



Webaddon3D extension

Application Programming Interface (API)

Version: 1.5.0.0

Author: Adrian Egli

Contact:

www.as-software.ch

info@as-software.ch

++41 41 544 00 02

All rights reserved by AS Software GmbH, Switzerland
July, 2009

Index

Webaddon3D extension	1
General description of webaddon3D.....	3
Integration and configuration.....	3
Webaddon3D interaction	4
Implemented technology.....	4
Communication.....	5
Application Programming Interface.....	5
Quick and Easy: JavaScript API.....	6
For Experts: XML based configuration.....	6
Webaddon3D XML API.....	6
Content modification / Commands	6
XML Database modifications	7
XML Database	7
XML Model: OpenSceneGraph Model or osgEarth.....	7
XML GOTO object.....	7
XML Shaders.....	7
XML visibility	7
XML collision	7
XML texturing.....	8
XML model manipulation (Draggers).....	8
XML animation.....	8
Command execution with webaddon3D XML	9
XML screen command.....	9
Events and Callbacks.....	10
Console output / extension debug	11
Build-in environment variables.....	12
APPENDIX	13
Document Type Definition (DTD).....	14
Table of Functions.....	15
Webaddon3D Actions/Callbacks.....	17

General description of webaddon3D

Webaddon3D technology allows 3D rendering in html based web content and reads the 3D data from an http server. It supports the Mozilla Gecko Extensions API for Firefox and the Windows ActiveX interface for Internet Explore. The concept of webaddon3D offers an easy integration into existing web contents/pages. Nevertheless application developers should well understand our technology.

With deep understanding of the webaddon3D's ability, application developers can quickly develop robust, high-performance and fantastic looking 3D applications. You can develop a graphical user interface (GUI) with pure html or Macromedia Flash. The latest Ajax web technology is completely compatible with our technology. Your Ajax understanding is similar to webaddon3D, once you have well studied the Ajax concept you will be able to understand how complete our interface is to write nice Ajax and JavaScript based web application using webaddon3D. It's much easier to get a nice looking web based 3D application than a pure C++ written GUI application. The web technology gives complete new possibility to visualize content in 3D and 2D. Images, text and other kind of information can all be visualized and displayed around our extension. You can even pass some information to display into the 3D content.

The underlying 3D engine is the famous OpenSceneGraph render engine. The whole extension is purely written in C++. The webaddon3D technology supports the latest OpenGL standard, GLSL shaders and is optimized for multi core architectures.

Integration and configuration

Webaddon3D contains of a web administration tool. This administration tool allows configuring your web content with a GUI. The graphical user interface works with latest Ajax web technology. For more details and trials, just visit webaddon3D administration tool at <http://3d.assoftware.ch>. You can open your own account. The first ten days are for free (demo period).

With the administration tool you can modify all startup settings in a graphical and easy to understand way. By using the administration tool you get a fully configured integration template. This template allows integrating webaddon3D content into any existing web content by 'copy & past'. The template looks similar to this example:

```
<div id="W3D127" style="position:relative; left:0px; top:0px; width:800px; height:600px; z-index:1">
<!-- webaddon3D (c) by AS SOFTWARE GMBH -->
<script type="text/javascript" src="http://webaddon3d.assoftware.ch/webaddon3d.js"></script>
<script type="text/javascript" src="http://3d.assoftware.ch/user/59/127/project.js"></script>
<!-- support: www.webaddon3d.com -->
</div>
```

Your 3D content/model can be hosted on our infrastructure or on yours. The advantage of hosting on our infrastructure is that you don't need care about the servers and theirs administrations. If you prefer using your own servers - no problem, you can use them without any restrictions.

The webaddon3D displays a start button in the center of the 3D container. The user of the 3D application has to start manually the 3D rendering. For some kind of applications we like to start the 3D renderer automatically. Webaddon3D has a function called StartRenderer. This function forces the extension to open its 3D context and start the renderer without user interaction. But for Internet Explorer and Firefox we shouldn't start the 3D rendering with a StartRenderer call (generally). For a small number of special applications, we have to start the renderer by such a JavaScript call. But take care about the behavior - especially with Firefox 3 based applications.

