

RPG-X2 Lua Documentation

Ubergames Walter Julius 'GSIO01' Hennecke

October 20, 2010

Contents

1	Introduction	5
1.1	General Information	5
1.2	Prerequisites	5
2	Lua Hooks	6
2.1	What is a Lua Hook	6
2.2	Static Lua Hooks	6
2.2.1	InitGame	6
2.2.2	ShutdownGame	6
2.2.3	RunFrame	6
2.2.4	GClientPrint	6
2.2.5	GPrint	7
2.3	Dynamic Lua Hooks	8
2.3.1	luaThink	8
2.3.2	luaTouch	8
2.3.3	luaUse	8
2.3.4	luaHurt	8
2.3.5	luaDie	8
2.3.6	luaFree	9
2.3.7	luaReached	9
2.3.8	luaReachedAngular	9
2.3.9	luaTrigger	9
2.3.10	luaSpawn	9
3	RPG-X2 Map Scripting	10
3.1	Map scripts	10
3.2	Calling Functions	10
4	RPG-X2 Lua Libraries	11
4.1	game	11
4.1.1	game.Print	11
4.1.2	game.ClientPrint	11
4.1.3	game.CenterPrint	11
4.1.4	game.MessagePrint	11
4.1.5	game.LevelTime	11
4.1.6	game.SetGlobal	11
4.1.7	game.GetGlobal	12

4.2	qmath	13
4.2.1	qmath.abs	13
4.2.2	qmath.sin	13
4.2.3	qmath.cos	13
4.2.4	qmath.tan	13
4.2.5	qmath.asin	13
4.2.6	qmath.acos	13
4.2.7	qmath.atan	13
4.2.8	qmath.floor	13
4.2.9	qmath.ceil	14
4.2.10	qmath.fmod	14
4.2.11	qmath.modf	14
4.2.12	qmath.sqrt	14
4.2.13	qmath.log	14
4.2.14	qmath.log10	14
4.2.15	qmath.deg	14
4.2.16	qmath.rad	14
4.2.17	qmath.frexp	14
4.2.18	qmath.ldexp	15
4.2.19	qmath.min	15
4.2.20	qmath.max	15
4.2.21	qmath.random	15
4.2.22	qmath.crandom	15
4.3	vector	16
4.3.1	vector.New	16
4.3.2	vector.Construct	16
4.3.3	vector.Set	16
4.3.4	vector.clear	16
4.3.5	vector.Add	16
4.3.6	vector.Subtract	16
4.3.7	vector.Scale	16
4.3.8	vector.Length	17
4.3.9	vector.Normalize	17
4.3.10	vector.RotateAroundPoint	17
4.3.11	vector.Perpendicular	17
4.4	entity	18
4.4.1	entity.Find	18
4.4.2	entity.FindNumber	18
4.4.3	entity.FindBModel	18
4.4.4	ent.GetNumber	18
4.4.5	ent.SetKeyValue	18
4.4.6	entity.Remove	18
4.4.7	ent.GetOrigin	18
4.4.8	ent.IsClient	19

4.4.9	ent.GetClientname	19
4.4.10	ent.GetClassname	19
4.4.11	ent.SetClassname	19
4.4.12	ent.GetTargetname	19
4.4.13	ent.SetupTrigger	19
4.4.14	entity.GetTarget	19
4.4.15	entity.Use	19
4.4.16	entity.Spawn	19
4.4.17	entiy.CallSpawn	20
4.4.18	entity.DelayedCallSpawn	20
4.4.19	entity.RemoveSpawns	20
4.5	mover	21
4.5.1	mover.Halt	21
4.5.2	mover.HaltAngles	21
4.5.3	mover.AsTrain	21
4.5.4	mover.SetAngles	21
4.5.5	mover.SetPosition	21
4.5.6	mover.ToAngles	21
4.5.7	mover.ToPosition	21
4.6	sound	22
4.6.1	Sound Channels	22
4.6.2	sound.PlaySound	22
5	Examples	23
5.1	Example 1 - HelloWorld	23
5.1.1	Hello World for game	23
5.1.2	Hello World for a client	23
5.1.3	Hello World for all clients	24
5.2	Example 2 - Finding Entities	24
5.2.1	Finding entities by their targetnames	24
5.2.2	Finding entities by their entity number	25
5.2.3	Finding entities by thier brush model	25
5.3	Example 3 - Spawning entities	25
6	How to ...	27
6.1	add RPG-X2 Turbolifts to older maps	27
6.2	add Transporters with ui_transporter to older maps	27
6.3	convert func_usable force field from older maps to func_forcefield	27

