_READY.
<code>ogOnLoad(int object, callback);</code>	Specify function which is called when object finished loading (async objects only)
<code>ogOnFailure(int object, callback);</code>	Specify function which is called when object loading failed (for example URL not available or wrong format etc.) (async objects only)

3.2 Context-Object

Context objects are a combination of render window and render engine. There is also the concept of "virtual contexts" where no visible render window is available (use for example to create a query engine).

Create Context	
context_id = ogCreateContext (options , cbfInit, cbfExit, cbfResize);	Create a render window "options" is a string containing a key-value pair list with the following possible keys: fullscreen (default false) false,true canvas (null) string: canvas-id
context_id = ogCreateContextFromCanvas (canvasid, [optional] fullscreen , [optional] cbfInit, [optional] cbfExit, [optional] cbfResize);	creates an OpenWebGlobe context from an existing canvas. This is a convenience function which simplifies the call "ogCreateContext"
Destroy Context	
void ogDestroyContext (context_id);	Free the context and all of its sub objects.
Context Operations	
width = ogGetWidth (context_id);	Returns the width of the context
height = ogGetHeight (context_id);	Returns the height of the context
scene = ogGetScene (context_id)	Retruns the scene attached to this context. If no scene is defined, -1 is returned.
Render Engine	
ogSetBackgroundColor (context_id, r, g, b, a);	Set the background color of the context.
ogDrawText (context_id, text, x, y);	Draw ASCII (latin1) text directly on the screen.
ogSetTextColor (context_id, r, g, b);	Set text color.
[width, height] = ogGetTextSize (int context, text);	Returns size of text (in pixel).
ogSetViewport (context_id, x, y, width, height);	Set the viewport

Parent Object

ogContext is a top level object and has no parent object

Child Objects

Object	Min	Max
ogScene	0	1

Note

In future versions of OpenWebGlobe SDK there will also be the concept of a "shared context", this will be a context which shares resources among different contexts.

However, this will not be available before OpenWebGlobe SDK 1.1.

3.3 Setting Context-related Callback Functions

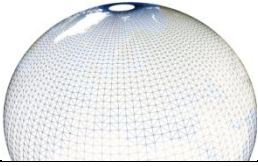
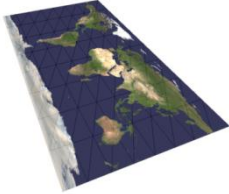
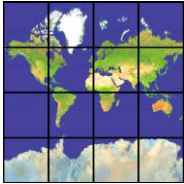
ogSetMouseDownFunction (context_id, callback);	Set MouseDown callback
ogSetMouseUpFunction (context_id, callback);	Set MouseUp callback
ogSetMouseMoveFunction (context_id, callback);	Set MoveMove callback
ogSetMouseWheelFunction (context_id, callback);	Set MouseWheel callback
ogSetKeyDownFunction (context_id, callback);	Set KeyDown callback
ogSetKeyUpFunction (context_id, callback);	Set KeyUp callback
ogSetResizeFunction (context_id, callback);	Set Resize callback
ogSetRenderFunction (context_id,);	Set Render callback.
ogSetTimerFunction (context_id, callback);	Set Timer callback
ogSetRenderGeometryFunction (context_id, callback);	Set Render-Geometry callback
ogSetBeginRenderFunction (context_id, callback);	Set "Begin-Render" callback
ogSetEndRenderFunction (context_id, callback);	Set "End-Render" callback
ogSetNumRenderPasses (context_id, numPasses);	Set number of render passes

Definitions of Callback functions:

CallBack_MouseDown (context_id, button, x, y);	Called when mouse button is pressed
CallBack_MouseUp (context_id, button, x, y);	Called when mouse button is released
CallBack_MouseMove (context_id, x, y);	Called when mouse is moved
CallBack_MouseWheel (context_id, delta);	Called when mouse wheel is used
CallBack_KeyDown (context_id, key);	Called when key is pressed
CallBack_KeyUp (context_id, key);	Called when key is released
CallBack_Render (scene_id);	Main Render Loop
CallBack_Resize (context_id, width, height);	Called when window is resized
CallBack_Timer (context_id, delta);	Called every frame for timing
CallBack_Init (context_id);	Called when Render API is ready
CallBack_Exit (context_id);	Called before Render API is closed
CallBack_RenderGeometry (mesh_id, pass)	Called to render Geometry (custom)
CallBack_BeginRender (scene_id, pass)	Called when render pass begins
CallBack_EndRender (scene_id, pass)	Called when render pass ends
CallBack_Object (object_id)	Object specific callback (for example ready, failure, ...)

