User manual

TPMirror

version 2.0 (March 6, 2012)

english & deutsch

Contents

1 English

1.1 Introduction

TPMirror is a teleport device, working grid-wide by using the RLV technology. TPMirror was once developed as a RLV teleporter. Now, in the version 2, it was migrated to RTS technology. RTS is a shorthand for '*RLV Teleporter System*'. It is a framework for building of RLV teleporters. RTS offers this way a number of features to TPMirror.

Why is TPMirror different? TPMirror is a grid-wide teleporter, it acheives any location that one can visit via a landmark. The avatar is not moved physically, the teleport not fails if the target location is on different sim or continent. TPMirror also doesn't open the map window, like map teleporters do. instead, TPMirror passes over the target coordinate to the viewer, enforcing the immediate teleport.

You can imagine TPMirror as a tunnel entrance. You can not see through the tunnel, you can only see the entrance, perhaps a picture on the wall, or on floor, but as soon you're pasing the entrance the tunnel brings you at the opposite side – the teleport destination.

Basically, RLV teleporters require the viewer to support a special RLV interface to pass over the target coordinate. TPMirror works this way, too. But since not every viewer suports the interface, TPMirror works also without it: TPMirror allows to switch the RLV support off and turns than to an usual landmark giver.

Also the supplied TP Relay turns any RLV teleporter (also TPMirror with active RLV support) into a map teleporter. This allows to use RLV teleporters, TPMirror and every map teleporter on same way and with any viewer. For this reason TPMirror passes this relay over to users not running RLV.

1.1.1 Audience

Who is TPMirror designed for? Well, it allows to jump over to places on the same sim, like a skybox, club or another floor in the house. This are local teleports, and there are also simplier and lighter solutions to acheive that goals.

But if the task is to make any place on the grid accessible as if it was next door, if you need connnect places like rooms in a house, no matter if those places are on same sims or spread over the whole grid, all belongs to you or to you business partner or even any stranger, then is TPMirror one of devices of choice.

RTS was designed for managing multiple teleport destinations. For traditional reasons, TPMirror is a single-target device. But with a little effort, one can extend it by multi-target ability. The package supplies a step-by-step tutorial about how to do this, and also the rebuilt teleporter: TPArch is a RTS Teleporter which manages up to 80 destinations.

1.1.2 Scope

The documentation structure is this: The *introduction* explains shortly the content of the product package and how to update it. Also, it explains what RLV is, why we need this interface, which viewers support it and, finally, how to use TPMirror without running RLV-capable viewer.

The section 1.2 handles the *installation* and application of the teleporter. Its *configuration* is, again, explained in the section 1.3.

The section 1.4 follows, giving a brief introduction in the how the teleporter works and also a stepwise tutorial about how to extend TPMirror to TPArch, a teleporter using the multitarget feature of RTS.

TPMirror is a RTS teleporter. Hence, the RTS documentation is important for background information going farther than both last sections. The RTS documentation you can download on the RTS product page[1], it is linked as a single PDF file.

1.1.3 Package content

The package content is shown in the figure 1.1. Most importnt content is listed:

- '**TPMirror**' and '**TPArch**': The teleporter and its extension. Both are ready to use, but need a reconfiguration, for instance to define the teleport destinations.
- '**PortalFrame**' makes an open-air instlallation of the teleporters possible.
- '**KeyHolder**' is a device, required for requesting manual redelivery and update, as described in section 1.1.3.2.

[1] https://marketplace.secondlife.com/p/RLV-Teleporter-System-private-edition/2944310

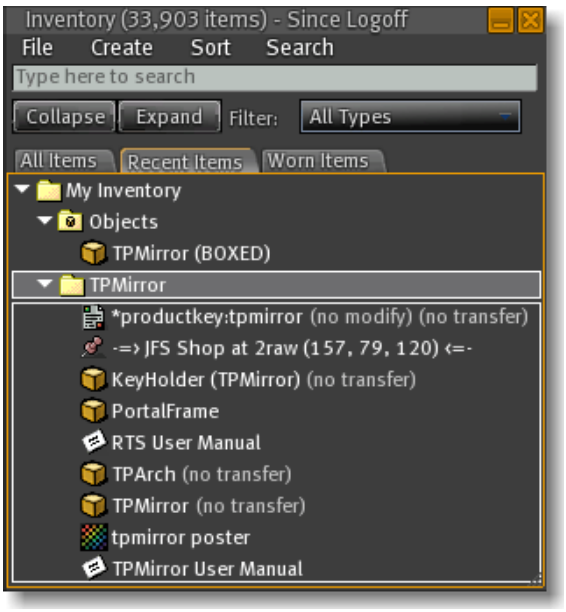

Figure 1.1: TPMirror, package content

- '***productkey**' is a script working inside the KeyHolder.
- '**TPMirror User Manual**' and '**RTS User Manual**' are help resources for the teleporter and the core system driving it. Both include further notecards with partial documentation and also those in german.

1.1.3.1 Unpacking

There are two ways to unpack the package, traditional and automated. In first case, please rezz it on floor, ignore the menu, open the package and copy the content. You receive than a folder named '**TPMirror (BOXED)**'.

You can also let the package box open itself. To do so, rezz the box on floor or wear in hand (if rezzing is not allowed.) If you are on a place allowing scripts, a menu opens. If you ignore it, you can reopen the menu by clicking the box.

In this menu, please hit the '**Open**' button. The box will give you the product folder '**TPMirror**' and then opens the menu again, where you need the '**Remove**'. This button destroys the box or detaches it if worn.

Please, keep the product package as backup for the files, or at least the script '***productkey**' from it. It is the only way to request product updates manually. There is also an online tutorial about automated unpacking product boxes available.[2]

[2] http://jennassl.blogspot.com/2011/11/unpacking-jfs-boxes.html

1.1.3.2 Update

TPMirror is inside a vending system at time, which allows automatic delivery of updates. But you also can request updates manually. To do so, you have two ways: Via the product box itself and via the supplied key holder.

In both cases the script '***productKey**' is working, it connects to an update orb installed in one of JFS stores. The connection is local via chat, hence you need to visit the shop for it. Since they move sometimes, a blog post provides the list.[3]

Instead of the *product box* you can use any prim with installed '***productKey**' script. This is how to request updates by using this way:

Come closer than 10m to the update orb. Now rez please the product box on the ground, a menu opens. Here, hit the '**Update**' button. The update orb will hang out the update, the menu opens again, you can hit the '**Remove**' button there.

The *KeyHolder* is a watch-like device made to keep up to 12 product keys. This way you can update up to 12 products by using a single device. To make the key holder work, you need to install the '***productKey**' script into. In the supplied device the script is already installed. To request updated by using key holder, please do this:

Come closer than 10m to the update orb. Wear the key holder and touch it. A menu opens. The RTS product should be selected. If not, please select it. After a click on the '**Update**' button, the update orb hang out the product update. You can take off the key holder now.

In both cases, the update is a redelivery of the actual product box. There is also an online tutorial about updating UFS products available.[4]

1.1.4 Why RLV teleporters

RLV was a special viewer originally, made with a goal of support the immersion of the user (the feeling, being in the shown scene.) Herefor, the viewer offers a special interface, which allows scripts to gain a limited control over the viewer.

With this interface, scripts can run a number of actions on the viewer, the user of the viewer can do usually, like standing up if sitting, or teleport to a given position. The list of possible actions is relatively huge, but TPMirror needs only one of them: It sends the teleport command to the viewer.

If received this command, the viewer connects immediately to the target sim and location. The user sees only the progress bar and short time later the avatar standing

[3] http://jennassl.blogspot.com/2010/05/actual-shop-list.html
[4] http://jennassl.blogspot.com/2011/11/updating-jfs-products.html

on target place. Quite same like by using a landmark or world map, but without opening of landmark or map window.

Exactly here is the difference of RTS teleporters (or in common of RLV teleporter) against the landmark distributors or map teleporters: The user is not pulled out of the scene by opened landmark or world map window.

Originally, only the RLV implemented this interface, but since the developer of the viewer has published it,[5] other creators could make theirs objects RLV-capable and make the viewer more and more popular. Now is this interface also available for all viewer developer as '*RLVa*'[6] and is implemented by many other third party viewers.

1.1.4.1 Without RLV

A disadvantage of RLV teleporters is, not all viewers support the required interface. A user of a viewer which not supports it, cannot use those teleporters in described way. Especially the official viewer doesn't support the interface.

While developing of RTS this was taken in account: RTS teleporters, and hence also TPMirror, allow to switch the RLV support off in the configuration. The teleporter works with any viewer than, but instead of accessing the RLV interface just provides a landmark or SLURL to the target place.

1.1.4.2 With a teleport relay

Another possibility to use RLV/RTS teleporters (hence TPMirror) with a viewer without support of the RLV interface, is to use the teleport relay.

To use a RLV teleporter is generally not only a RLV-capable viewer required, but also a special device, so-called relay. RLV has a built-in technical safety gate: Only commands from an owned device are accepted. RLV teleporters belong usually not to the user of the viewer and cannot access the viewer directly.

The relay resolves this problem. It belongs to the user of the viewer and is heard by the viewer. The relay receives commands, sent by the RLV devices and relies them to the viewer.

Usually, the relay performs a security check: Is the device allowed to control the viewer or not? In case of doubts, the relay asks the user via menu, but if a device is on black or white list of the relay, the relay declines or accepts the commands automatically.

[5] http://wiki.secondlife.com/wiki/LSL_Protocol/RestrainedLifeAPI
[6] http://rlva.catznip.com/blog/2009/10/release-rlva-1-0-5/

Common relays follow the specification, they require running the RLV-capable viewer and relay all commands to the viewer. But to use RLV teleporters, only three commands are relevant: The *version check*, the *unsit command* and the *teleport command*.

Only this three commands the teleport relay accepts and relies to the viewer and declines every other. This allows to use a RLV teleporter without risk of an unwanted action of RLV devices. By using other relays one has to deal with theirs security settings first.

If a viewer without the RLV interface is used, the teleport relay emulates the interface by sending a faked version response to the teleport device, and by opening the map window for teleport.

This way inworld devices have a limited control over RLV-capable viewer. With a viewer that can't RLV, RLV teleporters become also usable – they turn to a regular map teleporters. The relay was created with this purpose in mind and is installed into TPMirror for hanging out if the user has no relay in use.

1.1.4.3 RLV-capable viewers

There are a few viewers meanwhile that support the RLV interface and make it possible to use RLV teleporters with an usual relay. Some of them are listed below.

- Classical RLV (http://www.erestraint.com/realrestraint/)
- Cool Viewer (http://sldev.free.fr/)
- Firestorm and Phoenix viewers (http://www.phoenixviewer.com/)
- Imprudence (http://blog.kokuaviewer.org/)
- Rainbow (http://my.opera.com/boylane/blog/)
- Singularity (http://www.singularityviewer.org/)

The list must not be complete and should give just an overview.

1.1.5 Global coordinates

Every place in SL can be given by a local and global coordinate. Local coordinate defines the position of a point relative to a special given origin.

For example you can see in the symbol bar the local position of the avatar relative to the sim origin. Be it a point with coordinate `<128, 128, 24>` (exactly in the middle the sim.) On the whole SL world, there are as many points with local coordinate

<128, 128, 24>, as many sims there are. So, only if the sim is specified, you can position a point by a local coordinate clearly in the SL world.

A SLURL gives, hence, both, the local position and the sim which origin is used to define the coordinate. Such a description is distinct in the SL world:

http://slurl.com/secondlife/DaBoom/128/128/24.

All sims join to a *Grid*, the world of SL. Hence, for every sim exists a global, i.e. in the whole grid clear coordinate of the sim origin. So, one can determine that the sim *Da Boom* has the origin on the global position **<256000, 256000, 0>**. How to determine this? For example via the *Grid Survey* service, we just call this URL:

http://api.gridsurvey.com/simquery.php?region=DaBoom

The response contains two numbers: **x=1000**, **y=1000**. We multiple them by 256 (the width of a sim) and assume **z=0** to get the global position.

If we combine the local coordinate of a place in a sim, with the global coordinate of the sim origin, we get the global position of the point in the grid. So, the point given by position '**Da Boom/128/128/24**' is located on the position of **<256128, 256128, 24>**, relative to the grid origin, i.e. global.

The global position is distinct for the whole world of SL, it is also simple (a single vector value.) Map and RLV/RTS teleporters work for this reason with the global position.

The global position has one disadvantage: Sometimes, move sims across the grid. Moving a sim means changing the global coordinate of the sim origin. This also changes the global positions of all places on the sim. Teleporters cannot achieve the places after the move by using old coordinates and must recalculate them.

As a work-around, some teleporters calculate the target position for every teleport again and again, which stresses the sim. RTS (hence also TPMirror) avoids this permanent calculation. Instead, the teleporter caches the calculated value and uses it until the calculation was too long ago. Also, the device owner can delete the cached value making the teleporter calculate the position for the next teleport.

1.2 Installation and application

Figure 1.2: Installation

TPMirror is a device, using a single prim, formed to a square plane. The installation takes three steps:

1. Rezz the TPMirror and align at the workplace, for example a wall or on the floor, figure 1.2. After owner change, the device initializes automatically. This takes a few seconds.

2. Optionally you can configure the teleporter, section 1.3.

3. Now you can activate the device as described in section 1.2.1.

If you not know RTS or TPMirror yet, best is to activate the teleporter first, play with it and than continue with the configuration, changing destination and possibly the redesign.

The teleporter must not stay a square. Anything an avatar can bump into can become a teleporter: A tree, an invisible wall, a spider net, even a beach ball laying around. In every case, it is a device which reacts on collision and supports a single target. In the section 1.4 is explained how to extend the teleporter with selection of destinations.

1.2.1 Taking in operation

The TPMirror is ready to use straight out of the box, but it will teleport to the preconfigred destination. To change the destination, the configuration must be changed, section 1.3.4.

You can control the teleporter via owner menu, which is shown only to the owner. TPMirror uses a single prim, there is no extra menu button. To open the owner menu, please use a longclick.[7] The menu has four buttons:

- '**start**', '**stop**' to start or stop the teleporter.
- '**reset**': Resets all scripts and initializes the device. This also reads the configuration and sets up the device. As soon you changed the configuration, or the device, please remember to reset the teleporter.
- '**refresh**': This button removes the cached global coordinate (section 1.1.5) and enforces so the calculation for the next teleport.

1.2.2 Application

To travel with TPMirror, you must collide with it: E.g. walk into or over it. What happens next, depends on if it is in RLV or LM mode (RLV mode is on per default) and also on if the user uses RLV capable viewer and an active relay. The complete process is shown in figure 1.3.

1.2.2.1 In the RLV mode

Is the RLV mode active, the teleporter sends a teleport request naming the user. The request is received by the worn relay owned by the user.

Is the request accepted, the relay passes it over to the viewer. *The viewer performs the teleport.* The teleporter receives the positive response from the relay and (if configured so) gives the SLURL of the own position to the user via IM (the return-SLURL.)

Is the request declined, the teleporter can give the user only the LM or SLURL to the target place, if the LM mode is active.

Is the relay absent, switched off or the user took too long to answer the relay menu, the teleporter detects timeout and hangs out the teleport relay to the user (or another file set up in the configuration.)

[7] Please click the teleporter and hold the left mouse button one second.