1 Introduction

1.1 General Information

The RPG-X2 Lua Documentation documents and describes all Lua functions available in RPG-X2. The version you are reading right now is for **RPG-X2 version 2.2 beta 4.4.5**. The RPG-X2 Lua Documentation will be updated with every new release of RPG-X2.

1.2 Prerequisites

- In Lua variables are not declared with their type. In order to provide you information of what type a variable is the types will be written in front of variables in italic (example: *integer* **clientNum**).
- There are three different types of function calls in RPG-X2 Lua.
 - Function calls from Lua base libraries (example: **tostring(clientNum)**).
 - Function calls from RPG-X2 libraries which have the library name in front **library.function()** (example: **entity.Spawn()**).
 - Function calls on variables. This is possible on entities and vectors for example (example: **ent.Remove(ent)**).
 - Function calls where the variable a function is called on is the first argument **var.function(var)** can be written as **var:function()** (example: **ent.Remove(ent)** is the same as **ent:Remove()**).

2 Lua Hooks

2.1 What is a Lua Hook

A Lua Hook is a function that gets called when a specific event in the game logic happens. For example if the game is initialized in the game logic `G_InitGame` function gets called. This function has a Lua Hook which means when the `G_InitGame` function is called in the game logic the corresponding Lua function gets called as well. There are Lua Hooks with static function names and Lua Hooks with dynamic function names.

2.2 Static Lua Hooks

Static Lua hooks always have the same function name.

2.2.1 InitGame

`InitGame(integer leveltime, integer randomssed, integer restart)`

Gets called at game start or after a `map_restart` command was issued.

leveltime current level time in milliseconds

restart is 1 when call is result of a `map_restart`

2.2.2 ShutdownGame

`ShutdownGame(integer restart)`

Gets called when the game shuts down (disconnect, game is closed, map change, map restart).

restart is 1 when call is result of a `map_restart`

2.2.3 RunFrame

`RunFrame(integer leveltime)`

Gets called everyframe. Should be used with caution because this is called every frame and the frametime is 50ms.

leveltime current leveltime in milliseconds

2.2.4 GClientPrint

`GClientPrint(string text, entity client)`

Gets called when the game logic function `G_PrintfClient` gets called.

text text that gets printed

client the client the text gets printed for

2.2.5 GPrint

GPrint(*string* **text**)

Gets called when the game logic function G_Print is called.

text text that gets printed to the game console

2.3 Dynamic Lua Hooks

These hooks can have different function names. All of the hooks are for entities. The function names for these are defined in radiant by key-value pairs. As the function names depend on these pairs the function names for these hooks in this documentation are the keys that are used to define the function names in Radiant.