Webaddon3D interaction

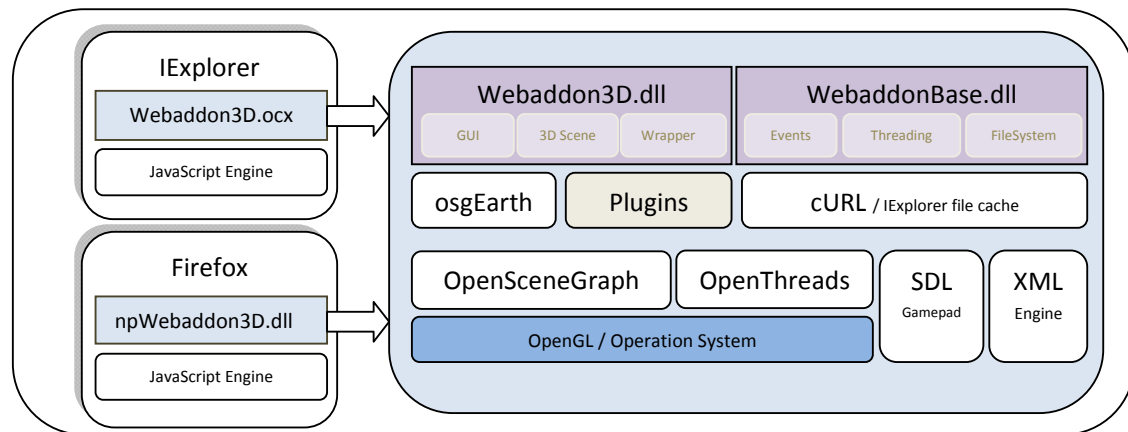
The webaddon3D technology supports bi-directional communications. It's quite important to well understand the main features and their impacts on application development. This chapter should explain in detail the concepts of our webaddon3D communication layer.

Since the browsers use multiple threads to optimize user interactions, we have to insure that all communications are thread-safe too. Our technology is completely optimized on multithreading and synchronizes automatically all communications. It's robust and thread-safe - the application developers don't need to care about thread-safety of their web applications using webaddon3D. We solved for you the huge synchronization problem, especially for real time applications.

Implemented technology

Microsoft Windows

The implemented interaction technology is based on scriptable Mozilla Firefox extension (Gecko) / ActiveX component. Both implementations load the same dynamic linked webaddon3D library. The extension implements a wrapper around our library and the Internet Explorer/Firefox API.



Webaddon3D shares resources between instances. This reduces the memory consumption by avoiding multiple instances of the webaddon3D core. Starting and ending of instances force a well designed synchronization. Adding or removing of instance need to stop all active rendering and restart the thread management with changed number of instances. Even for the JavaScript Engine we have to guarantee that each instances runs correctly under multi threading. The synchronization is implemented in WebaddonBase as one of its main feature. Webaddon3D is our abstraction for the

OpenSceneGraph render engine / scene graph and for the wrappers. Our library manages also the external OpenSceneGraph libraries as osgEarth, Quick Time technology or others plugins.

One difficult problem is the closing/shutting of the browser instance. All of the allocated webaddon3D resources must be unloaded before the browser instance gets closed.

Macintosh OS/X

Under Macintosh OS/X operating system we currently support Appel's Safari browser . We implemented a similar concept to the Microsoft Windows extension (Firefox) . The OS/X extensions is based on WebKit Technology and it's Window Manager is fully integrated into OpenSceneGraph (Cocoa). There should not be any complex integration stuff to solve. It should work very similar like Firefox (embed) extensions. We implemented the same API as we did for Microsoft Windows ActiveX based extension, and Mozilla Firefox based. We are looking for the OS/X Firefox integration in near future .

Communication

Webaddon3D offers a bi-directional communication. You can call webaddon3D functions from JavaScript to interact with our extension. There a functions to change the behavior of the 3D content. This functions delegates some command to the extension. It acts on the called command in safe way and as soon as possible. Some calls are immediately invoked others gets handled on next synchronized phase. Others functions help to retrieve information from the extension. Many other technologies use only this kind of functions. But webaddon3D technology offers a second communication layer; you can connect to callbacks and events. The callbacks get invoked after a important state has changed or some important information gets for the first time retrieved. Like information about webaddon3D license, OpenGL support and graphical processing unit (GPU) with driver version. – There are also events ready to bind with you JavaScript environment. If you are interested in mouse events you should bind to our events.