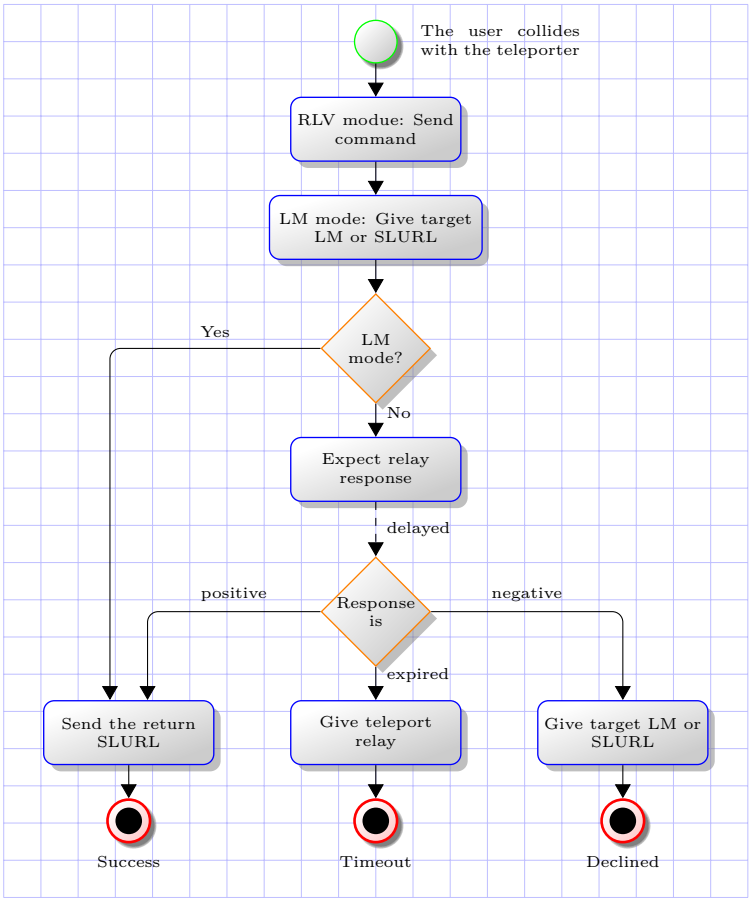

Figure 1.3: Teleport process

1.2.2.2 In the LM mode

Is the LM mode active, no matter if the RLV mode is active or not, the teleporter gives imediately the target LM or SLURL. The teleport is then valued as successed because the user can use the given LM or SLURL.

Is the RLV mode active at same time, the teleport request is sent via RLV anyway, and processed by the viewer eventually, but the teleporter doesn't await the relay response.

```
# TPMirror Configuration
[*MAIN]
user         = all

image        = 09BB628B-F97A-44FD-9503-6E4EC62DE59E
side         = all

title        = %n\n...step into and visit
color        = lavender
fadeout      = 0

mode         = RLV+LM
fail         = LM
return       = LM
delay        = 10

norelay      = TP Relay (r upper arm)
offset       = -3
fasttp       = OFF
refresh      = 24

format       = APP
msg:url      = This link brings you at target location:
msg:lm       = This landmark brings you at target location...
msg:return   = This link brings you back:
msg:norelay  = Please, use this item in order to use me...

[@sensor]
mode         = collide
interval     = 20

# The only target
[Welcome to Endora]
image        = 8cee492c-917e-341d-b3a9-63d83ed44d34
target       = Endora/127/169/42
```

Figure 1.4: Configuration of the TPMirror

1.3 Configuration

The configuration of TPMirror (like the RTS teleporters) is done by a notecard, which is inside the object inventory of the teleporter and has the name '**config**'. If delivered, TPMirror is configured by a text shown in 1.4.

For the detailed configuration info, please read the RTS documentation. Here we only discuss this configuration text and setting up the TPMirror specially.

1.3.1 Syntax

The char '**#**' starts the comment: The char and following text until the end of line is ignored. The text before the comment char is processed (except the spaces.) Comments are always closed by the end of line.

Empty lines have no relevance for TPMirror, they just improve the readability.[8]

Other lines have either section names or parameter values. A section name is always enclosed in square brakets. Section names are case-insensitive. Around the square brakets are only spaces allowed.

Params are separated from their values by assignment char. Both, param and value must be noted. Params are case-insensitive, some values are also, but not generally.

All param values belong to a last section that was started before the param line. The order in which the params are noted is not relevant.

1.3.2 Main section

```
[*MAIN]
```

The main section has the name '***main**' in any case. Upper case is preferred to emphasize its meaning: this section configures the teleporter generally.

[8] For performance reasons, please avoid empty lines and pure comment lines.

1.3.2.1 User parameter

```
user        = all
```

The param '**user**' defines who is allowed to use the teleporter, i.e. to travel with it. The value is a list of string flags, connected by '**+**'. Allowed are the flags '**owner**', '**!owner**', '**group**', '**!group**', '**all**' and '**!all**'.

Are the flags '**all**', '**!all**' used, others will be ignored. In the first case everyone can use the teleporter. In the second one – nobody. The flags '**owner**', '**!owner**' allow and forbid the device owner to use the teleporter. The flags '**group**' und '**!group**' allow or forbid every person from the device group to travel with the teleporter.

In the original configuration 1.4 everyone is allowed to use the teleporter, while the value '**group+!owner**' would mean: Everyone off the device group may travel with the teleporter but not the owner.

1.3.2.2 Picture parameters

```
image       = 09BB628B-F97A-44FD-9503-6E4EC62DE59E
side        = all
```

The param '**image**' defines the default picture. It is used if a selected target not defines an own. Allowed is either the UUID of the picture, or name of the image in object inventory. It must be full-perm there, otherwise its UUID can not be resolved.

The param '**side**' gives the face number which will show the target image. Possible values are '**all**', '**none**' and a number between 0 and 9. The value '**all**' means: the target picture is shown at all faces, and the value '**none**' suppress displaying the picture. If a number is given, this is the number of the face that takes the picture.

1.3.2.3 Title parameters

```
title       = %n\n...step into and visit
color       = lavender
fadeout     = 0
```

The param '**title**' defines the title (hover text), the value supports placeholders:

- The placeholders '**\n**' and '**\t**' are replaced by the line break and tabulator.
- The placeholders '**%n**' and '**%s**' are replaced by the destination name and name of the defining landmark or target sim respectively.

- The placeholders '**%p**' and '**%c**' are replaced by the number of the selected target in the list (1 means first target) and amount of targets in the list.

To hide the title, just remove the title param, or use '**-**' as value.

The param '**color**' defines the color of shown hover text. The param has also alias '**colour**'. The value is basically a vector like in example above. But some popular names can be used by color name instead of vector. Names, the RTS understands are given in the RTS documentaion. The color name is case-insensitive, '**White**' means the same as '**WHITE**' or '**white**'.

The param '**fadeout**' defines finally, how long the title is visible before it disappears. The time is set in seconds, the timer starts after the title is changed. 0 means the title remains permanently.

1.3.2.4 Teleport modes

```
mode        = RLV+LM
fail        = LM
return      = LM
delay       = 10
```

The value of params '**mode**', '**fail**' and '**return**' is a combination of string flags '**RLV**', '**LM**', '**N**' and '**OFF**'. The flags are combined via '**+**' and define the behaviour of the teleporter. The flags work additively: The value '**LM+OFF**' means the same as '**LM**' or '**lm**', flag names are case-insensitive.

Is the flag '**RLV**' used, the teleporter access the RLV interface of the viewer. The flag '**LM**' allows handing out the target LM or SLURL, the teleporter becomes a landmark distributor. The flags '**N**' and '**OFF**' deactivate the mode using this flag.

The param '**mode**' defines how the teleport works, if via RLV or by handing out the LM or SLURL. Allowed value flags are '**RLV**' and '**LM**'. If the flag '**LM**' is set, the teleport is taken as succeed, since the agent can use given LM or SLURL any time.

The param '**fail**' defines what to do if the teleport fails: The teleporter can give the target LM or do nothing. Allowed are the values '**LM**', '**N**' and '**OFF**'.

The param '**return**' defines what to do if the teleport succeed: If allowed, the teleporter will send the SLURL for teleporter's position, so the traveller can return. Allowed values are '**LM**', '**N**' and '**OFF**'.

The param '**delay**' defines how long the teleporter has to await the response of the user's relay until teleport failure is detected and the '**fail**' mode is active. The time is given in seconds, minimum is 5. This time covers the delays due lag, as well time the user took to react on their relay, figure 1.3.

Settings

The params '**mode**', '**fail**' and '**return**' allow just two reasonable values: '**LM**' or '**OFF**': The value '**N**' equals '**OFF**' and the value '**N+LM**' equals '**LM**'.

For the param '**mode**' these values are reasonable:

- '**RLV**' - Teleport attempt via RLV interface, further actions according to the params '**fail**' and '**return**'.
- '**LM**' - Don't use RLV, instead, give the LM or SLURL to the destination, further actions according to the '**return**' param.
- '**RLV+LM**' - Teleport attempt via RLV and also the target LM/SLURL is provided. Further actions as defined by the '**return**' param.
  This setting suits both, users with and without RLV, especially if the destinations are defined by SLURL: Users with RLV are teleported automatically, and users without RLV can use the SLURL from the chat history.

1.3.2.5 Teleport relay

```
norelay     = TP Relay (r upper arm)
```

The param '**norelay**' defines a file to be given if teleport failed because the user has no relay. The file is addressed by 'teleport relay' in the figure 1.3, the timeout branch.

This way the teleporter can give out a notecard with further information. TPMirror is configured by giving out the teleport relay, since it allows to use the teleporter even without a RLV-capable viewer.

1.3.2.6 Offset parameter

```
offset = -3          # means...
offset = <-3, 0, 0>
```

The param '**offset**' gives the return position (rezzing position for returning avatars) relative to the teleporter. You can define it by a single number or by a vector, while the single number equals the vector with that number as x. Both definitions in the example above are equal.

What means this value? If the offset is zero, the return position is the position of the teleporter itself, than avatar using the return SLURL will rez at the position of the teleporter.

For TPMirror it is very awkward. Because of *returncollision* with the teleporter, the avatar is teleported repeatelly to the destination, and thus can't return anymore. The offset setting moves the return position into a place safe to rezz.

The offset vector is always the vector in the global xy plane around the system prim. If the system prim is not rotated, than the vector `<3, 0, 0>` aims a point 3 meters towards the positive x axis of the system prim. If we rotate the prim around the z axis, the return point follows the rotation. But it ignores the rotations around x or y axis.

This way we can use a teleporter placed on a wall and configure the offset vector in a way, the return position is 3 meters in front of the teleporter. Than we can place the teleporter on any other wall and the return position remains in front of it. But if we turn the device to place is flat on the floor, the return position not moves above the teleporter, it remains 3 meters aside.

1.3.2.7 FastTP mode

```
fasttp      = OFF
```

This param is included into the TPMirror configuration for compatibility reasons. The param value is one of four flags: '`Y`' and '`ON`' switch the mode on and flags '`N`' and '`OFF`' switch the mode off. Since TPMirror is a teleporter for a single target, this mode has no relevance.

In a teleporter for multiple targets and with a possibility to select one of them in any way, you can switch the FastTP mode on. If active, the teleport starts immediately as soon a destination is selected in any way. If the mode is inactive, you can scroll through targets and trigger the teleport after you chosen the destination.

1.3.2.8 Target refreshing

```
refresh     = 24
```

For teleport, TPMirror needs a global coordinate of the target position. Sims move sometimes, which change this value. The global position needs a recalculation and that means delays and server load.

To lower the server load, the resolved coordinate is cached. The param '**`refresh`**' states how long the saved coordinate remains valid. The value is in hours, the minimum is 6.

The position is not refreshed permanently but due a teleport to an expired position. Thus, it is advisable to use a value of one day to one week here, i.e. 24 to 168 hours. Also, the 'refresh' button in owner menu sets the targets as expired manually.

1.3.2.9 Message format

```
format = APP
```

The param '**format**' defines the format for sent SLURL. Here you can define one of four values, '**MAPS**', '**SLURL**', '**APP**' and a freetext.

'**SLURL**': Send a *SLURL*.[9] It is classical, all viewers understand the link and open the landmark window if it is clicked in chat history.

'**MAPS**': Send a *MapURL*.[10] It was introduced with the version 1.23 of SL viewer. Older viewers open the web browser instead a landmark window.

APP: Send a *teleport link*.[11]. This string enforces an immediate teleport if clicked in chat history, without to open the landmark window.

A freetext is a text with placeholders `%s`, `%x`, `%y`, `%z`, replaced by the sim name, and local coordinates on the sim, respectivelly. For example the format string '`%s (%x, %y, %z)`' produces a text like 'Endora (123, 234, 345)'.

Note: You can emulate the flags '**MAPS**', '**SLURL**' and '**APP**' by using the freetext format, for example the format '`secondlife:///app/teleport/%s/%x/%y/%z`' produces the same teleport link as for using the '**APP**' flag (but takes more processor time to calculate).

1.3.2.10 Message parameters

```
msg:url     = This link brings you at target location:
msg:lm      = This landmark brings you at target location...
msg:return  = This link brings you back:
msg:norelay = Please, use this item in order to use me...
```

The params '**msg:...**' take text values which introduce given files and SLURLs. They also accept placeholders, section 1.3.2.3.

The param '**msg:url**' saves the text to introduce the target SLURL. The message for traveller is a combination of this text with the SLURL, sent in a single IM.

The param '**msg:lm**' saves the text to be sent if a target LM is given. The message and the landmark are sent separately.

The param '**msg:return**' saves the text to introduce the return SLURL.

The param '**msg:norelay**' saves the text sent if the teleporter detects missing of the relay and sends file noted under '**norelay**' param.

[9] A link like http://slurl.com/secondlife/Endora/123/234/345
[10] A link like http://maps.secondlife.com/secondlife/Endora/123/234/345
[11] A string like secondlife:///app/teleport/Endora/123/234/345

1.3.3 Sensor section

```
[@sensor]
```

The section with name '**@sensor**' is meant for a sensor script, which again is detecting a potential user of the teleporter. The sensor supports different detection modes, TPMirror uses only the collision mode, hence we describe this mode only.

1.3.3.1 Sensor mode

```
mode        = collide
```

The param **mode** chooses the sensor mode. The value can be one of five: **touch**, **collide**, **scan**, **sit** and **chat** activate the touch, collision, radar, sit and chat modes respectivelly. However, the TPMirror uses the collistion mode only, hence please leave this param unchanged.

1.3.3.2 Scan interval

```
interval = 20
```

The '**interval**' param gives a value in seconds and states how long a sensored avatar remains in avatar filter and not reported again. Otherwise a simple run over a carpet would create a bunch of triggered teleports.

The collisiion mode knows only this param, hence other possible params are not relevant for TPMirror and thus not described in this documentation. RTS documentation provides a complete documentation of the sensor section.

1.3.4 Target sections

A target section provides three Entries: The *target name*, the *target image* and the *target definition*. Since TPMirror has no possibility to select targets, only the first target is selected permanently, even if more than one target section is given. Hence, only a single section makes sense for TPMirror, but you can define up to 80.

1.3.4.1 Target name

```
[Welcome to Endora]
```

The target name is the name of the section. It must start by a letter or underscore and can be off multiple words. It must be distinct and 32 letters at last. The target name can be injected via placeholder '**%n**' into the title (hover text) and sent messages.

1.3.4.2 Target image

```
image       = 8cee492c-917e-341d-b3a9-63d83ed44d34
```

The param is used as target picture. The param '**image**' is optional, if omitted, the image defined in main section is used instead. The value can be an UUID or an image in object inventory, but than it must be fullperm.

1.3.4.3 Target definition

```
target      = Endora/127/169/42
```

The param '**target**' is required to localise the target, the value can be a LM or a SLURL.

If this is a LM, it muss be in in the object inventory. In this case the LM is given to the user if LM mode is active, section 1.3.2.4. This way they can visit the place later by using the landmark.

If the target is defined by a SLURL, user's inventory is not overfilled by sent LMs, but the traveller can visit the place again only via the teleport history or manually taken LMs. The SLURL can be given by three manners:

- SLURL: http://slurl.com/secondlife/Endora/127/169/42
- MapURL: http://maps.secondlife.com/secondlife/Endora/127/169/42
- RawURL: Endora/127/169/42

RawURL starts with the sim name and thus it is universal. **Note**: The RTS documentation may call this form 'Stripped SLURL', the actual name was met later.

