Documentation for
GMI - The Irrlicht Engine port for Game Maker

written by **Xception**

Irrlicht Device

irrlicht _window_handle=**device_create**(videodriver, x, y, width, height, bitsperpixel, fullscreen, stencilbuffer, vsync, parented)
Starts the Irrlicht engine with selected driver at position x,y and dimension width,height and bitsperpixel colodepth(16/32)
fullscreen, stencilbuffer, vertical synchronistation(avoids tearing): set to true or false
parented(inside GM window): window_handle() for attached to GM window or 0 for not attached

device_makeChild(parent, x, y, width, height)
attach or detach Irrlicht window to/from GM window
parent=window_handle() to attach it or 0 to detach it

device_setKeyboardFocus(window_handle)
sets focus to a new window

device_destroy()
Stops the engine freeing all resources used by it.

device_drop(object)
drops the object, decreases the reference counter by one
returns true if the object was deleted

device_grab(object)
grabs the object increasing the reference counter by one

active=**device_isWindowActive**()
Returns true if window is active. If the window is inactive, nothing need to be drawn.

device_setResizeAble(mode)
true/false

device_setWindowCaption(string text)
Sets the caption of the Irrlicht window.

Videodriver functions

vd_addDynamicLight(light)

texture=**vd_addTexture**(string name, image)
Creates a texture from a loaded image.

texture=**vd_addEmptyTexture**(width, height,string name)
creates an empty texture with dimension width, height and name for the texture

result=**vd_beginScene**(back, z, a, r, g, b)
Applications must call this method before performing any rendering.
args: clear backbuffer?, clear zbuffer?, color:a, r, g, b

returns false if failed

image=**vd_createImageFromFile**(string name)
Creates a software image from a file.
No hardware texture will be created for this image.
This method is useful for example if you want to read a heightmap for a terrain renderer.

vd_deleteAllDynamicLights()
Deletes all dynamic lights which were previously added with addDynamicLight

vd_draw2DImage(img, x, y, rx, ry, rx2, ry2, color, usealpha)
Draws an 2d image, using a color (if color is other than Color(255,255,255,255)) and the alpha channel of the texture if wanted.

vd_draw2DLine(x1, y1, x2, y2, a, r, g, b)
Draws a 2d line.

vd_draw2DPolygon(x1, y1, radius, a, r, g, b, vc)

vd_draw2DRectangle(a, r, g, b, x1, y1, x2, y2)

vd_draw3DBox(minx, miny, minz, maxx, maxy, maxz, a, r, g, b)

vd_draw3DLine(x, y, z, x2, y2, z2, a, r, g, b)

vd_draw3DTriangle(mb, x1, y1, z1, x2, y2, z2, x3, y3, z3, color)

vd_drawMeshBuffer(mb)

vd_drawStencilShadow(clear, color1, color2, color3, color4)

vd_drawStencilShadowVolume(tri, count, zfail)

result=**vd_endScene**()
Presents the rendered image on the screen.
Applications must call this method after performing any rendering.
returns false if failed

type=**vd_getDriverType**()

light=**vd_getDynamicLight**(index)

count=**vd_getDynamicLightCount**()
Returns current amount of dynamic lights set

fps=**vd_getFPS**()
returns frames per second

maxlights=**vd_getMaximalDynamicLightAmount**()
Returns the maximal amount of dynamic lights the device can handle

maxpolys=**vd_getMaximalPrimitiveCount**()

name=**vd_getName**()

polys=**vd_getPrimitiveCountDrawn**()

width=**vd_getScreenWidth**()

height=**vd_getScreenHeight**()

texture=**vd_getTexture**(filename)

flag=**vd_getTextureCreationFlag**(flag)

vd_makeColorKeyTextureFromPos(texture, x, y)
Creates an 1bit alpha channel of the texture based of an color key position.
This makes the texture transparent at the regions where this color key can be found
when using for example draw2DImage with useAlpha = true.

vd_makeColorKeyTexture(texture, a, r, g, b)

vd_makeNormalMapTexture(tex, amp)

result=**vd_queryFeature**(feature)
Queries the features of the driver, returns true if a feature is available

vd_removeAllTextures()
Removes all texture from the texture cache and deletes them, freeing lot of memory.
Please note that after calling this, the pointer to all ITextures may not be longer valid,
if they were not grabbed before by other parts of the engine for storing them longer.
So it would be a good idea to set all materials which are using textures to null first.

