Fachhochschule Nordwestschweiz
Hochschule für Architektur, Bau und Geomatik

OpenWebGlobe SDK Specification

VERSION 1.0 rev 17

Thursday, 28. April 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

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 serveral 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.

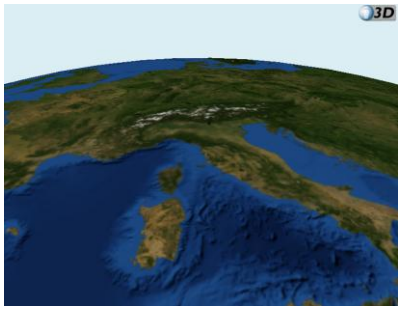
Inside the scene there is a camera which controls what is visible. There can be several cameras in a scene, but only one is active at the same time. The camera has a navigation controller which allows navigation through the scene - for example with the mouse and keyboard as input. There is a default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

Using these objects a simple “Hello World” type application can be created. This example creates a virtual globe and adds image and elevation data to it.

```
#include "og.h"

int main(void)
{
   int context = ogCreateRenderWindow("Hello Globe", 640, 480);
   int globe = ogCreateGlobe(context);
   ogAddImageLayer(globe, "url=http://.../img/ layer=world500");
   ogAddElevationLayer(globe, "url=http://.../elv/ layer=srtm");
   ogExec();
}
```



The function **ogCreateRenderWindow** is a simple function to create a render window. 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 create several render windows at the same time.

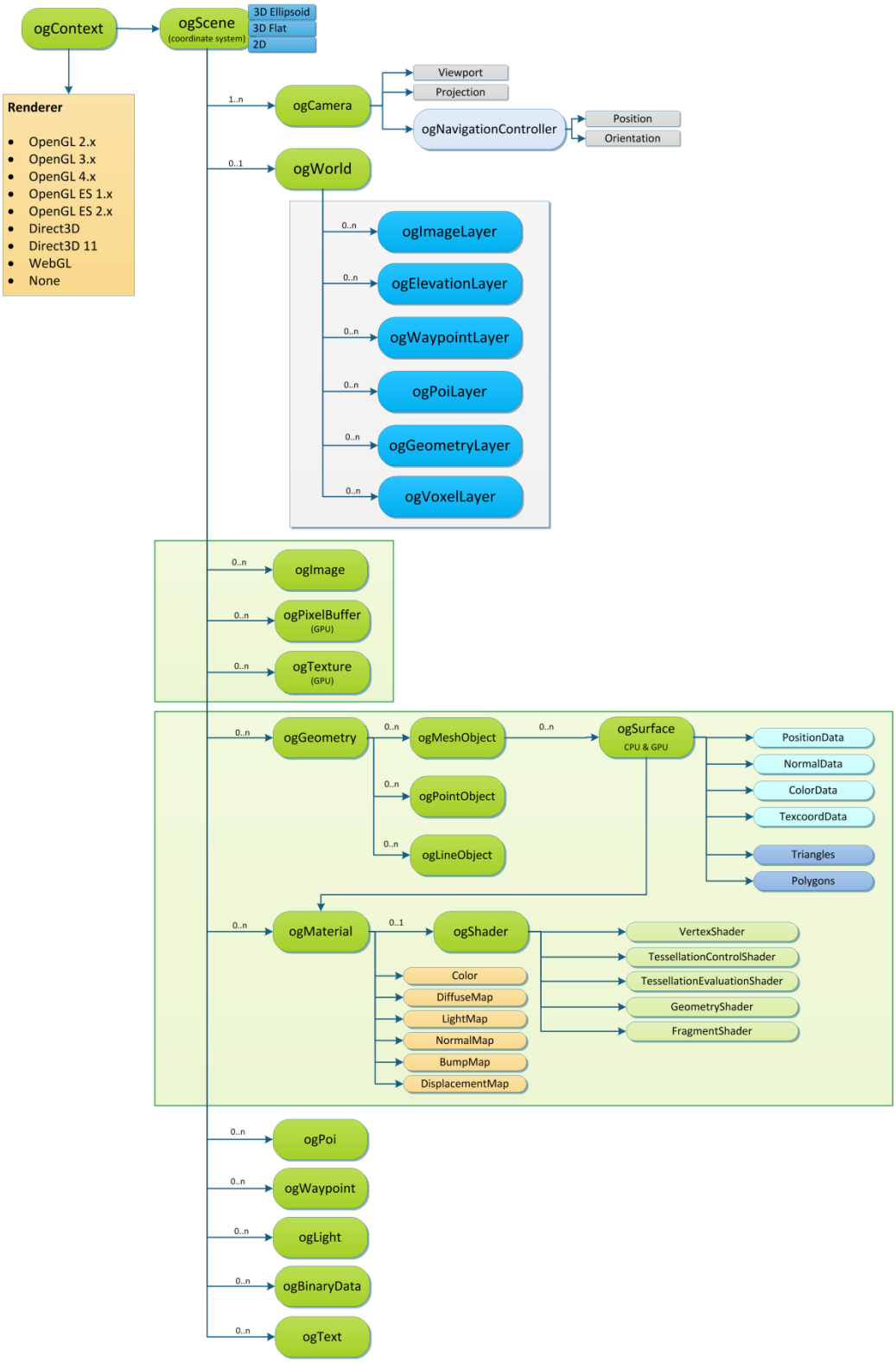
The second function **ogCreateGlobe** directly creates a WGS84 scene containing a camera and a world object. This is a convienience function as there are also functions available to create scene objects, camera objects and world objects manually.

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.

With **ogExec** the virtual globe application starts until a terminate command is used or the window is closed. You can add custom events for example to control the mouse input, or the keyboard input, or the rendering mechanism, or the loading of asynchronous objects.

2 OpenWebGlobe Objects overview

The OpenWebGlobe SDK has the following object hierarchy:



3 OpenWebGlobe Functions

3.1 Application Specific functions

Application Name	
void **ogSetApplicationName**(const char* name);	Set Name of Application. This is important to have an application specific settings directory. Default Name is "OpenWebGlobe". All application data including web cache is stored in %appdata%/OpenWebGlobe or $HOME/OpenWebGlobe or the name you specified. **Warning**: This must be the first OpenWebGlobe command called by the application. You can't change the application time at a later time. (This may lead to unexpected behavior.)
const char* **ogGetApplicationName**();	Returns application name.
Settings for Streaming (out of core loading data)	
void **ogSetDiskCacheDirectory**(const char* directory);	Set directory of disk cache. (for new world objects only)
void **ogSetDiskCacheSize**(float size);	Set cache size in MB (10^6 Bytes) for Disk Space. (Default setting is 250 MB). (for new world objects only)
void **ogSetMemoryCacheSize**(float size);	Set cache size in MB (10^6 Bytes) for Memory Cache. This cache contains raw files stored in memory. (Default setting is 50 MB). (for new world objects only)
void **ogSetTileCacheSize**(float size);	Tile cache size in MB. This is used for tiles (image, elevation, poi, voxel, ...). This memory or parts of it may be on GPU. (Default setting is 128 MB). (for new world objects only)
void **ogSetNumDownloadThreads**(int numthrd);	Set the number of download threads. If you specify 0, everything will be downloaded synchronously. (for new world objects only)
void **ogClearDiskCache**();	Delete all files on disk used for cache.
Application Settings	
const char* **ogGetSettingsDirectory**();	Get directory where application specific settings are stored. This is platform specific.