1.4 Extension

TPMirror is a RTS teleporter, but it uses not all possibilities RTS offers: TPMirror was originally developed as a teleporter for a single destination. RTS allows to define multiple destinations, but for traditional reasons, TPMirror offers no possibility to select one of them.

In this section we extend the teleporter by the missing ability. We don't have to write or change scripts. All we'll do, is just linking properly named prims to the teleporter. The result is a teleporter called TPArch, figure 1.5 (for better view, the picture is taken without hover text.)

Figure 1.5: TPArch (PortalFrame installation)

This section is written for purchaser of TPMirror – who can travel grid-wide, very soon needs a teleport device for many destination around the grid. But also for purchaser of the RTS package, as a demonstration of RTS and tutorial for working with this framework.

1.4.1 TPMirror architecture

First, we take a brief look at the design and working way of TPMirror. This may help to understand how RTS teleporters work. Detailed information is given in the RTS documentation, here we really take an overview.

1.4.1.1 Files

This files are instaled into TPMirror:

Script '**.core**': The core script, controls other scripts, performs teleports and enforces sending of messages and files. It also opens the owner menu and runs a number of core services.

Script '**.config**': Configuration script. Reads the '**config**' notecard and shares the read data over other scripts. Other script have no access to this notecard, which simplifies the script code.

Script '**.sender**': Sender script. Sends files like LMs or the teleport relay and messages vie IM as well, to the user and device owner, like error messages produced while configuration and run time.

Skript '**.sensor**': Sensor skript, used to detect potential users. The script reports every detected avatar. The core script receives the reports, decides if the avatars are allowed users and teleports them in positive case.

Script '**.targets**': Target list. Can take up to 80 targets given in the configuration. Manages the target data: Stores the specified data, resolves the global position, provides data for a selected target.

Object '**TP Relay (r upper arm)**': The teleport relay, as given in the configuration, param '**norelay**' (section 1.3.2.5.)

1.4.1.2 Core services

RTS separates the functionality of the teleporter in single services. The services are executed by the core script, but the execution can be taken over by so-called driver scripts. TPMirror uses this services:

The *text service* is responsible for displaying the title (hover text) with respect of title parmeters, section 1.3.2.3.

The *message service* sends script and error messages to the owner via IM, the messages are passed over to the sender script, which than resends them.

The *image service* displays the target picture in respect of picture parameters, expecially the '**side**' param, section 1.3.2.2.

The *menu service* opens the owner menu if the teleporter is long-clicked.

1.4.2 Design

Back to the extension challenge. The teleporter is a linkset while the prims have roughly four functions. The *base prim* runs all scripts and is in fact the unchanged TPMirror.

The *thumbnail prims* are used to select destinations. RTS offers also other ways to select the teleport destination but we not use them. The *menu button* makes opening the owner menu much easier than the longclick.

Additionally we build a frame from stone, which allows us optically to place the teleporter open-air.

1.4.2.1 Base prim

At first, we need a base teleporter: We rezz TPMirror and set the prim size to $(0.02, 1.5, 2)m$ and zero rotation $(0, 0, 0)°$[12], figure 1.6. The prim we rename now – to '*TPArch*'.

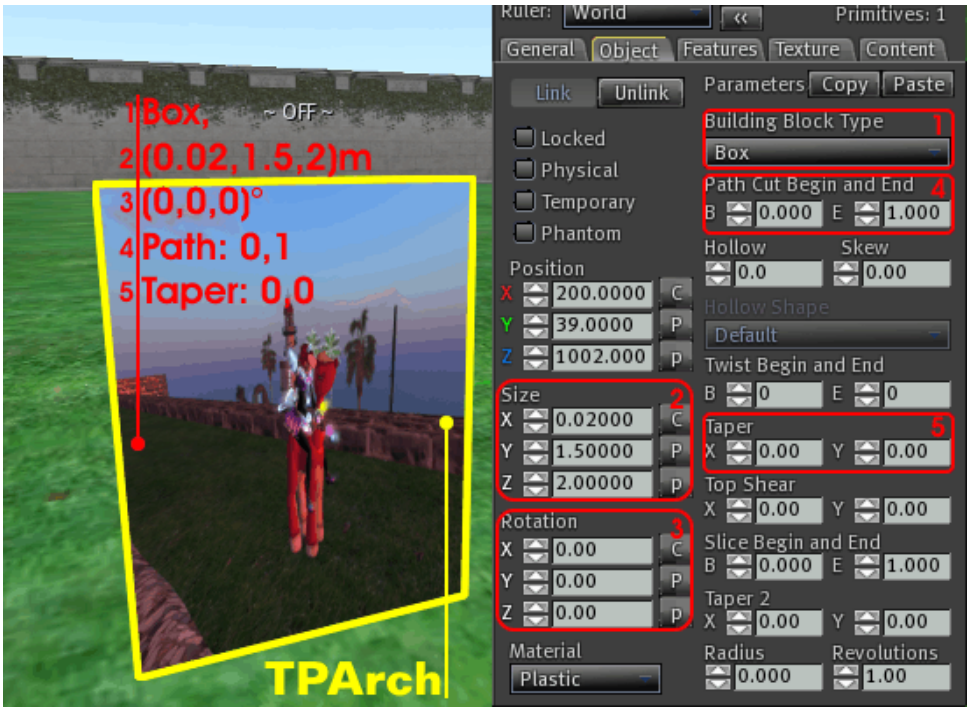

Figure 1.6: Rezzed base (with legend)

The picture also introduces the legend for further pictures: Yellow are the prim names and status in the linkset, red – their geometrical properties and relative

[12]The size and rotation we denote as a vector (x, y, z)

position. The prim type is given directly, position and rotation denoted by units at the end. Further properties are introduced by prefix. Default values are omitted if possible (here is only size non-default.)

1.4.2.2 Thumbnail prims

RTS supports up to 20 tumbnail prims but we use just 6. The prims display destination pictures, clicking one selects shown destination. The thumb prims are reason of the teleporter name – they are aligned arch-like above the base prim.

Rather to say the thumbs are sectors of a half-circle above the base. One can define their geometrical data by using a calculator, but we use a building trick for that.

We rezz two boxes: a thumb prim in the size $(0.02, 0.38, 0.35)m$ and the help prim in the size $(0.05, 0.05, 0.25)m$, both in zero rotation and distance of $55cm$. Now we taper the preview prim and link to the help prim making this root. The prim pair we justify $1.004m$ above the base, figure 1.7.

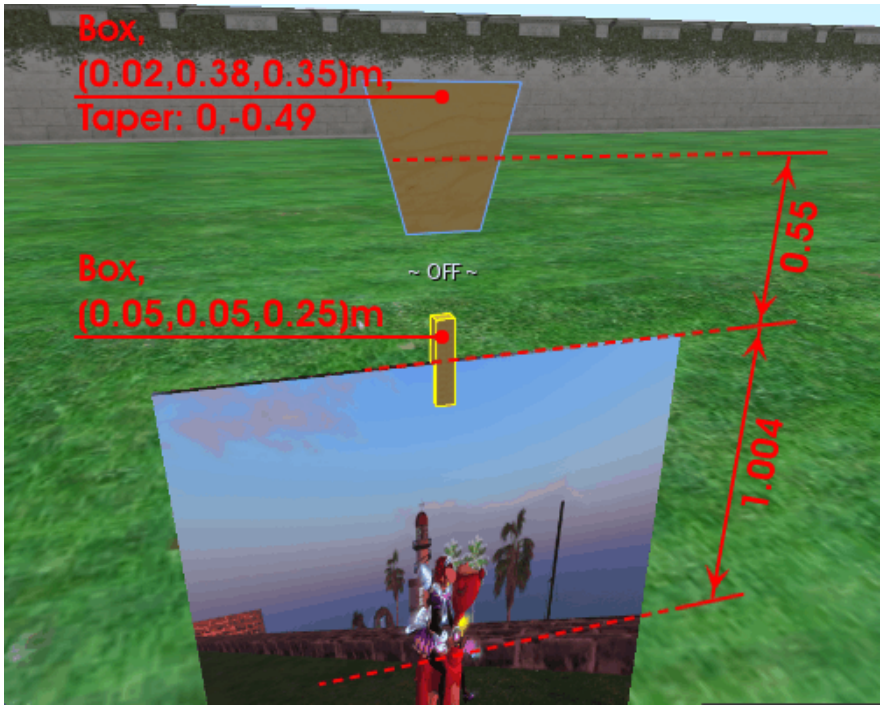

Figure 1.7: Thumb prim, help construction

Now we copy the prim pair 5 times at front: Edit, press and hold the shift key and pull on the green arrow, figure 1.8 left.

We than fold the copies (replace the y position by those of the base) and rotate every copy in a distance of 30°, figure 1.8 right. The building trick, mentoined above is to move the thumb prims into position just by rotating the help prims.

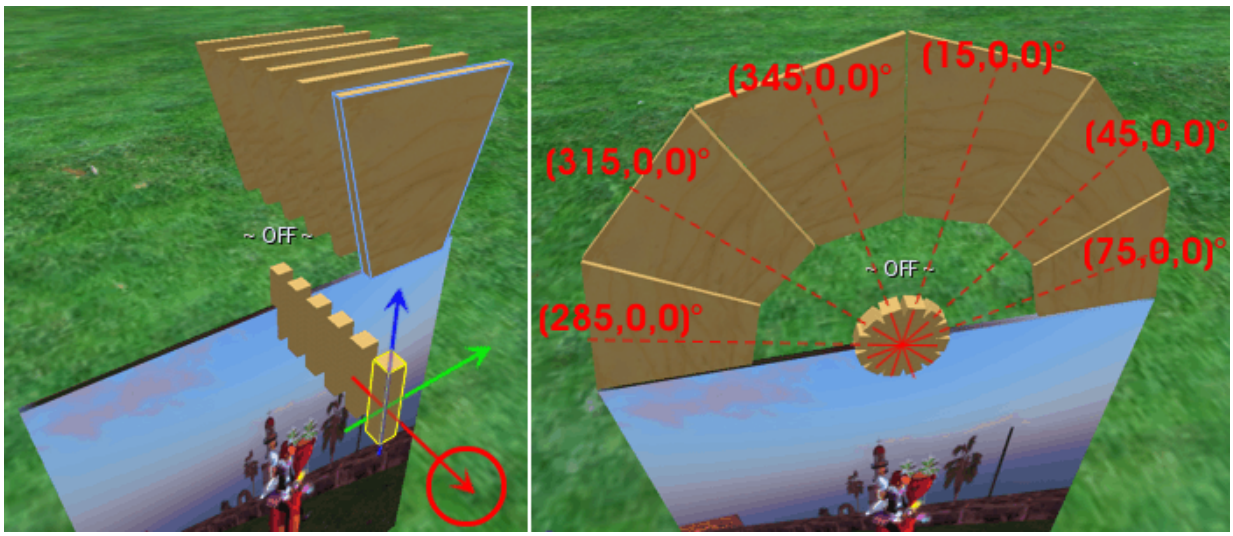

Figure 1.8: Thumb prim, copied and aligned

Now we can unlink the prim pairs (break links) and delete all help prims except one, it will be the later menu button. In final step we rename the thumbs counterclockwise from top by '`thumb 1`' through '`thumb 6`', figure 1.9. Right than the prims are accepted as thumbnail prims.

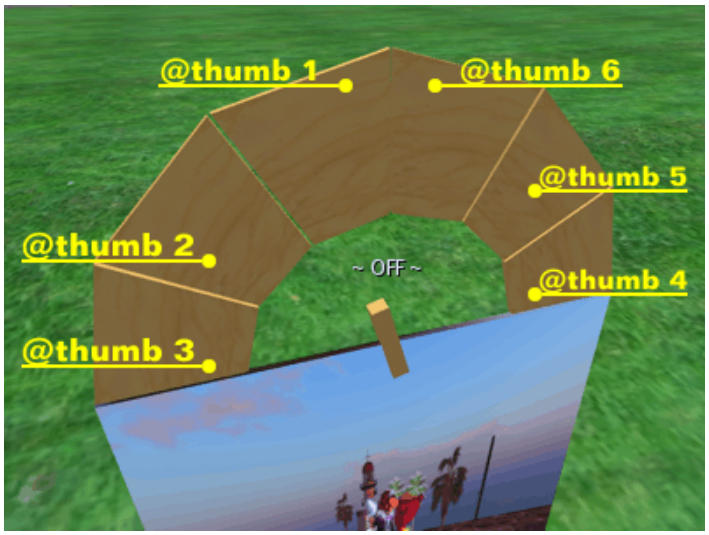

Figure 1.9: Names of thumb prims

1.4.2.3 Menu button

A menu button is any prim named '`.menu`'. For it, we use the remained help prim and fill optically the space to the base – we glass this place.

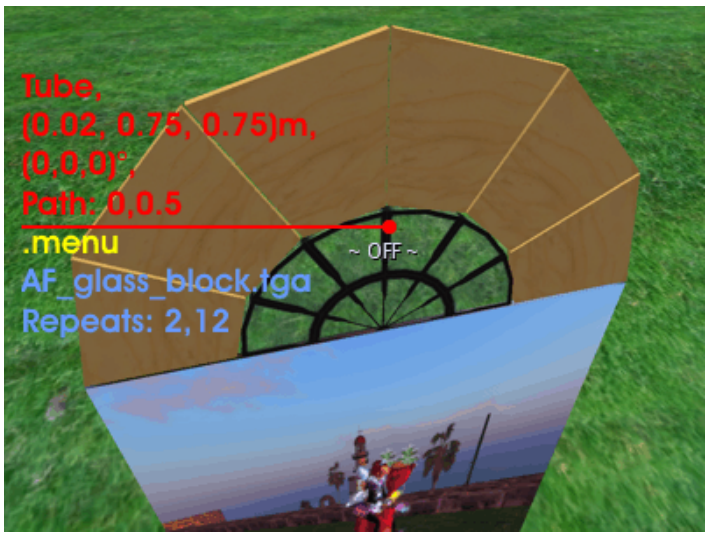

Figure 1.10: Menu button, reshaped, renamed and textured

A half circle with a radial mosaic pattern is a good idea. We could use a cylinder with an appropriate texture, but we use a trick to bring a glass block texture radially in a half circle: We use a flat closed tube.

The help prim gets the name '**`.menu`**', becomes a tube in the size $(0.02, 0.75, 0.75)m$, zero rotation and path cut of $(0, 0.5)$. As texture we use '**`AF_glass_block.tga`**' from the texture viewer (RTS package) and set the repeats to $(2, 12)$, figure 1.10.

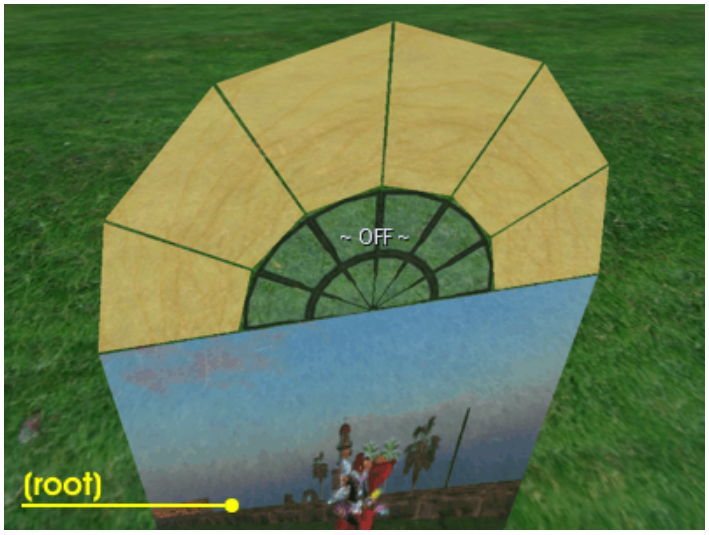

Figure 1.11: Finalisation

1.4.2.4 Finalisation

Now we link all prims in a way, the base prim is root. In one step we can now polish and brighten the teleporter. To create an illusion of a filmy construction, we take the teleporter its depth: We set all side faces invisible.

