Configuration & API

RLV Teleporter System

personal edition

version 2.1 (December 16, 2011)

english & deutsch

Contents

1 English

1.1 Introduction

The RLV Teleporter System (in short *RTS*) is a framework for building teleport devices. Basically, in SL exist two technologies for teleporting. Local and global teleporters. Local teleporters move the traveling avatar to the target position, thus they work only if the teleport path stays inside any sim at every time.

Global teleporters let the viewer disconnect from the current sim and connect to the target one. This works over every distance. Before RLV only one system worked so: *Map teleporters* open a world map with selected position and 'Teleport' button. This window takes the agent off the actual environment destroying the immersion.

RLV Teleporters pass the teleport command to the viewer over the RLV interface. Is the command accepted, the viewer performs the teleport without opening the map window. This kind or teleport is only possible if the viewer implements the RLV interface. The RTS belongs to this category. Some important features of this system are introduced.

RTS works *grid-wide*. It can achieve every place, given by a landmark or SLURL, except of places secured by landing points – similar to using a landmark, the landing point overtakes the teleport. However, RTS handles local targets on same manner.

RTS supports *multiple targets* and *teleport modes*: Due *TriggerTP* teleport mode, the teleport target is selected from a list given in the device configuration. Due *DirectTP* teleport mode, the target is given on-the-fly by the teleport command itself. Finally, due the *RepeatTP* teleport mode, the teleport goes to a location used prevously, for example to follow an avatar.

The *FastTP* technique allows to initiate the teleport by a single click. RTS supports also four other *trigger modes*: Collision, sitting, radar and chat.

The system architecture is *modular* and *extensible*: By selecting of scripts, one can realize different teleport projects. Open *script interfaces* allow to extend RTS by further scripts.

Link-robust semantics of prims: Prims take a special roles, e.g. buttons, preview prims etc. Scripts find the prims by name instead of link numbers. This way the devices are not sensitive against changes on the linkset.

Comprehensive *configuration*: Configuration data is given by a notecard and separated in sections. This improves the handling of data and extensibility of the system, if different sections take data for different scripts.

1.1.1 Audience

Who is this product intended for? Well, if you look for a teleporter which simply bring you to another place on the same sim, like an elevator to the next floor of your home, than RTS is not for you. There are many teleporter scripts around, which will do their job too, and possibly better than RTS.

But if you are looking for a teleporter for places across the grid, if a single teleport device or a ring of partner teleporters, than you might find RTS usefull, even if you are not using RLV or a capable viewer.

Are you familiar with scripting or building? If not, no problem: In the RTS package you'll find example teleporters, preconfigured and ready to use. All you need than to do is setting them up by replacing target places with your favorites. In that case you can even skip reading this manual after the configuration section (1.2.)

If you know how to build and script, you can easilly adapt the teleporters to your own needs, extend them by own scripts and also create the own tleporter style by using RTS as core. In that case you need the whole manual and also the *RTS Extensions* package, available for free.

RTS is released in two versions, a personal and professional editions. If you only want teleporters for your own needs, and let your friends, guests and strangers to use them, than you need the personal edition. If you want give the teleporters away, or perhaps sell them, than the professional edition is yours.

1.1.2 Scope

RTS belongs to a bundle of two products: *RTS* itself and *RTS Extensions*. RTS includes a development kit and example teleporters, built with this kit. This documentation explains how to configure and use teleporter system. It also specifies interfaces, which the RTS scripts offer for control and interaction.

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, how this interface works, which viewers support it and, finally, how to use RLV teleporters without running RLV-capable viewer.

The *configuration* of the teleport systems and of packaged example devices is explained in the section 1.2. The section 1.3 introduces the general *architecture* of the RTS, the system scripts and how they fit into the teleport system.

The interfaces of the scripts are again specified in the section 1.4. The section 1.5 provides a list of *error messages*. System scripts generate them while configuration and in runtime if something goes wrong.

Finally, the chapter 3 gives the color palettes listing colors by names. This colors can be used by name instead of vector values in the configuration.

The *RTS Extensions* package handles development of teleporter projects by using the development kit and writing special scrpts. The package documentation starts where this one stops and provides a step-by-step building tutorial while the package itself includes ressources used in the tutorial.