It's important to understand why you have to explicitly bind callbacks and events. All callbacks or events define a function header (API). You have to implement this header in you JavaScript scripts. And you have to register the callback function with webaddon3D. Once webaddon3D gets a callback or event registered with a JavaScript function, it connects the internal callback or event pipeline. Only with a connected pipeline the callback or event gets transferred to the JavaScript engine (browser). To get high performance application, you should only register callbacks or events when you really need the information. Have also a look at [Events and Callbacks](#) in this document.

Application Programming Interface

Webaddon3D has two different interfaces to modify the behavior of current content. The easiest way is to use the well known JavaScript API. With our JavaScript API application developers can interact with the extension.

Quick and Easy: JavaScript API

Webaddon3D JavaScript API: You can modify in real time the behavior of webaddon3D. Have a look at the complete table of methods. The API gives access to Level of Detail, Sun position, Fog, Anaglyph Stereo rendering, Camera, Camera Path. For complex 3D content modification you should use the webaddon3D XML instead of JavaScript API. The complete [Table of Functions](#) can be found in the [APPENDIX](#) of this document.

For Experts: XML based configuration

There is also an XML based API for more user control in scene graph manipulation. With the XML API application developers can modify 3D content in real time and have easy access to the 3D content (database). Through this API you act user interactions and change interactively the behavior of the content. The XML API is more complex to use because you need to convert the communication/interaction into webaddon3D XML syntax.

Webaddon3D XML API

For complex interactions with webaddon3D you should use our XML based API. It allows modifying interactively the content as well as the JavaScript does, but under full user control. There are two JavaScript methods for XML support. If you prefer to pass the configuration by a string: call `HandleXML(...xml string...)` or if you prefer to pass a simple URL to a XML file/stream call: `HandleXMLAsFile(...url...)`. The XML based API allows content changing and as well the execution of complex commands. A sample of such complex command is the scene manipulation. You can append an object to manipulate the scene, the so called Dragger. A Dragger gets attached to a sub scene (model). For more detail have a look at the XML Description for [Draggers](#).

The [webaddon3D XML](#) is described in the Document Type Definition (DTD) language, which is fully listened in the [APPENDIX](#) of this document.

Example:

<http://webaddon3d.assoftware.ch/webaddon3D/tests/webaddon3d.xml>

http://webaddon3d.assoftware.ch/webaddon3D/tests/webaddon3d_commands.xml

Content modification / Commands

Each file starts with a tag called `<WEBADDON3D>` and the attribute `ACTION` has to be. If you like to append a new model or just modify the database, then the `ACTION` attribute has to be set to `ADD` for appending some content, for modifications use `UPDATE`, and for removing use `REMOVE`. If you like to execute a command: `EXECUTE`. When the attribute `ACTION` is set to `EXECUTE` only the `<COMMANDS>` tag get parsed, the `<DATABASE>` tag gets ignored. Or vice-versa if the `ACTION` tag is `ADD`, `UPDATE` or `REMOVE` then the `<COMMANDS>` tag gets ignored. So don't mix commands with database updates. When you like to `UPDATE` the webaddon3D content by XML you need only to rewrite the involved part of XML. Webaddon3D will replace the current XML behavior with the rewritten one.

XML Database modifications

XML Database

This sub-chapter describes the database modification in detail. The webaddon3D needs one `<DATABASE>` tag defined and at least one or more `<MODEL>` tags. Important is the model's attribute `ID`. Each model needs numerical identifier defined by the attribute `ID`. All further operations are bind with this passed object/model identifier (`ID`).

XML Model: OpenSceneGraph Model or osgEarth

If you like to add a new model to the content use a webaddon3D XML with a `<WEBADDON3D ACTION="ADD">` and a model tag with a identifier `<MODEL ID ="#">`. Then you need to pass an OpenSceneGraph 3D readable file `<FILE>filename</FILE>` or an osgEarth root file `<EARTH>filename</EARTH>`. The database gets updated as soon as possible, in a thread-safe way.

The `<MODEL>` can now have optional further tags. (The `<FILE>` or `<EARTH>` tag is only required when you execute `<WEBADDON3D ACTION="ADD">`). If you wish to get a mouse trigger object, append a `<GOTO>` object declaration; for others functionality you have just to set the required XML tag.