3.4 Scene Object

There are 3 different scene types: 3d-Ellipsoid-WGS84, 3D-Flat-Cartesian and 2D-Screen.

3D-Ellipsoid-WGS84 	A virtual globe on the WGS-84 Ellipsoid. You can map point in geocentric cartesian coordinates (x,y,z) or using (longitude, latitude, elevation).
3D-Flat-Cartesian 	A 3D-terrain using cartesian coordinates. Usually this is only a part of the world and not a whole planet.
2D-Screen 	A 2D-Scene which is mainly used for 2D-Maps
Custom	A custom scene where you provide your own coordinate system (and model/view/projection matrices).

Create Scene

scene_id = ogCreateScene (context_id, sceneType);	<p>Creates a scene, sceneType is one of the following: OG_SCENE_3D_ELLIPSOID_WGS84 OG_SCENE_3D_FLAT_CARTESIAN OG_SCENE_2D_SCREEN OG_SCENE_CUSTOM</p> <p>When creating a scene there is a default camera and a default navigation controller.</p>
--	--

Destroy Scene

ogDestroyScene (scene_id);	Removes the scene and frees all memory.
-----------------------------------	---

Scene Operations

context_id = ogGetContext (int scene);	Returns the parent context object.
world_id = ogGetWorld (int scene);	Returns the world object within this scene or -1 if no world object is available.

Camera Management

ogSetActiveCamera (scene_id, camera_id);	Set the specified camera to the active camera. Please refer to the camera object for more information about cameras.
camera_id = ogGetActiveCamera (scene_id);	Returns the active camera of the scene. Please refer to the camera object for more information about cameras.
num = ogGetNumCameras (scene_id);	Returns the number of cameras in the scene (can't be smaller than 1). Please refer to the camera object for more information about cameras.

<code>camera_id = ogGetCameraAt(scene_id, index);</code>	Returns the camera at specified index. Please refer to the camera object for more information about cameras.
Picking	
<code>[geometry_id, x, y, z] = ogPickGeometry(scene_id, mx, my);</code>	Pick on Geometry. (All geometry but not terrain/globe). mx, my: Mouse Position
<code>[mesh_id, surface_id, x, y, z] = ogPickMesh(scene_id, mx, my);</code>	Pick on Geometry. Returns more detailed infos.

3.4.1 3D-Ellipsoid-WGS84 specific functions

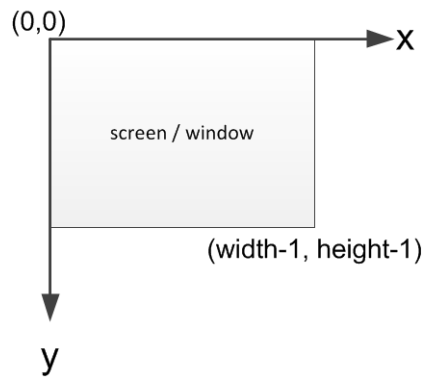
3D-Ellipsoid-WGS84 specific functions (only works with WGS84 Ellipsoid scenes)	
<code>world_id = ogCreateGlobe(int context)</code>	Extends context object. Creates a scene (OG_SCENE_3D_ELLIPSOID_WGS84) and adds a world object. This is a convenience function. You can retrieve the scene object with: <code>scene_id = ogGetParentObject(world_id);</code>
<code>[hit, lng, lat, elv] = ogPickGlobe(scene_obj, mx, my);</code>	Extends scene object. Pick on the globe. If there is a hit "hit" is true and there is a valid WGS84 Position in lng, lat, elv.
<code>[x,y,z] = ogToCartesian(scene_id, lng, lat, elv);</code>	Extends scene object. Convert lng,lat,elv coordinate to geocentric cartesian.
<code>[lng, lat, elv] = ogToWGS84(scene_id, x, y, z);</code>	Extends scene object. Convert a geocentric cartesian coordinate to lng, lat, elv
<code>ogSetPositionWGS84(camera_id, lng, lat, elv);</code>	Extends camera object. Set the camera position as WGS84 coordinate. Please refer to the camera object for more information about cameras.
<code>ogSetOrientationWGS84(camera_id, yaw, pitch, roll);</code>	Extends camera object. Set the orientation (body frame). The orientation depends on current WGS84 coordinate. (angles are in radiant). Please refer to the camera object for more information about the camera.
<code>ogLookAtWGS84(camera_id, lng, lat, elv);</code>	Extends camera object. Sets yaw, pitch, roll so that it looks at the specified position.