vd_removeTexture(texture)
Removes a texture from the texture cache and deletes it

vd_setAmbientLight(a, r, g, b)
Sets the dynamic ambient light color.

vd_setFog(a, r, g, b, linear, start, end , density, pixel, range)
Sets the fog mode. These are global values attached to each 3d object rendered, which has the
fog flag enabled in its material.

vd_render(back, z, a, r, g, b)

vd_setMaterial(mat)

vd_setRenderTarget(tex)
Sets a new render target.
This will only work, if the driver supports the EVDF_RENDER_TO_TARGET feature, which can
be queried with queryFeature().

vd_setTextureCreationFlag(flag, state)
Enables or disables a texture creation flag.
This flag defines how textures should be created.
By changing this value, you can influence the speed of rendering a lot.
But please note that the video drivers take this value only as recommendation

vd_setTransform(state, matrix)

vd_setViewPort(x1, y1, x2, y2)

Sets a new viewport. Every rendering operation is done into this new area

Camera functions

cam_mapKey(i, key)
sets new keys for the fps camera, default are cursor keys
argument0: 0/1/2/3 -> forward/backward/left/right
argument1: new key as key constant

aspectratio= **cam_getAspectRatio**(cam)

farvalue=**cam_getFarValue**(cam)

fov=**cam_getFOV**(cam)

nearvalue=**cam_getNearValue**(cam)

cam_getProjectionMatrix(cam)

x=**cam_getTargetX**(cam)

y=**cam_getTargetY**(cam)

z=**cam_getTargetZ**(cam)

x=**cam_getUpVectorX**(cam)

y=**cam_getUpVectorY**(cam)

z=**cam_getUpVectorZ**(cam)

result=**cam_isDebugDataVisible**(cam)

result=**cam_isInputReceiverEnabled**(cam)

cam_setAspectRatio(cam, asp)

cam_setFarValue(cam, f)

cam_setFOV(cam, fov)

cam_setInputReceiverEnabled(cam, mode)

cam_setNearValue(cam, n)

cam_setProjectionMatrix(cam, n)

cam_setTarget(cam, x, y, z)

cam_setUpVector(cam, x, y, z)

Animated Mesh Functions

mesh=**am_getMesh**(m, frame, level, startf, endf)

returns mesh

type=**am_getMeshType**(m)

Animated Mesh SceneNode functions

node=**amsn_addShadowVolume**(n, id, zf, i)
Creates shadow volume scene node as child of this node and returns a pointer to it.
The shadow can be rendered using the ZPass or the zfail method.
ZPass is a little bit faster because the shadow volume creation is easier,
but with this method there occur ugly looking artifacs when the camera is inside
the shadow volume. These error do not occur with the ZFail method.

frame=**amsn_getFrameNr**(obj)
Returns the current displayed frame number.

joint=**amsn_getMS3DJointNode**(obj, string name)

amsn_setAnimationSpeed(obj, speed)
Sets the speed with witch the animation is played.

amsn_setCurrentFrame(obj, frame)
Sets the current frame number. From now on the animation is played from this frame

amsn_setFrameLoop(obj, start, end)
Sets the frame numbers between the animation is looped. The default is 0 - MaximalFrameCount
of the mesh.

amsn_setLoopMode(obj, mode)

MD2

result=**amsn_setMD2Animation**(obj, string ani)
starts md2 animation, argument=name of animation

count=**md2_getAnimationCount**(obj)

name=**md2_getAnimationName**(obj, nr)

MS3D

count=**ms3d_getJointCount**(obj)

name=**ms3d_getJointName**(obj, nr)

number=**ms3d_getJointNumber**(obj, string name)

matrix=**ms3d_getMatrixOfJoint**(obj, nr, frame)

Microsoft X File Format

count=**x_getAnimationCount**(obj)

name=**x_getAnimationName**(obj, idx)

count=**x_getJointCount**(obj)

name=**x_getJointName**(obj, idx)

number=**x_getJointNumber**(obj, string name)

matrix=**x_getMatrixOfJoint**(obj, nr, frame)

x_setCurrentAnimation(obj, string name)

x_setCurrentAnimationNr(obj, name)