XML GOTO object

A `<GOTO>` object can highlight some text, when the mouse cursor touch the 3D content of the `<MODEL>`, you can append a `<GOTO>` camera location when the user double-click (left mouse button) the `<GOTO>` object (`<MODEL>`). The location is defined by 3D world coordinates `<POSITION> <X>double</X> <Y>double</Y> <Z>double</Z> <POSITION>`. The camera orientation is defined by heading, pitch and roll `<ORIENTATION> <H>double</H> <P>double</P> <R>double</R> </ORIENTATION>`. You should define also the displayed text by the tag `<DESCRIPTION>text</DESCRIPTION>`. For the movement duration you can define the time in milliseconds with the tag `<DURATION>double</DURATION>`.

XML Shaders

There also a way to append a predefined GLSL shader like WIND, in a next version of webaddon3D you can pass a fragment shader and a vertex shader in GLSL slang. But not yet implemented in the current version. `<SHADER>`

XML visibility

You can switch off the model by passing `<VISIBILITY MODE="0"/>` or switch on `<VISIBILITY MODE="1"/>`. If you don't change the visibility explicitly, it's always on.

XML collision

You can disable collision detection (motion model collision check) by `<COLLISION MODE="0"/>`, if you pass one the collision detection gets active. If you don't change the value, collision is set to the default value defined in the administration tool.

XML texturing

To change the model texturing you can access to all texture layers in the model. Disable, enable a specified layer is as easy as the replacement of the blend function; change blend function's color, transparency and the OpenGL function itself. Have a look following the example:

```
--<TEXTURING>
--<TEXTURE UNIT="0">
  <VISIBILITY MODE="1">
  --<BLENDFUNCTION MODE="REPLACE">
  --<COLOR>
    <RED>123</RED>
    <GREEN>123</GREEN>
    <BLUE>225</BLUE>
    <ALPHA>255</ALPHA>
  </COLOR>
  </BLENDFUNCTION>
</TEXTURE>
--<TEXTURE UNIT="1">
  <VISIBILITY MODE="0">
</TEXTURE>
</TEXTURING>
```

The texture layer zero is visible and we change the blend function to *REPLACE*. The blend function color gets updated by the new RGBA color. {0,..,255}

The second texture layer (one) gets disabled.

XML model manipulation (Draggers)

You can append or remove a dragger from the **<MODEL>** by the tag **<MANIPULATOR>** which needs a concrete **<DRAGGER>** defined. You can chosen from several draggers for your application behavior. Webaddon3D support following draggers:

```
TAB_PLANE_DRAGGER
TAB_PLANE_TRACKBALL_DRAGGER
TRACKBALL_DRAGGER
TRANSLATE_1D_DRAGGER
TRANSLATE2_D_DRAGGER
TRANSLATE_AXIS_DRAGGER
TAB_BOX_DRAGGER
```

To select a dragger use the tag **<DRAGGER TYPE = " TAB_BOX_DRAGGER"/>**. To remove the dragger from the **<MODEL>** or deactivate it use **<DRAGGER TYPE="REMOVE_ANY_DRAGGERS"/>**.

XML animation

If your model contains an animation you start, pause, stop it with **<ANIMATION LOOPMODE="NO_LOOPING" AUTO_PAUSE_TIME="1.5">**. The **<ANIMATION>** has many attribute to set: **LOOPMODE**, **TIME_MULTIPLIER**, **TIME_OFFSET**, **PAUSE** and **AUTO_PAUSE_TIME**. The **LOOPMODE** can be set to {*SWING*, *LOOP*, *NO_LOOPING*}. The **TIME_MULTIPLIER** defines a factor for speed up or slow down. And there are some attribute for pausing the animation. You can pause the animation or un-pause it with attribute **PAUSE** {0, 1}. You can define an auto pause time, position in the animation execution with attribute **AUTO_PAUSE_TIME** {time/position} or you can start the animation at time/position by settings the attribute **TIME_OFFSET**.

Command execution with webaddon3D XML

You can invoke some commands using webaddon3D XML. You need to define `<WEBADDON3D ACTION = "EXECUTE ">` and the `<COMMANDS>` tag then you can the command to execute.

XML screen command

Use for full screen enable (`<SCREEN MODE="OPEN_FULL" DEVICE="0"/>`) and for disable (`<SCREEN MODE="CLOSE_FULL" DEVICE="0"/>`). The mode defines the screen command, and the device which display should get the renderer window attached.

