RPG-X2 Lua Documentation

Written by
Walter Julius 'GSIO01' Hennecke

Updated by
Hendrik 'Harry Young' Gerritzen

For
RPG-X V. 2.2 Beta 8.4.7

Date
2011-10-21

Contents

1. **Introduction** **3**
    1. General Information 3
    2. Prerequisites 3
2. **Lua Hooks** **3**
    1. What are Lua Hooks? 3
    2. Static Lua Hooks 4
    3. Dynamic Lua Hooks 5
3. **RPG-X2 Lua Libraries** **7**
    1. game 7
    2. qmath 8
    3. vector 10
    4. entity 11
        1. Important Information 11
        2. Getters and Setters 12
            1. string 13
            2. entity 14
            3. float 14
            4. vector 14
            5. boolean 15
            6. integer 16
        3. Library Functions 17
    5. mover 18
    6. sound 19
4. **How to...** **20**
    1. ...Find Entities? 20
    2. ...Add Turbolifts to older Maps? 21
    3. ...Add Transporters to older Maps? 24
    4. ...Convert Forcefields to func_forcefields? 25
    5. ...add automatic Map Reload after map_restart? 26
    6. ...add Doorlock using Lua? 27
    7. ...move a Brushmodel using Lua? 28
5. **Debugging** **29**
    1. Logical Bugs 29
    2. Case-Specific Bugs 29
6. **Appendix** **31**
    1. locedit 31

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 4.4.7. 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)**).
    - If the Variable is also the Functions first Argument the Function can be written as **var:function()** (example: **ent:Remove()**)
  - Also please note that Lua is differentiating between capitalized and uncapitalized letters. So ent:remove() won't remove the Entity but will rather put an Error-Message into the game-console.

2.: Lua Hooks

2.1: What are Lua Hooks?

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.

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

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

RunFrame(*integer* **leveltime**)

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

- **leveltime** current leveltime in milliseconds

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

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 functions 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. You will see how this is done in the last Segment of this Documentation.

luaThink(*entity* **ent**)

Gets called each time the entity thinks.

- **ent** the entity itself

luaTouch(*entity* **ent**, *entity* **other**)

Gets called each time the entity is touched.

- **ent** the entity itself
- **other** the entity that touched ent

luaUse(*entity* **ent**, *entity* **other**, *entity* **activator**)

Gets called each time the entity is used.

- **ent** the entity itself
- **activator** the entity that used ent

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

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

luaFree(*entity* **ent**)

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

- **ent** the entity itself

luaReached(*entity* **ent**)

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

- **ent** the entity itself

luaReachedAngular(*entity* **ent**)

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

- **ent** the entity itself

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

luaSpawn(*entity* **ent**)

Gets called when the entities spawn function is called.

- **ent** the entity itself.

3.: RPG-X2 Lua Libraries

3.1: game

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

game.Print(*string* **text**)

Prints **text** to the server console.

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.

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.

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.

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.

game.GetGlobal(*string* **name**)

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

game.LevelTime()

Returns the current level time in milliseconds.

3.2: qmath

This library provides access to mathematical functions available in the game code. Basic arithmetic is available via Lua.

qmath.abs(*float* **f**)
Returns the integer part of **f**.

qmath.sin(*float* **degree**)
Returns the sine of **degree**.

qmath.cos(*float* **degree**)
Returns the cosine of **degree**.

qmath.tan(*float* **degree**)
Returns the tangent of **degree**.

qmath.asin(*float* **f**)
Returns the arcsine of **f**.

qmath.acos(*float* **f**)
Returns the arccosine of **f**.

qmath.atan(*float* **f**)
Returns the arctangent of **f**.

qath.ceil(*float* **f**)
Returns the ceiled value of **f**.

qmath.floor(*float* **f**)
Returns the floored value of **f**.

qmath.fmod(*float* **f**, *float* **n**)
Returns the remainder of f=n.

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.**

qmath.sqrt(*float* **f**)
Returns the square root of **f**.

qmath.log(*float* **f**)
Returns the logarithm of **f**.

qmath.log10(*float* **f**)
Returns the logarithm to the base of 10 of **f**.

qmath.deg(*float* **radian**)
Converts from **radian** to degrees.

qmath.rad(*float* **degree**)
Converts from **degree** to radian.

qmath.frexp(*float* **f**)
Breaks **f** into its binary significant and an integral exponent for 2.
x = significant 2exponent

qmath.ldexp(*float* **f**, *float* **n**)
Returns the result from multiplying **f** by 2 raised to the power of **n**.

qmath.min(*integer* **array[]**)
Return the lowest value in **array[]**.

qmath.max(*integer* **array[]**)
Return the highest value in **array[]**.

qmath.rand)
Returns random integers.

qmath.random()
Returns random floats.

qmath.crandom()
Returns random floats (crazy random function).