The easiest way to do so, is to set the whole linkset invisible, activate the face selection, select every front and back side, than set them half-transparent. The result is shown in the figure 1.11.

1.4.2.5 Placement

Only configuration remains, than we can install the teleporter in a way through, or as a window. But not realy outside: Without a frame the teleporter looks irreal. We build one, so we can install the teleporter at wayside or similar places.

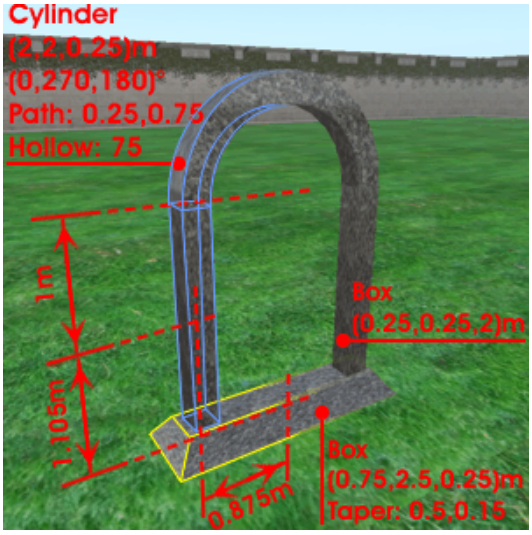

Figure 1.12: Portal frame

The simpliest variant is made from four prims with a marmor texture, figure 1.12, but other solutions are also possible. The frame we can link to the teleporter (than please reset) but must not: Than we can also install the teleporter pure if needs.

```
# TPArch Configuration
[*MAIN]
title       = Destination %p of %c\n%n\n...step into and visit
# --- The rest like for TPMirror --------------------------

[@sensor]
mode        = collide
interval    = 20

# Target list
[Linden Playground (M)]
image       = 1b5e8133-3fb8-5488-c498-e63ec41b2010 # Da Boom (143, 148, 41)
target      = Da Boom/128/128/35

[Protected Ocean (G)]
image       = 16c62e5b-0b9c-9d78-1aba-e1f45af68bdb # Pravatch (230, 229, 2)
target      = Pravatch/221/227/3

[MYST Island (G)]
image       = 5a6b672a-6c6a-d2a1-a020-04217cdaf0da # Age of Myst (171, 46, 33)
target      = Age of Myst/175/14/24

[Great Wall Of China (G)]
image       = 10594b0c-2ae4-3f7b-052b-a77a098fefcd # China Sichuan (130, 10, 24)
target      = China Sichuan/133/16/24

[Museum of Illusions & Magic (G)]
image       = 76999bdb-11ba-c08d-4d5a-24ea1bed073a # AmazingIllusionMuseumEI
target      = Enchantment Island/59/120/45

#This target was in TPMirror configuration
[Welcome to Endora (A)]
image       = 8cee492c-917e-341d-b3a9-63d83ed44d34
target      = Endora/127/169/42

[The Grand Canyon (M)]
image       = d835568f-fb32-5fe4-2ba1-99191261eced # Grand Canyon (106, 173, 110)
target      = Grand Canyon/89/186/110

[Happy Mood (G)]
image       = 525ea54c-49dd-bc1d-327e-c64cd90610db # HappyMood (107, 144, 79)
target      = HappyMood/109/144/80

# --- 3 more target sections follow ---------------------
```

Figure 1.13: Configuration of TPArch

1.4.3 Configuration

The design is complete. Last step is extending the configuration. To do so, we edit the teleporter and the '**config**' notecard inside.

Here we simply change the title (to reflect multiple targets) and replace a single target section by a list of sections – each one for every teleport destination. We can enter here up to 80 of them. The TPArch is already configured by 11 example destinations, figure 1.13.

The confguration gives some tips by handling the destination list:

- Target names include the sim rating: (G, M, A.) The target name is inserted into the title via placeholder '**%n**', this allows us to see the sim rating before we teleport there.
- Target pictures are given via UUID. This saves up installing of the pictures and synchronization their names with the setting in the notecard.
- Comment sign after the UUID allows us to name the picture. The name is not important for the teleporter itself but allows to find the picture in our inventory later.
- The target place is given by RawURL. This way, while preparing the destinations list, we must not take a LM, but can use the teleport history or simply take the sim name and position from the taken screen shot.

After the configuration text is extended and saved, we have to reset the teleporter: Click on the menu button, than '**reset**' button in the opened menu. After the configuration is complete we can start the teleporter via the owner menu again. The device is ready to use, as shown in figure 1.5.

1.4.4 Extended architekture

We have no script changed, added or removed in the original teleporter, so why it works as it works? The answer is in the architecture and interfaces of RTS. This section should explain how they work together in the context of TPArch. For more information, please use the RTS documentation.

The targetlist script '**.targets**' manages the destinations given in the configuration. If one is selected in any way, the script provides a set of data for the target, including the target picture and pictures before and after the selected in the list.

As soon the sensor script detects a collision, the script reports the avatar via a message, which again triggers the teleport. The core script receives the message and checks if the avatar is allowed to be teleported. If so, the teleport is started.

By adding and naming of prims, we activated two more core services, the *thumb service* and the *button service*.

The name prefix '**@thumb**' denotes prims as thumbnails. Than they fall under control of the *thumb service*. If a target is selected, this service displays the pictures of the neighbour-targets on the thumbnail prims. If one of them is clicked, the service produces a control message, which moves the target selection in a way the target is selected that was shown on the clicked prim.

The prim with name '**.menu**' is meant to be button and covered by the *button service*. The service produces a control message, if the prims is clicked. If the button name starts by a dot, the produced message is meant for the core script. The menu button has the name '**.menu**', this produces a control message opening the owner menu.

Hence, TPArch uses only one possibility to select the destination. But the targetlist script offers a few more: The exact target number to select (button prefix '**pos**'), selection of the first or last destination (buttons '**first**', '**last**'), Random target selection (button '**random**') and a free text search.

That's all. I wish you much fun and success with experimenting with the teleporter and RTS.

2 Deutsch

2.1 Einleitung

TPMirror ist ein Teleportgerät, das mittels der RTV Technologie gridweit funktioniert. TPMirror war bereits als RLV Teleporter entwickelt, jetzt, in der Version 2, wurde er mit der RTS Technologie neu aufgebaut. RTS steht für '*RLV Teleporter System*', es ist ein Framework zum Bau von RLV Teleportern. TPMirror erbt auf diese Weise viele Features von RTS.

Was ist an TPMirror anders? TPMirror ist ein gridweiter Teleporter. Er erreicht jeden Ort, der via einer LM erreichbar ist. Der Avatar wird nicht physikalisch bewegt, der Teleport bricht nicht ab, wenn die Zielposition auf der anderen Sim oder Kontinent liegt. TPMirror öffnet auch keine Weltkarte, wie Kartenteleporter das tun. Statdessen überreicht TPMirror die Zielkoordinate an den Viewer und löst den unmittelbaren Teleport aus.

Man kann TPMirror als Tunneleingang vorstellen. Man kann nicht durch den Tunnel sehen, man sieht nur den Eingang, vielleicht ein Bild auf der Wand or am Boden. Doch sobald man den Eingang passiert, bringt der Tunnel einen zum anderen Ende – dem Zielort.

Grundsätzlich, erfordern RLV Teleporter, dass der Viewer eine spezielle RLV Schnittstelle unterstützen, um die Zielkoordinate zu übergeben. TPMirror funktioniert auch auf diese Weise. Da aber nicht jeder Viewer diese Schnittstelle anbietet, funktioniert TPMirror auch ohne sie: Man kann zum Einem den RLV Support ausschalten und den Teleporter dadurch zu einem Landmarkverteiler machen.

Zum Anderen macht das mitgelieferte TP Relay jeden RLV Teleporter (und auch den TPMirror mit aktiven RLV Support) zu einem Kartenteleporter. Dies erlaubt es, RLV Teleporter, TPMirror und auch Kartenteleporter auf dieselbe Weise und mit jedem Viewer zu benutzen. Aus diesem Grund überreicht TPMirror dieses Relay an den Benutzer, sofern er kein RLV oder kompatible Viewer einsetzt.

2.1.1 Zielgruppe

Für wen ist TPMirror geeignet? Nun, er erlaubt zum Ort auf derselben Sim zu springen, ob eine Skybox, Club oder die nächste Etage im Haus. Das sind aber lokale Teleports und es gibt einfachere und leichtere Lösungen für diese Aufgabe.

Möchte man aber jeden Ort auf der Grid mit nur einem Klick erreichen, als ob das ein Nebenraum wäre, möchte man Orte verbinden als wären das Räume im Haus, unabhängig ob die Orte auf derselben Sim befinden oder über die Grid verteilt, ob sie einem gehören oder den Partnern oder gar Fremden, dann ist TPMirror eines der Geräte der Wahl.

RTS wurde entwickelt, um mehre Teleportziele zu verwalten. Aus traditionällen Gründen unterstützt TPMirror aber nur ein Teleportziel. Mit ein wenig Aufwand kann man ihn aber um die Mehrzielfähigkeit erweitern. Im Paket ist ein schrittweiser Tutorial enthalten wo das behandelt wird, so wie das Ergebnis: TPArch ist ein RTS Teleporter, für bis zu 80 Teleportziele.

2.1.2 Struktur

Die Dokumentstruktur ist die folgende: Die *Einleitung* stellt den Inhalt des Produktpakets vor und wie Sie das Produktupdate erhalten. Es wird erklärt, was RLV ist, warum diese Schnittstelle benötigt wird, welche Viewer sie unterstützen und wie man TPMirror ohne dieser Viewer nutzen kann.

Der Abschnitt 2.2 behandelt die *Installation* des Teleporters und seinen Einsatz. Seine *Konfiguration* wird im Abschnitt 2.3 erklärt.

Anschließend folgt der Abschnitt 2.4 mit einer kurzen Einführung in die Arbeitsweise des Teleporters, und einer schrittweisen Anleitung zum Erweitern von TPMirror zu TPArch, einem Teleporter, der mit einer Liste von Zielen arbeitet, und somit die Mehrzielfähigkeit von RTS ausnutzt.

TPMirror ist ein RTS Teleporter. Deshalb ist die RTS Dokumentation für Hintergrundinformation wichtig, die über die beiden letzten Abschnitte hinaus geht. Auch in diesen Abschnitten wird darauf verwiesen. Die Dokumentation steht als eine PDF Datei zur Verfügung, verlinkt auf der RTS Produktseite.[1]

2.1.3 Paketinhalt

Inhalt des Produktpakets ist in der Abbildung 2.1 dargestellt. Am wichtigsten sind diese Dateien:

[1] https://marketplace.secondlife.com/p/RLV-Teleporter-System-private-edition/2944310

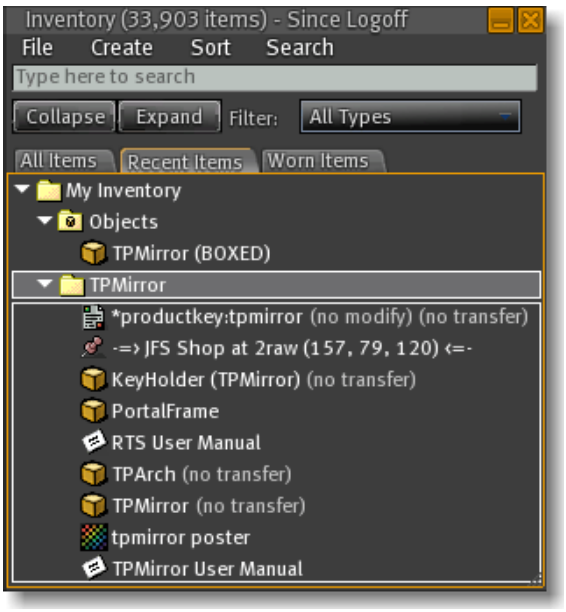

Figure 2.1: TPMirror, Paketinhalt

- '**TPMirror**', '**TPArch**': Der Teleporter und seine Erweiterung. Beide sind einsatzfertig, benötigen aber eine Rekonfiguration, etwa zur Einstellung von Teleportziele.
- '**PortalFrame**' erlaubt eine freie Installation der Teleporter.
- '**KeyHolder**' ist ein Gerät zum manuellen Anfordern von Updates und Nachlieferungen, Abschnitt 2.1.3.2.
- '***productkey**' – Dieser Skript arbeitet im KeyHolder.
- '**TPMirror User Manual**' und '**RTS User Manual**' – Hilfsmittel für den Teleporter und das Kernsystem. Beide enthalten weitere Notekarten mit Teildokumentation und in der deutschen Sprache.

2.1.3.1 Auspacken

Auspacken können Sie das Paket auf traditionelle Weise und automatisch: Im ersten Fall rezzen Sie es am Boden, öffnen und kopieren Sie den Paketinhalt. Sie erhalten dann ein Verzeichnis namens '**TPMirror (BOXED)**'.

Alternativ können Sie das Paket sich selbst öffnen lassen. Dazu rezzen Sie es am Boden oder tragen in der Hand (falls rezzen nicht erlaubt ist.) Sind Sie auf einem Ort mit erlaubter Skriptausführung, öffnet sich das Paketmenü. Ignorieren Sie es, können Sie das Paket anklicken und erhalten Sie das Menü erneut.

In diesem Menü brauchen Sie den Button '**Open**'. Nach dem Klick darauf übergibt die Box das Produktverzeichnis '**TPMirror**'. Anschließend öffnet sich das Paketmenü erneut, wo Sie den '**Remove**' Button anklicken können. Er zerstört die Box, falls sie gerezzt ist oder legt sie ab, wenn getragen.

Bitte bewahren Sie die Produktbox als Sicherheitskopie auf, oder wenigstens den Skript '***productkey**', er ist zum Anfordern von Produktupdates erforderlich. Es steht auch ein Onlinetutorial zum automatischen Entpacken von JFS Produktboxen zur Verfügung.[2]

2.1.3.2 Update

TPMirror befindet sich derzeit in einem Verkaufssystem, das eine automatische Auslieferung von Updates erlaubt. Möchten Sie dagegen Updates manuell anfordern, haben Sie zwei Möglichkeiten: Via der Produktbox selbst oder des mitgelieferten KeyHolders.

In beiden Fällen wird der '***productkey**' Skript aktiv, der Updates vom Updateorb anfordert. Der Orb muss dabei in der Nähe sein, deshalb müssen Sie dafür erst einen der JFS Shops aufsuchen, wo der UpdateOrb installiert ist. Die Liste solcher Shops finden Sie in einem dafür eingerichteten Blogartikel.[3]

Anstatt der *Produktbox* können Sie auch jedes Prim nehmen, in das der Skript '***productkey**' installiert ist. So fordern Sie Produktupdates damit an:

Kommen Sie näher als 10m an den Updateorb. Rezzen Sie nun die Produktbox, das Menü erscheint. Wenn nicht, bitte die Box anklicken. In diesem Menü klicken Sie bitte den '**Update**' Button. Der Updateorb wird das Update ausliefern. Im darauf geöffneten Menü klicken Sie bitte den '**Remove**' Button, damit zerstört sich die gerezzte Box automatisch.

Der *KeyHolder* ist ein armbandähnliches Gerät, das Schlüssel für bis zu 12 Produkte verwaltet und das Herumschleppen von Produktboxen überflüssig macht. Der Skript '***productkey**' muss ins Gerät installiert sein, damit er arbeitet.

Im mitgelieferten KeyHolder ist der Produktschlüssel bereits installiert. Um damit die Produktupdates anzufordern, verfahren Sie bitte wie folgt:

Kommen Sie ebenfalls in die Nähe des Updateorbs. Tragen Sie den Keyholder und klicken den an, ein Menü erscheint. Bitte wählen Sie das RTS Produkt aus, falls nicht bereits ausgewählt. Nach einem Klick auf den '**Update**' Button liefert der UpdateOrb das Update aus. Sie können den KeyHolder wieder ablegen.