2.3.1 luaThink

luaThink(*entity ent*)

Gets called each time the entity thinks.

ent the entity itself

2.3.2 luaTouch

Gets called each time the entity is touched.

ent the entity itself

other the entity that touched **ent**

2.3.3 luaUse

Gets called each time the entity is used.

ent the entity itself

activator the entity that used **ent**

2.3.4 luaHurt

luaHurt(*entity ent, entity inflictor, entity attacker*)

Gets called each time the entity gets hurt.

ent the entity itself

inflictor the inflictor

attacker the attacker

2.3.5 luaDie

luaDie(*entity ent, entity inflictor, entity attacker, integer dmg, integer mod*)

Gets called when the entity dies.

ent the entity itself

inflictor the inflictor

attacker the attacker

dmg the amount of damage

mod the means of death

2.3.6 luaFree

luaFree(*entity ent*)

Gets called when the entity is freed which means it is removed.

ent the entity itself

2.3.7 luaReached

luaReached(*entity ent*)

Gets called when movement of the entity has reached its endpoint.

ent the entity itself

2.3.8 luaReachedAngular

luaReachedAngular(*entity ent*)

Gets called when angular movement of the entity has reached its endangles.

ent the entity itself

2.3.9 luaTrigger

luaTrigger(*entity ent, entity other*)

Gets called when the entity is triggered. Note that this is not the same as when the entity is used this is for trigger entities.

ent the entity itself

other the entity that triggerd **ent**

2.3.10 luaSpawn

luaSpawn(*entity ent*)

Gets called when the entities spawn function is called.

ent the entity itself.

3 RPG-X2 Map Scripting

3.1 Map scripts

Currently on script file can be loaded for each map. This script file has to be located in *scripts/lua/;mapname;* and must have the name *;mapname;.lua*. *;mapname;* is the name of the .map file and .bsp file.

3.2 Calling Functions

There are Dynamic Lua Hooks for use in Radiant (listed below und Dynamic Lua Hooks in this documentation). You can use these hooks on entities by adding the corresponding Lua Hook key and the function name as value to an entity.

For example if you want a function *PrintText* to be called when a *func_usable* is used you have to add the key *luaUse* and the value *PrintText* to this entity.

4 RPG-X2 Lua Libraries

4.1 game

This library provides access to some game logic function such as `G_Printf` and `G_ClientPrintf`.

4.1.1 game.Print

`game.Print(string text)`

Prints **text** to the game console (the server console).

4.1.2 game.ClientPrint

`game.ClientPrint(integer clientNum, string text)`

Prints **text** to the clients console that has the client number **clientNum**. If **clientNum** is -1 the text gets printed to all clients consoles.

4.1.3 game.CenterPrint

`game.CenterPrint(integer clientNum, string text)`

Prints **text** to the center of the screen of the client with client number **clientNum**. If **clientNum** is -1 the text gets printed for all clients.

4.1.4 game.MessagePrint

`game.MessagePrint(integer clientNum, string text)`

Prints **text** to the lower right corner of the screen of the client with client number **clientNum**. If **clientNum** is -1 the text gets printed for all clients.

4.1.5 game.LevelTime

`game.LevelTime()` Returns the current level time in milliseconds.

4.1.6 game.SetGlobal

`game.SetGlobal(string name, value)`

Sets a global lua variable which is called **name** to **value**. Creates a new global variable if a variable of **name** does not exist. **value** can be of any type.

4.1.7 game.GetGlobal

`game.GetGlobal(string name)`

Returns the value of the global variable **name**. Returns *nil* if the variable does not exist.

4.2 qmath

This library provides access to mathematical functions available in the game code.

4.2.1 qmath.abs

`qmath.abs(float f)`

Returns the integer part of **f**.

4.2.2 qmath.sin

`qmath.sin(float degree)`

Returns the sine of **degree**.

4.2.3 qmath.cos

`qmath.cos(float degree)`

Returns the cosine of **degree**.

4.2.4 qmath.tan

`qmath.tan(float degree)`

Returns the tangent of **degree**.

4.2.5 qmath.asin

`qmath.asin(float f)`

Returns the arcsine of **f**.

4.2.6 qmath.acos

`qmath.acos(float f)`

Returns the arccosine of **f**.

4.2.7 qmath.atan

`qmath.atan(float f)`

Returns the arctangent of **f**.