Events and Callbacks

Some Important notes

Application developers can easily connect their own JavaScript callback functions with webaddon3D callbacks or events. You have to write a JavaScript function fitting the callback API. We are offering several callbacks or events. But you should only connect with a minimum number of them. Each callback interacts with the JavaScript engine implemented in the web browser. This consumes some milliseconds. – But there cost is minimal as long as you don't implement complex stuff inside such callback methods. Take care about the efficiency of the function. Think how and when you have to process the retrieved information. It's important to understand that the callback gets transferred immediately. And it has a (minimal) impact of the render performance. Think about the JavaScript technology, the Ajax concept before you work with callbacks and events. How you can store temporarily the retrieved information for further processing in a second new thread.

Interface for register a callback or action

The JavaScript callback functions needs to be connected by a string (key) and needs to fit to a fixed number of arguments. Webaddon3D invoke the JavaScript method when an event (action) gets raised. With the information passed through the arguments you are now able to implement your own application. To register a event callback you have to use `RegisterAction('*** KEY ***, ' *** your own JavaScript function name ***');` from the JavaScript API. And this should be implemented in a default callback function called *WebaddonRegisterActions*. This callback get raised just after extension initialization, before any 3D content gets loaded.

```
<SCRIPT>

function WebaddonRegisterActions() {

    Webaddon3D.RegisterAction('JS_ACTION_OPENGLINFORMATION','OpenGLInformationAction');

    ...

}

</SCRIPT>
```

The webaddon3D raises a special mouse event. The Mouse3D event is the closest intersection point in the 3D scene (at mouse position clicked).

Webaddon3D Event identifier / key: *JS_ACTION_MOUSE3D*

```
function Mouse3DAction(x,y,z,button,dblClk)
```

The camera path information callback sends information about the current playing camera path.

Webaddon3D Event identifier / key: *JS_ACTION_CAMERAPATHINFORMATION*

```
function CameraPathInformationCallback(information)
```

Webaddon3D collects at starting/initialization useful OpenGL information and raises a OpenGL Information event. By listening to this event you can decide which version of OpenGL is installed. You can choose right 3D content (shaders) to get best rendering/quality on the client system.

Webaddon3D Event identifier / key: **JS_ACTION_OPENGLINFORMATION**

```
function OpenGLInformationCallback(vendor,renderer,version,textureSize,GPU,GLSL)
```

For some reason you need information about the webaddon3D license, so you should connect to License information callback.

Webaddon3D Event identifier / key: **JS_ACTION_LICENCEINFORMATION**

```
function LicenceCallback(valid,expired,expiredDate,validModules)
```

After initialization and all root 3D content get loaded, webaddon3D raises the Load Callback.

Webaddon3D Event identifier / key: **JS_ACTION_LOADCALLBACK**

```
function LoadCallback(loaded)
```

When you have defined a [XML GOTO object](#), webaddon3D raises on touch, on leave event. It raises the event when the mouse cursor pass over the XML GOTO object.

Webaddon3D Event identifier / key: **JS_ACTION_WEBADDONGOTOTOUCH**

```
function GotoTouchAction(objectID,onEnter,onLeave)
```

Console output / extension debug

In application development we work often with limited information. As long as we don't get strange behavior we don't care about log files or console output. But once we do not really understand the application's behavior we wish to get more detailed. For such case webaddon3D has an option to get detailed information at run time. Start Firefox browser with argument `-console` and you get a second window displaying all standard outputs. - Webaddon3D writes still very limited information into this console window. If application developers need more information, you have to set environment/system variables. To force webaddon3D writing out a detailed log, you have to turn on the webaddon3D console output by settings the environment variable **WEBADDON_CONSOLE**. You can put this variable before starting the Firefox in console mode. Under windows set **WEBADDON_CONSOLE =ON**; Unix based system (Linux/osX) export **WEBADDON_CONSOLE =ON**. If you wish to get more information about the OpenSceneGraph rendering system, set the **OSG_NOTIFY_LEVEL** to INFO or DEBUG.

We just explained how to enable the console output in Firefox. The problem is that Internet Explorer doesn't have any console implemented. But when you have some strange behavior with the ActiveX based extension, and you can use Visual Studio or the OCX test container. Normally you get enough information from the Firefox based application.