Billboard Functions

width=**bb_getWidth**(n)

height=**bb_getHeight**(n)

bb_setSize(n, w, h)

Terrain SceneNode Functions

SceneManager

node=**sm_addAnimatedMesh**(m, parent)
Adds a scene node for rendering an animated mesh model.

node=**sm_addBillboard**(parent, w, h)
Adds a billboard scene node to the scene graph.
A billboard is like a 3d sprite:
A 2d element, which always looks to the camera.
It is usually used for things like explosions, fire, lensflares and things like that.

node=**sm_addCamera**(parent)
Adds a camera scene node to the scene graph and sets it as active camera.

node=**sm_addCameraFPS**(parent, rotatespeed, movespeed)
Adds a camera scene node which is able to be controled with the mouse and keys like in most first person shooters (FPS):

node=**sm_addCameraMaya**(parent, rot, zoom, trans)
Adds a camera scene node which is able to be controlled with the mouse similar like in the 3D Software Maya

node=**sm_addDummyTransformationSceneNode**(p)

node=**sm_addEmptyNode**(parent)
Adds an empty scene node.

node=**sm_addLight**(parent)

Adds a dynamic light scene node to the scene graph.

node=**sm_addMesh**(m, parent)
Adds a scene node for rendering a static mesh.

node=**sm_addOctTree**(m, parent, polys)
Adds a scene node for rendering using a octtree to the scene graph.
This a good method for rendering scenes with lots of geometry.
The Octree is built on the fly from the mesh, much faster then a bsp tree.

node=**sm_addParticleSystem**(defemit, parent)
Adds a particle system scene node to the scene graph.

node=**sm_addSkyBox**(top, bottom, left, right, front, back)
Adds a skybox scene node to the scene graph.
A skybox is a big cube with 6 textures on it and is drawed around the camera position.

node=**sm_addTerrain**(bmp filename, parent, maxlod, patchsize)

node=**sm_addTestSceneNode**(size, parent)

node=**sm_addText**(f, string text, parent, id)

node=**sm_addWaterSurface**(m, h, sp, l, parent)
Adds a scene node for rendering a animated water surface mesh.
Looks ly good when the Material type EMT_TRANSPARENT_REFLECTION is used.

sm_clear()
clears the whole scene, all scene nodes are removed

animator=**sm_createCollisionResponseAnimator**(world, n, erx, ery, erz, gx, gy, gz, sv)
Creates a special scene node animator for doing automatic collision detection and response.

animator=**sm_createDeleteAnimator**(zeit)
Creates a scene node animator, which deletes the scene node after some time automaticly.

animator=**sm_createFlyCircleAnimator**(cx, cy, cz, radius, sp)
Creates a fly circle animator, which lets the attached scene node fly around a center.

animator=**sm_createFlyStraightAnimator**(spx, spy, spz, epx, epy, epz, t, loop)
Creates a fly straight animator, which lets the attached scene node fly or move along a line between two points.

selector=**sm_createMetaTriangleSelector**()
Creates a meta triangle selector which is nothing more than a collection of one or more triangle selectors providing together the interface of one triangle selector.
In this way, collision tests can be done with different triangle soups in one pass.

selector=**sm_createOctTreeTriangleSelector**(mesh, node, minimal)
Creates a simple ITriangleSelector, based on a mesh.
Triangle selectors can be used for doing collision detection.
This triangle selector is optimized for huge amounts of triangle, it organizes them in an octtree.
Please note that the created triangle selector is not automaticly attached to the scene node.

Parameters:
mesh: Mesh of which the triangles are taken.

node: Scene node of which visibility and transformation is used.
minimalPolysPerNode: Specifies the minimal polygons contained a octree node. If a node gets less polys the this value, it will not be splitted into smaller nodes.

animator=**sm_createRotationAnimator**(rx, ry, rz)
Creates a rotation animator, which rotates the attached scene node around itself.

selector=**sm_createTerrainTriangleSelector**(node, lod)

selector=**sm_createTriangleSelector**(m, node)
Creates a simple ITriangleSelector, based on a mesh.
Triangle selectors can be used for doing collision detection.
Don't use this selector for a huge amount of triangles like in Quake3 maps.
Instead, use for example createOctTreeTriangleSelector().