3.4.2 3D-Flat-Cartesian

3D-Flat-Cartesian specific functions	
[hit, x, y, z] = ogPickTerrain (scene_id, mx, my);	Extends scene object. Pick on the flat cartesian terrain

3.4.3 2D-Screen

This scene type is 2D. Everything is done with window coordinates, this means (0,0) is top left.



z-Values in the range [-1,1] are possible to create "Layers".

Parent Object

The parent object always is from type ogContext

Child Objects

Object	Min	Max
ogCamera	1	∞
ogWorld	0	1
ogImage	0	∞
ogPixelBuffer	0	∞
ogTexture	0	∞
ogGeometry	0	∞
ogMaterial	0	∞
ogPoi	0	∞
ogWaypoint	0	∞
ogLight	0	∞
ogBinaryData	0	∞
ogText	0	∞

3.5 Camera Object

Create Camera	
camera_id ogCreateCamera (scene_id);	Creates a new camera.
Destroy Camera	
ogDestroyCamera (camera_id);	Destroy Camera. This function has no effect on cameras that are currently active.
Camera Operations	
nc_id ogGetNavigationController (camera_id);	Retrieve current navigation controller.
ogLookAt (camera_id, x, y, z);	Look at specified point (cartesian)
ogSetOrientation (camera_id, yaw, pitch, roll);	Set orientation (cartesian)
Set Matrices manually (only works with OG_SCENE_CUSTOM)	
ogSetProjectionMatrix (camera_id, mat);	Set the projection matrix manually. mat is an array with 16 values. (column major)
ogPerspectiveProjection (camera_id, fovy, aspect, zNear, zFar);	Create a perspective projection
ogOrtho2D (camera_id, left, right, bottom, top);	Create a 2d-ortho matrix.
ogSetViewMatrix (camera_id, m);	Set view matrix manually using 4x4 double values. (not recommended)

Parent Object

The parent object always is from type ogScene

Child Objects

Object	Min	Max
ogNavigationController	1	1

3.6 Texture Object

Create Texture Object	
<code>texture_id = ogLoadTextureAsync(scene_id, url);</code>	Loads a texture asynchronously
Destroy Texture Object	
<code>ogDestroyTexture(texture_id);</code>	Remove texture and free all memory
Texture Operations	
<code>ogBlit(texture_id, x,y, ...)</code>	blit texture on screen (2D)

Parent Object

The parent object always is from type ogScene

Child Objects

There are no child objects.

3.7 Navigation Controller Object

The navigation controller has special callback functions which are called even if you overload other events. In theory you could implement a navigation without