Build-in environment variables

To get the service of some special debug functionality you have to set some system variables before starting the webaddon3D extension (Firefox). Due of the implementation of Internet Explorer you can't enable the console output. (To debug Internet Explorer and webaddon3D extension you need to attach the visual studio debugger to a running instance). Enable system variables:

`WEBADDON_CONSOLE`

`OSG_NOTIFY_LEVEL`

Debug shadow

Turn on Parallel split shadow map: `WEBADDON_PSSM`, and set a second variable to get debug coloring of the texture/shadow maps (`WEBADDON_DEBUG_PSSM`)

Light space perspective shadow map: `WEBADDON_LISPSM_PSSM`

Simulate and test the sun and sky GLSL shader:

`WEBADDON_SIM_SKY`

Write all parsed webaddon3D XML files/string into this folder:

`WEBADDON_XML_DEBUG_PATH`

Display current frame rate:

`WEBADDON_FRAME_RATE`. Press F1 to toggle the frame information, F2 writes detailed information out (console).

APPENDIX

The appendix contains of the

[Document Type Definition \(DTD\)](#)

[Table of Functions](#)

[Webaddon3D Actions/Callbacks](#)

Document Type Definition (DTD)

The Document Type Definition describes the webaddon3D XML syntax, all webaddon3D XML has to fit to this document schema.

```

<!ELEMENT WEBADDON3D (DATABASE+|COMMANDS+)>
<!ATTLIST WEBADDON3D ACTION (EXECUTE|UPDATE|ADD|REMOVE) #REQUIRED >
<!ELEMENT DATABASE (MODEL+)>
<!ELEMENT MODEL ((EARTH?|FILE?),GOTO?,SHADER?,VISIBILITY?,COLLISION?,TEXTURING?,TRANSFORMATIONS?,MANIPULATOR?,
ANIMATION?)>
<!ATTLIST MODEL ID CDATA #REQUIRED >
<!ELEMENT EARTH (FILE)>
<!ATTLIST EARTH MOTIONMODEL (WEBADDON|TERRAIN|TRACKBALL_AUTO_RESET) #REQUIRED >
<!ELEMENT FILE (#PCDATA)>
<!ELEMENT GOTO (POSITION?,ORIENTATION?,DESCRIPTION?,DURATION?)>
<!ELEMENT SHADER (#PCDATA)>
<!ELEMENT VISIBILITY (#PCDATA)>
<!ATTLIST VISIBILITY MODE (0|1|2) #REQUIRED >
<!ELEMENT COLLISION (#PCDATA)>
<!ATTLIST COLLISION MODE (0|1) #REQUIRED >
<!ELEMENT TEXTURING (TEXTURE+)>
<!ELEMENT POSITION (X,Y,Z)>
<!ELEMENT ORIENTATION (H,P,R)>
<!ELEMENT DESCRIPTION (#PCDATA)>
<!ELEMENT SEQUENCE (DURATION?)>
<!ELEMENT DURATION (#PCDATA)>
<!ELEMENT X (#PCDATA)>
<!ELEMENT Y (#PCDATA)>
<!ELEMENT Z (#PCDATA)>
<!ELEMENT H (#PCDATA)>
<!ELEMENT P (#PCDATA)>
<!ELEMENT R (#PCDATA)>
<!ELEMENT TEXTURE (VISIBILITY?,BLENDFUNCTION?,COLOR?)>
<!ATTLIST TEXTURE UNIT CDATA #REQUIRED >
<!ELEMENT BLENDFUNCTION (COLOR?)>
<!ATTLIST BLENDFUNCTION MODE (DECAL|MODULATE|BLEND|REPLACE|ADD) #REQUIRED >
<!ELEMENT COLOR (RED,GREEN,BLUE,ALPHA)>
<!ELEMENT RED (#PCDATA)>
<!ELEMENT GREEN (#PCDATA)>
<!ELEMENT BLUE (#PCDATA)>
<!ELEMENT ALPHA (#PCDATA)>
<!ELEMENT MANIPULATOR (DRAGGER+)>
<!ELEMENT DRAGGER (#PCDATA)>
<!ATTLIST DRAGGER TYPE
(REMOVE_ANY_DRAGGERS|TAB_PLANE_DRAGGER|TAB_PLANE_TRACKBALL_DRAGGER|TRACKBALL_DRAGGER|TRANSLATE_1D_DRAGGER|TRANSLATE2
_D_DRAGGER|TRANSLATE_AXIS_DRAGGER|TAB_BOX_DRAGGER) #REQUIRED ><!ELEMENT ANIMATION (#PCDATA)>
<!ATTLIST ANIMATION LOOPMODE (SWING|LOOP|NO_LOOPING) #REQUIRED >
<!ATTLIST ANIMATION TIME_MULTIPLIER CDATA #IMPLIED >
<!ATTLIST ANIMATION TIME_OFFSET CDATA #IMPLIED >
<!ATTLIST ANIMATION PAUSE (0|1) #IMPLIED >
<!ATTLIST ANIMATION AUTO_PAUSE_TIME CDATA #IMPLIED >
<!ELEMENT COMMANDS (SCREEN?)>
<!ELEMENT SCREEN (#PCDATA)>
<!ATTLIST SCREEN MODE (OPEN_FULL|CLOSE_FULL) #REQUIRED >
<!ATTLIST SCREEN DEVICE (0|1) #REQUIRED >