selector=**sm_createTriangleSelectorFromBB**(node)
Creates a simple dynamic ITriangleSelector, based on a axis aligned bounding box.
Triangle selectors can be used for doing collision detection.
Every time when triangles are queried, the triangle selector gets the bounding box of the scene node, an creates new triangles.
In this way, it works good with animated scene nodes.

sm_drawAll()
Draws all the scene nodes.
This can only be invoked between beginScene() and endScene().

camera=**sm_getActiveCamera**()
The active camera is returned.
Note that this can be NULL, if there was no camera created yet.

mesh=**sm_getMesh**(string fn)
Gets pointer to an animateable mesh.
Loads it if needed.

node=**sm_getRootSceneNode**()
Returns the root scene node.
This is the scene node wich is parent of all scene nodes.
The root scene node is a special scene node which only exists to manage all scene nodes.
It is not rendered and cannot be removed from the scene.

node=**sm_getSceneNodeFromID**(id, start)
Returns the first scene node with the specified id.

sm_setActiveCamera(cam)
Sets the active camera. The previous active camera will be deactivated

sm_setShadowColor(a, r, g, b)

Scenenodes

node_addAnimator(n, animator)
Adds an animator which should animate this node

node_addChild(n, child)

Adds a child to this scene node.
If the scene node already has got a parent, it is removed from there as child.

x=**node_getAbsolutePositionX**(n)
Returns the current absolute position of the node

y=**node_getAbsolutePositionY**(n)

z=**node_getAbsolutePositionZ**(n)

result=**node_getAutomaticCulling**(n)
Gets the automatic culling state.

id=**node_getID**(n)
Returns the id of the scene node. This id can be used to identify the node

count=**node_getMaterialCount**(n)
Returns amount of materials used by this scene node.

name=**node_getName**(n)

parent=**node_getParent**(n)

x=**node_getPositionX**(n)
Returns the current position of the node relative to the parent

y=**node_getPositionY**(n)

z=**node_getPositionZ**(n)

x=**node_getRotationX**(n)
Returns the current rotation of the node relative to the parent

y=**node_getRotationY**(n)

z=**node_getRotationZ**(n)

x=**node_getScaleX**(n)

y=**node_getScaleY**(n)

z=**node_getScaleZ**(n)

selector=**node_getTriangleSelector**(n)
Returns a pointer to the TriangleSelector or NULL, if there is none.

result=**node_isCulled**(node)
returns true if a scenenode is currently culled, outside of the view frustrum

visible=**node_isDebugDataVisible**(n)

visible=**node_isVisible**(n)
Returns true if the node is visible.
This is only an option, set by the user and has nothing to do with geometry culling

node_remove(n)

Removes this scene node from the scene, deleting it.

node_removeAll(n)
removes all children of scene node

node_removeAnimator(n, animator)
Removes an animator from this scene node.

node_removeAnimators(n)
Removes all animators from this scene node.

result=**node_removeChild**(n, child)
Removes a child from this scene node.
Returns true if it was successful.

node_setAutomaticCulling(n, mode)
Enables or disables automatic culling based on the bounding box.
Automatic culling is enabled by default.
Note that not all SceneNodes support culling (the billboard scene node for example)
and that some nodes always cull their geometry because it is their only reason for existance,
for example the OctreeSceneNode.

node_setDebugDataVisible(n, mode)
Sets if debug data like bounding boxes should be drawed.
Please note that not all scene nodes support this feature.

node_setID(n, id)
sets the id of the scene node. This id can be used to identify the node.

node_setMaterialFlag(n, fl, mode)
Sets all material flags at once to a new value.

node_setMaterialTexture(n, tex, layer)
Sets the texture of the specified layer in all materials of this scene node to the new texture.

node_setMaterialType(n, mt)
Sets the material type of all materials of this scene node to a new material type.

node_setName(n, string name)
Sets the name of the node.

node_setParent(n, p)
Changes the parent of the scene node.

node_setPosition(obj, x, y, z)
Sets the position of the node. Note that the position is relative to the parent.

node_setRelativePosition(obj, r, x, y, z)

node_lift(n, speed)

node_move(n, speed)

node_strafe(n, speed)

node_setRotation(obj, x, y, z)