1.1.3 Package content

The delivered package contains besides of satellite files also a development kit, example projects and help files. The files inside the package are unstructured. But the package can unpack itself in folders automatically as described in section 1.1.3.

The development kit consists off the folders '**System**' and '**Drivers**' with system and driver scripts inside for building teleporter devices. The example projects are ready build and pre configured devices inside the folder '**> Projects**'.

The leading '>' char is an indicator for further subfolders. By delivery, the folder contains only objects – the example teleporters. The idea behind the folder name is to provide you a space for further teleporter projects, where you can keep every of them in its own subfolder.

Help files are in the folder '**Resources**', that is a package with sounds and animations, a picture manager and a color manager. The picture manager is load by few images used for example teleporters. The color manager allows to easily select a color for using in the configuration. It knows also the color palettes given in chapter 3.

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. This works overall but all files come in a single folder without any order.

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 '**RLV Teleporter System**' and four others: '**> Projects**', '**Drivers**', '**Resources**'

and '**System**'. Scripts can't give scoped folder structure. Thus, please, move the last given folders into the product folder, so it looks like in figure 1.1.

After all folders are given, the menu opens again where you can hit the '**Remove**' button to detach it or clean from the world. Otherwise the box destroys itself automatically after 30 minutes.

Please, keep the product package as backup for the files, or at least the scirpt '***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.[1]

Update

The RTS product 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 instaling 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.[2]

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.[3]

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

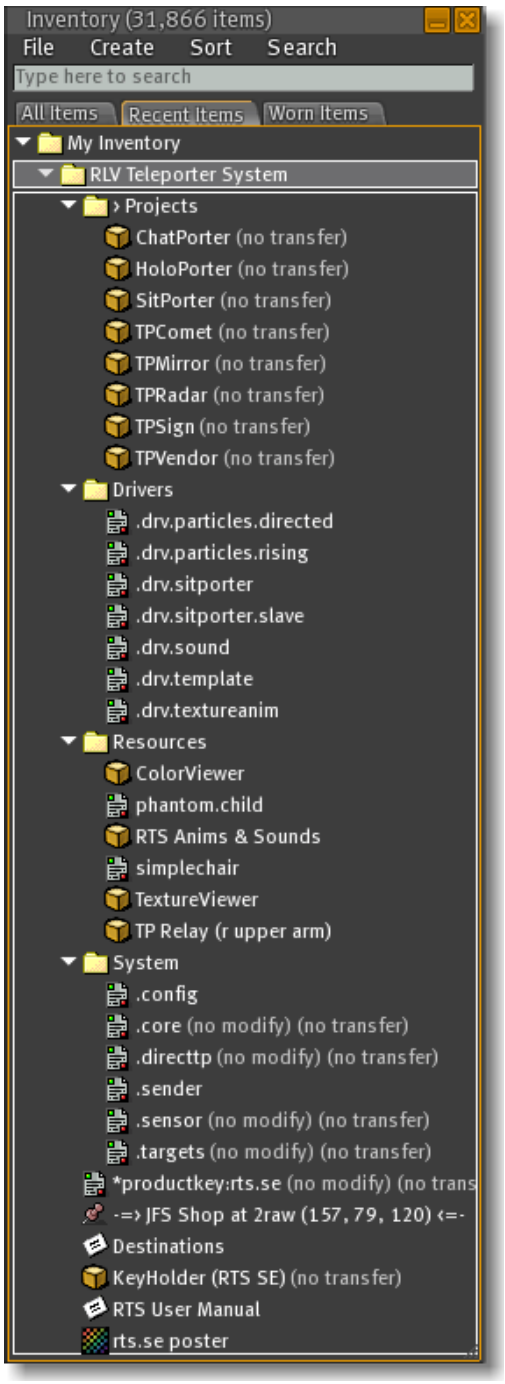


Figure 1.1: Package content, unpacked automatically

Development kit

The development kit contains system and driver scripts, required for building of teleport devices. The system scripts form a generic 'operation system' of the device. The driver scripts adjust it on needs of the concrete device.

The *core script* is a script '**.core**'. It is essential for the device and required for every teleport project, It performs the teleport process and runs also core services around the device.

The *configuration* script '**.config**' is required for every teleport project. It centralizes the configuration of the device by reading the configuration notecard and forwarding data to other scripts.

