



## Lua command reference document



**Version information:** This document is based on the commands available in IRB  
SVN revision 578

## Content

Global LUA pre-defined functions (basicFunctions.lua).....	4
---- internals .....	4
---- Global functions .....	4
Global Functions.....	6
Global variables creation .....	6
Lighting / ambiance commands.....	6
Background.....	7
Special FX post-process commands .....	7
Cameras commands .....	8
Cameras commands (continued) .....	9
Cameras commands (continued) .....	10
GUI commands .....	10
Object search commands .....	10
Sound commands .....	11
Player affecting commands .....	11
Game Commands .....	12
Callbacks for the global function: .....	12
Misc Commands: .....	13
Dynamic objects functions: .....	14
Label management.....	14
Object type management.....	14
Position/rotation management .....	14
Position/rotation management (continued) .....	15
Getting information .....	15
Animation/AI specific functions.....	16
Default animation names .....	16
Dynamic objects: (properties and combat) .....	17
Current properties and combat commands: .....	18
Misc commands.....	19



Special Variables.....	19
Callbacks:.....	19



## Global LUA pre-defined functions (basicFunctions.lua)

These functions can be called from anywhere in IRB. But these have been written in LUA, so they could be modified by the user.

### ---- internals

**IRBStoreGlobalParams()**  
**IRBRestoreGlobalParams()**

### ---- Global functions

**setObjectName(string 'name')**  
set the global name of the current object

**chaseObject(string 'name', float speed, float near, float far)**  
chase the specified object  
speed = move speed  
near = stop when reaching this distance (also IDLE state is triggered)  
far = start walking from this position (also WALK state is triggered)

**walkRandomly(float radius, float speed)**  
Walk randomly in a specified radius  
*radius* = random point to get there specified from a radius around the object  
*speed* = move speed

**walkToObject(otherObj)**  
Walk to an object, current command doesn't account the animations

**CustomDynamicObjectUpdate()**  
update the dynamic objects, work with the 'walkToObject' command  
Should be placed in the 'step' callback

**programAction(time, action, parameter...)**  
Call a specified custom function at the specified time interval.  
*time* = time interval in second  
*action* = function name to call  
*parameters* = parameter (up to 10 parameters)  
Ex:     programAction(10, print, 'hello world')  
          Will call: print "hello world" after 10 seconds.

**CustomDynamicObjectUpdateProgrammedAction()**  
Update programmed action if it exist



*action* = **hasActionProgrammed()**

Will return the name of the action that is programmed. if any.

**enableObject()**

will enable the current object

**disableObject()**

Will disable the current object

**decreasePlayerLife(points)**

Remove 'point' from the current life of the player

**increasePlayerLife(points)**

Increase the current player life with 'points'

**decreasePlayerMoney(points)**

Remove 'points' from the current player money

**increasePlayerMoney(points)**

Increase the current money of the player with 'point'

**playSound(soundName, looped)**

Play a 2D sound

**emitSound(soundName, looped)**

Play a sound from the current position of the object (3D Sound)

**sleep(time)**

Will 'sleep' until the time is reached. 'time' is in seconds.



## Global Functions

These functions can affect about anything in IRB: Objects, lighting, setting of the game, camera... They are not restricted to the object from which they have been invoked.

### Global variables creation

**setGlobal**(string *'name'*, *value*)

Create a global unique name

*value* = **getGlobal**(string *'name'*)

Retrieve the value of the global

**deleteGlobal**(string *'name'*)

Delete the specified global

### Lighting / ambiance commands

**setTimeOfDay**(int *time*)

set the time of day

**setAmbientLight**(int *r*, int *g*, int *b*)

Set the ambient light to the specified RGB colors

*r,g,b* = **getAmbientColor**()

Give back the ambient color

**setFogColor**(int *r*, int *g*, int *b*)

Set the color of the fog to the specified RGB color

*r,g,b* = **getFogColor**()

Retrieve the current fog color in RGB color

**setFogRange** (float *near*, float *far*)

Define the range of the fog

*near, far* = **getFogRange**()

Retrieve the range of fog

*String* = **getLanguage**()

Return the current language for the game, the currently supported languages strings are:

"en-us" -> English, US

"pt-br" -> Portuguese, Brazil

"fr-ca" -> French, Canada

"de-ge" -> Deutsch, Germany



## Background

**setSkydomeTexture**(string *'texturefilename'*)

Change the texture of the skydome

**setSkydomeVisible**(bool *visible*)

Set the visibility of the skydome

**setBackgroundColor**(int *r*, int *g*, int *b*)

Set the background rendering color

## Special FX post-process commands

**setPostFX**(string *'name'*)

Create a postprocess FX, These can be useful in a lot of circumstances. Applied FX will change all of the display. The shader modify the rendered output.

name can be:

adaptive\_bloom  
blur  
color  
depth\_of\_field  
desaturate  
displacement  
dream\_vision  
embossed  
gray\_scale  
invert  
motion\_blur  
none  
night\_vision  
pencil  
posterize  
pulsing  
radial\_blur  
scratched  
sepia  
shake  
sharpen  
simple\_bloom  
tiling  
vignette  
water

Theses FX are based on shaders and there might be modified in the future.