3.2 General Object functions

Object Operations	
unsigned int **ogGetObjectType**(int object);	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
unsigned int **ogGetNumObjects**();	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.
int **ogGetObjectAt**(unsigned int index);	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.
int **ogGetParentObject**(int object);	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.
void **ogSetObjectName**(int object, const char* name);	Set name of the object. This is a convienience function, at it does the same as ogSetStringProperty(object, "name", "ObjectName");
int **ogGetObjectByName**(const char* name);	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. (Property key "name").
Functions for asynchronous objects	
unsigned int **ogGetObjectStatus**(int object);	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) Syncronous objects always have the value OG_OBJECT_READY.
void **ogOnLoad**(int object, CallBack_Object callback);	Specify function which is called when object finished loading (async objects only)
void **ogOnFailure**(int object, CallBack_Object callback);	Specify function which is called when object loading failed (for example URL not available or wrong format etc.) (async objects only)
Properties	
void **ogSetStringProperty**(int object, const char* key, const char* value);	Set string property. If the key already exists the property is overwritten (if not readonly). Sets the property type to OG_PROPERTY_STRING
void **ogSetIntegerProperty**(int object, const char* key, int value);	Set integer property. If the key already exists, the property is overwriten (if not readonly). Sets the property type to OG_PROPERTY_INTEGER
void **ogSetDoubleProperty**(int object, const char* key, double value);	Set double precision floating point property. If the key already exists, the property is overwriten. (if not readonly). Sets the property type to OG_PROPERTY_DOUBLE

void **ogSetBoolProperty**(int object, const char* key, bool value);	Set boolean property. If the key already exists, the property is overwriten. (if not readonly) Sets the property type to OG_PROPERTY_BOOL
const char* **ogGetStringProperty**(int object, const char* key);	Get property as string
int **ogGetIntegerProperty**(int object, const char* key);	Get property as integer
double **ogGetDoubleProperty**(int object, const char* key);	Get property as double
bool **ogGetBoolProperty**(int object, const char* key);	Get property as bool
unsigned int **ogGetPropertyType**(int object, const char* key);	Returns the native type of the property. This can be one of the following values: OG_PROPERTY_STRING OG_PROPERTY_INTEGER OG_PROPERTY_DOUBLE OG_PROPERTY_BOOL If the property is not found OG_INVALID is returned.
bool **ogHasProperty**(int object, const char* key)	Returns true if property for given key exists.
bool **ogPropertyIsReadonly**(int object, const char* key)	Returns true if property is readonly.
bool **ogPropertyIsProtected**(int object, const char* key)	Returns true if property is protected. (can't be removed)
unsigned int **ogGetNumProperties**(int object)	Get total number of properties of the object
const char* **ogGetPropertyKey**(unsigned int index);	Returns key name of the property at the specified index. Index is a value in the range [0, oGetNumNumProperties(o)-1]. Returns 0 if an invalid index is given.
void **ogRemoveProperty**(int object, const char* key)	Removes property. (only works for unprotected properties).
const char* **ogToXML**(int object)	create XML representation of the object

Properties for all OpenWebGlobe Objects

Key	Type	Description
name	OG_PROPERTY_STRING **(protected)**	The name of the object. The default name of an object always is its class name (for example "ogContext", "ogScene", "ogImage")

3.3 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	
int **ogCreateContext**(const char* options , CallBack_Init cbfInit, CallBack_Exit cbfExit, CallBack_Resize cbfResize);	Create a render window "options" is a string containing a key-value pair list with the following possible keys: title (no default) string width (default 640) 1-n height (default 480) 1-n bpp (default 32) 16, 32 fullscreen (default false) false,true stereo (default false) false,true virtual (default false) false,true example options string: "title='hello' width=320 height=200 bpp=32 fullscreen=false" Strings containing spaces must be in double quotes or single quotes, for example: title="hello world" or title='hello world'.
int **ogCreateRenderWindow**(const char* title, unsigned int width, unsigned int height);	Quick creation of a render window by using window name, width and height.
Destroy Context	
void **ogDestroyContext**(int context);	Free the context and all of its sub objects. If the context is a window, the window will be closed. If the last context is destroyed the ogExec-Function will terminate.
Context Operations	
int **ogGetWidth**(int context);	Returns the width of the context
int **ogGetHeight**(int context);	Returns the height of the context
int **ogGetScene**(int context)	Retruns the scene attached to this context. If no scene is defined, -1 is returned.
Render Engine	
void **ogSetBackgroundColor**(int context, float r, float g, float b, float a);	Set the background color of the context.
void **ogGetBackgroundColor**(int context, float* r, float* g, float* b, float* a);	Retrieve the background color of the context
void **ogSetDrawColor**(int context, float r, float g, float b, float a);	Set drawing color for standard primitives
void **ogDrawText**(int context, char* sText, int x, int y);	Draw ASCII (latin1) text directly on the screen.
void **ogSetTextColor**(int context, float r, float g, float b);	Set text color.
void **ogGetTextSize**(int context, char* sText, int* width, int* height);	Returns size of text (in pixel).
void **ogSetViewport**(int context, int x, int y, int width, int height);	Set the viewport

Properties for ogContext

Key	Type	Description
width	OG_PROPERTY_INTEGER **(readonly, protected)**	The width of the context .
height	OG_PROPERTY_INTEGER **(readonly, protected)**	The height of the context.
bpp	OG_PROPERTY_INTEGER **(readonly, protected)**	Number of bits per pixel (invalid on virtual context)
fullscreen	OG_PROPERTY_BOOL **(readonly, protected)**	fullscreen or windowed (invalid on virtual context)
stereo	OG_PROPERTY_BOOL **(readonly, protected)**	true if stereoscopic rendering is active. (invalid on virtual context, not available in WebGL)
virtual	OG_PROPERTY_BOOL **(readonly, protected)**	true if context is virtual (no visible window or screen)

Parent Object

ogContext is a top level object and has no parent object

Child Objects

Object	Min	Max
ogScene	0	∞

Note

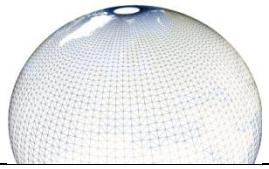
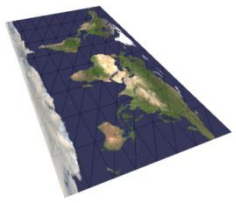
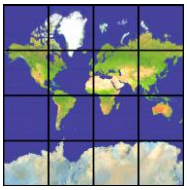
In future versions of OpenWebGlobe SDK there will also be the concept of a "shared context", this will be a context which shares resources among different contexts.
However, this will not be availble before OpenWebGlobe SDK 1.1.