Sets the rotation of the node in degrees. This only modifies the relative rotation of the node.

node_rotate(obj, x, y, z)

node_setScale(obj, x, y, z)

node_setTriangleSelector(node, triselector)
Sets the triangle selector of the scene node.
The Selector can be used by the engine for doing collision detection.
You can create a TriangleSelector with createTriangleSelector() or
createOctTreeTriangleSelector().
Some nodes may create their own selector by default,
so it would be good to check if there is already a selector in this node by calling
getTriangleSelector().

node_setVisible(node, visible)
Sets if the node should be visible or not. All children of this node won't be visible too.

node_updateAbsolutePosition(n)

Cursor functions

x=**cursor_getPositionX**()

y=**cursor_getPositionY**()

x=**cursor_getRelativePositionX**()

y=**cursor_getRelativePositionY**()

result=**cursor_isVisible**()
Returns true if the cursor is visible, false if not.

cursor_setPosition(x, y)
New position of the cursor. The coordinates are pixel units
cursor_setVisible(mode)
sets cursor visibility: true/false

Filesystem

stringparameters_setParameter(string par, string value)

result=**filesystem_addZipArchive**(string name)

result=**filesystem_changeWorkingDirectoryTo**(string name)

xmlreader=**filesystem_createXMLReader**(string name)

xmlwriter=**filesystem_createXMLWriter**(string name)

result=**filesystem_existFile**(string name)

directory=**filesystem_getWorkingDirectory**()

XML Reader

count=**xmlreader_getAttributeCount**(xml)

name=**xmlreader_getAttributeName**(xml, idx)

text=**xmlreader_getAttributeValue**(xml, string name)

text=**xmlreader_getAttributeValueNr**(xml, idx)

value=**xmlreader_getAttributeValueAsFloat**(xml, string name)

value=**xmlreader_getAttributeValueAsInt**(xml, string name)

text=**xmlreader_getAttributeValueSafe**(xml, string name)

text=**xmlreader_getNodeData**(xml)

name=**xmlreader_getNodeName**(xml)

type=**xmlreader_getNodeType**(xml)

result=**xmlreader_isEmptyElement**(xml)

result=**xmlreader_read**(xml)

XML Writer

xmlwriter_writeClosingTag(xml, string text)

xmlwriter_writeComment(xml, string text)

xmlwriter_writeElement(xml, string wn, string wan, string wav)

xmlwriter_writeLineBreak(xml)

xmlwriter_writeText(xml, string text)

xmlwriter_writeXMLHeader(xml)

SceneCollisionManager

result=**scm_getCollisionPoint**(x1, y1, z1, x2, y2, z2, ts)
Finds the collision point of a line and lots of triangles, if there is one.

x=**scm_getCollisionPointX**()

y=**scm_getCollisionPointY**()

z=**scm_getCollisionPointZ**()

node=**scm_getSNFromCameraBB**(cam, mask)
Returns the scene node, at which the overgiven camera is looking at

and which id matches the bitmask.
A ray is simply casted from the position of the camera to the view target position,
and all scene nodes are tested against this ray.

node=**scm_getSNFromRayBB**(x1, y1, z1, x2, y2, z2, mask)
Returns the nearest scene node which collides with a 3d ray and which id matches a bitmask.

node=**scm_getSNFromScreenCoordBB**(x, y, mask)
Returns the scene node, which is currently visible under the overgiven screencoordinates, viewed from the currently active camera

x=**scm_getScreenCoordinatesFrom3DPositionX**(x, y, z, cam)

y=**scm_getScreenCoordinatesFrom3DPositionY**(x, y, z, cam)

falling=**scm_setCollisionResultPosition**(world, n, dx, dy, dz, slide, gx, gy, gz)

Axis Aligned Bounding Box functions

result=**aabb_intersectsWithBox**(n1, n2)

result=**aabb_intersectsWithLine**(n1, x1, y1, z1, x2, y2, z2)

result=**aabb_isPointInside**(n1, x, y, z)

Meta Triangle Selector functions

mts_addTriangleSelector(mts, ts)

mts_removeAllTriangleSelectors(mts)

mts_removeTriangleSelector(mts, ts)