## Cameras commands

**setCameraTarget(float x, float y, float z)**

Set a new position for the camera, use the specified coordinates

**setCameraTarget(string 'objectname')**

Set a new position for the camera, use the object name as reference

**x,y,z = getCameraTarget()**

Get the current camera target

**x,y,z = getCameraPosition(string 'objectname')**

Retrieve the position of the current camera

**setCameraPosition(float x, float y, float z)**

Set a new position for the camera, use the specified coordinates

**setCameraPosition(string 'objectname')**

Set a new position for the camera, use the object name as reference

**setCameraRange(float start, float end)**

Set the range of the ingame camera zoom (mouse wheel)

Default values are min=72, max=2000

**setCameraZoom(float zoom)**

Set distance for the camera zoom

Default value is 600

**Start, end = getCameraRange()**

Retrieve the current range limits of the camera zoom

**setCameraAngleLimit(float low, float high)**

Define the angle limit (up/down) limits of the game camera

The default values are -25, 89

**low, high = getCameraAngleLimit()**

Retrieve the current rotation information of the RTS camera

**setCameraRTSRotation(float X, float Y)**

Redefine the angle of the rotation of the RTS camera

The default values are 135.0, 45.0

**Start, end = getCameraRTSRotation()**

Retrieve the current rotation information of the RTS camera





## Cameras commands (continued)

### **cutsceneMode()**

switch the current camera to the cutscene camera

### **gameMode()**

switch the current camera to the ingame camera

### **setRTSView()**

switch the ingame camera view to a RTS game type view

### **setRTSFixedView()**

switch the ingame camera view to a RTS game type view, rotation is locked

### **setRPGView()**

switch the ingame camera view to third person view type view

### **setFPSView()**

switch the ingame camera view to first person view type view

### **defineKeys(string key, string command)**

Define key ingame keyboard keys

key is a string to the keyboard key desired: ex: "A"

command is the assigned command for the key:

"FORWARD" → Move forward

"BACK" → Move backward

"LEFT" → Move on the left

"RIGHT" → Move on the right

"INVENTORY" → Open the inventory panel

"INTERACTION" → Activate the interaction (like clicking on the object)

#### *New shortcuts:*

key="ARROWS" → Will automatically set the movement for the player to the arrow keys.

Key="WASD" or "QWERTY" -> use WASD for movement

key="ZQSD" or "AZERTY" -> use ZQSD for movement

key="QZERTY" -> use ZASD for movement

When using shortcuts, put "" for the command ex: **defineKeys("QWERTY","")**



## Cameras commands (continued)

**setCameraAttachment**(string bone)

Will define a bone in the player character where to attach a camera. This is only used in the FPS Camera mecanic. If you don't want to use a bone as attachment, just enter this: `setCameraAttachment("")`. Then the camera system will place the camera at 0,0,0 locally in the player character.

**setCameraOffset**(float x,y,z)

Will set the position for the offset of the camera in FPS camera mecanic.

## GUI commands

**printToConsole**(string text)

Will print the specified text to the IRB main console

## Object search commands

**findInSphere**(float radius, x,y,z, string object type)

Will return a table with the names of the object from the specified position in a specified radius of the specified types.

radius = contain the size of the sphere

x,y,z = position of the center of the sphere

object type = type of objects to selects

Current object types are the following:

“player” -> the player

“npc” -> a NPC

“interactive” -> an interactive object

“non-interactive” -> a non-interactive object

“all” -> all the object in the radius



## Sound commands

**playSound2D**(string 'filename', boolean looped)

Play the specified 2D sound, 'looped' is optional.

**playSound3D**(string 'filename', bool looped, float x, float y, float z)

Play the specified sound, at the specified coordinates. Looped is for a looping sound

**setSoundListenerPosition**(float x, int y, int z)

Define the position of the listener for a 3D sound

*Note: is now updated directly by the engine to the player position. Will need to be overridden to change the position.*

**setSoundVolume** (float volume)

Set the volume to a specified level

**stopSounds()**

Stop all sounds

## Player affecting commands

**setPlayerLife**(int life)

Set the player life to the specified value.