3.4 Setting Context-related Callback Functions

void **ogSetMouseDownFunction**(int context, CallBack_MouseDown callback);	Set MouseDown callback
void **ogSetMouseUpFunction**(int context, CallBack_MouseUp callback);	Set MouseUp callback
void **ogSetMouseMoveFunction**(int context, CallBack_MouseMove callback);	Set MouseMove callback
void **ogSetMouseWheelFunction**(int context, CallBack_MouseWheel callback);	Set MouseWheel callback
void **ogSetKeyDownFunction**(int context, CallBack_KeyDown callback);	Set KeyDown callback
void **ogSetKeyUpFunction**(int context, CallBack_KeyUp callback);	Set KeyUp callback
void **ogSetResizeFunction**(int context, CallBack_Resize callback);	Set Resize callback
void **ogSetRenderFunction**(int context, CallBack_Render callback);	Set Render callback.
void **ogSetTimerFunction**(int context, CallBack_Timer callback);	Set Timer callback
void **ogSetRenderGeometryFunction**(int context, CallBack_RenderGeometry callback);	Set Render-Geometry callback
void **ogSetBeginRenderFunction**(int context, CallBack_BeginRender callback);	Set "Begin-Render" callback
void **ogSetEndRenderFunction**(int context, CallBack_EndRender callback);	Set "End-Render" callback
void **ogSetNumRenderPasses**(int context, int numPasses);	Set number of render passes

3.5 Scene Object

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

3D-Ellipsoid-WGS84 	Ein virtueller Globus auf dem WGS-84 Ellipsoid. Für dieses System gibt auch eine interne kartesische Repräsentation.
3D-Flat-Cartesian 	Ein Terrain in einem (flachen) kartesischen Koordinatensystem. Dies wird v.a. für lokale Gebiete verwendet und nur selten für den gesamten Planeten.
2D-Screen 	Eine 2D Szene (kartesisch). Diese kann z.B. für 2D-Karten verwendet werden.

Create Scene	
int **ogCreateScene**(int context, unsigned int sceneType);	Erstellen einer Szene. sceneType ist folgendes: OG_SCENE_3D_ELLIPSOID_WGS84 OG_SCENE_3D_FLAT_CARTESIAN OG_SCENE_2D_SCREEN **OG_SCENE_CUSTOM** Beim Erstellen der Szene wird automatisch eine Standardkamera mit einem Standard-navigation-controller angelegt. Die Szene ist also sofort navigierbar. (Diese Standard Kamera / Navigation kann auch durch ein eigenes Programm ersetzt werden.)
int **ogLoadScene**(int context, const char* url);	Lädt ein Szenen Object (XML Definition) von der angegebenen URL.
int **ogLoadSceneAsync**(int context, const char* url);	Lädt ein Szenen Object asynchron von einer URL. Ist das Bild fertig geladen wird der ObjectReady-Event ausgelöst.
Destroy Scene	
void **ogDestroyScene**(int scene);	Szene (und alle Unterobjekte) freigeben.
Save Scene	
void **ogSaveScene**(int scene, const char* filename);	Speichert das Szenen Object als XML (lokales file).
Scene Operations	
int **ogGetContext**(int scene);	Returns the parent context object.
int **ogGetWorld**(int scene);	Returns the world object within this scene or -1 if no world object is available.
Camera Management	

void **ogSetActiveCamera**(int scene, int camera);	Set the specified camera to the active camera. Please refer to the camera object for more information about cameras.
int **ogGetActiveCamera**(int scene);	Returns the active camera of the scene. Please refer to the camera object for more information about cameras.
unsigned int **ogGetNumCameras**(int scene);	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.
int **ogGetCameraAt**(int scene, unsigned int index);	Returns the camera at specified index. Please refer to the camera object for more information about cameras.
Picking	
int **ogPickGeometry**(int scene,float mx, float my, double* x, double* y, double* z);	Picking auf Geometrie (alles ausser Terrain/Globus). Gibt das Geometrie-Objekt zurück oder -1 falls nichts an der angegebenen Mausposition (mx,my) gefunden wurde.
void **ogPickMesh**(int scene,float mx, float my, double* x, double* y, double* z, int* meshobject, int* surfaceobject);	Picking auf Geometrie (alles ausser Terrain/Globus). Gibt das Mesh-Objekt (Teilobjekt der Geometrie) und dessen Surface object zurück oder -1 nichts an der angegebenen Mausposition (mx,my) gefunden wurde.

3.5.1 3D-Ellipsoid-WGS84 specific functions

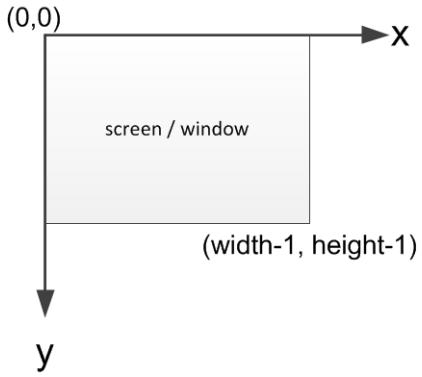
3D-Ellipsoid-WGS84 specific functions (only works with WGS84 Ellipsoid scenes)	
int **ogCreateGlobe**(int context)	**Extends context object.** direktes Erstellen eines virtuellen Globus auf dem context, ohne vorher eine Szene zu generieren. (Die automatisch generierte Szene ist OG_SCENE_3D_ELLIPSOID_WGS84) Diese Szene kann folgendermassen erhalten werden: scene = ogGetParentObject(globe);
int **ogPickGlobe**(int scene,float mx, float my, double* lng, double* lat, double* elv);	**Extends scene object.** Picking auf dem Globus. Gibt die Position in Länge, Breite und Ellipsoid-Höhe zurück.
void **ogToCartesian**(int scene, double lng, double lat, double elv,double* x,double* y,double* z);	**Extends scene object.** Konvertiert eine WGS84 Position (Länge, Breite, Höhe) in das interne kartesische System.
void **ogToWGS84**(int scene, double x, double y, double z, double* lng,double* lat,double* elv);	**Extends scene object.** Konvertiert eine interne kartesische Koordinate nach WGS84 (Länge, Breite, Höhe).
void **ogSetPositionWGS84**(int camera, double longitude, double latitude, double elv);	**Extends camera object.** Set the camera position as WGS84 coordinate. Please refer to the camera object for more information about cameras.
void **ogSetOrientationWGS84**(int camera, double yaw, double pitch, double roll);	**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.
void **ogLookAtWGS84**(int camera, double longitude, double latitude, double elv);	**Extends camera object.** Create a view matrix for the current camera looking at specified wgs84 position.