[2] http://jennassl.blogspot.com/2011/11/unpacking-jfs-boxes.html
[3] http://jennassl.blogspot.com/2010/05/actual-shop-list.html

In beiden Fällen ist das Update eine Nachlieferung des aktuellen Produktpakets. Es steht ebenfalls ein Onlinetutorial zum Updaten von JFS Produkten zur Verfügung.[4]

2.1.4 Warum RLV Teleporter

RLV war ursprünglich ein spezieller Viewer, der mit dem Ziel entwickelt wurde, Immersion (das Gefühl, im Spiel zu sein) des Benutzers zu unterstützen. Hierfür bietet der Viewer eine besondere Schnittstelle, die den Skripten erlaubt, den Viewer eingeschränkt zu kontrollieren.

Mithilfe dieser Schnittstelle können Skripte einige Aktionen des Viewers auslösen, die sonst nur der Viewerbenutzer ausführen kann, wie etwa vom gesessenen Objekt aufstehen oder zu einem Ort teleportieren. Die Liste von ausführbaren Aktionen ist relativ umfangreich, wobei TPMirror nur eine benötigt: Er versendet nur den Teleportbefehl.

Erreicht dieser Befehl den Viewer, setzt er sich sofort in Bewegung. Der Benutzer sieht nur den Ladebalken und kurze Zeit später den Avatar am Zielort stehen. Genau so, wie beim Benutzen einer Landmarke oder Weltkarte, nur ganz ohne eines geöffneten Landmarkenfenster oder Kartenfenster.

Genau darin liegt der Vorteil von RTS Teleportern (oder allgemein der RLV Teleportern) gegenüber Landmarkverteilern oder Kartenteleportern: Der Benutzer wird aus der Spielsituation durch das geöffnete Landmark- oder Kartenfenster nicht herausgerissen.

Ursprünglich wurde die RLV Schnittstelle nur durch den RLV implementiert, jedoch seit die Entwicklerin des Viewers sie veröffentlicht hat[5], konnten andere Hersteller ihre Objekte RLV-fähig machen und so die Verbreitung des RLV fördern. Inzwischen steht diese Schnittstelle für alle Viewerentwickler als *RLVa* zur Verfügung[6] und wird von vielen Drittanbieterviewern implementiert.

2.1.4.1 Ohne RLV

Ein Nachteil von RLV Teleportern ist, dass nicht jeder Viewer diese Schnittstelle unterstützt. Benutzer eines Viewers ohne RLV Schnittstelle kann solche Teleporter nicht auf die beschriebene Weise nutzen. Vor allem der offizielle Viewer kennt die RLV Schnittstelle nicht.

[4] http://jennassl.blogspot.com/2011/11/updating-jfs-products.html
[5] http://wiki.secondlife.com/wiki/LSL_Protocol/RestrainedLifeAPI
[6] http://rlva.catznip.com/blog/2009/10/release-rlva-1-0-5/

Beim Entwickeln von RTS wurde dies berücksichtigt: RTS Teleporter, daher auch TPMirror, lassen den RLV Support in der Konfiguration ausschalten. Der Teleporter funktioniert dann mit jedem Viewer, spricht jedoch das RLV Interface nicht an, sondern übergibt lediglich die LM oder SLURL zum Zielort.

2.1.4.2 Mit einem Teleportrelay

Eine andere Möglichkeit, RLV Teleporter (daher RTS Teleporter und daher TPMirror) mit einem Viewer zu benutzen, der die RLV Schnittstelle nicht hat, bietet das Teleportrelay.

Zum Nutzen von RLV Teleporter ist generel neben dem RLV-fähigen Viewer, auch ein spezielles Objekt erforderlich, das Relay. RLV besitzt eine technische Sicherheitschranke: Es werden nur Befehle angenommen, die von Objekten kommen, die dem Viewerbenutzer gehören. RLV Teleporter sind Inworldobjekte, die zumeist anderen gehören und so den Viewer gar nicht direkt ansprechen können.

Das Relay löst das Problem. Es gehört dem Viewerbenutzer und kann so den Viewer kontrollieren. Das Relay empfängt Befehle der RLV Geräte und leitet sie zum Viewer weiter.

Meistens führt das Relay dabei auch eine Sicherheitprüfung durch: Darf das Gerät den Viewer kontrollieren oder nicht? Im Zweifelsfall fragt das Relay den Benutzer via Menü, steht das Gerät oder Gerätebesitzer auf einer schwarzen (oder weißen) Liste, verweigert oder akzeptiert das Relay automatisch.

Übliche Relays folgen der Spezifikation, sie setzen die Benutzung eines RLV-fähigen Viewers voraus und leiten alle RLV Befehle der Inworldobjekte an den Viewer weiter. Um RLV Teleporter nutzen zu können, sind aber nur drei Befehle von Bedeutung: Die *Versionsprüfung*, der *Unsitbefehl* und der eigentliche *Teleportbeffehl*.

Nur diese Befehle werden vom Teleportrelay an den Viewer weitergeleitet, alle anderen Befehle werden abgewiesen. Dadurch kann man den RLV ohne Gefahr einer unerwünschten Aktion seitens RLV Geräte nutzen. Bei anderen Relays ist dafür eine Auseinandersetzung mit deren Sicherheitseinstellungen erforderlich.

Wird dabei ein Viewer benutzt, der die RLV Schnittstelle nicht hat, emuliert sie das Teleportrelay, indem es eine gestellte Versionsantwort an das Teleporter versendet und zum Teleportieren das Kartenfenster öffnet.

Auf diese Weise bekommen Inworldobjkte eine stark eingeschränkte Kontrolle über den RLV-fähigen Viewer. Mit Viewern, die RLV nicht verstehen, kann man RLV- und RTS Teleporter trotzdem nutzen – sie werden zu normalen Kartenteleporter. Das Relay wurde mit diesem Ziel geschaffen und ist im TPMirror installiert, zum Aushändigen, wenn der Benutzer kein Relay benutzt.

2.1.4.3 RLV-fähige Viewer

Es gibt mehrere Viewer, die RLV Schnittstelle implementieren und RLV Teleporter mit einem üblichen Relay nutzen lassen. Einige davon sind aufgelistet:

- Klasisches RLV (http://www.erestraint.com/realrestraint/)
- Cool Viewer (http://sldev.free.fr/)
- Firestorm und Phoenix Viewers (http://www.phoenixviewer.com/)
- Imprudence (http://blog.kokuaviewer.org/)
- Rainbow (http://my.opera.com/boylane/blog/)
- Singularity (http://www.singularityviewer.org/)

Die Liste dient dem Überblick und wird nicht vollständig sein.

2.1.5 Globale Koordinaten

Jeden Ort in SL kann man mit einer lokalen und einer globalen Koordinate angeben. Lokale Koordinate beschreibt die Position eines Punktes bezüglich eines speziell festgelegten Ursprungs.

Oben in der Symbolleiste kann man etwa die lokale Position des Avatars bezüglich Ursprung der Sim einsehen. Sei es ein Punkt mit der Koordinate **<128, 128, 24>** (genau in der Mitte der Sim.) Auf der ganzen SL-Welt gibt es so viele Punkte mir der lokalen Koordinate **<128, 128, 24>**, wieviele Sims es gibt.

Erst mit Angabe der Sim kann eine Lokalkoordinate den Punkt eindeutig positionieren. Eine SLURL gibt deshalb die lokale Position, so wie die Sim an, von der aus die Position gemessen wurde. Solch eine beschreibung ist in der SL Welt eindeutig:

http://slurl.com/secondlife/DaBoom/128/128/24.

Alle Sims fügen sich zu einer *Grid* zusammen, das ist die Welt von SL. Jede Sim besitzt deshalb eine globale, d.h. für die Grid eindeutig bestimmte Koordinate ihres Ursprungs. Es lässt sich etwa ermitteln, dass der Ursprung der Sim *Da Boom* die globale Position **<256000, 256000, 0>** hat. Wie kommt man auf Globalkoordinaten einer Sim? Z.B. über den Dienst *Grid Survey*, mit diesem Aufruf:

http://api.gridsurvey.com/simquery.php?region=DaBoom

Die Antwort enthält zwei Angaben: **x=1000**, **y=1000**. Diese Werte multiplizieren wir mit 256 (die Größe einer Sim), und nehmen **z=0** an. Fertig.

Kombiniert man nun die lokale Koordinate eines Ortes auf der Sim mit der globalen Koordinate ihres Ursprungs, erhält man die globale Koordinate des Ortes.

So befindet sich der Punkt '`Da Boom/128/128/24`' auf einer Position `<256128, 256128, 24>`, gemessen vom Ursprung der Grid, d.h. global.

Die globale Position ist eindeutig in der ganzen Grid, sie ist außerdem einfach (nur eine Vektorzahl), aus diesen Gründen arbeiten Kartenteleporter und RLV/RTS Teleporter mit dieser Angabe.

Die globale Position hat aber einen Nachteil: Gelegentlich kommt es vor, dass die Sim sich verschiebt. Dann erhält sie eine andere Ursprungskoordinate. Dadurch ändern sich die globale Koordinaten aller Orte auf der Sim. Teleporter, können die Orte nach der Verschiebung via alter Koordinaten nicht mehr erreichen, die Zielposition muss erneut berechnet werden.

Einige Teleporter berechnen deshalb die Zielposition für jeden Teleport neu. RTS (daher auch TPMirror) verzichtet auf die permanente Neuberechnung (sie belastet die Sim) sondern cacht den Wert und erneuert ihn erst wenn die Berechnung zu lange zurück liegt. Außerdem kann der Gerätebesitzer den gecachten Wert als veraltet kennzeichnen und so beim nächsten Teleport berechnen lassen.

2.2 Installation und Gebrauch

Figure 2.2: Installation

TPMirror ist ein Gerät, das nur ein Prim belegt, welches als eine quadratische Fläche geformt ist. Die Installation erfolgt in drei Schritten:

1. Rezzen Sie den TPMirror und positionieren es am Einsatzort, etwa an einer Wand oder Boden, Abbildung 2.2. Beim Besitzerwechsel initialisiert sich das Gerät automatisch. Dieser Vorgang dauert wenige Sekunden.

2. Optional können Sie den Teleporter konfigurieren, Abschnitt 2.3.

3. Nun aktivieren Sie das Gerät, wie im Abschnitt 2.2.1 beschrieben.

Kennen Sie RTS oder TPMirror noch nicht, ist es empfohlen, den Teleporter erst zu aktivieren, ausprobieren und erst dann an die Konfiguration wagen, Zieldefinition und evtl. auch Umbau.

Der Teleporter muss nicht seine rechteckige Form behalten. Alles kann ein Teleporter sein, womit sich der Avatar stoßen kann: Ein Baum, unsichtbare Wand, Spinnennetz oder gar ein Strandball, der herum liegt. In allen Fällen bleibt es ein Teleportergerät, das auf Collision reagiert und nur ein Ziel unterstützt. Im Abschnitt 2.4 wird erklärt, wie man den Teleporter so erweitert, dass man vor dem Teleport auch das Ziel auswählen kann.

2.2.1 Inbetriebnahme

Der TPMirror ist mit einem Teleportziel vorkonfiguriert und kann sofort nach dem Auspacken eingesetzt werden, allerdings teleportiert er zum voreingestellten Ziel. Das Teleportziel wird in der Konfiguration festgelegt bzw geändert, Abschnitt 2.3.4.

Bedient wird der Tleporter über das Besitzermenü, das nur dem Besitzer angezeigt wird. TPMirror besteht aus nur einem Prim (kein extra Prim als Menübutton), daher lässt sich das Besitzermenü nur via Langklick öffnen.[7] Das Menü enthält vier Buttons:

- '**start**', '**stop**' um den Teleporter starten oder anzuhalten.
- '**reset**': Initialisiert alle Skripte. Hierdurch wird die Konfiguration eingelesen und das Gerät eingestellt. Diesen Vorgang müssen Sie deshalb nach Änderungen in der Konfiguration oder am Teleporter erneut auslösen.
- '**refresh**': Dieses Button löscht die gespeicherten globale Zielkoordinate (Abschnitt 2.1.5) und erzwigt so ihr Neuberechnen beim nächsten Teleport.

2.2.2 Gebrauch

Um mit TPMirror zu reisen, muss man dami kollidieren, etwa ins Bild oder darüber laufen. Was dann passiert, hängt davon ab, ob der Teleporter in einem RLV oder LM Modus eingestellt ist (ausgeliefert ist er mit dem aktiven RLV Modus) und ob der Benutzer einen RLV-fähigen Viewer und das Relay einsetzt. Der genaue Ablauf ist in der Abbildung 2.3 dargestellt.

2.2.2.1 Im RLV Modus

Ist der RLV Modus aktiv, versendet der Teleporter eine Teleportanfrage mit Angabe des Benutzers, die Anfrage empfängt sein getragenes Relay.

Ist die Anfrage akzeptiert, leitet sie das Relay an den Viewer weiter. *Der Viewer führt den Teleport aus.* Der Teleporter empfängt dann die positive Rückmeldung des Relays und (falls eingestellt) übergibt die SLURL eigener Position an den Benutzer per IM (die Rückkehr-SLURL.)

Weist das Relay die Anfrage ab, kann der Teleporter nur noch die LM oder SLURL zum Ziel an den Benutzer übergeben, falls der LM Modus aktiv ist.

Ist das Relay dagegen nicht vorhanden, ausgeschaltet, oder hat der Benutzer seine Rückfrage nicht rechtzeitig beantwortet, stellt der Teleporter das Timeout fest und

[7] Bitte den Teleporter anklicken und die Maustaste erst nach einer Sekunde los lassen.

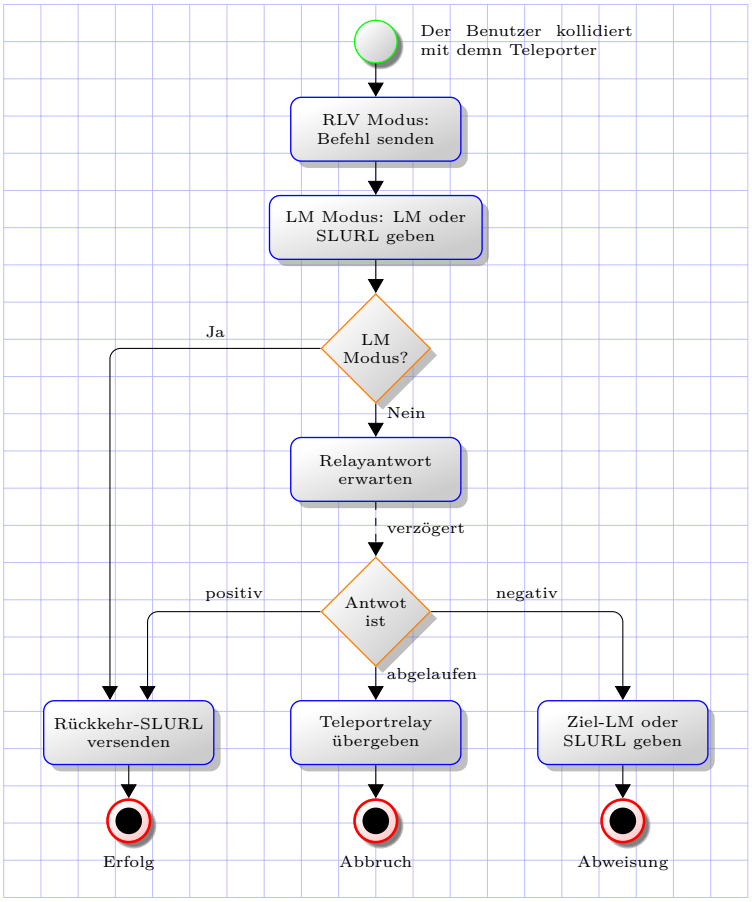

Figure 2.3: Teleportvorgang

händigt das Teleportrelay (oder eine andere in der Konfiguration angegebene Datei aus) dem Benutzer aus.