```

Table of Functions

The complete JavaScript API is listed and documented in the above list of functions. All functions have a C++ style description. The keyword *void* declares that the method returns no value. Each function defines the returned data, the function name to call and the parameter(s) to pass. Webaddon3D JavaScript API supports two data types; double stands for numerical information, string for text/characters.

void SetLOD(double factor)

Changes the Level of Detail factor. The factor should be a value greater zero.

double GetLOD()

Returns the Level of Detail factor.

void GotoNAVIGATE(double mode, double speedFactor)

This method interact with the webaddon3D motion model. You can rotate, translate the camera and zoom.

mode	description	speed_factor suggestion
0	stop the motion model. Stops all motions	0
1	forward, zoom in	8
2	backward, zoom out	8
3	step left, move left on view plane	8
4	step right, move right on view plane	8
5	step up, move up on view plane	4
6	step down, move down on view plane	4
6	rotate left	0.2
7	rotate right	0.2
8	rotate up	0.1
9	rotate down	0.1

void GotoXYZHPR(double posX, double posY, double posZ, double heading, double pitch, double roll, double time)

This method moves the camera to the Position (posX,posY,posZ) with Orientation (heading, pitch,roll). The movement from current camera position to the new GOTO position has duration of time passed in seconds.

void GotoXYHPR(in double x,in double y,in double groundElevation,in double h,in double p,in double r,in double t)

This method moves the camera to the Position (posX,posY) with Orientation (heading, pitch,roll). The movement from current camera position to the new GOTO position has duration of time passed in seconds. The groundElevation defines the height over terrain.

void PlayPath(in string filename)

Plays a camera Path passed by a filename. The file gets downloaded from the server root directory.

void StopPath()

Stops playing the current camera path

void ResetPath()

Resets the current playing camera path and restarts it.

void ToggleStereo()

Switch anaglyph stereo on/off.

void ToggleGLSLSkyShader()

Switch GLSL sky shader (sun) on/off

double GetX()

Returns the camera position X.

double GetY()

Returns the camera position Y.

double GetZ()

Returns the camera position Z.

double GetHeading()

Returns the camera orientation Heading.

double GetPitch()

Returns the camera orientation Pitch.

double GetRoll()

Returns the camera orientation Roll.

double Get3DMousePosX()

Returns last mouse 3D position X.

double Get3DMousePosY()

Returns last mouse 3D position Y.

double Get3DMousePosZ()

Returns last mouse 3D position Z.

void SetSunPosition(in double phi,in double rho)

Set the sun position by angle phi e [0° .. 360°] and rho = [-90° .. 90°] (height).

void SetFogLinear(dist_min, dist_max, r, g, b, sel)

Set linear fog in the scene. We have only fog within distance greater than dist_min and we have 100% fog for distances greater than dist_max. The r,g,b represent the colour in RGB colour mode. { r,g,b e [0 .. 255] } [sel = 0 : Scene | sel = 1 : Sky System]

void SetFogLinearColour (r, g, b, sel)

Set linear fog in the scene. The r,g,b represent the colour in RGB colour mode. { r,g,b e [0 .. 255] } [sel = 0 : Scene | sel = 1 : Sky System]

void SetFogExp(density, r, g, b, exptype, sel)