Material functions

sets material for each layer, use scene node functions to change all layers/materials
mat_setAmbientColor(n, layer, a, r, g, b)

mat_setBackFaceCulling(n, layer, bfc)

mat_setBilinearFilter(n, layer, bf)

mat_setDiffuseColor(n, layer, a, r, g, b)

mat_setEmissiveColor(n, layer, a, r, g, b)

mat_setFog(n, layer, fog)

mat_setGouraudShading(n, layer, gs)

mat_setLighting(n, layer, l)

mat_setMaterialType(n, layer, mt)

mat_setShininess(n, layer, s)

mat_setTrilinearFilter(n, layer, tf)

mat_setWireframe(n, layer, wf)

mat_setZBuffer(n, layer, zb)

mat_setZWriteEnable(n, layer, zw)

Light Functions

light_setAmbientColor(lsn, r, g, b)

light_setCastShadows(lsn, cs)

light_setDiffuseColor(lsn, r, g, b)

light_setRadius(lsn, r)

light_setSpecularColor(lsn, r, g, b)

Particlesystem SceneNode functions

ps_addAffector(ps, pa)

setBox(x1, y1, z1, x2, y2, z2)
sets bounding box for ps_createBoxEmitter
args: x1,y1,z1,x2,y2,z2

emitter=**ps_createBoxEmitter**(ps, dirx, diry, dirz, minP, maxP, minLife, maxLife, maxAngle)

affector=**ps_createFadeOutParticleAffector**(ps, a, r, g, b, t)

affector=**ps_createGravityAffector**(ps, x, y, z, t)

ps_setColor(a1, r1, g1, b1, a2, r2, g2, b2)

emitter=**ps_createPointEmitter**(ps, dirx, diry, dirz, minP, maxP, minLife, maxLife, maxAngle)

ps_removeAllAffectors(ps)

ps_setEmitter(ps, e)

ps_setParticleSize(ps, w, h)

SceneNodeAnimatorCollisionResponse functions

x=**cr_getEllipsoidRadiusX**(crn)

y=**cr_getEllipsoidRadiusY**(crn)

z=**cr_getEllipsoidRadiusZ**(crn)

x=**cr_getEllipsoidTranslationX**(crn)

y=**cr_getEllipsoidTranslationY**(crn)

z=**cr_getEllipsoidTranslationZ**(crn)

selector=**cr_getWorld**(crn)

result=**cr_isFalling**(crn)

cr_setEllipsoidRadius(crn, x, y, z)

cr_setEllipsoidTranslation(crn, x, y, z)

cr_setGravity(crn, x, y, z)

cr_setWorld(crn, world)

Gui Environment

button=**gui_addButton**(x1, y1, x2, y2, parent, id)

checkbox=**gui_addCheckbox**(checked, x1, y1, x2, y2, parent, id)

combobox=**gui_addCombobox**(x1, y1, x2, y2, parent, id)

menu=**gui_addContextMenu**(x1, y1, x2, y2, parent, id)

editbox=**gui_addEditBox**(x1, y1, x2, y2, border, parent, id)

openfiledialog=**gui_addFileOpenDialog**(string title, modal, parent, id)

gui_image=**gui_addImage**(x1, y1, x2, y2, parent, id)

gui_image=**gui_addImage2**(tex, x1, y1, alpha, parent, id)

fader=**gui_addInOutFader**(x1, y1, x2, y2, parent, id)

listbox=**gui_addListBox**(x1, y1, x2, y2, parent, id)

menu=**gui_addMenu**(parent, id)

meshviewer=**gui_addMeshViewer**(x1, y1, x2, y2, parent, id)

mbox=**gui_AddMessageBox**(string title, string body, modal, flags)

scrollbar=**gui_addScrollBar**(hor, x1, y1, x2, y2, parent, id)

statictext=**gui_addStaticText**(x1, y1, x2, y2, border, wrap, parent, id)

tab=**gui_addTab**(x1, y1, x2, y2, parent, id)

tabcontrol=**gui_addTabControl**(x1, y1, x2, y2, fill, border, parent, id)

toolbar=**gui_addToolBar**(parent, id)

window=**gui_addWindow**(x1, y1, x2, y2, modal, parent, id)

font=**gui_getBuiltInFont**()

font=**gui_getFont**(string fname)

gui_drawAll()

result=**gui_hasFocus**(element)

gui_removeFocus(element)

gui_setFocus(element)