Create Navigation Controller	
nc_id = ogCreateNavigationController (camera_id, type);	<p>Create a navigation Controller. If the camera already has a navigation controller, a new one is created and the old one is removed. The navigation controller can be initialized with:</p> <p>OG_NAVIGATION_DEFAULT (default navigation) OG_NAVIGATION_CUSTOM (custom, overload appropriate navigation events)</p> <p>(Standard navigation types are planned and will be implemented over time.)</p>
Navigation Command	
void SendNavigationCommand (nc_id, command);	Send navigation specific commands.
Navigation Controller Operations	
ogSetNavigationEventInit (nav, cbf);	Set callback for navigation
ogSetNavigationEventExit (nav, cbf);	Set callback for navigation
ogSetNavigationEventMouseDown (nav, cbf);	Set callback for navigation
ogSetNavigationEventMouseUp (nav, cbf);	Set callback for navigation
ogSetNavigationEventMouseMove (nav, cbf);	Set callback for navigation
ogSetNavigationEventMouseWheel (nav, cbf);	Set callback for navigation
ogSetNavigationEventKeyDown (nav, cbf);	Set callback for navigation
ogSetNavigationEventKeyUp (nav, cbf);	Set callback for navigation
ogSetNavigationEventPreRender (nav, cbf);	Set callback for navigation
ogSetNavigationEventPostRender (nav, cbf);	Set callback for navigation
ogSetNavigationEventResize (nav, cbf);	Set callback for navigation
ogSetNavigationEventTimer (nav, cbf);	Set callback for navigation
ogSetNavigationEventCommand (nav, cbf);	Set callback for navigation

Callback Definitions for navigation

CallBack_NavigationInit (camera_id);	Called when camera is created. (This callback is done for convenience and called as soon as SetNavigationEventInit is set. You can also ignore this callback and setup everything without this initialization. It is considered best practice to create a "init callback" for custom navigation controllers.)
CallBack_NavigationExit (camera_id);	Called when camera is deleted
CallBack_NavigationMouseDown (camera_id, button, x, y);	Called when mouse button is pressed
CallBack_NavigationMouseUp (camera_id, button, x, y);	Called when mouse button is released
CallBack_NavigationMouseMove (camera_id, x, y);	Called when mouse is moved
CallBack_NavigationMouseWheel (camera_id, delta);	Called when mouse wheel is used
CallBack_NavigationKeyDown (camera_id, key);	Called when key is pressed
CallBack_NavigationKeyUp (camera_id, key);	Called when key is released
CallBack_NavigationPreRender (camera_id, scene_id);	Called before anything is rendered
CallBack_NavigationPostRender (camera_id, scene_id);	Called when everything is rendered
CallBack_NavigationResize (camera_id, width, height);	Called when window is resized
CallBack_NavigationTimer (camera_id, delta_time);	Called every frame for timing (time in milliseconds)
CallBack_NavigationCommand (camera_id, command);	Called when navigation command is sent.

3.8 World Object (Terrain / Globe)

Create World Object	
world_id ogCreateWorld (scene_id);	Creates a world object (for example virtual globe)
Destroy World Object	
ogDestroyWorld (world_id);	Entfernt das Welt-Objekt
World Object Operations	
ogShowWorld (world_id);	Show world (which is hidden with ogHideWorld)
ogHideWorld (world_id);	Hide world.
ogSetRenderoption (world_id, renderoption);	Set render option. [This is currently not available]
ogRenderWorld (world_id);	manual render world. This is required if you use a custom Render callback.

Parent Object

The parent object always is from type ogScene

Child Objects

Object	Min	Max
ogImageLayer	0	∞
ogElevationLayer	0	∞
ogWaypointLayer	0	∞
ogPoiLayer	0	∞
ogGeometryLayer	0	∞
ogVoxelLayer	0	∞

3.9 POI Object

Create POI Object	
poi_id = ogCreatePOI (poilayer_id, options);	<p>Create a POI</p> <pre>{ text: "poi text", icon: "url to icon", position: [lng, lat, elv], flagpole: true, false flagpoleColor: [r,g,b] visibilityRange: [0, 1000] mode: "absolute" size: s }</pre> <p>text optional if there is an icon icon optional if there is a text optional, default is true optional, default is [0,0,0] optional, default is [0,∞] optional, default is "absolute" optional, default is 40</p> <p>text: utf-8 encoded text flagpole-color: r,g,b components are in range [0,1] mode: can be "absolute" or "relativeToGround" visibility-range: is specified in km. default is [0,∞] size: how big one pixel is [m].</p> <p>Please note that the parent of the POI is a scene object and not the layer.</p>
Destroy POI Object	
ogDestroyPOI (poi_id);	Destroy POI object and free all memory. The POI is also removed from a layer, if it is in a layer.
POI Operations	
ogChangePOIText (poi_id, text)	change poi text
ogChangePOIIcon (poi_id, url)	change icon of poi.
ogChangePOISize (poi_id, size)	change size of poi
ogChangePOIPosition (poi_id, lng, lat, elv)	change position of poi. if SRS is WGS84: use (lng, lat, elv) if SRS is cartesian: use (x,y,z) instead
ogHidePOI (poi_id)	Hide POI
ogShowPOI (poi_id)	Show a previously hidden POI
Picking	
poi_id = ogPickPOI (scene_id, mx, my)	Pick POI at mouse position (mx,my). Returns the POI-id or -1 if no POI is picked.