The *sender* script '**.sender**' is required for all projects. Its job is to send files and messages to avatars, for instance error messages to the device owner.

The *sensor* skript '**.sensor**' is optional. It is used for triggering the teleport. The script refines the click trigger mode (the core script can do it) and triggers also by collision, sit, radar and chat.

The *targetlist* script '**.targets**' is required for teleporters working with a list of target given by configuration notecard. The script saves up to 80 target locations and calculates in run time global coordinates, required for teleporting there.

The script resolves also global coordinate of a position used in *DirectTP* command if the position provides local coordinate. This allows to build direct teleporters without manual calculating of global coordinates.

The *DirectTP* script '**.directtp**' is helpfull for building direct teleporters: The target positions are not defined in the configutation than but provided in the teleport command in local form, so this script has to resolve and forward the global position of the target location.

The *sound driver* script '**.drv.sound**' plays sounds if the device is started, stoped, or if a teleport process starts or is completed.

The *particle drivers* '**.drv.particles.rising**' and '**.drv.particles.directed**' spawn particle salves while the teleport – either rising or directed towards the traveler.

The *animation driver* '**.drv.textureanim**' assigns and animates the face texture while the teleport and depends of its process.

Finally, the *driver template* script '**.drv.template**' is also available. It supports most system interfaces and can be easily extended to a passive driver script.

Teleporter devices

The project folder has seven teleporters inside, prepared for using out of the box. Some of them have scroll prims and a logo prim for calling owner menu, some are made just of one prim and have none for owner menu.

If the logo prim is missing, you can open the owner menu via long click: Click the device with the left mouse button and hold it a second. The long click works for all teleporters, if with or without logo prim.

TPSign

TPSign is a single prim, figure 1.2(a) and uses no sensor script: The teleport starts if the target picture is clicked. Because scroll possibility is missing, the teleporter handles a single target but is potentially able to handle many.

TPVendor

TPVendor takes 11 prims and extends TPSign by preview buttons, scroll buttons and the logo prim, figure 1.2(b). The device uses also no sensor script (actually the scripts are the same as in TPPSign.) Through scroll buttons the device becomes multi target one. It is pre configured by eleven destinations.

The destination is chosen by scroll buttons or thumbnails, the teleport starts by clicking the target picture. The teleporter uses also no sensor script, the scripts are the same as inside TPSign.

TPMirror

TPMirror uses a single prim, figure 1.3(a) and runs the sensor script in collision mode. The teleporter is made for using as wall picture or a curtain. The teleport starts by walking into the picture. It can be also put like carpet on floor, teleporting avatars running over it.

TPRadar

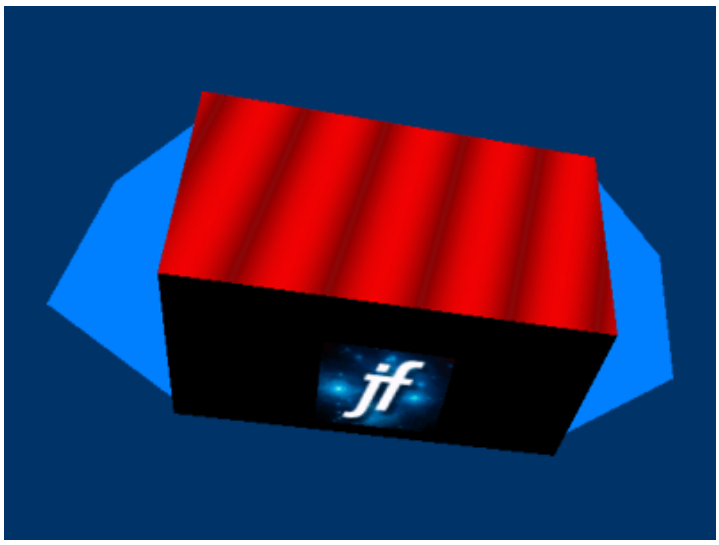
TPRadar is made off four prims for mounting inconspicuously on the wall or floor, figure 1.3(b). Attention by unpacking: The device is (5 x 5 x 15) cm^3 small. The target picture and text are hidden.