3.3: vector

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

vector.New()

Allocates and returns a new vector (0|0|0).

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

Allocates and returns a new vector (**x**|**y**|**z**).

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

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

vector.Clear(*vector* **v**)

Clears vector **v** by setting it to (0|0|0).

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

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

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

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

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

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

vector.Length(*vector* **a**)

Returns the length of **a**.

vector.Normalize(*vector* **a**)

Normalizes **a**.

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
- **dest** point after rotation

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)

3.4: 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. Dependent on if those entities are stowed in a variable they can be written ent.function(ent, parm) or ent:function(parm).

3.4.1: Important Information

Some of the keys in the Radiant are renamed in the radiant for editor convenience. They are stowed in their Game-Variable by the game code. Unfortunately Lua is not able to use these renamed keys, so you have to put your information directly in the Game-Variable. Since these are also only accessible via the game code You may ask for specific information at ubergames.net.

3.4.2: Getters and setters

Probably the most important addition entity-wise is the ability to ask for the value of an entities key and revalue it. This is done by the functions ent.Set<key>(*entity* **ent**, **value**) and ent.Get<key>(*entity* **ent**) where **ent** is the entity the **value** is stowed on. There is also the function ent.SetKeyValue(*entity* **ent**, *string* **key**, **value**), but this one is not as versatile as the Setters regarding possible keys and therefore should be considered as backwards capability only.

The following lists show the available Keys for Getters and Setters sorted by Type. Also some Keys may only be available as Getters or Setters so there will be check boxes for those. If there is a 'V' the Key is available; if there is a 'X' it is not. Numbers refer to notes at the end of the table.

3.4.2.1: string

Key	Available as Getter?	Available as Setter?
Classname	V	V
Targetname	V	V
Bluename	V	V
Bluesound	V	V
Falsename	V	V
Falsetarget	V	V
Greensound	V	V
LuaDie	V	V
LuaFree	V	V
LuaHurt	V	V
LuaReached	V	V
LuaReachedAngular	V	V
LuaSpawn	V	V
LuaHurt	V	V
LuaSpawn	V	V
LuaThink	V	V
LuaTouch	V	V
LuaTrigger	V	V
LuaUse	V	V
Message	V	V
Model	V	V
Model2	V	V (1)
Paintarget	V	V
Redsound	V	V
Swapname	V	V
Clientname	V	X
Parm (2)	V	V

1. Will only take effect after the Entity is respawned
2. Gives one of four Lua Parameters of ent. Codes are **ent.GetParm**(*entity* **ent**, *integer* **parm**) and **ent.SetParm**(*entity* **ent**, *integer* **parm,** *string* **value**)

3.4.2.2: entity

Key	Available as Getter?	Available as Setter?
Activator		X
Enemy		X
LastEnemy	V	
Parent	V	
entity.GetTarget (1)	V	X

1. The total function is entity.GetTarget(*entity* **ent**) and it will give back the Entity **ent** is targeting.

3.4.2.3: float

Key	Available as Getter?	Available as Setter?
Angle	V	V
Distance	V	V
PhysicsBounce	V	V
Random	V	V
Speed	V	V

3.4.2.4: vector

Key	Available as Getter?	Available as Setter?
Apos1	V	X
Apos2	V	X
Movedir	V	V
Pos1	V	V
Pos2	V	V
Origin	V	X

3.4.2.5: boolean

Key	Available as Getter?	Available as Setter? (1)
Booleanstate	V	V
FreeAfterEvent	V	V (2)
NeverFree	V	V
PhysicsObject	V	V
Takedamage	V	V
ent.IsClient (3)	V	X
ent.IsLocked (3)	V	X
ent.IsRocket (3)	V	X
ent.IsGrenade (3)	V	X

1. For setting to the opposite state use the following code: ent:Set<Key>(not ent:Get<Key>())
2. As the name implies this will free the entity immediately as this is considered an Event.
3. Code is **ent.Is <type>**(*entity* **ent**)