4.2.8 qmath.floor

`qmath.floor(float f)`

Returns the floored value of **f**.

4.2.9 `qmath.ceil`

`qmath.ceil(float f)`

Returns the ceiled value of `f`.

4.2.10 `qmath.fmod`

`qmath.fmod(float f, float n)`

Returns the remainder of f/n .

4.2.11 `qmath.modf`

`qmath.modf(float f)`

Breaks f apart into its integer part and its fractional part. The fractional part is returned while the integer part is assigned to f

4.2.12 `qmath.sqrt`

`qmath.sqrt(float f)`

Returns the square root of `f`.

4.2.13 `qmath.log`

`qmath.log(float f)`

Returns the logarithm of `f`.

4.2.14 `qmath.log10`

`qmath.log10(float f)`

Returns the logarithm to the base of 10 of `f`.

4.2.15 `qmath.deg`

`qmath.deg(float radian)`

Converts from radian to degrees.

4.2.16 `qmath.rad`

`qmath.rad(float degree)`

Converts from degree to radian.

4.2.17 `qmath.frexp`

`qmath.frexp(float f)`

Breaks `f` into its binary significand and an integral exponent for 2.

$x = \text{significand} * 2^{\text{exponent}}$

4.2.18 **qmath.ldexp**

qmath.ldexp(*float f*, *float n*)

Returns the result from multiplying **f** by 2 raised to the power of **n**.

4.2.19 **qmath.min**

qmath.min(*integer array*[])

Return the lowest value in **array**[].

4.2.20 **qmath.max**

qmath.max(*integer array*[])

Return the highest value in **array**[].

4.2.21 **qmath.random**

qmath.random()

Returns random integers.

4.2.22 **qmath.crandom**

qmath.crandom()

Returns random floats (crazy random function).

4.3 vector

This provides a new type vector along with mathematical functions for it.

4.3.1 vector.New

`vector.New()`

Allocates and returns a new vector $\begin{pmatrix} 0 \\ 0 \\ 0 \end{pmatrix}$.

4.3.2 vector.Construct

`vector.Construct(float x, float y, float z)`

Allocates and return a new vector $\begin{pmatrix} x \\ y \\ z \end{pmatrix}$.

4.3.3 vector.Set

`vector.Set(vector v, float x, float y, float z)`

Set the vector **v** to the specified values.

4.3.4 vector.clear

`vector.Clear(vector v)`

Clears **vector** by setting it to $\begin{pmatrix} 0 \\ 0 \\ 0 \end{pmatrix}$.

4.3.5 vector.Add

`vector.Add(vector a, vector b, vector c)`

Adds **a** and **b** and stores the result in **c**.

4.3.6 vector.Subtract

`vector.Subtract(vector a, vector b, vector c)`

Subtracts **b** from **a** and stores the result in **c**.

4.3.7 vector.Scale

`vector.Scale(vector a, float b, vector c)`

Scales **a** by the value of **b** and stores the result in **c**.

4.3.8 vector.Length

vector.Length(*vector a*)

Returns the length of **a**.

4.3.9 vector.Normalize

vector.Normalize(*vector a*)

Normalizes **a**.

4.3.10 vector.RotateAroundPoint

vector.RotateAroundPoint(*vector dest, vector dir, vector point, float degrees*)

Rotates **point** around a given vector.

dir vector around which to rotate (must be normalized)

point point to be rotated

degrees how many degrees to rotate the point by

dst point after rotation

4.3.11 vector.Perpendicular

vector.Perpendicular(*vector dest, vector src*)

Finds a vector perpendicular to the source vector. **src** source vector **dest** a vector that is perpendicular to **src** (the result is stored here)

4.4 entity

This library holds function for entities. All functions listed with `entity` in front here are calls from the library. All functions listed with `ent` are called on a variable of the type `entity`.

4.4.1 `entity.Find`

`entity.Find(string targetname)`

Returns the first entity found that has a `targetname` of `targetname`.