3.5.2 3D-Flat-Cartesian

3D-Flat-Cartesian specific functions	
int **ogPickTerrain**(int scene,float mx, float my, double* x, double* y, double* z);	**Extends scene object.** Picking auf dem Terrain. Gibt die Position in Länge, Breite und Höhe zurück. Dies sind kartesische Koordinaten.

3.5.3 2D-Screen

Bei dieser Szene wird in Screen-Koordinaten gearbeitet. Es ist zu beachten, dass der Nullpunkt oben links ist.



z-Werte beim Zeichnen der 2D-Grafiken müssen verwendet werden, da Objekte nach der Tiefe sortiert werden ("Layer").

z muss im Bereich [-1,1] sein, wobei 1 ganz vorne und -1 ganz hinten ist.

z-Werte dürfen niemals für Höhendaten verwendet werden. (Dafür könnte ein Bild mit float-Werten verwendet werden). There is no elevation data in this scene format.

Properties for ogScene

Key	Type	Description
type	OG_PROPERTY_STRING **(read-only, protected)**	The type of the scene. Possible values are: "3D_ELLIPSOID_WGS84", "3D_FLAT_CARTESIAN", "2D_SCREEN" or other scene types provided by extensions.

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	∞

3.6 Camera Object

Create Camera	
int **ogCreateCamera**(int scene);	Erstellen einer neuen Kamera. Hat Default-Werte für eine perspektivische Kamera.
int **ogLoadCamera**(int scene, const char* url);	Lädt eine Kamera von einer URL. (OpenWebGlobe XML Format).
int **ogLoadCameraAsync**(int scene , const char* url);	Lädt eine Kamera von einer URL asynchron. Ist die Kamera fertig geladen wird der ObjectReady-Event ausgelöst.
Destroy Camera	
int **ogDestroyCamera**(int camera);	Destroy Camera. This function has no effect on cameras that are currently active.
Camera Operations	
int **ogGetNavigationController**(int camera);	Retrieve current navigation controller.
void **ogSaveCamera**(int camera, const char* file);	Saves the camera (XML)
void **ogSetProjectionMatrixd**(int camera, double* m);	manuelles setzen der 4x4 Projektionsmatrix.
void **ogSetProjectionMatrixf**(int camera, float* m);	manuelles setzen der 4x4 Projektionsmatrix.
void **ogPerspectiveProjection**(int camera, float fovy, float aspect, float zNear, float zFar);	Erstellt eine perspektivische Projektionsmatrix. Fovy gibt das Betrachtungsfeld (Field of View) entlang der x-Achse. aspect ist das Verhältnis Breite zu Höhe.

void **ogOrtho2D**(int camera, float left, float right, float bottom, float top);	Definiert eine 2-dimensionale orthogonale Projektionsmatrix.
void ogSetViewMatrixd(int camera, double* m);	Set view matrix manually using 4x4 double values.
void ogSetViewMatrixf(int camera, float* m);	Set view matrix manually using 4x4 float values
void **ogLookAt**(int camera, double toX, double toY, double toZ);	Look at specified point (cartesian)
void **ogSetOrientation**(int camera, double yaw, double pitch, double roll);	Set orientation (cartesian)

Properties for ogCamera

Key	Type	Description

Parent Object

The parent object always is from type ogScene

Child Objects

Object	Min	Max
ogNavigationController	1	1

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	
int **ogCreateNavigationController**(int camera, unsigned int type);	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: OG_NAVIGATION_DEFAULT (default navigation) OG_NAVIGATION_CUSTOM (custom, overload appropriate navigation events) (Standard navigation types are planned and will be implemented over time.)
Navigation Command	
void **SendNavigationCommand**(int navigation, const char* command);	Send navigation specific commands. Example: flyto(lat, lng, elv)
Navigation Controller Operations	
void **ogSetNavigationEventInit**(int nav, CallBack_NavigationInit cbf);	Set callback for navigation
void **ogSetNavigationEventExit**(int nav, CallBack_NavigationExit cbf);	Set callback for navigation
void **ogSetNavigationEventMouseDown**(int nav, CallBack_NavigationMouseDown cbf);	Set callback for navigation
void **ogSetNavigationEventMouseUp**(int nav, CallBack_NavigationMouseUp cbf);	Set callback for navigation
void **ogSetNavigationEventMouseMove** (int nav, CallBack_NavigationMouseMove cbf);	Set callback for navigation
void **ogSetNavigationEventMouseWheel** (int nav, CallBack_ NavigationMouseWheel cbf);	Set callback for navigation
void **ogSetNavigationEventKeyDown** (int nav, CallBack_NavigationKeyDown cbf);	Set callback for navigation
void **ogSetNavigationEventKeyUp**(int nav, CallBack_NavigationKeyUp cbf);	Set callback for navigation
void **ogSetNavigationEventPreRender** (int nav, CallBack_NavigationPreRender cbf);	Set callback for navigation
void **ogSetNavigationEventPostRender**(int nav, CallBack_NavigationPostRender cbf);	Set callback for navigation
void **ogSetNavigationEventResize**(int nav, CallBack_NavigationResize cbf);	Set callback for navigation
void **ogSetNavigationEventTimer**(int nav, CallBack_NavigationTimer cbf);	Set callback for navigation
void **ogSetNavigationEventCommand** (int nav, CallBack_NavigationCommand cbf);	Set callback for navigation

Callback Definitions for navigation

typedef void (***CallBack_NavigationInit**)(int);	(camera)	Called when camera is created. (This callback is done for convienience 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.)
typedef void (***CallBack_NavigationExit**)(int);	(camera)	Called when camera is deleted
typedef void (***CallBack_NavigationMouseDown**)(int, int, int, int);	(camera, button, mouseX, mouseY)	Called when mouse button is pressed
typedef void (***CallBack_NavigationMouseUp**)(int, int, int, int);	(camera, button, mouseX, mouseY)	Called when mouse button is released
typedef void (***CallBack_NavigationMouseMove**)(int, int, int);	(camera, mouseX, mouseY)	Called when mouse is moved
typedef void (***CallBack_NavigationMouseWheel**)(int, float);	(camera, delta)	Called when mouse wheel is used
typedef void (***CallBack_NavigationKeyDown**)(int, unsigned int);	(camera, key)	Called when key is pressed
typedef void (***CallBack_NavigationKeyUp**)(int, unsigned int);	(camera, key)	Called when key is released
typedef void (***CallBack_NavigationPreRender**)(int);	(camera, scene)	Called before anything is rendered
typedef void (***CallBack_NavigationPostRender**)(int);	(camera, scene)	Called when everything is rendererd
typedef void (***CallBack_NavigationResize**)(int, int, int);	(camera, width, height)	Called when window is resized
typedef void (***CallBack_NavigationTimer**)(int, float);	(camera, delta_time)	Called every frame for timing
typedef void (***CallBack_NavigationCommand**)(int, const char*);	(camera, command)	Called when navigation command is sent.