Gui Element

gui_addChild(element, child)

result=**gui_bringToFront**(element, child)

x=**gui_getAbsolutePositionX**(element)

y=**gui_getAbsolutePositionY**(element)

x=**gui_getRelativePositionX**(element)

y=**gui_getRelativePositionY**(element)

element=**gui_getElementFromId**(element, id, search)

element=**gui_getElementFromPoint**(element, x, y)

id=**gui_getID**(element)

element=**gui_getParent**(element)

text=**gui_getText**(element)

result=**gui_isEnabled**(element)

result=**gui_isVisible**(element)

gui_moveElement(element, x, y)

gui_removeElement(element)

gui_removeChildElement(element, child)

gui_setEnabled(element, mode)

gui_setID(element, id)

gui_setRelativePosition(element, x1, y1, x2, y2)

gui_setText(element,string text)

gui_setVisible(element, mode)

gui_updateAbsolutePosition(element)

Gui Button

result=**gui_isButtonPressed**(button)
returns true or false

gui_setButtonImage(button, image)

gui_setIsPushButton(button, mode)

gui_setOverrideFont(button, font)

gui_setButtonPressed(button, mode)

gui_setPressedButtonImage(button, image)

Gui Checkbox

gui_cbIsChecked(button)
returns true or false

gui_cbSetChecked(button, mode)
set to true or false

Gui Combobox

index=**combobox_addItem**(combo,string text)
adds a text item to a combobox
returns index of item

combobox_clear(combo)

text=**combobox_getItem**(combo, idx)
returns text of item idx

count=**combobox_getItemCount**(combo, idx)

index=**combobox_getSelected**(combo)

combobox_setSelected(combo, id)
-1 for no item

Gui ContextMenu

index=**contextmenu_addItem**(cm, string text, enabled, submenu)

contextmenu_addSeperator(cm)

id=**contextmenu_getItemCommandID**(cm, idx)

count=**contextmenu_getItemCount**(cm)

text=**contextmenu_getItemText**(cm, idx)

index=**contextmenu_getSelectedItem**(cm)

contextmenu_getSubMenu(cm, idx)
returns a pointer to the submenu or 0 if there is no submenu at index idx

enabled=**contextmenu_isItemEnabled**(cm, idx)
returns true if item idx is enabled

contextmenu_removeAllItems(cm)

contextmenu_removeItem(cm, idx)

contextmenu_setItemCommandID(cm, idx, id)

contextmenu_setItemEnabled(cm, idx, mode)

contextmenu_setItemText(cm, idx, string text)

Gui EditBox

editbox_enableOverrideColor(edit, mode)

max=**editbox_getMax**(edit)
returns maximum amount of characters set by editbox_setMax

editbox_setMax(edit, max)

editbox_setOverrideColor(edit, r, g, b)

editbox_setOverrideFont(edit, font)

Gui OpenFileDialog

filename=**opendialog_getFilename**(open)
argument: openfiledialog
returns chosen filename

Gui Image

guiImage_setImage(img, image)

guiImage_setUseAlphaChannel(img, use)

Gui InOutFader

InOutFader_fadeIn(fader, time)

InOutFader_fadeOut(fader, time)

result=**InOutFader_isReady**(fader)
returns true or false

InOutFader_setColor(fader, r, g, b)

Gui Listbox

index=**listbox_addItem**(lb, string text, string icon)
adds an item to a listbox
args: listbox, text, icon name
returns index of item

listbox_clear(lb)
clears all item entries

count=**listbox_getItemCount**(lb)
returns number of items

text=**listbox_getListItem**(lb, id)
returns text of listbox item id

index=**listbox_getSelected**(lb)
returns index of selected item

listbox_setIconFont(lb, font)

listbox_setSelected(lb, idx)

Gui Meshviewer

meshviewer_setMaterial(mv, mat)

meshviewer_setMesh(mv, mesh)

Gui Scrollbar

pos=**scrollbar_getPos**(sb)

scrollbar_setMax(sb, max)

scrollbar_setPos(sb, pos)