(a) TPSign

(b) TPVendor

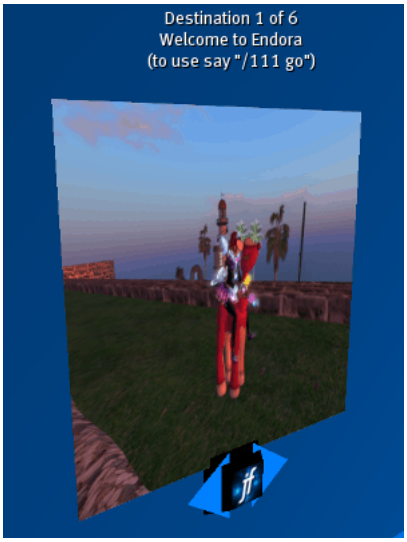
Figure 1.2: Clickteleporters

(a) TPMirror

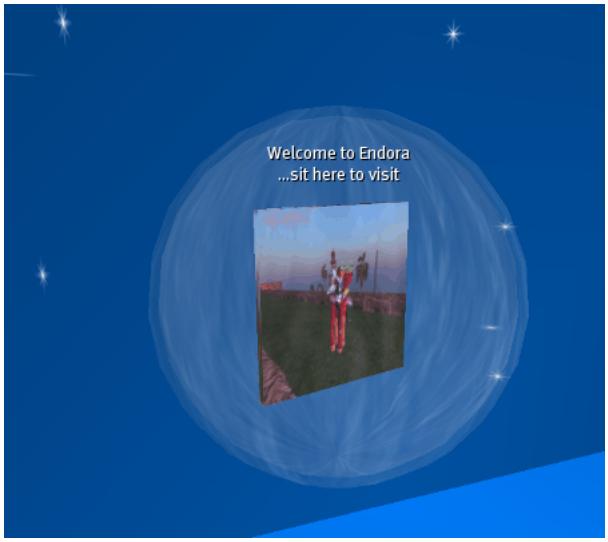


(b) TPRadar

Figure 1.3: Scanning teleportera



(a) Chatporter



(b) Sitporter

Figure 1.4: Reacting teleporters

Figure 1.5: The Holoporter

The teleporter observes the room and teleports intruders at the selected target. To do so, the sensor script runs in radar mode. The menu prim is also used as particle source: The particle driver fires particle rings towards the travelers.

Chatporter

The Chatporter is made as a wall screen showing the target picture with scroll and menu prims (four overall,) figure 1.4(a). The teleport starts if the traveller gives the command '`go`' on the channel `111`, i.e. '`/111 go`'. The command is received by the sensor script in chat mode.

Here you can see the power of this sensor mode. If the teleporter is started, it understands not only the command '`go`' (teleport) but also '`.menu`' to open the owner menu, '`prev`' and '`next`' to scroll through targets. Also a target search is possible: The command '`/111 lind`' selects for example a target with name starting by 'lind', in this case 'Linden Playground'.

But also you can say '`/111.tpto:HappyMood/179/161/26`' to generate the (local) DirectTP command towards the stated location. The teleporter resolves the global coordinate of the location and sends you there.

Sitporter

The teleporter is made as a sphere with rotating target picture inside, figure 2.4(b). The teleport starts if the user sits on it. To do so, the sensor script works in sit mode. Again, no scroll possibility, thus only a single target is possible here,

Holoporter

Holoporter is build off 6 prims, figure 1.5, as a platform with a target picture over it. It manages multiple targets and reconfigured also by six. To travel, just stand on it (simple walk over, its no posestand) and click the teleporter.

To recognize the teleporter is clicked by a person over the platform, the sensor script is used in touch mode with distance restricted to 2m – avatars not on the platform would stand further away and their clicks are ignored.

TPComet

In the package you find also a bonus teleporter. This one was made for taking product picture (title page of this document) but is really working. TPComet is a direct teleporter: The target positionss are defined by target pictures (i.e. their prim names) and not by the config notecard.

To explain the teleporter, the knowledge about RTS architecture and core services are required. To configure target locations you must know how to edit prims. For this reasons, the teleporter is introduced in the documentation for the RTS Extensions package and not in this document.