3.4.2.6: integer

Key	Available as Getter?	Available as Setter?
Clipmask	V	V
Count	V	V
Damage	V	V
EventTime	V	V
Flags	V	V
Freetime	V	X
Health	V	V
MethodOfDeath	V	V
Moverstate	V	V
N00bCount	V	V
Nextthink	V	V
NoiseIndex	V	V
OldHealth	V	V
PainDebounceTime	V	V
Sound1To2	V	V
Sound2To1	V	V
SoundLoop	V	V
SoundPos1	V	V
SoundPos2	V	V
Spawnflags	V	V
SplashDamage	V	V
SplashMethodOfDeath	V	V
SplashRadius	V	V
Number (1)	V	X

1. Gives out the entity Number of ent

3.4.3: Library Functions

entity.Find(*string* **targetname**)
Returns the first entity found that has a targetname of **targetname**.

entity.FindNumber(*integer* **entnum**)
Returns the entity with the entity number **entnum**.

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.

entity.Remove(*entity* **ent**)
Removes/frees **ent**.

ent.SetupTrigger(*entity* **ent**, *float* **x**, *float* **y**, *float* **z**) or
ent.SetupTrigger(*entity* **ent**, *vector* **size**)
Does some setup for entities spawned by script that are to be used as trigger.

- **ent** the entity
- **x** length along the X-Axis
- **y** length along the Y-Axis
- **z** length along the Z-axis
- can also be stowed in a vector **size**

entity.Use(*entity* **ent**)

- Uses **ent**.

entity.Teleport(*entity* **client,** *entity* **target**)

- Teleports **client** to **target's** position

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

entity.CallSpawn(*entity* **ent**)
Calls the game logic spawn function for the class of **ent**.

entity.DelayedCallSpawn(*entity* **ent**, *integer* **delay**)
Calls the game logic spawn function for the class of **ent** after a delay of **delay** milliseconds.

entity.RemoveUnnamedSpawns()
Removes all spawn points from the maptaht don't have a targetname.

entity.RemoveSpawns()
Removes all spawn points from the map.

entity.RemoveType(*string* **classname**)
Removes all entities of type **classname** from the map.

ent.Lock(*entity* **ent**)
Looks the entity **ent**. Works with anything that can be locked (doors, turbolifts, usables, ...).

ent.Unlock(*entity* **ent**)
Unlocks the entity **ent**. Works with anything that can be locked (doors, turbolifts, usables, ...).

3.5: mover
Important note: always call mover.Halt or mover.HaltAngles before you move a mover again otherwise the movement won't work correctly.

mover.Halt(*entity* **ent**)
Stops translational movement on **ent** immediately.

mover.HaltAngles(*entity* **ent**)
Stops rotational movement on **ent** immediately.

mover.AsTrain(*entity* **mover**, *entity* **target**, *float* **speed**)
Moves an entity like a func_train entity. Targets have to be

path_corner entities.

- **ent** the entity to move
- **target** the first path_corner to move to
- **speed** speed to move to first path_corner with

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

Sets the angles of ent to the specified **value(s)**. Values are sorted Pitch (around Y-Axis), Yaw (around Z-Axis) and Roll (around X-Axis). These can also be stowed in a vector **angles**.

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)**. Can also be stowed in a vector **pos**.

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

Rotates **ent** with speed **speed** (in degrees per second) to the specified **value(s)**. Values are sorted Pitch (around Y-Axis), Yaw (around Z-Axis) and Roll (around X-Axis). These can also be stowed in a vector **angles**.

mover.ToPosition(*entity* **ent**, *vector* **pos**) or
mover.ToPosition(*entity* **ent**, *float* **x**, *float* **y**, *float* **z**)

Moves **ent** with speed **speed** to the specified **value(s)**. Can also be stowed in a vector **pos**.

3.6: sound

This library enables sounds to be played as part of a Lua-Script. There is only one function available in this library.

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

- **ent** the entity the sound will be played on
- **sound** the sound file which will be played

- **chan** the sound channel the sound will be played on

CHAN_AUTO (recommended)	0
CHAN_LOCAL	1
CHAN_WEAPON	2
CHAN_VOICE	3
CHAN_ITEM	4
CHAN_BODY	5
CHAN_LOCAL_SOUND	6
CHAN_ANNOUNCER	7
CHAN_MENU1	8

3.7: trace

This Library enables Lua-managed traces.

trace.DoTrace(*vector* **start**, *vector* **mins**, *vector* **maxs,** *vector* **end**, *float* **passEnt**, *float* **contents**)

Does a trace.