3.8 World Object (Terrain / Globe)

Create World Object	
int **ogCreateWorld**(int scene);	Erstellen einer Welt (z.B. virtueller Globus)
int **ogLoadWorld**(int scene, const char* url);	Lädt ein World Object (XML Definition) von der angegebenen URL.
int **ogLoadWorldAsync**(int scene, const char* url);	Lädt ein World Object asynchron von einer URL. Ist das Bild fertig geladen wird der ObjectReady-Event ausgelöst.
Destroy World Object	
void **ogDestroyWorld**(int world);	Entfernt das Welt-Objekt
World Object Operations	
void **ogSaveWorld** (int world, const char* filename);	Speichert das World Object als XML (lokales file).
void **ogShowWorld**(int world);	Die Welt wieder sichtbar machen (welcher zuvor mit ogHideWorld versteckt wurde).
void **ogHideWorld**(int world);	Virtuelles entfernen der Welt. Die Welt wird nicht mehr gerendert und neue Daten werden nicht mehr heruntergeladen. Aktuelle Daten bleiben jedoch im Speicher. Collision Checks und andere abfragen sind nicht mehr möglich. Hinzufügen und Entfernen von Layern funktioniert zwar, ist aber nicht sichtbar.
void **ogSetRenderoption**(int world, unsigned int renderoption);	Setzen von Render Optionen für den Globus. Dies ist zur Zeit beschränkt auf: OG_RENDEROPTION_WIREFRAME (Wert 0 oder 1)
void **ogRenderWorld**(int world);	Manuelles Rendern des Terrain / Globus. Diese Funktion muss aufgerufen werden, wenn ein Render-Callback definiert wurde. (Es ist somit z.B. auch möglich den Globus mehrfach zu rendern in verschiedenen Render-Pässen oder mit verschiedenen Viewports).

Properties for ogWorld

Key	Type	Description

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	∞

3.8.1 Image Layer

Create Image Layer Object	
int **ogAddImageLayer**(int world, const char* setup);	Hinzufügen eines Bild-Layers.
Destroy Image Layer Object	
void **ogRemoveImageLayer**(int imagelayer);	Entfernen (und Löschen) eines Bild-Layers.
Image Layer Operations	
void **ogSwapImageLayers**(int layer1, int layer2);	Ändert Layer Reihenfolge durch Austauschen zweier Layer
int **ogSetImageLayerTransparency**(int imagelayer, float transparency);	Setzt die Transparenz eines Image Layers. Transparenz ist immer im Range [0,1].
void **ogSetImageLayerMaxLod**(int imagelayer, unsigned int maxlod);	Setzt das maximale LOD eines Image Layers. (nur für externe Dieste, z.B. OpenStreetMap)
unsigned int **ogGetNumImageLayers**();	Get number of layers.
int **ogGetImageLayerAt**(unsigned int index);	Retrieve layer at specified index. Index ranges from 0 to GetNumImageLayers()-1.

Properties for ogImageLayer

Key	Type	Description

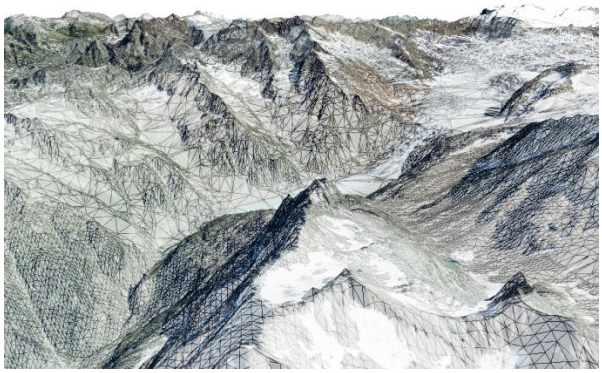
Parent Object

The parent object always is from type ogWorld

Child Objects

There are no child objects.

3.8.2 Elevation Layer



DHM 25, SWISSIMAGE © swisstopo (JA100071)

Create Elevation Layer Object	
int **ogAddElevationLayer**(int world, const char* setup);	Hinzufügen eines Höhendaten-Layer.
Destroy Elevation Layer Object	
void **ogRemoveElevationLayer**(int elevationlayer);	Entfernen (und Löschen) eines Höhendaten-Layers.
Elevation Layer Operations	
void **ogSwapElevationLayers**(int layer1, int layer2);	Ändert Layer Reihenfolge durch Austauschen zweier Layer.
unsigned int **ogGetNumElevationLayers**();	Get number of layers.
int **ogGetElevationLayerAt**(unsigned int index);	Retrieve layer at specified index. Index ranges from 0 to GetNumElevationLayers()-1.

Properties for ogElevationLayer

Key	Type	Description

Parent Object

The parent object always is from type ogWorld

Child Objects

There are no child objects

3.8.3 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.

Properties for ogWaypointLayer

Key	Type	Description

Parent Object

The parent object always is from type ogWorld

Child Objects

There are no child objects

3.8.4 POI Layer

Create POI Layer Object	
int **ogAddPOILayer**(int world, const char* setup);	
Destroy POI Layer Object	
void **ogRemovePOILayer**(int poilayer);	
POI Layer Operations	
void **ogSwapPOILayers**(int layer1, int layer2);	Ändert Layer Reihenfolge durch Austauschen zweier Layer.
unsigned int **ogGetNumPOILayers**();	Get number of layers.
int **ogGetPOILayerAt**(unsigned int index);	Retrieve layer at specified index. Index ranges from 0 to GetNumPOILayers()-1.