A change in the value will rise a "injured state" on the player

A value of 0 will set the "die" state on the player

*life* = **getPlayerLife()**

Get the current player life

**setPlayerMoney**(int money)

Set the amount of money the player will have

*money* = **getPlayerMoney()**

Retrieve the current amount of money the player have

**moveObjectLoot()**

Add the current item to the player loot. Item must be a "loot type" object.

**destroyAfterUse**( boolean )

Will set if the loot object will be destroyed once used. Good for potions for example.



## Dialogs /GUI commands

**showBlackScreen**(string 'optional text')

Show a black screen with an optional text (mostly used when player is dead)

**hideBlackScreen**()

Remove the black screen

**showDialogMessage**(string 'message')

Show the specified message

*answer* = **showDialogQuestion**(string '*question*')  
*question*

Show the specified question. (note: this can only be called from within a dynamic object)

*Bool* = **getAnswer**()

Will return the answer of the dialog question. Used principally in the **OnAnswer()** callback to do an action based on the response of the Question Dialog.

## Game Commands

**saveGame**()

save the game

**loadGame**()

load back the game

### Callbacks for the global function:

These functions are for the internal use of the engine. They should not be tampered with unless you know what you are doing.

**IRBStoreGlobalParams**()

**IRBRestoreGlobalParams**()



## Misc Commands:

*x,y,z* = **getObjectPosition**(string 'objectname')

Retrieve the position of the specified object

**addObjectLoot**(String objectname, loot template name)

Will create a loot item and place it in the desired object loot inventory

*int*=**getObjectItemCount**(String itemname)

give the number of duplicate items in the loot with this name

**useGlobalFunction**(String function name)—DISABLED

Will use a simple function defined in the LUA global script.

**getObjectProperty**(string object name, property name)

Retrieve the current value of the named object property.

Look on page 15 for the properties names.

**setObjectProperty**(string object name, property name, float value)

Set the named property of the specified object to a new value

Look on page 15 for the properties names.

*Boolean* = **checkObjectItem**(string object name, string item name)

Return true if the named item is found in the specified object inventory. False if not.

**destroyObjectItem**(string object name, string item name)

Will seek for the first occurrence of the item name in the specified object and remove it.

**setObjectVisible**(string object name, Boolean visible)

set the specified object visible or invisible

*Boolean* = **isObjectVisible**(string object name)

Return the visibility of the object (true/false)

**attachObject**(string object name, string attachment name)

Attach the current object, to another object attachment point.



## Dynamic objects functions:

These function can only be used inside the dynamic object. It will affect only the object it has been invoked with.

### Label management

#### **showObjectLabel()**

Display the object label

#### **hideObjectLabel()**

Hide the object label

#### **setObjectLabel(string 'newlabel')**

Define the new object label

Ex: *setObjectLabel("baddie")*

### Object type management

#### **setObjectType("type")**

Allow the developer to override the dynamic object type from LUA.

Type is a string and can contain theses values:

"interactive", "non-interactive"

### Position/rotation management

#### **setPosition(float x,float y,float z)**

Move the object to this specified position

#### **x,y,z = getPosition()**

Give back the position of the object (x,y,z)

#### **setRotation(float x,float y,float z)**

Rotate the object to the specified angle

#### **x,y,z = getRotation()**

Give back the current angle rotation of the object (x,y,z)

#### **x,y,z = getParentPosition()**

Give back the position of the object (x,y,z)

#### **x,y,z = getParentRotation()**

Give back the position of the object (x,y,z)



## Position/rotation management (continued)

**turn**(float *degrees*)

Turn relative of a specified degree angle

**move**(float *speed*)

Set the walk speed of the object

Also activate the "walk" animation state

**walkTo**(float x,float y,float z)

Set the destination of the object and the object will walk there with the current speed defined in the XML file

Ex: **walkTo(10,0,10)**

**true/false = hasReached()**

Will tell if a walking character has reached the destination point

**lookAt**(float x,float y,float z)

The object turn to face the specified coordinates

Ex: **lookAt(10,0,10)**

**lookToObject**(string '*objectname*')

The object turn to face the specified object name

## Getting information

string '*objectname*' = **getName()**

Return the defined name of the template used

**setName**(string name)

set the new name for this object. The object keep the internal name, but this set the alias

*distance* = **distanceFrom**(float x, float y, float z)

Give back the specified distance from the specified coordinates to the object.

*distance* = **distanceFrom**(string '*object*')

Give back the specified distance from the the object to the specified object.