4.4.2 `entity.FindNumber`

`entity.FindNumber(integer entnum)`

Returns the entity with the entity number `entnum`.

4.4.3 `entity.FindBModel`

`entity.FindBModel(integer bmodelnum)`

Returns the entity with the brush model `*bmodelnumber`. This is the only failsafe way to find brush entities as the entity number is different when you load a map local or join a server.

4.4.4 `ent.GetNumber`

`ent.GetNumber(entity ent)` or `ent:GetNumber()`

Returns the entity number of the entity.

4.4.5 `ent.SetKeyValue`

`ent.SetKeyValue(entity ent, string key, string value)` or `ent:SetKeyValue(string key, string value)`

Sets a key-value pair for `ent` like in Radiant. Only works if the `key` is part of `fields_t` (predefined keys).

4.4.6 `entity.Remove`

`entity.Remove(entity ent)`

Removes/frees `ent`.

4.4.7 `ent.GetOrigin`

`ent.GetOrigin(entity ent)` or `ent:GetOrigin()`

Returns the origin of `ent` as vector.

4.4.8 ent.IsClient

ent.IsClient(*entity* ent) or **ent:IsClient**()

Returns boolean. True if **ent** is a client.

4.4.9 ent.GetClientname

ent.GetClientname(*entity* ent) or **ent:GetClientname**()

Returns the clientname of **ent**.

4.4.10 ent.GetClassname

ent.GetClassname(*entity* ent) or **ent:GetClassname**()

Returns the classname of **ent**.

4.4.11 ent.SetClassname

ent.SetClassname(*entity* ent, *string* classname) or

ent:SetClassname(*string* classname)

Sets the classname of **ent** to **classname**.

4.4.12 ent.GetTargetname

ent.GetTargetname(*entity* ent) or **ent:GetTargetname**()

Returns the targetname of **ent**.

4.4.13 ent.SetupTrigger

ent.SetupTrigger(*entity* ent) or **ent:SetupTrigger**()

Does some setup for entities spawned by script that are to be used as trigger.

4.4.14 entity.GetTarget

entity.GetTarget(*entity* ent) Returns the target of **ent**.

4.4.15 entity.Use

entity.Use(*entity* ent)

Uses **ent**.

4.4.16 entity.Spawn

entity.spawn()

Tries to spawn a new entity and returns it. If no new entity can be spawned *nil* is returned.

4.4.17 entity.CallSpawn

entity.CallSpawn(*entity* **ent**)

Calls the game logic spawn function for the class of **ent**.

4.4.18 entity.DelayedCallSpawn

entity.DelayedCallSpawn(*entity* **ent**, *integer* **delay**)

Calls the game logic spawn function for the class of **ent** after a delay. **delay** delay in milliseconds

4.4.19 entity.RemoveSpawns

entity.RemoveSpawns()

Removes all spawn points from the map.

4.5 mover

Important note: always call `mover.Halt` or `mover.HaltAngles` before you move a mover again otherwise the movement wont work correctly.

4.5.1 mover.Halt

`mover.Halt(entity ent)`

Stops movement immediately.

4.5.2 mover.HaltAngles

`mover.HaltAngles(entity ent)`

Stops angular movement immediately.

4.5.3 mover.AsTrain

`mover.AsTrain(entity mover, entity target, float speed)`

Moves an entity like a *func_train* entity. Targets have to be *path_corner* entities.

`target` the first *path_corner* to move to.

4.5.4 mover.SetAngles

`mover.SetAngles(entity ent, vector angles)` or `mover.SetAngles(entity ent, float x, float y, float z)`

Sets the angles of `ent` to the secified value(s).

4.5.5 mover.SetPosition

`mover.SetPosition(entity ent, vector pos)` or `mover.SetPosition(entity ent, float x, float y, float z)`

Set the position of `ent` to the specified value(s).