1.1.4 RLV, RLVa and Immersion

RLV[4] is a special viewer, developed with a goal to support immersion of the user (a feeling to be in the game.) To understand RLV, we must not enter the scene, for which the viewer was created. Imagine for example a situation where an evil magician turns you to stone.

Usually, you must do now as if you were off stone: don't move, nothing say, and never touch a thing. The feeling, to be a stone is missing. But immersion begins even if we use the standard instrument of SL: An attachment that locks the pose and blocks movement by arrow keys. More its not possible.

The RLV goes further. It allows this attachment to block chat and clicking on objects. And detaching the stone attachment would be impossible too. You must not pretend being of stone, you feel so.

The standard viewer is made very safe. Its abilities are never restricted and it performs no unexpected actions. That is safe but not optimal for the mentioned situation.

RLV is different: Scripts can deactivate certain features like chat, detaching of attachments, and run some actions, like stand up of an object or *teleport to a location inworld*. To do so, scripts access a special interface provided by the viewer.

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.

[4] Originally 'Restrained Life Viewer', later 'Restrained Love Viewer'
[5] http://wiki.secondlife.com/wiki/LSL_Protocol/RestrainedLifeAPI
[6] http://rlva.catznip.com/blog/2009/10/release-rlva-1-0-5/

Why we need this interface actually? LSL (the programming language of SL) offers no possibility by now to move the avatr grid-wide other than by opening the map window. RLV, or to be more exact, the teleport command is the only way to send the avatar without to break the situation.

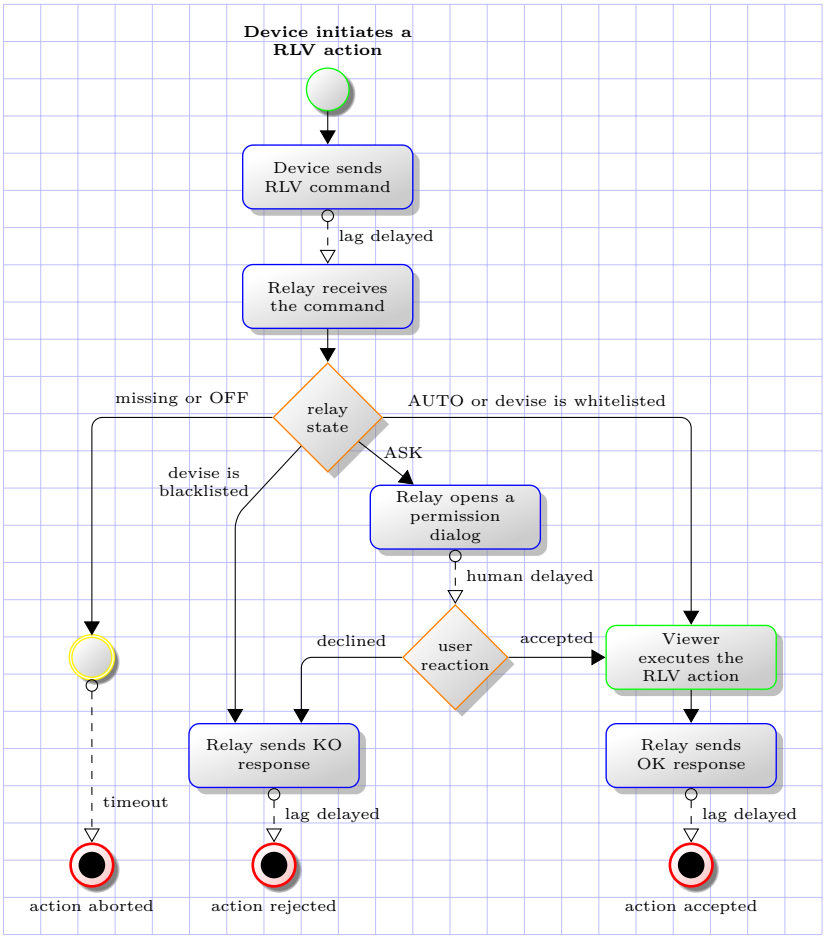


Figure 1.6: Executing of a RLV action, client view

Functionality

From the whole set of RLV commands, the RTS teleporters use only the teleport command and the unsit command. What exactly happens if an arbitrary device sends a RLV command, displays the figure 1.6.