- **start** start-point of trace
- **mins** minimal distance of trace (nil if unused)
- **maxs** maximal distance of trace (nil if unused)
- **end** end-point of trace
- **passEnt** Number of ents to pass
- **contents** ????????

trace.FreeTrace(*trace* **tr**)

Ends trace-process for **tr**.

3.8: cinematic

This library manages the cinematic camera... but I don't know how so these are all guesstimates ^^.

cinematic.Activate(*entity* **ent**, *entity* **target**)

Activates Camera on **ent** and points it at **target.**

cinematic.Deactivate(*entity* **ent**)

Deactivates camera on **ent**.

cinematic.ActivateGlobal(*entity* **target**)
Activates broadcasting of **target**.

cinematic.DeactivateGlobal()
Deactivates broadcasting.

4.: How to...

In this segment I will show you some example functions and basic operations you can use for your own Lua projects. They are mostly based on real projects I did and released. You may feel free to just copy them if you wish, but I'll tell you work will be easier if you understand what you are doing. Also please understand that I'm sharing my Experience using Lua with you and that I won't hold back my opinion on whether something works good or bad. That shouldn't hold you back from trying tough.

In the Code you will often find -- or --[[followed by some Text and in the latter case a]]-- . These are comments I put in to explain certain things that may not be selfexplaining.

4.1: ...Find Entities?

There are three different ways to locate a specific entity: by Targetname, by Entitynumber and by Brushmodelnumber. They all have their requirements, their advantages and their disadvantages and I will limit myself to these as the code will be shown in the following examples.

Locating Entities by Targetname in the beginning seems like the most straight forward way to dial into an entity. If you get into it tough you'll realize that Lua will only return the first Entity with matching Targetname and that Entity might not be the one you're looking for. Still this is the only way to locate all sorts of non-Brush-Entities (exluding misc_model_breakables) so if you are looking for one of those there's no way you get around using this. If however you are building a Lua script for a map of yours there's no better thing than

locating your Entity by it's Targetname as everything else can swift from compile to compile and if you need the Entity to be available by another Targetname you can simply put a target_relay in. You can find out the targetname of a visible Entity ingame by logging in as an Admin, pointing your Croshair at it and use the command '/getEntInfo'.

Locating Entity by their Entitynumber is kind of a last resort since those vary if you are on a local or dedicated server. Therefore you have to build a switch in your script that checks if it's executed on a local or a dedicated Server (I had to make one for the Refit of the Borg-Maps). You can get the Entitynumber by logging in as an Admin and scanning the Entity with the Tricorder.

As the name implies, finding Entities by their Brushmodelnumber only works for Brushmodels. The advantage over Entitynumber is, that the Brushmodelnumber is the same on any kind of Server. Just as the Targetname you get it by logging in as an Admin, pointing your Croshair at it and use the command '/getEntInfo'.

4.2: ...Add Turbolifts to older Maps?

In the history of RP-Maps target_turbolift is still a fairly new function and therefore one of the best things to refit on older maps. It has tough some Requirements. First of all the map has to feature a Turbolift-system, that is based on the Idea of nearly seamless transportation. So if the map you wish to refit has a func_train-turbolift you're stuck with that system. Beyond that you need at least one Usable at each Turbolift-Location.

There are two stages to adding a Turbolift: Creating and Programming the Entity and redirecting the Usable to target it. Additionally you may wish to modify the doors to be able to target them. All of these steps have to be repeated for each Location which isn't as hard as it sounds cause once you have the system in place you can copy and paste it as often as you need with only the needed adaptions like Targetnames.