Parent Object

The parent object always is from type ogScene

Child Objects

There are no child nodes.

3.9.1 Image Layer



Create Image Layer Object	
layer_id = ogAddImageLayer (world_id, options);	Add an image layer.
Destroy Image Layer Object	
ogRemoveImageLayer (layer_id);	Remove image layer (and free all memory).
Image Layer Operations	
ogSwapImageLayers (layer_id1, layer_id2);	Swap order of image layers.
ogSetImageLayerTransparency (layer_id, transparency);	Set transparency of two layers, transparency is in range [0,1].
ogSetImageLayerMaxLod (layer_id, maxlod);	Set max lod of the image layers.
num = ogGetNumImageLayers ();	Get number of layers.
layer_id = ogGetImageLayerAt (index);	Retrieve layer at specified index. Index ranges from 0 to GetNumImageLayers()-1.

Parent Object

The parent object always is from type ogWorld

Child Objects

There are no child objects.

3.9.2 Elevation Layer



DHM 25, SWISSIMAGE © swisstopo (JA100071)

Create Elevation Layer Object	
layer_id = ogAddElevationLayer (world_od, options);	Add Elevation Layer
Destroy Elevation Layer Object	
ogRemoveElevationLayer (layer_id);	Remove elevation layer (and free all memory).
Elevation Layer Operations	
ogSwapElevationLayers (layer_id1, layer_id2);	Swap order of elevation layers.
num = ogGetNumElevationLayers ();	Get number of layers.
layer_id = ogGetElevationLayerAt (index);	Retrieve layer at specified index. Index ranges from 0 to GetNumElevationLayers()-1.

Parent Object

The parent object always is from type ogWorld

Child Objects

There are no child objects

3.9.3 Waypoint Layer

Create Waypoint Layer Object	
int ogAddWaypointLayer (int world, const char* setup);	
Destroy Waypoint Layer Object	
void ogRemoveWaypointLayer (int waypointlayer);	
Waypoint Layer Operations	
void ogSwapWaypointLayers (int layer1, int layer2);	Ändert Layer Reihenfolge durch Austauschen zweier Layer.
unsigned int ogGetNumWaypointLayers ();	Get number of layers.
int ogGetWaypointLayerAt (unsigned int index);	Retrieve layer at specified index. Index ranges from 0 to GetNumWaypointLayers()-1.

Parent Object

The parent object always is from type ogWorld

Child Objects

There are no child objects

3.9.4 POI Layer



Create POI Layer Object

`poilayerid = ogCreatePOILayer(world_id, layername, [textstyle], [iconstyle]);`

Creates a new POI layer.

world: the world
layername: name of the layer

textstyle and icon style are optional. If you don't use the default styles will be used. If you want to use the default text style and a custom icon style, use "null" for textstyle.

textstyle =
{
 "font" : 'SansSerif',
 "fontSize" : 48,
 "backgroundColor" : [r,g,b,a],
 "fontColor" : [r,g,b,a],
 "lineWidth" : 3,
 "strokeColor" : [r,g,b,a],
 "textAlign" : 'left',
 "shadowOffsetX" : 2,
 "shadowOffsetY" : 2,
 "shadowBlur" : 5,
 "shadowColor" : [r,g,b,a]
}

iconstyle =
{
 "width" : 64,
 "height" : 64,
 "border" : 0,
 "backgroundColor" : [r,g,b,a],
 "shadowOffsetX" : 0,
 "shadowOffsetY" : 0,
 "shadowBlur" : 0,
 "shadowColor" : [r,g,b,a]
};

Destroy POI Layer Object

`ogRemovePOILayer(poilayer_id);`

Remove POI layer and free all memory. Also frees the memory of all POIs attached to this layer.