**setEnabled**(boolean)

Set the state of the dynamic object (enabled/disabled)

Ex: **setEnabled(false)**

*Note: Disabling a NPC will remove it from the map (as if the character had died)*



## Animation/AI specific functions

**setFrameLoop**(int *start*, int *end*)

Set the animation frameloop for the object

**setAnimationSpeed**(float *speed*)

set the animation speed to the specified value. (frame rate)

**setAnimation**(string '*animation name*')  
Set the current animation of the object to the specified animation name

ex: *setAnimation("Cutscene1")*

### Default animation names

Here are the default animation names for the specific animation states. The animation states are associated with the AI in the engine.

Using these names will trigger the proper animation state for the AI. Currently there only a few defined name. Will need more.

**idle**, set the state to *OBJECT\_ANIMATION\_IDLE*

❖ Special, must be there (xml) as this is the strict minimum for a NPC

**walk**, set the state to *OBJECT\_ANIMATION\_WALK*

❖ Used by the animation system for walking

**run**, set the state to *OBJECT\_ANIMATION\_RUN*

**attack**, set the state to *OBJECT\_ANIMATION\_ATTACK*

❖ Special, will trigger the attack damage if an attack event has been defined in the XML template definition

**hurt**, set the state to *OBJECT\_ANIMATION\_INJURED*

❖ Used by the animation/combat system for the injured state in combat

**die**, set the state to *OBJECT\_ANIMATION\_DIE*

❖ Used by the animation system/combat to show the die state

**spawn**, set the state to *OBJECT\_ANIMATION\_SPAWN*

**despawn**, set the state to *OBJECT\_ANIMATION\_DESPAWN*





## Dynamic objects: (properties and combat)

Current properties names:

Property name		Description
life	-->	Current life points
mana	-->	Current mana points
maxlife	-->	Maximum life points
regenlife	-->	Points of life regeneration
regenmana	-->	Points of mana regeneration
money	-->	Current money
level	-->	Current level
experience	-->	Current experience (or given experience if NPC)
mindamage	-->	minimum damage per hit
maxdamage	-->	maximum damage per hit
armor	-->	armor point
magic_armor	-->	magic armor point
hurt_resist	-->	hurt resistance probability
dotduration	-->	duration for the dot (damage over time, poison)
hit_prob	-->	probability of hitting the target
dodge_prob	-->	probability of dodging the hit from the attacker
mindefense	-->	minimum defense points
maxdefense	-->	maximum defense points
weight	→	base weight for this object
maxweight	→	maximum weight this object can carry
currentweight	→	current weight the object carries

## Current properties and combat commands:

### **setEnemy()**

This object will become an enemy of the player

### **setObject()**

This object will become an object (non-aggressive) (default)

### **getEnemyCount()**

Will return the number of active enemies on the map

### **setProperty** (string 'property name', float value)

This will set the named property to the value entered

Ex: `setProperty("life",100)`

### float value **getProperty**(string 'property name')

Give back the entered value from the named property entered

Ex: `value = getProperty("life")`

### **addLoot**(string "Template name")

Add a template (should be a loot template to work), inside the current object backpack directly. It can be a NPC or the player.

### string name **Spawn**(string "template name", x, y, z, r)

create a new dynamic object on the map, x,y and z are for the position

and r is for the rotation of the element. Once created, the function return the name of the new dynamic object. The Y position will be recalculated so the spawned item will be placed on the ground.

### **attack**(string 'enemy name')

This will trigger the combat system to do a battle with the current object and the object entered. The combat system will check the properties of each opponent and do damages on the defender.

This will not check of the faction, nor the distance.

Ex: `attack("player")`



## Misc commands

boolean *value* **isKeyPressed**(string 'key name')

Return true if the key on the keyboard is currently pressed.

Ex: *value* = **isKeyPressed**("1")

## Special Variables

**ObjName** = NAME OF THE OBJECT

**objType** = *ENEMY* or *OBJECT*

## Callbacks:

**onLoad()**

Called when the object is loading in the game

**onUpdate()** or **step()** -- old name

Called at each interval (default interval is 1/4 sec)

**onClicked()**

Called when the object has been clicked on with the mouse

**onCollision()**

Called when a collision occurs with the object

**onAnswer()**

Called when the user responds to the Question Dialog. The answer returns only to the dialog caller. Ex: If the player is invoking the question dialog, only him will receive the answer.

**onUse()**

Called when the user selects an item from the backpack inventory and presses the USE button

**onKeyPressed()**

Called when the user presses a key on the keyboard

**CustomDynamicObjectUpdate()**

**CustomDynamicObjectProgrammedAction()**