The following pieces of code are taken from my enterprise-e-v2-Upgrade. They have been adapted to the new setters as orriginaly they use 'SetKeyValue'.

```
--[[So usually the only thing you have is a set of doors and a bunch of
usables.
Since we do not wish to return to other entities later let us start with
he modification of the doors.
In the end it doesn't matter tough]]--

function InitGame(levelTime, randomSeed, restart)  --We want this at
mapload so we need this hook
     game.Print("Initializing Lua Map Upgrade ...");
     game.Print("-Turbolift Setup ..."); --These lines are for
debugging purposes as explained in Section5
          game.Print("--Deck 1 ...");
               game.Print("---renaming doors ...");
                    ent = entity.FindBModel(7); --finds the Entity by
Brushmodelnumber
                    ent:SetTargetname("tld1doors"); --Note that
strings have to be written in "..."
                    ent = entity.FindBModel(8);
                    ent:SetTargetname("tld1doors");
--[[With that the done the next step should be to create the turbolift-
entity itsself]]--
               game.Print("---Adding turbolift ...");
                    ent = entity.Spawn();
                    ent.SetupTrigger(ent, 144, 100, 98);
--[[technically the Turbolift is a trigger and needs to be defined as a
space ]]--
                    ent:SetClassname("target_turbolift");
                    ent:SetTargetname("tld1");
                    ent:SetSwapname(); - if you want to deactivate
them, I did not
                    ent:SetTarget("tld1doors");
                    ent:SetHealth("1"); --Health holds the Deck Number
                    ent:SetWait(3000);
                    entity.CallSpawn(ent);
                    mover.SetPosition(ent, -2976, 8028, 887); --all
ent's spawn at (0|0|0) first
                    mover.SetAngles(ent, 0, 270, 0); --to have it
'view' the doors
--[[You can get the position and angles either by decompiling your
target-map or using the command /clientpos ]]--
--[[With the turbolift set up all there is left is to hook it up to the
usables as shown on the next page]]--


               game.Print("---redirecting usables ...");
                    ent = entity.FindBModel(90);
                    ent:SetTarget("tld1");
                    ent:SetLuaUse("turbosound");
                    ent = entity.FindBModel(86);
```

```
                        ent:SetTarget("tld1");
                        ent:SetLuaUse("turbosound");
                        ent = entity.FindBModel(87);
                        ent:SetTarget("tld1");
                        ent:SetLuaUse("turbosound");
                        ent = entity.FindBModel(167);
                        ent:SetTarget("tld1");
                        ent:SetLuaUse("turbosound");
                        ent = entity.FindBModel(88);
                        ent:SetTarget("tld1");
                        ent:SetLuaUse("turbosound");
                        ent = entity.FindBModel(89);
                        ent:SetTarget("tld1");
                        ent:SetLuaUse("turbosound");
      game.Print("... done.");
end

--[[Now you may wonder what about that luaUse-Parameter? Remember the
'Specify deck desired' from the torumode? It will give either that or an
error sound along with a message. This is not yet implemented in the
Lua-Upgrade.]]--

function turbosound (ent, other, activator)
      tgt = entity.GetTarget(ent)   --Gets the turbolift
      if tgt:IsLocked() = true then --Checks if locked
            sound.PlaySound(ent, "sound/movers/switches/voyneg.mp3", 0)
--gives error sound
            game.MessagePrint(activator:GetNumber(), "=C= Unable to
complie: The Turbolift is offline.)
      else
            sound.PlaySound(ent,
"sound/voice/computer/tour/trblftmenu.mp3", 0) --Specify deck
      end
end
```

Now you may wonder: What about deckName? Well there's no way to stow this using Lua. You'll have to resort to create a <mapname>.turbolift in the maps-folder. If you haven't dealt with such a file this is how it should look:

```
1 "Deck 1: Bridge | Ready Room | Conference Room"
2 "Deck 2: Quarters | CO-Quarters | Mess hall"
3 "Deck 3: Quarters | Mess hall | Holodeck | Transporterroom"
7 "Deck 7: Shuttlebay | cargobay | Airlock"
8 "Deck 8: Sickbay | Astronomy | Armoury | Brig"
16 "Deck 16: Maintenance | Main Engineering"
18 "Deck 18: Maintenance | Science Lab | Computer Core"
```

As you probably figure the leading number gives the Deck-Number the following string shall be associated with.

4.3: ...Add Transporters to older Maps?

Transporters are yet another new feature for RP-Maps. Just push a few buttons, go to the pad in time and you end up at your target destination. This is also a function that is retrofittable and that the map will benefit from. All you need is a usable that originally gave a transporter-sound or activated an old fasion 2-way-transporter. Additionally you need location for the transporter to work. If there are none I will explain how to do these in the appendix.