2.2.2.2 Im LM Modus

Ist das LM Modus aktiv (unabhängig vom RLV Modus), wird die LM oder SLURL zum Ziel an den Benutzer übergeben. Der Teleport gilt dann als erfolgreich, da der Benutzer die LM bzw. SLURL benutzen kann.

Ist gleichzeitig der RLV Modus aktiv, wird die Teleportanfrage dennoch versendet und evtl. durch den Viewer ausgeführt, auf die Rückmeldung des Relays wartet der Teleporter dann nicht mehr.

```
# TPMirror Konfiguration
[*MAIN]
user        = all

image       = 09BB628B-F97A-44FD-9503-6E4EC62DE59E
side        = all

title       = %n\n...step into and visit
color       = lavender
fadeout     = 0

mode        = RLV+LM
fail        = LM
return      = LM
delay       = 10

norelay     = TP Relay (r upper arm)
offset      = -3
fasttp      = OFF
refresh     = 24

format      = APP
msg:url     = This link brings you at target location:
msg:lm      = This landmark brings you at target location...
msg:return  = This link brings you back:
msg:norelay = Please, use this item in order to use me...

[@sensor]
mode        = collide
interval    = 20

# Das einzige Ziel
[Welcome to Endora]
image       = 8cee492c-917e-341d-b3a9-63d83ed44d34
target      = Endora/127/169/42
```

Figure 2.4: Konfiguration von TPMirror

2.3 Konfiguration

Die Konfiguration von TPMirror wird (wie bei RTS Teleportern) in einer Notekarte vorgenommen, die im Objektinventar des Teleporters unter dem Namen '**config**' vorliegt. Beim Ausliefern ist der TPMirror so konfiguriert, wie in 2.4 abgebildet.

Für die ausführliche Erklärung der Gerätekonfiguration im Allgemeinen steht die RTS Dokumentation zur Verfügung. Hier wird lediglich auf den Konfigurationstext eingegangen und somit auf Einstellung speziell des TPMirror.

2.3.1 Syntax

Das Zeichen '**#**' leitet das Kommentar ein: Das Zeichen selbst und der nachfolgende Text bis zum Zeilenende wird ignoriert. Der Text vor dem Kommentarzeichen wird beachtet (außer Leerzeichen), Kommentare werden immer mit dem Zeilenende abgeschlossen.

Leere Zeilen spielen für TPMirror keine Bedeutung, sie erleichtern die Lesbarkeit durch den Benutzer.[8]

Andere Zeilen enthalten entweder Sektionsnamen oder Parameterwerte. Sektionsnamen werden immer in eckige Klammer eingeschlossen. Sektionsnamen erlauben beliebige Schreibweise, um die eckige Klammer dürfen nur Leerzeichen stehen.

Parameter und ihre Werte werden durch das Gleichheitszeichen getrennt. Es müssen beide vorhanden sein. Parameternamen erlauben beliebige Schreibweise, bestimmte Parameter akzeptieren auch Werte in beliebiger Schreibweise, jedoch nicht generell.

Alle Parameterwerte gehören der Sektion an, die zuvor mit dem Namen eingeleitet wurde. Die Reihenfolge, in der die Parameter auftreten, hat keine Bedeutung.

2.3.2 Hauptsektion

```
[*MAIN]
```

Die Hauptsektion hat den Namen '***main**' in beliebiger Schreibweise. Großschreibung wird bevorzugt um die Bedeutung hervorzuheben: Die Hautsekton konfiguriert den Teleporter generell.

[8] Aus Performancegründen wird es empfohlen, auf Leerzeilen und reine Kommentarzeilen zu verzichten.

2.3.2.1 Benutzerparameter

```
user        = all
```

Der Parameter '**user**' gibt an, wer den Teleporter benutzen kann, d.h. damit reisen darf. Der Wert ist eine Liste von Stringflags, verbunden via '**+**'. Es sind Stringflags '**owner**', '**!owner**', '**group**', '**!group**', '**all**' und '**!all**' erlaubt.

Sind die Flags '**all**', '**!all**' angegeben, werden andere ignoriert. Im ersten Fall kann jeder den Teleporter nutzen, im zweiten – niemand. Die Flags '**owner**', '**!owner**' erlauben oder verbieten, dem Gerätebesitzer, mit dem Teleporter zu reisen. Die Flags '**group**' und '**!group**' erlauben oder verbieten jeder Person aus der Objektgruppe, den Teleporter zu nutzen.

In der originalen Konfiguration 2.4 darf jeder den TPMirror benutzen. Der Wert '**group+!owner**' würde dagegen bedeuten: Jeder aus der Objektgruppe kann mit dem Teleporter reisen, der Besitzer jedoch nicht.

2.3.2.2 Bildparameter

```
image       = 09BB628B-F97A-44FD-9503-6E4EC62DE59E
side        = all
```

Der Parameter '**image**' gibt das Defaultbild an, dieses wird angezeigt, wenn die Zielsektion kein Bild angibt. Erlaubt ist entweder eine UUID des Bildes, oder ein Bild im Inventar des Systemprims. Dann muss es Fullperm sein, damit seine UUID ermittelt werden kann.

Der Parameter '**side**' gibt die Primseite zum Anzeigen des Bildes an. Mögliche Werte sind '**all**', '**none**' und eine Zahl zwischen 0 und 9. Der Wert '**all**' bedeutet: Jede Primseite zeigt das Zielbild an, der Wert '**none**' unterdruckt die Bildanzeige. Ist dagegen eine Zahl angegeben, ist dies die LSL Nummer der Primseite für die Bildanzeige.

2.3.2.3 Titelparameter

```
title       = %n\n...step into and visit
color       = lavender
fadeout     = 0
```

Der Parameter '**title**' legt den Titel fest (den Hovertext.) Der Wert kann Platzhalter enthalten:

- Die Platzhalter '**\n**' und '**\t**' werden durch einen Zeilenumbruch und Tabulatorzeichen ersetzt.
- Die Platzhalter '**%n**' und '**%s**' stehen für den Namen des aktuell angezeigten Ziels, so wie Namen der LM oder der Sim, mit der das Ziel definiert ist.
- Die Platzhalter '**%p**' und '**%c**' werden durch die Nummer des angezeigten Ziels in der Zielliste ersetzt (1 steht fürs erste Ziel) und durch die Anzahl Ziele in der Liste.

Zum Verbergen des Titels kann man den Parameter auslassen oder '**-**' als Wert angeben.

Der Parameter '**color**' gibt die Titelfarbe an. Der Parameter kann auch als '**colour**' angegeben werden. Der Farbwert wird grundsätzlich ein LSL Vektor. Einige gebräuchliche Farben können aber auch mittels ihren Namen angegeben werden. Die erkennbaren Farbnamen sind in der RTS Dokumentation angegeben. Der Farbname beachtet die Schreibweise nicht, '**White**' bedeutet dasselbe wie '**WHITE**' oder '**white**'.

Der Parameter '**fadeout**' legt die Anzeigedauer in Sekunden fest, bevor der Titel ausgeblendet wird. Der Timer startet immer, wenn der Titeltext sich ändert. 0 als Wert bedeutet: Kein Ausblenden.

2.3.2.4 Teleportmodi

```
mode        = RLV+LM
fail        = LM
return      = LM
delay       = 10
```

Der Wert von Parametern '**mode**', '**fail**' und '**return**' ist eine Kombination von Stringflags '**RLV**', '**LM**', '**N**' und '**OFF**'. Die Flags können via '**+**' kombiniert werden und bestimmen die Arbeitsweise des Teleporters. Sie wirken additiv.

Ist der Flag '**RLV**' gesetzt, spricht der Teleporter die RLV Schnittstelle des Benutzerviewers an. Der Flag '**LM**' erlaubt die Herausgabe der LM bzw SLURL zum Ziel, der Teleporter wird zum Landmarkverteiler. Die Flags '**N**' und '**OFF**' deaktivieren den entsprechenden Modus, falls '**RLV**' oder '**LM**' nicht gesetzt sind.

Der Parameter '**mode**' bestimmt, ob der Teleportprozess via RLV oder mittels Übergabe der LM bzw. SLURL zum Ziel auszuführen ist. Erlaubte Flags sind: '**RLV**' und '**LM**'. Ist das Flag '**LM**' angegeben, gilt der Teleport immer als erfolgt, da der Agent die übergebene LM bzw. SLURL nutzen kann.

Der Parameter '**fail**' gibt an, ob die LM bzw. SLURL zum Ziel an den Agenten übergeben wird, falls der Teleport via RLV von ihm abgelehnt wurde. Erlaubt sind Flags '**LM**', '**N**' und '**OFF**'.

Der Parameter '**return**' gibt an, ob eine SLURL zum Standort des Teleporters an den Agenten übergeben wird, falls er teleportiert wurde. Erlaubt sind Flags '**LM**', '**N**' und '**OFF**'.

Der Parameter '**delay**' bestimmt, wie lange auf die Rückmeldung des Realay gewartet wird, bis das Timeout festgestellt ist und der Modus '**fail**' in Kraft tritt. Die Zeit ist in Sekunden angegeben, das Minimum ist 5 Sekunden. In diese Zeit fallen sowohl lagbedingte Verzögerungen, als auch die Reaktionszeit des Benutzers, auf die Anfrage seines Relays zu reagieren, Abbildung 2.3.

Einstellungen

Für Parameter '**mode**', '**fail**' und '**return**' sind nur zwei Werte sinnvoll: '**LM**' oder '**OFF**': Der Wert '**N**' entspricht '**OFF**' und der Wert '**N+LM**' entspricht '**LM**'.

Beim Parameter '**mode**' sind folgende Werte sinnvoll:

- '**RLV**' - Teleportversuch via RLV Schnittstelle, verfahre weiter entsprechend der Parameter '**fail**' und '**return**'.
- '**LM**' - Benutze RLV nicht, übergebe stattdessen die LM bzw. SLURL zum Ziel, weiter entsprechend des '**return**' Parameters.
- '**RLV+LM**' - Teleportversuch via RLV, und übergabe der LM und SLURL. Verfahre weiter entsprechend des '**return**' Parameters.
  Diese Einstellung eignet sich für Benutzer mit und ohne RLV, besonders wenn Ziele wia SLURL angegeben sind: Benutzer mit RLV werden automatisch teleportiert, Benutzer ohne RLV können die SLURL aus dem Chatverlauf benutzen.

2.3.2.5 Teleportrelay

```
norelay     = TP Relay (r upper arm)
```

Der Parameter '**norelay**' gibt die Datei an, die übergeben wird, falls keine Antwort vom Relay des Benutzers kam. Die Datei ist mit 'Teleportrelay' in der Abbildung 2.3 bezeichnet (das Timeout-Zweig.)

Auf diese Weise lässt sich etwa eine Notekarte mit weiteren Informationen übergeben. TPMirror ist auf die Übergabe des Teleportrelays eingestellt, da damit der Teleporter auch ohne RLV-fähigen Viewer benutzbar ist.

2.3.2.6 Offsetparameter

```
offset = -3          # entspricht...
offset = <-3, 0, 0>
```

Der Parameter '**offset**' gibt die relative Position des Rückkehrpunktes zum Teleporter an, das ist der Punkt wo die zurückkehrende Avatare gerezzt werden. Die Position kann mittels einer Zahl oder eines Vektors angegeben werden. Die Angabe per Einzelzahl entspricht der Angabe via Vektor, wo die Zahl als x Komponente gilt. Beide Definitionen im obigen Beispiel sind daher identisch.

Was bedeutet dieser Wert? Ist der Offset null, ist die Rückkehrposition die Position des Teleporters selbst, dann werden Avatare, welche die Rückkehr-SLURL benutzen, am Teleporters gerezzt.

Für TPMirror ist das besonders ungünstig: Durch die *Rückkehrkollision* mit dem Teleporter, wird der Avatar erneut zum Zielort teleportiert und kann daher nicht mehr zurückkehren. Die Offseteinstellung kann den Rückkehrpunkt zur Seite verschieben zu einem Ort das sicher zu rezzen ist.

Der Offsetvektor ist immer ein Vektor in der globalen xy Ebene um den Systemprim. Ist es nicht rotiert, zielt der Vektor **<3, 0, 0>** auf den Punkt 3m richtungs der lokalen x Achse des Prims. Wird das Systemprim um die z Achse rotiert, folgt der Offsetvektor dieser Rotation, ignoriert aber Rotationen um die x und y Achsen.

Auf diese Weise kann man einen Teleporter an der Wand so konfigurieren, dass der Rückkehrpunkt 3 Meter vor dem Teleporter liegt. Wird der Teleporter an jeder anderen Wand platziert, liegt der Rückkehrpunkt immer noch 3m vor ihm. Wird der Teleporter aber flach auf den Boden gelegt, verschiebt sich der Rückkehrpunkt nicht über den Teleporter, sondern bleibt 3m seitlich von ihm.

2.3.2.7 FastTP Modus

```
fasttp      = OFF
```

Dieser Parameter ist aus Kompatibilitätsgründen in der TPMirror Konfiguration enthalten. Der Parameterwert '**Y**' und '**ON**' schaltet den FastTP Modus ein, und der Wert '**N**' und '**OFF**' schaltet ihn aus. Da der TPMirror für nur ein Ziel ausgelegt ist, hat hier der FastTP Modus keine Bedeutung.

Erlaubt der Teleporter die Auswahl aus mehreren Zielen, lässt sich dieser Modus aktivieren. Ist der FastTP Modus an, erfolgt bei einer Zielauswahl sofort der Teleport zum ausgewählten Ziel. Ist der Modus ausgeschaltet, kann man durch die Ziele blättern, den Teleport muss man dann zusätzlich starten.

2.3.2.8 Zielaktualisierung

```
refresh     = 24
```

Zum Teleportieren benötigt TPMirror die globale Koordinate der Zielposition. Sims bewegen sich manchmal, was diesen Wert verändert. Die globale Zielkoordinate muss erneut aufgelöst werden, das bedeutet Serverlast und Verzögerungen.

Um die Server zu entlasten, werden die aufgelöste Koordinaten zwischengespeichert. Der Parameter '**refresh**' gibt an, wie lange gespeicherte Daten gültig sind. Die Angabe ist in Stunden, der Minimalwert ist 6. Das Wiederauflösen findet dabei nicht permanent statt, sondern erst beim Teleport auf eine veraltete Position.

Es empfiehlt sich ein Wert von einem Tag bis einer Woche zu nehmen (24 bis 168 Stunden.) Auch lassen sich die Zielpositionen via 'refresh' Button im Besitzermenü manuell als veraltet kennzeichnen.

2.3.2.9 Nachrichtformat

```
format = APP
```

Der Parameter '**format**' legt das Format von übergebenen SLURLs fest. Hier kann man einen der vier Werte angeben: '**MAPS**', '**SLURL**', '**APP**' und Freitext.

'**SLURL**': Versende eine *SLURL*.[9] Sie ist klassisch, alle Viewer verstehen den Link und öffnen das Landmarkenfenster, wenn der Link im Chatverlauf angeklickt ist.

'**MAPS**': Versendet eine moderne *MapURL*.[10] Sie wurde ab der Version 1.23 des SL Viewers eingeführt. Ältere Viewer öffnen den Webbrowser.

APP: Versendet einen *Teleportlink*.[11] Dieser String löst beim Klick im Chatverlauf einen Teleport ohne eines geöffneten Landmarkenfenster aus.

Der Freitext ist ein String mit Platzhaltern `%s`, `%x`, `%y`, `%z`, die durch den Simnamen und die lokale Koordinaten ersetzt werden. Zum Beispiel produziert der Format '`%s (%x, %y, %z)`' einen String wie 'Endora (123, 234, 345)'.