The device initiates the RLV action and sends a chat command naming the target avatar. The command is not directly received by their viewer, but by a special object owned by the agent, a so-called *relay*.

The reason is a security barrier build into RLV protocol: The viewer can not execute commands coming from foreign objects. The teleporters installed overall belong to others. Thus, the relay comes in game.

The relay receives the command and may pass over to the viewer or may not, depends on its security setting. If it is missing or *off*, the device rceives no response and knows, the action is aborted.

Is the relay there but the device is blacklisted, the relay sends the `KO` response and the device detects the action failure. If the device is whitelisted by the relay or the relay is on *auto*, it accepts the command, passes it over and sends the `OK` response. The device detects the action success.

Finally, the relay can be set on *ask*, than it needs the user's decision. For that it opens a menu asking what to do and performs depends on the answer.

Teleportrelay

In the product package there is also an object called '**TP Relay**'. Despite the specification[7] this relay passes over only three RLV commands to the viewer: The teleport command, the version checking command and the unsit command. If no RLV-capable viewer is used, the relay simulates it by sending a faked version response to inworld devises and opening map window for teleports.

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 example teleporters to hang out if the user has no relay in use.

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[8]
- Cool Viewer[9]
- Firestorm and Phoenix viewers[10]

[7] http://wiki.secondlife.com/wiki/LSL_Protocol/Restrained_Love_Relay/Specification

[8] http://www.erestraint.com/realrestraint/, Blog: http://realrestraint.blogspot.com/

[9] http://sldev.free.fr/

[10] http://www.phoenixviewer.com/

- Imprudence[11]
- Rainbow[12]
- Singularity[13]

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

1.1.5 The grid and coordinates

The world of second life is displayed by so-called *sims*. A sim displays a part of the world as a block of 256 meters length and width and unlimited height (but nothing can be build over 4 km height.)

Every position of the sim can be given by a *local coordinate*, the origin is the south-western edge of the sim, the north-eastern edge have the coordinate `<256, 256, 0>` and the position `<128, 128, 1000>` is someplace 1 km above the middle of the sim.

Local coordinates are not distinct, there are as many different points with coordinate `<128, 128, 1000>` as many sims are. To describe a point by a local coordinate clearly, you have to name the sim from which origin the coordinate is measured.

Sims fit together and build a *grid*, it is the world of SL. Some sims join side-by-side and build large connections, the largest are continents. Other sims are single isles. If two sims are in same connection, there is a way between every two positions in the same or different sims. You can move along this way via a vehicle, or a local teleporter or spring over by using a global teleporter like RTS.

Sim isles or different connections are not connected together. Between those sims works only global teleporters.

Sims are always placed closely, without overlapping or distance. Thus, if two sims are joined by side, their origins are on distance of 256 meters. Just knowing this we are able to calculate relative position of two places. But sims have an exactly determined position on the grid: The *global coordinate* of a sim is the coordinate of its origin relative to the origin of the grid. The coordinate is a vector with `Z=0`.

By dividing the global coordinate by 256, one gets the *grid coordinate* of the sim. This is a vector measuring the distance from the origin in sims: `X` gives the number of sims from grid origin in the west-eastern direction and `Y` the number of sims along the south-northern direction. Grid coordinates of two adjacent sims differ in `X` or `Y` exactly by 1.

[11] http://blog.kokuaviewer.org/
[12] http://my.opera.com/boylane/blog/
[13] http://www.singularityviewer.org/

RTS teleporters work with global coordinates of targets. A global position of a point, given locally on a sim, is a vectoral sum of the global position of the sim (its origin) and the local coordinate of the point. The result is an unique vector, no other point on the grid would give the same global position.

Usually, the targets are given by SLURL or LM, we not need to calculate global coordinates of them, this does the teleporter itself. Except for DirectTP mode, than we have to specify the target by global coordinate. Hence we calculate global coordinates of a position exemplarily.

As example we take the sim '**Da Boom**', the first sim in SL at all. We determine the global position of the point **<120, 130, 500>** on the sim. First, we need the global position of the sim. The *Grid Survey*[14] service can help us here. Concretely, we call this url:

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

We receive a list of data, we need two of them: **x=1000**, **y=1000**. This is the grid position of the sim, while **z=0** since the sim is on ground. We turn it into global position by multiplying with 256 and writing as vector: **<256000, 256000, 0>**.