The following pieces of code are taken from my rpg_magnificent-Upgrade. They have been adapted to the new setters as originally they use 'SetKeyValue'.

```
function InitGame(levelTime, randomSeed, restart)  --We want this at
mapload so we need this hook
     game.Print("Initializing Lua Map Upgrade ...");
          game.Print("--Cargobay ..."); --Again Debugging-Notes
               game.Print("---redirecting usable ...");
                    ent = entity.FindBModel(68);  --this will call the
UI
                    ent:SetTarget("tr_ui-cb");
                    ent:SetLuaUse("transoff");
                    entity.CallSpawn(ent);
               game.Print("---setting up trigger ...");
                    ent = entity.Spawn(); --This will be the
Transporters
                    mover.SetPosition(ent, -2264, -1496, 4220);
                    ent.SetupTrigger(ent, 96, 96, 104);
                    ent:SetClassname("trigger_transporter");
                    ent:SetWait("5");
                    ent:SetTargetname("tr_trigger-cb");
                    entity.CallSpawn(ent);
               game.Print("---setting up UI ...");
                    ent = entity.Spawn(); --This will manage the UI
                    ent:SetClassname("ui_transporter");
                    ent:SetTargetname("tr_ui-cb");
                    ent:SetSwapname(); --Will deactivate the
transporter if you wish
                    ent:SetTarget("tr_trigger-cb");
                    entity.CallSpawn(ent);
               game.Print("---Clean-Up previous System ...");
                    ent = entity.Find("t417"); --There was a 2-way-
Transporter so I locked it
                    ent:Lock();
               game.Print("---prep beam-ins ...");
                    ent = entity.Spawn(); --This is where players will
spawn when beaemed here
                    ent:SetClassname("info_notnull");
                    ent:SetTargetname("Cargobay");
```

```
                    mover.SetPosition(ent, -2256, -1496, 4172);
                    mover.SetAngles(ent, 0, 270, 0);
                    entity.CallSpawn(ent);
                    […] --Placeholder only, here would be the other
spawns
     game.Print("... done.");
end

--[[To date I have not yet figured out a way to give out the tourmodes
“Transporter Activated. Please move to the Transporter Platform.” but
I'm working on that. In the mean time you may wish to give out an error
when the transporter is offline. It works just as the Turbolift does
with the exception that we do only need this in one case so no 'else' is
required]]--

function transoff (ent, other, activator)
     tgt = entity.GetTarget(ent)   --Gets the UI
     if tgt:IsLocked() = true then --Checks if locked
          sound.PlaySound(ent, "sound/movers/switches/voyneg.mp3", 0)
--gives error sound
          game.MessagePrint(activator:GetNumber(), "=C= Unable to
complie: The Transporter is offline.”)
end
end
```

4.4: ...Convert Forcefields to func_forcefields?

While actually being around for quite some time many maps did not have the advantage of having forcefields that have a knockback and are only visible on contact or use. Those maps had to either resort on usables or – in fewer cases – doors. Now there are problems with each that I came across. Take the rpg_magnificent for instance. There is a force field around the center bed that is a func_usable. However it actually consists of two usables, one for the inner Texture and one for the outer. You can certanly make those func_forcefields individually but the opposite side will never see the forcefield appear when contacted. If you have something like that just leave it for now.

As for the func_door the problem can be that the forcefield is not an Individual door but part of a bigger door like in the rpg_magnificents Shuttlebay. If that is the case you better forget about converting forcefields on that map at all cause having two Forcefield-Systems looks kind of stupid, at least to me.

The other thing you can come across with func_door-forcefields are

areaportals. Now there are only a few examples where a func_door consists of an areaportal and a transparent forcefield-texture as that will create a blur-effect. A prominent example of this however is the 'Nelson is down'-Forcefield on borg3. Now when I did the Lua-Upgrade for that map I wanted all forcefields to be converted to func_forcefield. Additionally this forcefield had no visible trigger in the story-mode (it was triggered by the end of combat) and was just seperating one area from another. So in order to have only one type of forcefields I just set the doors wait to -1 and opened it by Lua so it was clamped open.
So my general recommendation with func_door-forcefields is: either leave them and every other forcefield on that map as they are or try to hide it if you can. Else they won't be much different than usables.

As you'll see converting usables to func_forcefields is a one-line-deal tough there can be sorroundings that make the process a little more 'complicated' as this example from the enterprise-e-v2 will show.

```
function InitGame(levelTime, randomSeed, restart)  --We want this at
mapload so we need this hook
     game.Print("Initializing Lua Map Upgrade ...");
     game.Print("-Setting up force fields...");
          game.Print("--Sickbay..."); -- Again for debugging
               ent = entity.FindBModel(161);
               ent:SetClassname ("func_forcefield"); -- And that's the
conversion
               ent:SetTargetname ("sick-ff"); --There were sounds
associated with the old effect
               entity.CallSpawn(ent);
               ent = entity.FindBModel(157);
               ent:SetTarget("sick-ff"); - Retarget the triggering
unsble
               entity.CallSpawn(ent);
end
```

4.5: ...add automatic Map Reload after map_restart?