Hinweis: Der Freitext kann die Flags '**MAPS**', '**SLURL**' und '**APP**' emulieren. Zum Beispiel erzeugt der Formatstring '`secondlife:///app/teleport/%s/%x/%y/%z`' denselben Teleportlink wie das Flag '**APP**', benötigt jedoch mehr Prozessorressourcen zum Generieren des Strings.

[9] Ein Link wie http://slurl.com/secondlife/Endora/123/234/345
[10] Ein Link wie http://maps.secondlife.com/secondlife/Endora/123/234/345
[11] Ein String wie secondlife:///app/teleport/Endora/123/234/345

2.3.2.10 Nachrichtparameter

```
msg:url     = This link brings you at target location:
msg:lm      = This landmark brings you at target location...
msg:return  = This link brings you back:
msg:norelay = Please, use this item in order to use me...
```

Die Parameter '**msg:...**' erlauben ebenfalls die Angabe von Textplatzhaltern, Abschnitt 2.3.2.3 und speichern Textnachrichten die beim Übergeben von Items oder SLURLs mitversendet werden.

Der Parameter '**msg:url**' gibt den Text an, der die Ziel-SLURL einleitet. Die IM an den Reisenden besteht aus diesem Text, gefolgt von der SLURL.

Der Parameter '**msg:lm**' gibt den Text an, der beim Übergeben der Ziel-LM als IM versendet wird. Der Reisende bekommt die LM und die IM getrennt.

Der Parameter '**msg:return**' gibt den Text an, der die Rückkehr-SLURL einleitet.

Der Parameter '**msg:norelay**' gibt den Text an, der beim Übergeben des norelay-Elements versendet wird.

2.3.3 Sensorsection

```
[@sensor]
```

Die Section namens '**@sensor**' ist für den Sensorskript gedacht, der wiederum potentielle Gerätebenutzer meldet. Der Sensor unterstützt verschiedene Modi, TPMirror benutzt nur den Kollisionmodus, deshalb wird nur dieser beschrieben.

2.3.3.1 Sensormodus

```
mode        = collide
```

Der Parameter **mode** wählt den Sensormodus aus. Es kann einer der fünf Namen angegeben werden: **touch**, **collide**, **scan**, **sit**, **chat** aktivieren entsprechend den Touch-, Kollision-, Radar-, Sitz- und Chatmodus. Jedenfalls benutzt TPMirror nur den Kollisionmodus, lassen Sie bitte dieses Parameter deshalb unverändert.

2.3.3.2 Scaninterval

```
interval = 20
```

Der Parameter '**interval**' gibt den Wert in Sekunden an. Er bestimmt, wie lange der gemeldete Avatar im Avatarfilter verbleibt und nicht wieder zum Teleportieren gemeldet wird. Sonst würde ein einfaches Laufen über den Teppich eine Menge von Teleports auslösen.

Der Kollisionsmodus kennt nur dieses Parameter, andere mögliche Parameter sind für den TPMirror nicht relevant und werden in dieser Dokumentation nicht beschrieben. RTS Dokumentation spezifiziert die Sensorsektion aber vollständig.

2.3.4 Zielsections

Eine Zielsektion besteht aus nur drei Einträgen: Der *Zielname*, das *Zielbild* und die *Zielangabe*. Da TPMirror keine Möglichkeit vorsieht, Ziele auszuwählen, wird permanent nur das erste Ziel ausgewählt, deshalb macht für TPMirror nur eine Zielsektion Sinn. Definierbar sind jedoch bis zu 80 Zielsektions.

2.3.4.1 Zielname

```
[Welcome to Endora]
```

Der Zielname ist der Sektionsname. Er muss mit einem Buchstaben oder Unterstrich beginnen und kann aus mehreren Wörtern bestehen. Er muss aber eindeutig und höchstens 32 Zeichen lang sein. Der Zielname kann via Plazhalter '**%n**' in den Titel (Hovertext) und versendete Nachrichten eingebracht werden.

2.3.4.2 Zielbild

```
image       = 8cee492c-917e-341d-b3a9-63d83ed44d34
```

Das Zielbild wird stellvertretend für das Ziel angezeigt. Der Parameter '**image**' ist optional. Ist er ausgelassen, wird das Defaultbild aus der Hauptsektion verwendet.

Als Parameterwert kann sowohl die UUID des Bildes angegeben werden oder Name eines Bildes im Objektinventar des Systemprims. Dann muss es dort fullperm vorliegen, sonst kann seine UUID nicht ermittelt werden.

2.3.4.3 Zielangabe

```
target      = Endora/127/169/42
```

Der Parameter '**target**' ist obligatorisch und definiert die Zielposition. Die Angabe kann via LM oder einer SLURL erfolgen.

Ist das Ziel via LM angegeben, muss sie im Objektinventar vorliegen. In diesem Fall wird die LM an den Reisenden übergeben, falls LM-Übergabe aktiv ist, Abschnitt 2.3.2.4. Dadurch kann er das Ziel später noch mal besuchen, ohne den Teleportlog zu durchsuchen.

Ist das Ziel als SLURL angegeben, wird das Inventar des Reisenden nicht durch Objekte gefüllt, die er nicht mehr wieder braucht, das Wiederbesuchen ist nur über den Log oder manuell gezogenen LM möglich. Die SLURL kann auf drei Arten angegeben werden:

- SLURL: http://slurl.com/secondlife/Endora/127/169/42
- MapURL: http://maps.secondlife.com/secondlife/Endora/127/169/42
- RawURL: Endora/127/169/42

RawURL beginnt mit dem Simnamen und ist daher universell. **Hinweis**: In der RTS Dokumentation kann diese Form 'Stripped SLURL' bezeichnet werden, die Bezeichnung 'RawURL' wurde später eingeführt.

2.4 Erweiterung

TPMirror ist ein RTS Teleporter, benutzt aber nicht alle Möglichkeiten von RTS: TPMirror wurde ursprünglich als ein Teleporter für nur ein Teleportziel entwickelt. RTS kann mit einer Liste von Ziele arbeiten, TPMirror bietet jedoch traditionsbedingt keine Möglichkeit, das Teleportziel zu wählen.

In diesem Abschnitt wird der Teleporter um diese Möglichkeit erweitert. Wir müssen dafür keine Skripte schreiben oder verändern. Alles was wir brauchen ist, richtig benannte Prims zum Teleporter hinzu zu linken. Das Ergebnis wird ein Teleporter sein, den wir wegen seiner Form mit '*TPArch*' bezeichnen, Abbildung 2.5 (Das Bild wurde ohne den Titeltext aufgenommen.)

Figure 2.5: TPArch, installiert mit PortalFrame

Der Abschnitt richtet sich an Käufer des TPMirror – da man gridweit teleportieren kann, braucht man bald einen Teleporter für mehr als nur ein Ziel. Aber auch an Besitzer des RTS Pakets, als Demonstration von RTS und als Tutorial für die Arbeit damit.

2.4.1 TPMirror Architektur

Zuerst betrachten wir kurz den Aufbau und Arbeitsweise von TPMirror. Das mag es helfen, zu verstehen, wie TPMirror funktioniert. Detailierte Inforrmation findet

sich in der RTS Dokumentation, hier finden wir lediglich einen Überblick.

2.4.1.1 Dateien

Im TPMirror sind einige Dateien installiert.

Skript '**.core**': Kernskript. Er kontrolliert die anderen Skripte, führt den Teleport aus und initialisiert den Versand von Nachrichten und Dateien. Er präsentiert außerdem das Besitzermenü und führt eine Reihe von Kernservices.

Skript '**.config**': Konfigurationsskript. Er liest die '**config**' Notekarte und verteilt die gelesenen Daten an andere Skripte. Andere Skripte haben auf die Konfigurationsnotekarte kein Zugriff, das vereinfacht deren Code.

Skript '**.sender**': Senderskript. Er versendet Dateien wie LMs oder das Relay, so wie Nachrichten per IM an die Benutzer, aber auch Fehlermeldungen, die während der Konfiguration und im Betrieb entstehen.

Skript '**.sensor**': Sensorskript, dient zum Erkennen und Melden des potentielles Benutzers. Der Sensorskrpt meldet jeden erkannten Avatar. Der Kernskript empfängt die Meldungen, prüft, ob die Avatare erlaubte Benutzer sind und teleportiert sie falls ja.

Skript '**.targets**': Zielliste, kann bis zu 80 Ziele aufnehmen, verwaltet die in der Konfiguration angegebene Ziele: Speichert Daten aus der Konfiguration, löst erforderliche Zielkoordinaten auf, liefert Daten für ausgewähltes Ziel.

Objekt '**TP Relay (r upper arm)**': Das Teleportrelay, wie in der Konfiguration angegeben, Parameter '**`norelay`**', Abschnitt 2.3.2.5.

2.4.1.2 Kernservices

In RTS wird die Funktionalität des Teleporters in einzelne Kernservices unterteilt. Sie werden durch den Kernskript ausgeführt, sind aber übertragbar. Folgende Kernservices kommen im TPMirror zum Einsatz:

Das *Textservice* ist dafür zuständig, den Titeltext über dem Teleporter anzuzeigen. Es beachtet die Titelparameter, Abschnitt 2.3.2.3.

Das *Nachrichtenservice* versendet Fehlermeldungen an den Besitzer via IM. Die Nachrichten werden an den Senderskript übergeben, die er weiterleitet.

Das *Imageservice* stellt das Zielbild dar. Es werden Bildparamteter beachtet, insbesondere der Parameter '**`side`**', Abschnitt 2.3.2.2.

Das *Menüservice* öffnet das Besitzermenü beim Langklick auf den Teleporter.

2.4.2 Aufbau

Zurück zur ursprünglichen Aufgabe. Der Teleporter wird ein Linkset, wobei die Prims grob drei Aufgaben übernehmen. Die *Basis* ist ein Prim wo alle Skripte installiert sind, und das tatsächlich der unveränderte TPMirror ist.

Die *Vorschaubilder* erlauben es, Teleportziele auszuwählen. RTS bietet dafür auch andere Möglichkeiten, wir benutzen sie nicht.

Das *Menübutton* macht es einfacher, das Besitzermenü zu öffnen, als mit dem Langklick. Zusätzlich betten wir den Teleporter zum freien Aufstellen in einen Steinrahmen ein.

2.4.2.1 Basis

Als erstes benötigen wir den Basisteleporter: TPMirror wird gerezzt und auf die Größe $(0.02, 1.5, 2)m$ so wie Nullrotation $(0, 0, 0)°$ gebracht[12]. Das Prim benennen wir gleich um – zu '*TPArch*', Abbildung 2.6.

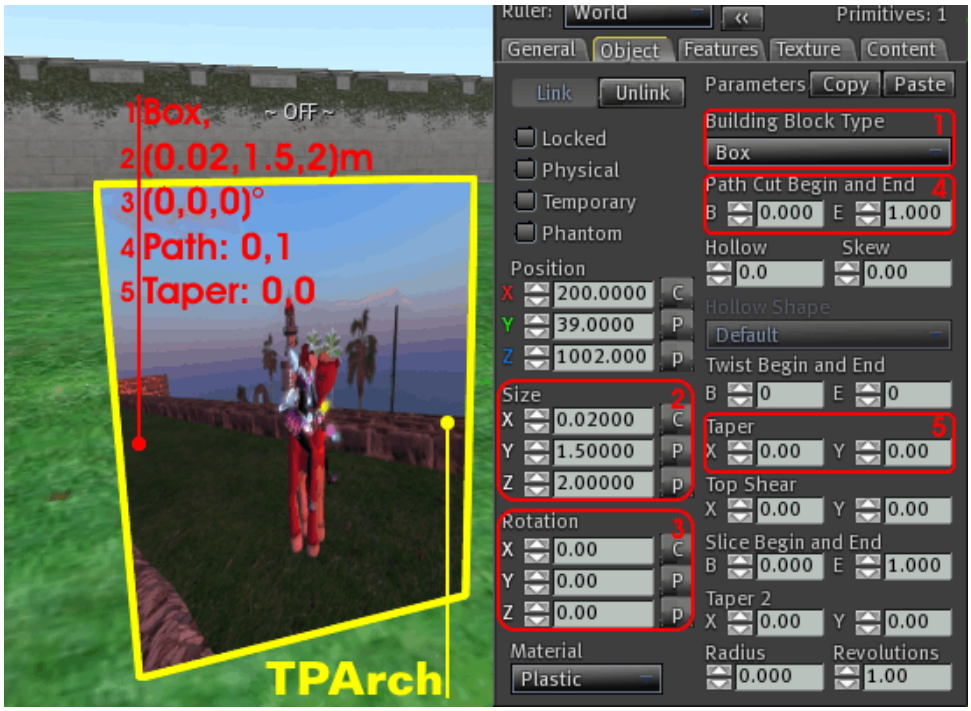

Figure 2.6: Rezzen der Basis (und Legende)

Die Abbildung führt auch die Legende für weitere Bilder ein: Gelb sind die Primnamen so wie Status im Linkset, rot – ihre geometrische Eigenschaften und relative Position. Der Primtyp wird direkt angegeben, Position und Rotation durch Einheiten

[12]Die Größe und Rotation wird als Vektor (x, y, z) angegeben.

am Vektorende. Weitere Eigenschaften werden via Präfix eingeleitet. Standardwerte werden nach Möglichkeit ausgelassen (hier wäre das alles außer der Größe).

2.4.2.2 Vorschauprims

RTS unterstützt bis zu 20 Vorschaubilder, wir nehmen nur 6. Auf den Prims werden die Zielbilder angezeigt, Klick darauf wählt das angezeigte Ziel. Die Vorschauprims geben dem Teleporter seinen Namen – wir ordnen sie bogenartig über der Basis.

Genauer gesagt blden die Vorschauprims Sektoren eines Halbkreises über dem Basisprim. Man könnte ihre geometrische Eigenschaften mit einem Taschenrechner berechnen, wir benutzen einen einfacheren Bautrick dafür.

Wir rezzen zwei Boxen: Das Vorschauprim in der Größe $(0.02, 0.38, 0.35)m$ und das Hilfsprim in der Größe $(0.05, 0.05, 0.25)m$, beide in Nulrotation und vertikalem Abstand von $55cm$.

Jetzt wird das Vorschauprim eingeengt und zum Hilfsprim so eingelinkt, dass dieses das Rootprim wird. Das Primpaar wird wiederum im vertikalen Abstand von $1.004m$ zur Basis positioniert, Abbildung 2.7.

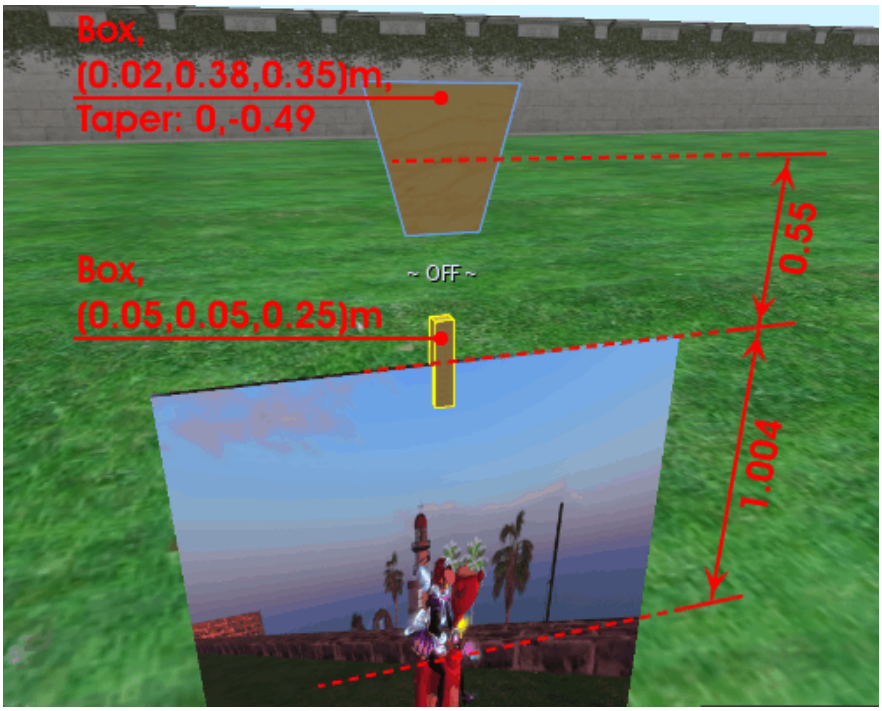

Figure 2.7: Vorschauprim, Hilfskonstruktion

Jetzt kopieren wir das Primpaar 5 Mal nach Vorne: Editieren, die Umschalttaste gedruckt halten und am roten Pfeil ziehen, Abbildung 2.8 links.