Whats left is to add the local position of the target point and we get our result: **<256120, 256130, 500>**.

The good news is, we don't need to calculate this even in DirectTP mode: If we use one of the targetlist or directtp scripts, than they do it for us. The section 1.4.3 explains how. Only if we want work script-reduced, we have to use the calculator.

[14] http://api.gridsurvey.com

1.2 Configuration

After the scripts inside development kit and the example teleporters were introduced, we start to handle their configuration. First in common, than of each device. Configuration means installing a notecard with the name '**config**' into the system prim and editing it.

1.2.1 Syntax

The format of the notecard describes a simple structuring of the data: The content consists off empty lines, comment lines and parameter-value pairs, grouped by sections, figure 2.7.[15]

```
# Comment line

[Section]
Parameter = Value # Comment
```

Figure 1.7: Configuration example

The char '**#**' and the following rest of the line is seen as comment and ignored.[16] The comment ends with the line, there is no block comment.

Square brackets enclose the section name. Sections separate the configuration data in parts. The partitioning is simple, sections can't be scoped. Spaces around section names are ignored, around the brackets only space char is allowed.

```
[Section]     # Correct
[ Section ]   # Correct, thesame as [Section]
[Section].    # Wrong, the period is not allowed
[Section      # Wrong, ']' is missing
```

Param and values must be both available. Spaces around the param name and value are ignored. Param names are converted to lower case.

```
parameter=Value      # Correct
PaRAmetER   = Value  # Correct, same as 'parameter=Value'
parameter   =        # Wrong, the value is missing
parameter            # wrong, too
            = Value  # Wrong: Param missing, line break not allowed
```

[15] The color highlighting is a service of this document. The notecard editor uses overall a black text color

[16] Hence, no names or values can have this char.

Note: Script can read at last 256 first characters of a line. Some params expect textual values, which can make the line long. Please think about the bound.

Note: Every empty line or pure comment line gives a delay of 0.1s at least while reading the notecard (its LSL.) Hence, to make the configuration run faster please, avoid empty and comment lines.

Comments put after section definition or param value produce no such delay.

1.2.2 Semantics

The content of the notecard builds three main parts, figure 1.8.

```
# Main section
[*MAIN]
# Config data

# Sections for further scripts
[@sensor]
# Config data

# Target sections
[Endora]
# Config data
```

Figure 1.8: Three major configuration parts

- The core script needs the *main section*, '**[*MAIN]**' (any case.)
- Other scripts, if configurable, might need other sections, like the *sensor section* '**[@sensor]**' in example,
- The targetlist script requires at least one *target section*. Those are sections with names starting by a letter or underscore.

The figure 1.9 shows a configuration text, used as basis for configuration of supplied example teleporters. The text of their config notecards are quite long and thus only changes against this text are shown.

```
# RLV Teleporter Configuration
[*MAIN]
user         = all

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

title        = »%n«\n.touch to visit.\n(is your relay on?)
color        = lavender
fadeout      = 0

norelay      = TP Relay (r upper arm)

mode         = RLV
fail         = LM
delay        = 15
return       = LM
offset       = -3

fasttp       = OFF
refresh      = 24
format       = MAPS

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 section
[@sensor]
mode         = touch
distance     = 5-10   # Touch only on distance between 5 and 10 meters

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

Figure 1.9: Base configuration

1.2.3 Main section

The main section has the name '***main**' (case-insensitive), figure 1.9.

Parameter user

```
user = group+!owner
```

The param '**user**' denotes who may use the teleporter, i.e. travel with it. The value can be a list of string flags connected by a '**+**' char. The string flags are shown in table 1.1.

Flag	Meaning
owner	The device owner
!owner	Not the device owner
group	Everyone from the object group
!group	Nobody from the object group
all	Everyone
!all	Nobody

Table 1.1: Flagames, param '**user**'

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**', '**group**' extend the user circle. The flags '**!owner**', '**!group**' reduce it. For example the definition above means: Everyone from the object group can travel with the teleporter but not the device owner.

Picture parameters image, side

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

The param '**image**' denotes the default image to use for targets which not define an image. You can use here an UUID of your texture or a texture in the object inventory. In the last case the texture must be fullperm, otherwise its UUID can't be resolved.

The param '**side**' denotes what face of the image prim should take the target picture, while preview pictures are shown on same face of thumbnail prims. Possible values are shown in table 1.2

Exception: If the setting is '**none**', the target picture is not shown, but the preview pictures are shown on all sides of thumbnail prims.

Flag	Meaning
all	Show picture on all faces
none	The picture is not shown at all
X	Use the face with the LSL-number X, while X is in range 0 to 8.

Table 1.2: Picture face, param '**side**'

Title parameters title, color, colour, fadeout