POI Layer Operations

`void ogHidePOILayer(poilayer_id);`

hide the POI-Layer: hides all POIs within this layer.

`void ogShowPOILayer(poilayer_id);`

show the previously hidden POI-Layer

Parent Object

The parent object always is from type ogWorld

Child Objects

There are no child objects

3.9.5 Geometry Layer

Create Geometry Layer Object	
geometrylayer_id = ogAddGeometryLayer (world_id, options);	
geometrylayer_id = ogCreateGeometryLayer (world_id, layername);	
Destroy Geometry Layer Object	
ogRemoveGeometryLayer (geometrylayer_id);	
Geometry Layer Operations	
ogHideGeometryLayer (geometrylayer_id);	
void ogRemoveGeometry (geometry_id);	

Parent Object

The parent object always is from type ogWorld

Child Objects

There are no child objects

3.9.6 Voxel Layer



Create Voxel Layer Object

voxellayer_id = ogAddVoxelLayer (world_id, options);	
---	--

Destroy Voxel Layer Object

ogRemoveVoxelLayer (voxellayer_id);	
--	--

Geometry Layer Operations

void ogHideGeometryLayer (geometrylayer_id);	
---	--

void ogShowGeometryLayer (geometrylayer_id);	
---	--

Parent Object

The parent object always is from type ogWorld

Child Objects

There are no child objects

3.10 Geometry

3.10.1 Geometry Object

Create Geometry Object	
<code>geometry_id = ogCreateGeometry(geometrylayer_id, json);</code>	Create a geometry from a geometry JSON.
<code>geometry_id = ogLoadGeometryAsync(geometrylayer_id, url);</code>	Loads a geometry JSON from specified url.
Destroy Geometry Object	
<code>ogDestroyGeometry(geometry_id);</code>	Destroy geometry object and free all memory
Geometry Object Operations	
<code>ogSetGeometryPosition(geometry_id, x, y, z);</code>	Set geometry position (cartesian)

Parent Object

The parent object always is from type ogScene

Child Objects

There are no child nodes.

3.10.2 Mesh Object

will be defined soon

3.10.2.1 Surface Object

will be defined soon

3.10.3 Point Object

will be defined soon

3.10.4 Line Object

will be defined soon

3.10.5 Material Object

will be defined soon

3.10.6 Creating Default Objekts

There are some convenience functions to create default objects. This will be extended over time.

geometry_id = ogCreateCube (scene_id, length);	Create a cube (center is 0,0,0).
geometry_id = ogCreateSphere (scene_id, radius, stacks, slices);	Create a sphere (center is 0,0,0)
geometry_id = ogCreateSquare (scene_id, length, numtiles)	Creates a square (checkerboard)

3.11 Text Data

<code>text_id = ogCreateText(scene_id, text);</code>	create new text (utf8)
<code>text_id = ogLoadTextAsync(int scene, const char* url);</code>	load text
<code>ogDestroyText(text_id);</code>	Textobjekt Löschen
<code>ogAddTranslation(text_id, text, LanguageCode)</code>	Adds a translation, language code should follow RFC 1766/ISO 639/ISO 3166/RFC 3066, this means for example "de_DE", "de_CH", "de_AT", "en_US" and so on.
<code>ogSetDefaultLanguage(context, LanguageCode)</code>	Set the default language. (default is "en_US".)
<code>text = ogGetText(text_id);</code>	Returns the text in default language
<code>text = ogGetTranslation(text_id, LanguageCode)</code>	Returns translated text. If there is no translation, the default language text is returned.
<code>t = ogHasTranslation(text, LanguageCode)</code>	Returns true if translation for specified language code exists.

4 Misc Functions

int ogExec ();	Start application. (Not required in WebGL version)
text = ogMemoryDump ();	Returns a memory dump
text = ogGetString (type);	Returns an info string OG_VERSION: Version of OpenWebGlobe
error = ogGetLastError ();	returns the last error or null if there is none
float ogRand01 ();	Returns a random value in the range [0,1]