4.5.6 mover.ToAngles

`mover.ToAngles(entity ent, float speed, vector angles)` or `mover.ToAngles(entity ent, float speed, float x, float y, float z)`

Rotates `ent` to the specified angles.

4.5.7 mover.ToPosition

`mover.ToPosition(entity ent, float speed, vector angles)` or

`mover.ToPosition(entity ent, float speed, float x, float y, float z)`

Moves `ent` to the specified position.

4.6 sound

This library adds function to play and handle sounds.

4.6.1 Sound Channels

In some function of the sound library you will be asked to specify a sound channel. General it will be ok to use `CHAN_AUTO` and let the engine choose the channel. Anyway you will be able to choose channels yourself.

Here is a table with the different channels and their numbers to use in functions:

<code>CHAN_AUTO</code>	0
<code>CHAN_LOCAL</code>	1
<code>CHAN_WEAPON</code>	2
<code>CHAN_VOICE</code>	3
<code>CHAN_ITEM</code>	4
<code>CHAN_BODY</code>	5
<code>CHAN_LOCAL_SOUND</code>	6
<code>CHAN_ANNOUNCER</code>	7
<code>CHAN_MENU1</code>	8

4.6.2 `sound.PlaySound`

`sound.PlaySound(entity ent, integer chan, string sound)`

Plays the sound file `sound` using the channel `chan` on the entity `ent`.

5 Examples

This section of the manual contains script examples which may help you to understand how certain functions should be used.

5.1 Example 1 - HelloWorld

This is a must have example I think as it always is there for any programming language you learn.

5.1.1 Hello World for game

Listing 5.1: Hello World for game

```
1 function HelloWorld(ent , other , activator)
2     game.Print("Hello World");
3 end
```

As you might not this is a function for luaUse (you can tell that from the function head).

5.1.2 Hello World for a client

Listing 5.2: Hello World for client

```
1 function HelloWorld(ent , other , activator)
2     game.ClientPrint(activator:GetNumber() ,
3         "Hello " .. activator:GetClientname());
4     game.CenterPrint(activator:GetNumber() ,
5         "Hello " .. activator:GetClientname());
6     game.MessagePrint(activator:GetNumber() ,
7         "Hello " .. activator:GetClientname());
8 end
```

As you might not this is a function for luaUse (you can tell that from the function head).

Listing 5.3: First function

```
1 game.ClientPrint(activator:GetNumber() ,
2     "Hello " .. activator:GetClientname());
```

This function prints a message to the clients console.

`activator:GetNumber()` gets the entity number of the activator which in this case is the client number as well.

`activator:GetClientname()` gets the clients clientname.

Listing 5.4: Second function

```
1 game.CenterPrint(activator:GetNumber(),
2     "Hello_" .. activator:GetClientname());
```

This function prints a message to the center of the screen of a client.

Listing 5.5: Third function

```
1 game.MessagePrint(activator:GetNumber(),
2     "Hello_" .. activator:GetClientname());
```

This function prints a message to the lower right corner of the clients screen.

5.1.3 Hello World for all clients

Listing 5.6: Hello World for all client

```
1 function HelloWorld(ent, other, activator)
2     game.ClientPrint(-1, "Hello_all");
3     game.CenterPrint(-1, "Hello_all");
4     game.MessagePrint(-1, "Hello_all");
5 end
```

This is very similar to the previous example the only difference is that instead of a client number -1 is the first arguments which results in the message to be printed to all clients.

5.2 Example 2 - Finding Entities

These examples will show the different ways of finding an entity.

5.2.1 Finding entities by their targetnames

Listing 5.7: Find an entity by its targetname

```
1 function Example()
2     local ent;
3     ent = entity.Find("doorbell");
4 end
```

You should note that `entity.Find()` only returns the first entity found which means if there are multiple entities with the same targetname and the one found first isn't yours you'll be unable to find the wanted entity by this way.