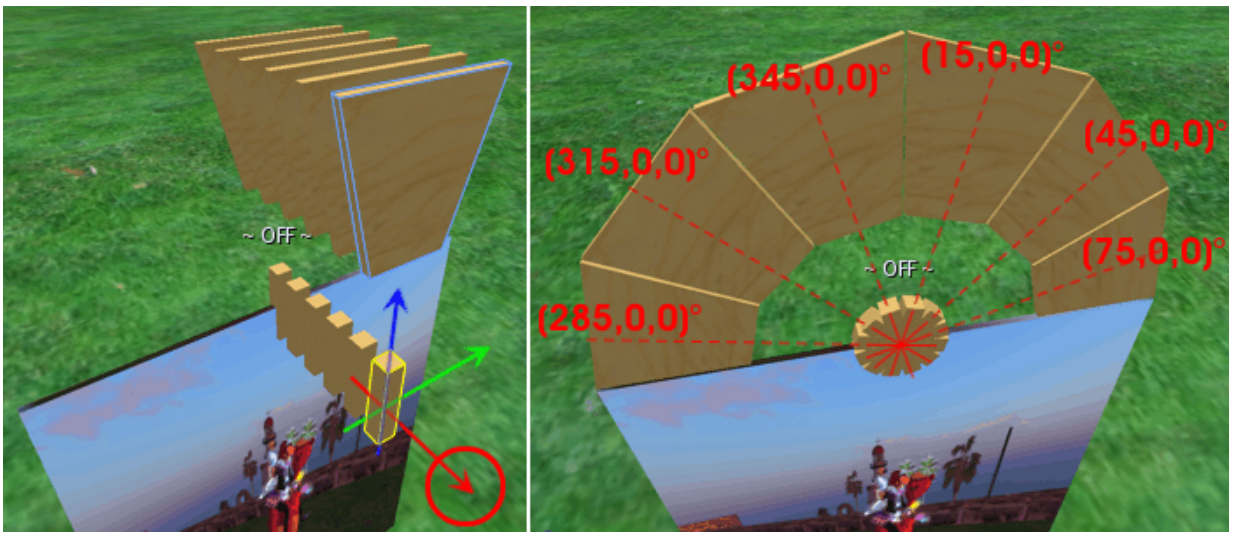

Figure 2.8: Vorschauprims, kopiert und ausgerichtet

Falten wir die Kopien zusammen (die y Position des Basisprim übernehmen) und rotieren jede Kopie im Abstand von 30°, Abbildung 2.8 rechts. Der angesprochene Bautrick besteht darin, dass durch Rotation des Hilfsprims die Vorschauprims in die erwünschte Position gebracht werden.

Jetzt kann man die Primpaare auslinken (Links auflösen) und alle Hilfsprims bis auf eines löschen: Das verbliebene Hilfsprim wird zum Menübutton. Anschließend werden die Vorschauprims von oben im Uhrzeigersinn mit '`@thumb 1`' bis '`@thumb 6`' benannt: Erst dadurch werden sie als Vorschaubilder ausgewiesen, Abbildung 2.9.

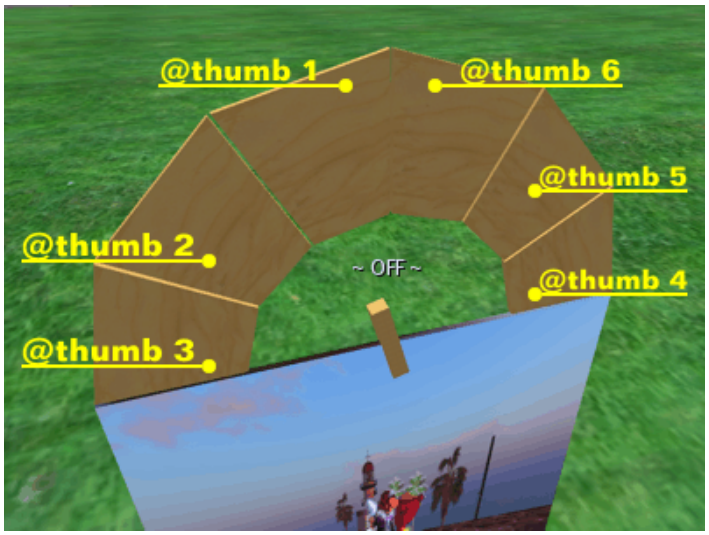

Figure 2.9: Namen der Vorschauprims

2.4.2.3 Menübutton

Ein Menübutton ist ein Prim mit dem Namen '**.menu**'. Die Form und Position im Linkset sind unwichtig. Wir nehmen dazu das Verbliebene Hilfsprim und füllen damit auch optisch den Zwischenraum zur Basis – die Stelle wird verglast.

Ein Halbkreis mit einem radialen Mosaikmuster bietet sich an. Dazu könnte man ein Zylinder nehmen mit einer geeigneten Textur, wir können aber einen Texturtrick anwenden um eine Glasblocktextur radial auf eine Halbscheibe auftragen: Wir benutzen als Scheibe ein flaches geschlossenes Rohr. Es hat dann eine Zylinderform und zeigt die Texturen radial auf den Basisseiten.

Das Hilfsprim bekommt den Namen '**.menu**' und wird zu einem Rohr der Größe $(0.02, 0.75, 0.75)m$, Nullrotation und Pfadschnitt $(0, 0.5)$. Als Textur nehmen wir '**AF_glass_block.tga**' aus dem Texturviewer (RTS Paket) und setzen die Wiederholung $(2, 12)$, Abbildung 2.10.

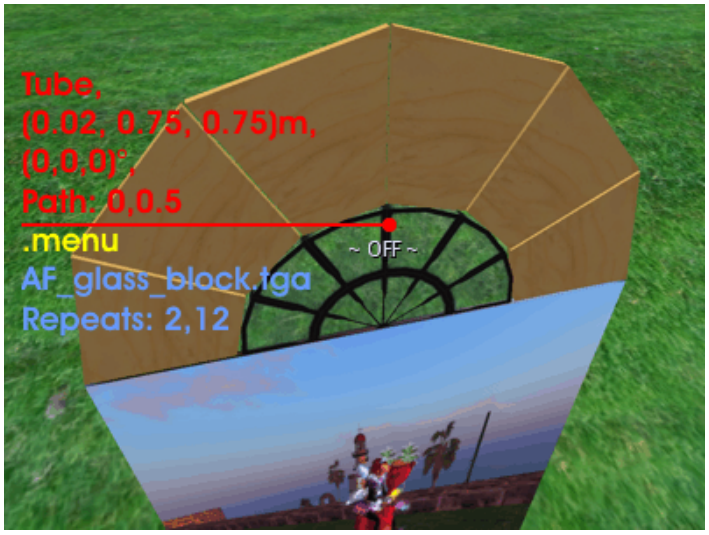

Figure 2.10: Menübutton: Umgeformt, umbenannt und texturiert

2.4.2.4 Abschluss

Jetzt werden alle Prims so verlinkt, dass die Basis das Rootprim wird. In einem Rutsch können wir nun den Glanz und das Leuchten zuweisen. Um die Illusion einer hauchdünnen Konstruktion zu erzeugen, nehmen wir dem Teleporter seine Tiefe: Wir setzen seitliche Flächen aller Prims transparent.

Am einfachsten geht das, indem man den Teleporter als ganzes transparent macht, die Seitenauswahl aktiviert und vordere bzw. hintere Flächen einzeln auswählt, die man anschließend halbdurchsichtig macht. Das Ergebnis ist in 2.11 abgebildet.

Figure 2.11: Abschluss

2.4.2.5 Platzierung

Es fehlt noch die Konfiguration, dann kann der Teleporter etwa im Durchgang oder als Fenster installiert werden. Aber nicht draußen: Ohne eines geeigneten Rahmen sieht er unwirklich aus. Wir machen eines, damit man den Teleporter frei am Wegrand oder ännlichen Plätzen aufzustellen kann.

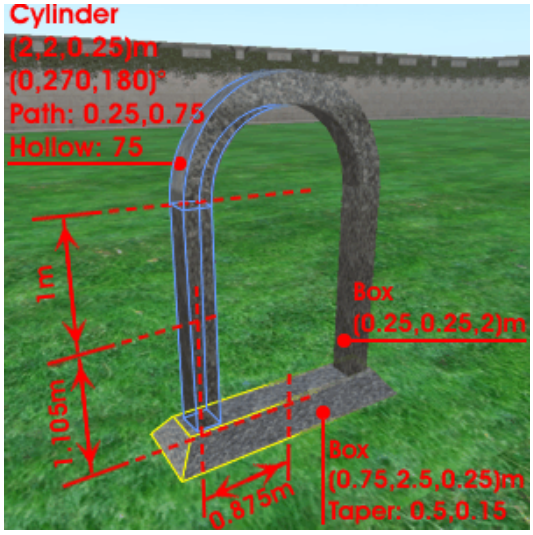

Figure 2.12: Portalramen

Die einfachste Variante besteht aus vier Prims, Abbildung 2.12. Der Rahmen kann mit dem Teleporter verlinkt werden (danach bitte resetten), muss aber nicht: So kann man den Teleporter auch pur installieren.

```
# TPArch Konfiguration
[*MAIN]
title       = Destination %p of %c\n%n\n...step into and visit
# --- Der Rest wie für TPMirror ----------------------------

[@sensor]
mode        = collide
interval    = 20

# Zielliste
[Linden Playground (M)]
image       = 1b5e8133-3fb8-5488-c498-e63ec41b2010 # Da Boom (143, 148, 41)
target      = Da Boom/128/128/35

[Protected Ocean (G)]
image       = 16c62e5b-0b9c-9d78-1aba-e1f45af68bdb # Pravatch (230, 229, 2)
target      = Pravatch/221/227/3

[MYST Island (G)]
image       = 5a6b672a-6c6a-d2a1-a020-04217cdaf0da # Age of Myst (171, 46, 33)
target      = Age of Myst/175/14/24

[Great Wall Of China (G)]
image       = 10594b0c-2ae4-3f7b-052b-a77a098fefcd # China Sichuan (130, 10, 24)
target      = China Sichuan/133/16/24

[Museum of Illusions & Magic (G)]
image       = 76999bdb-11ba-c08d-4d5a-24ea1bed073a # AmazingIllusionMuseumEI
target      = Enchantment Island/59/120/45

#Dieses Ziel war beim TPMirror angegeben
[Welcome to Endora (A)]
image       = 8cee492c-917e-341d-b3a9-63d83ed44d34
target      = Endora/127/169/42

[The Grand Canyon (M)]
image       = d835568f-fb32-5fe4-2ba1-99191261eced # Grand Canyon (106, 173, 110)
target      = Grand Canyon/89/186/110

[Happy Mood (G)]
image       = 525ea54c-49dd-bc1d-327e-c64cd90610db # HappyMood (107, 144, 79)
target      = HappyMood/109/144/80

# --- 3 weitere Ziele folgen -----------------------------
```

Figure 2.13: Konfiguration von TPArch

2.4.3 Konfiguration

Der Aufbau ist fertig. Es fehlt noch die Konfigurationserweiterung. Dazu bearbeiten wir den Teleporter und die Notekarte '**config**' darin.

Hier wird lediglich der Titel geändert (um die Mehrzielfähigkeit zu reflektieren) so wie die eine Zielsection durch eine Liste von Zielsections ersetzt – einmal für jedes Teleportziel. Wir können hier bis zu 80 davon registrieren. TPArch kommt mit 11 registrierten Beispielzielen, Abbildung 2.13.

Die Konfiguration gibt einige Tips bei der Handhabung der Zielliste:

- Zielnamen enthalten das Rating der Sim (G, M, A) in den Klammern. Der Zielname wird über den Platzhalter '**%n**' in den Titeltext integriert, das erlaubt es, im Betrieb, noch vor dem Teleport die Rating des Zielorts zu erkennen.
- Zielbilder sind via UUID angegeben. Dadurch erübrigt sich die Installation der Bilder und Synchronization der Bildnamen mit Angabe in der Notekarte.
- Kommentarzeichen bei der Bild-UUID erlaubt es, den Bildnamen anzugeben. Dieser ist nicht für den Teleporter bestimmt, er hilft später, das verwendete Bild wiederzufinden.
- Zielpositionen wurden via RawURL festgelegt. Dadurch braucht man keine Landmarken anfertigen um die Ziele vorzubereiten, man kann auch den Teleportverlauf benutzen oder Angaben der Sim und die Positionen von den Screenshots ablesen.

Nachdem die Konfiguraton erweitert und abgespeichert wurde, muss man den Teleporter starten: Den Menübutton anklicken, '**reset**'-Button im geöffneten Menü. Nach Ablauf der Konfiguration muss der Teleporter via erneut geöffneten Besitzermenü starten – der Teleporter ist einsatzbereit, wie in 2.5 abgebildet.

2.4.4 Erweiterte Architektur

Es wurden im Originalteleporter nichts an den Skripten geändert. Warum funktioniert die Erweiterung? Die Antwort liegt in der Architektur und Schnittstellen von RTS. Dieser Abschnitt soll ihre Wechselwirkung erklären, im Kontext des TPArchs. Für weitere Informationen schlagen Sie bitte die RTS Dokumentation nach.

Die Zielliste (Skript '**.targets**') verwaltet die in der Konfiguration angegebene Ziele. Wird ein Ziel ausgewählt, liefert die Zielliste eine Reihe von Zieldaten, unter anderem das Zielbild und Bilder der Nachbarziele.

Sobald der Sensorskript eine Avatarkollision erkennt, meldet er diese mit einer Nachricht, die den Teleport zum gerade ausgewählten Ziel auslöst. Der Kernskript prüft ob es für den Avatar zulässig ist und wenn ja, startet er den Teleport.

Durch Hinzufügen und Benennen von Prims werden zwei weitere Kernservices aktiviert, das *Thumbservice* und das *Buttonservice*.

Durch das Namenspräfix '**`@thumb`**' werden die Vorschauprims als solche gekennzeichnet und werden durch das *Thumbservice* erfasst. Wird ein Ziel ausgewählt, empfängt das Service die Vorschaubilder und zeigt sie auf den Vorschauprims an. Beim Klick eines davon produziert das Service eine Steuernachricht, welche die Zielauswahl derart verschiebt, dass genau das angeklickte Vorschauziel ausgewählt wird.

Das Prim mit Namen '**`.menu`**' gilt als Button und wird durch das *Buttonservice* erfasst. Das Service produziert beim Klick darauf eine Steuernachricht, die den Buttonnamen überträgt. Beginnt der Buttonname mit einem Punkt, ist die Nachricht für den Kernskript bestimmt. In diesem Fall ist die Steuernachricht genau die zum Öffnen des Besitzermenüs.

TPArch benutzt in der vorgestellten Ausführung also nur eine Auswahlmöglichkeit. Die Zielliste bietet aber auch Weitere: Die absolute Positionsangabe in der Zielliste (Buttonpräfix '**`pos`**'), Auswahl des ersten und letzten Ziels (Buttons '**`first`**', '**`last`**'), Zufallsauswahl (Button '**`random`**') und eine Textsuche.

Das wäre alles. Ich wünsche Ihnen viel Spaß und Erfolg beim Experimentieren mit dem Teleporter und dem RTS.