Properties for ogPoiLayer

Key	Type	Description

Parent Object

The parent object always is from type ogWorld

Child Objects

There are no child objects

3.8.5 Geometry Layer

Create Geometry Layer Object	
int **ogAddGeometryLayer**(int world, const char* setup);	
int **ogCreateGeometryLayer**(int globe, const char* layername);	Erstellt einen leeren Geometrie-Layer. In diesen Layer können zur Laufzeit neue Geometrie-Objekte hinzugefügt werden.
Destroy Geometry Layer Object	
void **ogRemoveGeometryLayer**(int geometrylayer);	
Geometry Layer Operations	
void **ogSwapGeometryLayers**(int layer1, int layer2);	Ändert Layer Reihenfolge durch Austauschen zweier Layer.
unsigned int **ogGetNumGeometryLayers**();	Get number of layers.
int **ogGetGeometryLayerAt**(unsigned int index);	Retrieve layer at specified index. Index ranges from 0 to GetNumGeometryLayers()-1.
void **ogAddStaticGeometry**(int layer, int geometry);	Fügt eine statische Geometrie dem Layer hinzu. Statisch heisst, dass die Position/Orientierung des 3D Objektes sich **nicht** ändern darf. (Eine GPU Version der Geometrie wird gegebenfalls erzeugt.) Die Geometrie muss entsprechend georeferenziert sein oder mit SetGeometryPosition definiert worden sein.
void **ogAddDynamicGeometry**(int layer, int geometry);	Fügt eine statische Geometrie dem Layer hinzu. Dynamisch heisst, dass die Position/Orientierung des 3D Objektes sich ändern darf. (Eine GPU Version der Geometrie wird gegebenfalls erzeugt.) Die Geometrie muss entsprechend georeferenziert sein oder mit SetGeometryPosition definiert worden sein.
void **ogRemoveGeometry**(int geometry);	Entfernt (und löscht!) die Geometrie vom Layer. Das Geometrie-Objekt kann nicht weiter verwendet werden.

Properties for ogGeometryLayer

Key	Type	Description

Parent Object

The parent object always is from type ogWorld

Child Objects

There are no child objects

3.8.6 Voxel Layer

Punktobjekte mit einer sehr grossen Anzahl Punkte (Punktwolken) können in ein Voxel-Objekt umgewandelt und über den Voxel Layer auf dem virtuellen Globus mit level of detail visualisiert werden.

Create Voxel Layer Object	
int **ogAddVoxelLayer**(int world, const char* setup);	
Destroy Voxel Layer Object	
void **ogRemoveVoxelLayer**(int voxellayer);	
Geometry Layer Operations	
void **ogSwapGeometryLayers**(int layer1, int layer2);	Ändert Layer Reihenfolge durch Austauschen zweier Layer.
unsigned int **ogGetNumGeometryLayers**();	Get number of layers.
int **ogGetGeometryLayerAt**(unsigned int index);	Retrieve layer at specified index. Index ranges from 0 to GetNumGeometryLayers()-1.

Properties for ogVoxelLayer

Key	Type	Description

Parent Object

The parent object always is from type ogWorld

Child Objects

There are no child objects

3.9 Image Object

Create Image Object	
int **ogCreateImage**(int scene, int width, int height, unsigned int pixelformat);	Erstellt ein leeres Bild. Format darf nur folgendes sein: OG_PIXELFORMAT_RGB (8 bit pro Kanal RGB) OG_PIXELFORMAT_RGBA (8 bit pro Kanal RGBA)
int **ogLoadImage**(int scene, const char* url);	Lädt ein Bild von einer URL. (jpg, png). Das Bildformat wird automatisch erkannt.
int **ogLoadImageAsync**(int scene, const char* url);	Lädt ein Bild asynchron von einer URL. Ist das Bild fertig geladen wird der ObjectReady-Event ausgelöst.
Destroy Image Object	
int **ogDestroyImage**(int image);	Zerstört ein Image Objekt (Speicherfreigabe).
Image Operations	
void **ogSaveImage**(int image, const char* filename, const char* formatoptions);	Speichert ein Bild (Binär-Format). Die Format-Option muss folgendes sein: "format=png" -> PNG-Format "format=jpg" -> JPG-Format (ohne alpha) "format=ppm" -> PPM-Format (ohne alpha) ist der format-string ungültig oder leer, so wird in im PNG-Format gespeichert.
int **ogGetImageWidth**(int image);	Gibt die Image-Breite zurück
int **ogGetImageHeight**(int image);	Gibt die Image-Höhe zurück
unsigned int **ogGetImagePixelFormat**(int image);	Gibt das Pixel format Format des Bildes zurück (mögliche Werte siehe ogCreateImage)
void **ogAddAlphaChannel**(int image);	Fügt einen Alpha-Kanal hinzu. Falls das Bild bereits einen Alpha Kanal besitzt, so geschieht nichts. Der neue Alpha Wert ist immer auf 255 gesetzt. Das Bildformat ist nach dieser Operation OG_PIXELFORMAT_RGBA.
int **ogResizeImage**(int image, int newwidth, int newheight);	Resize Image auf neue Grösse. Es wird ein Bilinearer Filter verwendet. Gibt ein neues image object zurück.
unsigned char* **ogLockImage**(int image);	Ein Lock auf die Image Daten. Nun kann direkt auf die zurückgegebene Adresse geschrieben werden. Achtung: Maximalgrösse abhängig von Format und Bild-Dimension: Falsches Schreiben in die zurückgegebene Adresse kann zu Programmabstürzen führen!
void **ogUnlockImage**(int image);	Ein zuvor mit Lock versehendes Bild freigeben.

Properties for ogImage

Key	Type	Description
width	OG_PROPERTY_INTEGER **(readonly, protected)**	Width of the image
height	OG_PROPERTY_INTEGER **(readonly, protected)**	Height of the image
url	OG_PROPERTY_STRING **(readonly, protected, optional)**	url of the image. If there is no URL available this key is not available.

Parent Object

The parent object always is from type ogScene

Child Objects

There are no child nodes.