I think we all know the headlock issue that arises when using a Turbolift on a map which has been going trough a map_restart. This can be prevented using Lua and I highly recommend adding this feature to any file as there might be other issues prevented by this as well.
This one is from the enterprise-e-v2 again.

```
function InitGame(levelTime, randomSeed, restart)
     game.Print("Initializing Lua Map Upgrade ...");
     game.Print("-map_restart ..."); - You should have gotten it by now
bv
     game.Print("--workaround-setup ...");
                ent = entity.Spawn(); -- This entity will reload the map
if needed
                ent:SetClassname("target_levelchange");
                ent:SetTargetname("map_restart");
                ent:SetTarget("enterprise-e-v2"); -- Remember to fill
yor map in here
                entity.CallSpawn(ent);
     game.Print("--trigger-setup ...");
          if restart == 1 then -- check if a restart happened and give
the following information
                game.ClientPrint(-1, "Please don't use map_restart, use
map or devmap instead.");
                game.Print("Please don't use map_restart, use map or
devmap instead.");
                ent = entity.Find("map_restart")
                entity.Use(ent) -- triggers the above entity
          end
end
```

4.6: ...add Doorlock using Lua?

Locking doors is nothing that is easily refittable (due to the lack of usables) yet something you can do by Lua with a simple script. This one is from the rpg_magnificent.

```
-- since we're going to do this by a luaUse-hook were going to refit one
first.
function InitGame(levelTime, randomSeed, restart)
     game.Print("Initializing Lua Map Upgrade ...");
     game.Print("-Intensive Care Lockdown ...")
          game.Print("--usable fit ...");
                ent = entity.FindBModel(205);
                ent:SetLuaUse("sicklock");
                entity.CallSpawn(ent)
end

-- now we need the function that will manage the lock

function sicklock ( ent , other , activator )
     ent = entity.FindBModel(202);
     if ent.Is Locked(ent) == true then
     ent.Unlock(ent);
     else
     ent.Lock(ent);
end
```

4.7: ...move a Brushmodel using Lua?

Moving a Brushmodel (and attached misc_models)is not a total revolution. However Lua presents us with a way to refit motions and

allows for very precise and simple directing over multiple points. It also allows for rotational movement which is not possible with a func_train. This one is from dn5. Is alo shows you how you can transfer between consecutive maps as this is about the Train at the end of the level.

```
-- since we're going to do this by a luaUse-hook were going to refit one
first.
function InitGame(levelTime, randomSeed, restart)
     game.Print("Initializing Lua Map Upgrade ...")
     game.Print("-Setting up train...");
                ent = entity.FindBModel(13);
                ent:SetClassname("func_usable");--makes console work
                ent:SetTarget("transfer");
                ent:SetLuaUse("transferstart");
                ent:SetLuaReached("transfermove");
                ent:SetSpawnflags("8");
                ent:SetWait("3");
                entity.CallSpawn(ent);
                ent = entity.Spawn();
                ent:SetClassname("target_levelchange"); --does the
levelchange
                ent:SetTargetname("transfer");
                ent:SetTarget("dn6");
                ent:SetWait("15");
                entity.CallSpawn(ent);
end
-- Now all we need are scripts that control our motion for us. First
we'll move a little to the Wall
function transferstart(ent)
     ent = entity.FindBModel(13);
     ent:SetClassname("func_static");-- Locks off the Usable
     mover.ToPosition(ent, 10, -768, -480, 1012);-- our direction and
speed
    sound.PlaySound(ent, "sound/movers/doors/largedoorstart.mp3", 0);
    -- Sounds make motions generally feel more realistic.
end
--Then it's down the track
function transfermove(ent)
     ent = entity.FindBModel(13);
     mover.Halt(ent); --we need toi stop first or else it can get nasty
    sound.PlaySound(entity.FindBModel(5),
"sound/movers/doors/largedoorstop.mp3", 0);
     mover.ToPosition(ent, 125, -768, 1696, 1012);
    sound.PlaySound(ent, "sound/movers/doors/largedoorstart.mp3", 0);
     ent = entity.FindBModel(76); --we have two seperate Brushmodels.
     mover.ToPosition(ent, 125, -568, 1820, 1144);
    sound.PlaySound(ent, "sound/movers/doors/largedoorstart.mp3", 0);
end
```