Also besides showing you how to find an entity you also can see how to use local variables in function here.

5.2.2 Finding entities by their entity number

Listing 5.8: Find an entity by its entity number

```
1 function Example()  
2     local ent;  
3     ent = entity.FindNumber(22);  
4 end
```

This is a quite failsafe way to find an entity there is just one thing you have to note: The entity number for an entity when the map is loaded locally is not the same as the entity number for an entity when running a dedicated server.

5.2.3 Finding entities by thier brush model

Listing 5.9: Find an entity by its brush model

```
1 function Example()  
2     local ent;  
3     ent = entity.FindBModel(22);  
4 end
```

This only works for brush entities of course but for these it is absolutly failsafe.

5.3 Example 3 - Spawning entities

This example shows you how to spawn entities from scripting. You can spawn almost all non brush entities as well as some brush entities that don't require a visible brush model (e.g. triggers).

Listing 5.10: Spawning an entity

```
1 function Example()  
2     local ent = entity.Spawn()  
3     if ent == nil then return;  
4     ent:SetKeyValue("classname", "info_notnull");  
5     mover:SetPosition(0, 0, 0);  
6     entity.CallSpawn(ent);  
7 end
```

So what does what and why?

```
1 local ent = entity.Spawn()
```

This tries to spawn a new entity and assign it to ent.

```
1 if ent == nil then return
```

This is a check to make sure that a new entity was successfully spawned. If that is not the case the further execution of the function is stopped.

```
1 ent:SetKeyValue("classname", "info_notnull");
```

This sets the classname and by this the entity is turned to an entity of a specific type.

```
1 mover:SetPosition(0, 0, 0);
```

This sets the origin of the entity.

```
1 entity.CallSpawn(ent);
```

This calls the spawn function of the entity.

6 How to ...

6.1 add RPG-X2 Turbolifts to older maps

Comming soon ...

6.2 add Transporters with ui_transporter to older maps

Comming soon ...

6.3 convert func_usable force field from older maps to func_forcefield

This HowTo shows you how you can convert a func_usable to a func_forcefield. Before we can start scripting we need to find out some things about the usable:

- How can you identify the usable 100 per cent failsafe.
- What are the current spawnflags of the entity.

You can obtain the information by doing the following things.

- Start RPG-X2 and laod the map.
- Login as admin or change to admin class.
- Goto the func_usable and make sure it is visible (the force field is activated).
- Target the usable with your crosshair.
- Open console and type **getEntInfo**.

You'll get a list of usefull information. Now if the entity has a targetname the next thing to do is to check if it is the only one with it. While you are still in console type **getEntByTargetname** followed by the targetname. If only one entity is listed the func_usable is the only one with this targetname and you are done otherwise just use the brushmodel of the func_usable. The next step is to see if the spawnflags are ok for your needs. This means check if any spawnflags of func_forcefield are included you don't want or if some are missing. No you start scripting. The best place to do this entity conversion is the **InitGame** function because this function is already called during map loading.

Listing 6.1: Example 1

```
1 function InitGame(levelTime , randomSeed , restart )
2     — adjust the targetname
3     local ent = entity.Find("forcefield1");
4     if ent == nil then return;
5     ent:SetKeyValue("classname" , "func_forcefield");
6     — setting the spawnflags is optional
7     — only change them if you have to
8     ent:SetKeyValue("spawnflags" , "0")
9     entity.CallSpawn(ent);
10 end
```

Listing 6.2: Example 2

```
1 function InitGame(levelTime , randomSeed , restart )
2     — adjust the model number
3     local ent = entity.FindBModel(22);
4     if ent == nil then return;
5     ent:SetKeyValue("classname" , "func_forcefield");
6     — setting the spawnflags is optional
7     — only change them if you have to
8     ent:SetKeyValue("spawnflags" , "0")
9     entity.CallSpawn(ent);
10 end
```