Set linear fog in the scene. The r,g,b represent the colour in RGB colour mode. { r,g,b e [0 .. 255] } [sel = 0 : Scene | sel = 1 : Sky System][exptype = 0 : EXP | exptype = 1 : EXP2]

void SetFogExpColour (r, g, b, exptype, sel)

Set linear fog in the scene. The r,g,b represent the colour in RGB colour mode. { r,g,b e [0 .. 255] } [sel = 0 : Scene | sel = 1 : Sky System][exptype = 0 : EXP | exptype = 1 : EXP2]

void DatabaseAddonAdd(string filename)

add a new 3D file to the sub-scene.

void DatabaseAddonShow(string filename)

show (and load if needed) the 3d file filename.

void DatabaseAddonHide(string filename)

Hide the 3D file filename

void MarkerAdd(string key, string description, string image, double x, double y, double z)

Add a marker to the scene. Markers are used to identify interesting places in the 3D scene.

[key : unique identifier. Key will be displayed in the 3D scene. Adding a Marker with a key that is already used will fail.]

[image : image displayed in the 3D scene.]

[x,y,z : absolute 3D marker Position.]

void MarkerAddOnGround(string key, string description, string image, double x, double y, double groundElevation)

Add a marker to the scene.

[Markers are used to identify interesting places in the 3D scene.]

[key : unique identifier. Key will be displayed in the 3D scene. Adding a Marker with a key that is already used will fail.]

[image : image displayed in the 3D scene.]

[x,y : absolute 3D marker position.]

[Elevation : elevation of the Marker in meters from ground.]

void MarkerRemove(string key)

Removes the Marker key in the scene

void MarkerRemoveAll()

Removes all Markers

void SetMotionModelType(int type)

int Get3DMouseButton()

returns the last mouse button click, each time you call the same Get3DMouse...() method, the system will lookup for values changes. If you need the correct 3D position of the click, call each method once, before you recall for a second time one of the method.

void RegisterAction(string key, string callback_function)

[key : JS_ACTION..... identifier]

[callback_function: JavaScript user defined callback function, who fits to the Webaddon3D defined interface.]

string GetLocalID()

return a identifier of the local machine (MAC address), to globally identify the current running session of webaddon3D

void CallPluginCommand(in string pluginIDKey, in string commandStr)

not implemented yet

void HandleXML(in string xmlStr)

parse and process the xml string

void HandleXMLAsFile(in string xmlFileUriRef)

parse and process the xml (file name)

void StartRenderer(in string infoStrMode);

starts the renderer (3D)

string GetVersion()

returns the webaddon3D version

Webaddon3D Actions/Callbacks

The following tables describe in detail the arguments for the callbacks, events raised by webaddon3D library.

KEY	ARGUMENTS		
JS_ACTION_MOUSE3D	X	double	3D coordinate x-axis
	Y	double	3D coordinate y-axis
	Z	double	3D coordinate z-axis
	Button	int	1 = left 2 = middle 4 = right
	Double_click	bool	double click

KEY	ARGUMENTS		
JS_ACTION_CAMERAPATHINFORMATION	Text	string	information

KEY	ARGUMENTS		
JS_ACTION_OPENGLINFORMATION	Vendor	string	OpenGL GPU Vendor
	Renderer	string	Renderer: OpenGL Engine
	Version	string	OpenGL Version
	Txt	int	Max. texture size supported
	Accel	bool	Hardware accelerated
	GLSL	bool	GLSL supported

KEY	ARGUMENTS		
JS_ACTION_LICENCEINFORMATION	Valid	bool	Licence valid
	Expired	bool	Licence expired
	ExpireDate	string	Date
	ValidModule	string	List of all valid modules

KEY	ARGUMENTS		
JS_ACTION_LOADCALLBACK	Loaded	bool	If no error, then flag true

KEY	ARGUMENTS		
JS_ACTION_WEBADDONGOTOTOUCH	ObjID	string	Object identifier
	Touch_Enter	bool	Move mouse into the object from outside
	Touch_Leave	bool	The mouse cursor leaves the touched object

AS Software GmbH has its expertise in web content technology and visualization of 3D content since 2004. One of our first references was the internet based visualization project of the CITY of CANNES, France. The database was the most complex and detailed database of a CITY in Europe at 2005. And since 2006 the virtual 3D city of Cannes is reachable at <http://3d.cannes.fr>. Webaddon3D is really fast and robust, even for such huge database like the city of Cannes.