5.: Debugging

As you start writing your Lua-files you're going to make mistakes. The first way you'll notice that is whenever your script does not work

in game. Most of them will likely be misspellings or forgotten symbols. Luckily the Game will give out debugging-information to the Console whenever an issue arises. That message will always be in yellow and have the following syntax.

```
Lua: <error-type>: [string
"<filepath>"]:<line of the Bug>: <Description of the issue>

example:

Lua: syntax error: [string
"scripts/lua/rpg_magnificent/rpg_magnificent.l..."]:413: function
arguments expected near 'Locked'
```

There are two basic types of Bugs: Logical Bugs and Case-Specific Bugs.

5.1: Logical Bugs

Logical Bugs are in the logical Structure of the File. They get checked for when the Game tries to load the file (LuaInit) and the entire file is unloaded if such a thing happens. Usually these are forgotten Symbols or logical words such as a 'then' before the first stage of an If-Statement or a ';' at the end of each line (except lines beginning with 'if' or 'function'). Missing Arguments in functions also go in this category.

5.2: Case-Specific Bugs

These Bugs are Case-Specific to their functions. Usually these are variables that have not been defined or entity.find's that have a non-existent target. If such a bug is found the Function is stoped at that point with aany previous modification not being undone. These Bugs are shown when the Function get's called. In the case of InitGame-Calls that is in the 'Map Loading'-Segment right above the ASCII-Combadge and has one additional advantage. The Debugging-Messages that I distributed all over the above examples will show up here and either show you that the file is loaded prefectly or where the bug is. In the case of my enterprise-e-v2 this is what it looks like.

```
Initializing Lua Map Upgrade ...
-map_restart ...
--workaround-setup ...
--trigger-setup ...
-Adding Transporter with UI ...
--redirecting usable ...
--setting up trigger ...
```

```
--setting up UI ...
-Turbolift Setup ...
--Deck 1 ...
---redirecting usables ...
---renaming doors ...
---Adding turbolift ...
--Deck 2 ...
---redirecting usables ...
---renaming doors ...
---Adding turbolift ...
--Deck 3 ...
---redirecting usables ...
---renaming doors ...
---Adding turbolift ...
--Deck 7 ...
---redirecting usables ...
---renaming doors ...
---Adding turbolift ...
--Deck 8 ...
---redirecting usables ...
---renaming doors ...
---Adding turbolift ...
--Deck 16 ...
---redirecting usables ...
---renaming doors ...
---Adding turbolift ...
--Deck 18 ...
---redirecting usables ...
---renaming doors ...
---Adding turbolift ...
-Setting up force fields...
--Sickbay...
--Brig...
---Cell A...
---Cell B...
---Cell C...
--Warpcore...
-Upgrading Engineering-Lockdown...
--removing Forcefields...
--usable fit...
-Adding weapon FX ...
--target setup ...
--usable setup ...
--Phaser ...
--Torpedo ...
... done.
```

As you see I build it somewhat collapsable for better orientation.

6.: Appendix

6.1: locedit

As I stated above you will need locations if you wish to use a Transporter on your map. Now a few years ago that was hard work either setting entities or getting the ressources to write a .locations-

file. The latter can now easily be done from ingame using locedit.

Locedit is very streight forward to use. You begin vy issuing the command '/locedit start X' where X is either 0 or 1. The difference between them is that if you set 1 you can restrict certain areas so only Admins can authorize a transport there. It also makes a slight difference in what code you use to add a location. If you set 0 the code is '/locedit add <locationname>' and if you set it to 1 it is '/locedit add X <locationname>' where for this X 1 is admin-protected and 0 is not. The command will then replace the 'add' with your current position and your yaw-angle (which will be the direction in which the beamed-in Player will look) and dump the entire line to the .locations-file.

If you wish to give the textfile some structure you can use '/locedit nl' to add an empty line. Once you're done issue '/locedit stop' to close off the file. It will be stowed as and in '.../RPG-X2/maps/<mapname>.bsp.locations' (if you use an IO-EF-based engine that will be in the approaming/stvef). All you need to do is remove the '.bsp' from the file name and your done. If you load the map now there should be locations available to you.