Gui StaticText

statictext_setPos(st, mode)

height=**statictext_getTextHeight**(st, mode)

statictext_setOverrideColor(st, r, g, b)

statictext_setOverrideFont(st, font)

statictext_setWordWrap(st, mode)

Gui Tab

number=**tab_getNumber**(tab)

tab_setBackgroundColor(tab, r, g, b)

tab_setDrawBackground(tab, mode)

Gui TabControl

tab=**tabcontrol_addTab**(tab, string text, id)

index=**tabcontrol_getActiveTab**(tab)

tab=**tabcontrol_getTab**(tab, idx)

count=**tabcontrol_getTabCount**(tab)

result=**tabcontrol_setActiveTab**(tab, idx)

Gui Toolbar

button=**toolbar_addButton**(tb, id, string text)

Gui Window

button=**window_getCloseButton**(window)

button= **window_getMaximizeButton**(window)

button=**window_getMinimizeButton**(window)

Gui Font

font_setColor(color)

font_drawText(font, string text, x, y)

index=**font_getCharacterFromPos**(font, string text, x)

width=**font_getDimensionWidth**(font, string text)

height=**font_getDimensionHeight**(font, string text)

Gui Skin

color=**skin_getColor**(defcolor)

text=**skin_getDefaultText**(deftext)

font=**skin_getFont**()

size=**skin_getSize**(def)

skin_setColor(def, color)

skin_setDefaultText(def, string text)

skin_setFont(font)

skin_setSize(def, size)

Image

bpp=**image_getBitsPerPixel**(img)

bytesperpixel=**image_getBytesPerPixel**(img)

format=**image_getColorFormat**(img)

width=**image_getDimensionWidth**(img)

height=**image_getDimensionHeight**(img)

size=**image_getImageDataSizeInBytes**(img)

size=**image_getImageDataSizeInPixels**(img)

color=**image_getPixel**(img, x, y)
returns 32bit color value

pointer=**image_lock**(img)
returns pointer to the locked image

image_unlock(img)

Texture

colorformat=**texture_getColorFormat**(tex)

width=**texture_getOriginalSizeWidth**(tex)

height=**texture_getOriginalSizeHeight**(tex)

pitch=**texture_getPitch**(tex)
returns pitch of texture

width= **texture_getSizeWidth**(tex)
returns width of texture

height=**texture_getSizeHeight**(tex)
returns height of texture

texture_lock(tex)
returns a pointer to the pixel data

texture_regenerateMipMapLevels(tex)

texture_unlock(tex)
must be called after a texture_lock call

MeshManipulator

mesh=**meshmanipulator_createMeshCopy**(mesh)
returns a copy of the mesh

mesh=**meshmanipulator_createMeshWithTangents**(mesh)
it calculates the tangent and binormal data which is needed there.
returns mesh consisting only of S3DVertexTangents vertices. If you no longer need the cloned mesh, you should call device_drop

meshmanipulator_flipSurfaces(mesh)

polys=**meshmanipulator_getPolyCount**(mesh)
returns number of polygons of a mesh

meshmanipulator_makePlanarTextureMapping(mesh, res)
applies planar texture mapping to mesh
res=resolution of the planar mapping. This is the value specifying which is the relation between world space and texture coordinate space. default=0.001

meshmanipulator_recalculateBoundingBox(mesh)

meshmanipulator_recalculateNormals(mesh)

meshmanipulator_scaleMesh(mesh, x, y, z)

meshmanipulator_setVertexColorAlpha(mesh, alpha)

Shaders

gpu_addShaderMaterial(string v, string p, call, mtype)
arguments: vertexshader program, pixelshader program, callback function, material type

gpu_addShaderMaterialFromFiles(string v, string p, call, mtype)
arguments: vertexshader program, pixelshader program, callback function, material type

Datatype Functions

datatype_delete(datatype)
deletes a created datatype
argument: the datatype created with a datatype_create function

line2d

line2D=**datatype_createLine2D**(xa, ya, xb, yb)
creates a line2D data type
returns pointer to line2D type

length=**line2d_getLength**(line2D)
returns length of a line2D datatype created with datatype_createLine2D

line2d_setLine(line2D, xa, ya, xb, yb)
sets new line2D properties

color=**makeColorARGB**(a, r, g, b)
returns 32bit color value
arguments: alpha, red, green, blue : 0-255