3.10 Texture Object

Create Texture Object	
int **ogLoadTextureAsync**(int scene, const char* url, const char* imageformat, unsigned int pixelformat);	Loads a texure asynchronously imageformat is one the of following strings: "png" or "PNG" for PNG image "jpg" or "JPG" for JPEG image "tga" or "TGA" for TGA image "gif" or "GIF" for GIF image Pixel Format can be: OG_PIXELFORMAT_RGB (4) OG_PIXELFORMAT_BGR (5) OG_PIXELFORMAT_RGBA (6) OG_PIXELFORMAT_BGRA (7)
int **ogCreateTextureFromImage**(int scene, int image);	Erstellt ein Textur-Objekt aus einem Image-Objekt. Rückgabe wert ist -1 falls dies nicht möglich ist (z.B. kein Speicher zu gross) Wählt automatisch die entprechende Grösse, Pixelformat und Typ.
int **ogCreateTexture**(int scene, unsigned int target, int width, int height, unsigned int pixelformat, unsigned int datatype);	Erstellt eine leere Textur. Target: OG_TEXTURE_2D (0) OG_TEXTURE_RECTANGLE (1) PixelFormat: OG_PIXELFORMAT_RED (0) OG_PIXELFORMAT_GREEN (1) OG_PIXELFORMAT_BLUE (2) OG_PIXELFORMAT_ALPHA (3) OG_PIXELFORMAT_RGB (4) OG_PIXELFORMAT_BGR (5) OG_PIXELFORMAT_RGBA (6) OG_PIXELFORMAT_BGRA (7) OG_PIXELFORMAT_LUMINANCE (8) OG_PIXELFORMAT_LUMINANCE_ALPHA (9) DataType: OG_TEXTUREDATATYPE_UNSIGNED_BYTE (0) OG_TEXTUREDATATYPE_FLOAT (1)
Destroy Texture Object	
void **ogDestroyTexture**(int texture);	Löscht eine Textur und gibt dessen Speicher frei.
Texture Operations	
void **ogGetMaxTextureSize**(int* w, int* h);	Gibt die maximale Breite / Höhe zurück die für eine Textur gültig ist. Dies ist i.d.R. Hardware-abhängig.
void **ogUpdateTexture**(int texture, int xoffset, int yoffset, int width, int height, unsigned char* data);	Textur mit neuen Daten auffüllen. Die Daten müssen im selben Format sein wie das welches bei ogCreateTexture angegeben wurde.

Properties for ogTexture

Key	Type	Description
width	OG_PROPERTY_INTEGER **(readonly, protected)**	Width of the texture
height	OG_PROPERTY_INTEGER **(readonly, protected)**	Height of the texture

Parent Object

The parent object always is from type ogScene

Child Objects

There are no child nodes.

3.11 Pixelbuffer Object

Create Pixelbuffer Object	
int **ogCreatePixelbuffer**(int scene, unsigned int bytes);	Erstellt einen Pixelbuffer mit angegebener Grösse in Bytes. Ein Pixelbuffer befindet sich (falls möglich) immer auf der GPU.
Destroy Pixelbuffer Object	
void **ogDestroyPixelbuffer**(int pixelbuffer);	Löscht den zuvor erstellten Pixelbuffer
Pixelbuffer Operations	
unsigned char* **ogLockPixelbuffer**(int pixelbuffer);	Lock auf den Pixelbuffer ermöglicht direktes Schreiben in die zurückgegebene Adresse
void **ogUnlockPixelbuffer**(int pixelbuffer);	Schützt den Pixelbuffer wieder. Dies muss nach der Schreiboperation aufgerufen werden.
void **ogCopyPixelbufferToTexture**(int pixelbuffer,int texture);	Kopiert den Pixelbuffer in eine Textur. Dies geschieht (falls möglich) 100% auf der GPU. Der Pixelbuffer Speicher muss gleich gross sein wie der Textur-Speicher. Achtung: Der Context der Textur und der Context des Pixelbuffers müssen übereinstimmen!

Properties for ogPixelbuffer

Key	Type	Description
size	OG_PROPERTY_INTEGER **(readonly, protected)**	size of the pixelbuffer object in bytes

Parent Object

The parent object always is from type ogScene

Child Objects

There are no child nodes.

3.12 Geometry

3.12.1 Geometry Object

Create Geometry Object	
int **ogCreateGeometry**(int scene, const char* xml);	Erstellt eine Geometrie aus einem OpenWebGlobe Geometrie-XML. Ein leerer String (oder 0) erstellt eine leere Geometrie (Null-Objekt).
int **ogLoadGeometry**(int scene, const char* url);	Lädt eine Geometrie von einer URL.
int **ogLoadGeometryAsync**(int scene, const char* url);	Lädt die Geometrie asynchron von einer URL. Ist das Bild fertig geladen wird der ObjectReady-Event ausgelöst.
Destroy Geometry Object	
int **ogDestroyGeometry**(int geometry);	Zerstört das Geometrie Objekt (komplette Speicherfreigabe).
Geometry Object Operations	
unsigned int **ogGetNumGeometries**();	Gibt die totale Anzahl der Geometrien zurück.
int **ogGetGeometryAt**(unsigned int index);	Gibt die Geometrie am angegebenen index zurück. Der Range ist: 0..ogGetNumGeometries()-1. Wird ein ungültiger Index angegeben, so wird -1 zurückgegeben.
void **ogSaveGeometry** (int geometry, const char* filename, unsigned int format);	Speichert die Geometrie. Format muss zur Zeit immer auf 0 gesetzt werden (OpenWebGlobe XML).
void **ogSetGeometryPosition**(int geometry, x, y, z);	Definiert die globale Position des 3D Objektes, Auf dem globus ist x die Länge, y die Breite und z die ellipsoidische Höhe.
void **ogSetGeometryTransformation**(int geometry, double* M);	Definiert eine Transformation. M hat 16 Werte und ist eine 4x4 Matrix.

Properties for ogGeometry

Key	Type	Description

Parent Object

The parent object always is from type ogScene

Child Objects

There are no child nodes.

3.12.2 Mesh Object

unsigned int **ogGetNumMeshes**(int geometry);	Gibt die Anzahl Meshes (Geometrie-Unterobjekte) zurück.
int **ogGetMeshAt**(int geometry, unsigned int index);	Gibt das Mesh am angegebenen index zurück. Der Range ist: 0..ogGetNumMeshes()-1. Wird ein ungültiger Index angegeben, so wird -1 zurückgegeben.

3.12.2.1 Surface Object

unsigned int **ogGetNumSurfaces**(int mesh);	Gibt die Anzahl Surfaces (Mesh-Unterobjekte) zurück.
int **ogGetSurfaceAt**(int mesh, unsigned int index);	Gibt die Surface am angegebenen index zurück. Der Range ist: 0..ogGetNumMeshes()-1. Wird ein ungültiger Index angegeben, so wird -1 zurückgegeben.

3.12.3 Point Object

3.12.4 Line Object

3.12.5 Material Object

3.12.6 Standard Objekte

Oft ist es hilfreich auf einige vordefinierte Geometrien zurückzugreifen. Die folgenden Objekte erzeugen geometrieobjekte.

int **ogCreateCube**(int scene, int length);	Erstellt einen Würfel mit der angegebenen Seitenlänge (in Meter) • Der Würfel ist texturierbar (alle Seiten haben dieselbe Textur) • Der Würfel hat Normalen • (0,0,0) ist das Zentrum des Würfels
int **ogCreateSphere**(int scene, int radius, int stacks, int slices);	Erstellt eine Kugel mit dem angegebenen Radius (in Meter). Stacks und Slices sind die anzahl geographischen Unterteilungen (Länge / Breite) • Die Kugel ist texturierbar (sphärische Projektion) • Der Kugel hat Normalen • (0,0,0) ist das Zentrum der Kugel
int **ogCreateSquare**(int scene, int length, int numtiles)	Erstellt ein Quadtrat mit "numtiles" Unterteilungen (xz-Ebene). Die Gesamtlänge ist "length". Texturkoordinaten existieren über das gesamte Objekt. Normalen sind vorhanden. (0,0,0) ist das Zentrum.

3.12.7 Text Data

int **ogCreateText**(int scene, const char* text);	Erstellen eines neuen Textobjektes in der default-Sprache (utf-8 encoded)
int **ogLoadText**(int scene, const char* url);	Lädt ein Textobjekt von einer URL. (OpenWebGlobe XML Format)
int **ogLoadTextAsync**(int scene, const char* url);	Lädt ein Textobjekt von einer URL asynchron. Ist das Textobjekt fertig geladen wird der ObjectReady-Event ausgelöst.
int **ogDestroyText**();	Textobjekt Löschen
void **ogSaveText**(int textobject, const char* file);	Speichert das Textobjekt im OpenWebGlobe-XML format.
void **ogAddTranslation**(int textobject, const char* text, const char* LanguageCode)	Fügt eine Übersetzung hinzu. die Language wird als Code übergeben. Der Code sollte nach Definition aus RFC 1766/ISO 639/ISO 3166/RFC 3066, als z.B. "de_DE", "de_CH", "de_AT", "en_US" usw. (Theoretisch ist der Anwender jedoch frei eigene Codes zu verwenden, dies ist aber nicht empfohlen)
void **ogSetDefaultLanguage**(const char* LanguageCode)	Setzt die Standard Sprache. (Standard ist "en_US".)
const char* **ogGetText**(int textobject);	Gibt den Text in der Standard-Sprache zurück
const char* **ogGetTranslation**(int textobject, const char* LanguageCode)	Gibt übersetzten Text zurück. Gibt es für den Text in der angegebenen Sprache keine Übersetzung so wird der Text in der Default-Sprache zurückgegeben (oder falls es diese nicht gibt ein leerer String).
bool ogHasTranslation(int textobject, const char* LanguageCode)	gibt true zurück, fall es für den Text eine Übersetzung in der angegebenen Sprache gibt.

3.12.8 Binary Data

int **ogCreateBinaryData**(int scene);	Erstellen eines neuen Binärobjektes (leer)
int **ogLoadBinaryData**(int scene, const char* url);	Lädt ein Binärdatenobjekt von einer URL.
int **ogLoadBinaryDataAsync**(int scene, const char* url);	Lädt ein Binärdatenobjekt von einer URL asynchron. Ist das Binärdatenobjekt fertig geladen wird der ObjectReady-Event ausgelöst.
void **ogDestroyBinaryData**(int bindata);	Binärobjekt Löschen
unsigned int **ogGetNumBinaryData**();	Gibt die Anzahl Binärdatenobjekte zurück. Kann nie kleiner als 1 sein.
int **ogGetBinaryDataAt**(unsigned int index);	Gibt das Binärobjekt am angegebenen index zurück. 0..ogGetNumBinaryData()-1. Wird ein ungültiger index angegeben, so wird -1 zurückgegeben.
void **ogSaveBinaryData**(int binarydata, const char* file);	Speichert das Binärdatenobjekt in einem File.

3.12.9 Light

int **ogCreateLight**(int scene, const char* xml);	Erstellen eines neuen Lichtes
int **ogLoadLight**(int scene, const char* url);	Lädt ein Licht von einer URL. (OpenWebGlobe XML Format)
int **ogLoadLightAsync**(int scene, const char* url);	Lädt ein Licht von einer URL asynchron. Ist das Licht fertig geladen wird der ObjectReady-Event ausgelöst.
void **ogDestroyLight**(int light);	Licht löschen
void **ogSaveLight**(int light, const char* file);	Speichert das Licht im OpenWebGlobe-XML format.

4 Operating System Specific Functions

4.1 Windows OS Specific

HWND **wogGetWindowHandle**(int context);	Gibt das HWND des Context zurück.
void **wogSetParentWindowHandle**(int context, HWND parent);	Setzt das Parent HWND des context. Dies funktioniert nur im Windowed Mode (nicht Fullscreen!). So kann man GUI Toolkits (z.B: Window.Forms) mit OpenWebGlobe mischen.

4.2 MacOS X Specific

not yet available

4.3 iPhone/iPad Specific

not yet available

4.4 Android Specific

not yet available

4.5 WebGL Specific

not yet available

5 Misc Functions

int **ogExec**();	Ausführen des Applikations-Loops.
void **ogMemoryDump**(const char* file);	Schreibt den aktuellen Speicher-Zustand in die angegebene Datei (überschreiben). Alle Objekte werden aufgelistet inkl. Speicherbedarf.
const char* **ogGetString**(unsigned int type);	Gibt einen informationsstring zurück. OG_VERSION: Die OpenWebGlobe Version OG_VENDOR: Der Hersteller der OpenWebGlobe Bibliothek.
const char* **ogGetLastError**();	Beschreibung des letzten Fehlers, 0 wenn kein Fehler vorhanden ist.
float **ogRand01**();	Returns a pseudo random value in the range [0,1]

6 Events

6.1 Event-Callback-Funktionen

typedef void (***CallBack_MouseDown**)(int, int, int, int);	(context, button, mouseX, mouseY)	Called when mouse button is pressed
typedef void (***CallBack_MouseUp**)(int, int, int, int);	(context, button, mouseX, mouseY)	Called when mouse button is released
typedef void (***CallBack_MouseMove**)(int, int, int);	(context, mouseX, mouseY)	Called when mouse is moved
typedef void (***CallBack_MouseWheel**)(int, float);	(context, delta)	Called when mouse wheel is used
typedef void (***CallBack_KeyDown**)(int, unsigned int);	(context, key)	Called when key is pressed
typedef void (***CallBack_KeyUp**)(int, unsigned int);	(context, key)	Called when key is released
typedef void (***CallBack_Render**)(int);	(scene)	Main Render Loop
typedef void (***CallBack_Resize**)(int, int, int);	(context, width, height)	Called when window is resized
typedef void (***CallBack_Timer**)(int, float);	(context, delta_time)	Called every frame for timing
typedef void (***CallBack_Init**)(int);	(context)	Called when Render API is ready
typedef void (***CallBack_Exit**)(int);	(context)	Called before Render API is closed
typedef void (***CallBack_RenderGeometry**)(int, int)	(mesh, pass)	Called to render Geometry (custom)
typedef void (***CallBack_BeginRender**)(int, int)	(scene, pass)	Called when render pass begins
typedef void (***CallBack_EndRender**)(int, int)	(scene, pass)	Called when render pass ends
typedef void(***CallBack_Object**)(int)	(object)	Object specific callback (for example ready, failure, ...)