```
title   = »%n«\n.touch to visit.\n(is your relay on?)
color   = <1.000000, 0.980390, 0.803920> # 'LemonChiffon'
fadeout = 0
```

The param '**title**' defines the title (hover text over the device.) The value can have placeholders, table 1.3.

\n	Line break on this place
\t	Tabulator char
%n	Name of the currently shown zarget
%s	Name of the LM or target sim, if defined via SLURL
%p	Target number in the list, first one has number 1
%c	Amount of targets in the list

Table 1.3: Text placeholders

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 chapter 3. The color name is case-insensitive, '**White**' means the same as '**WHITE**' or '**white**'. In this document we use lower case.

To recognize the color name, as much leading letters are compared as required to distinct the name from others. The significant name prefix is highlighted red in the tables. For example, this line has same effect:

```
colour = LemonCh # 'LemonChiffon'
```

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.

Teleport parameters mode, fail, delay, return, offset

```
mode   = RLV
fail   = LM
delay  = 15
return = LM
offset = -3
```

The value of params '**mode**', '**fail**' and '**return**' is a combination of string flags. Flag names, their meanings and which params accept them is displayed in the table 1.4.

Flag	mode	fail	return	Description
RLV	•			Access RLV of the user
LM	•	•	•	Give the target LM or SLURL
OFF	•	•	•	Do nothing
N	•	•	•	Same as '**OFF**'

Table 1.4: Teleport flags, teleport parameters

The flags are connected via '**+**' and work additively: The value '**LM+OFF**' means the same as '**LM**' or '**lm**': Flag names are case-insensitive.

The param '**mode**' defines how the teleport works, via RLV or by handing out the LM or SLURL. Also, 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.

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 traveler can return.

The param '**delay**' defines how long the teleporter has to await the response of the user's relay until teleport failure is detected. The time in seconds, minimum is 5.

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

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

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. In some cases it is awkward:

Returning avatars would collide with the teleporter and if it is working on collision, it will sent them again, which is not wanted. Also if the teleporter is mount on celling, returning avatars would see themselves falling on the floor. The offset setting could move the return position aside or place it directly on the floor.

Note: The position of the avatar is the middle of its bounding box, this point is roughly one meter above the ground. Hence, the return position must be a meter above the floor.

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 axix, the return point follows the roatation. But it ignores the rotations around x or y axis.

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

Param norelay

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

The param '**norelay**' defines a file to be given if teleport failed because the user has no relay. That can be a notecard, a package with items, or the TPRelay making the teleporter easier to use.

Param fasttp

```
fasttp = OFF
```

The param '**fasttp**' accepts one of values '**N**' or '**OFF**' to deactivate and '**Y**' or '**ON**' to activate the mode. If activated, the teleport starts as soon a target is selected. If the mode is off, we can browse through targets and have to trigger a teleport specially. FastTP allows to build teleporters working with a single click.

Param refresh

```
refresh = 24
```

For teleport, the global coordinate of target location is required. But sims change their position on grid sometimes. After that the global coordinate on the sim is not valid and must be resolved again, i.e. requested by the SL servers.

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 minumum is 6. Sims move very seldom, you can use something like 24 hours.

What means this value exactly? The position is not refreshed permanently, causing lag. Instead, the position is resolved on demand, as soon a teleport to an expired position should start. A smaller value does not increase lag as long the teleporter is not used. If it is used, the smaller the value is, the more often the target location is resolved.

Resolving the position causes a delay of 1 to 2 seconds, the teleporter seems to be inert. If greater value is used, the teleporter seems more fluent, but can easier miss moving of the sim: About half of set time the teleporter caches the wrong position.

Thus, it is advisable to use a value of one day to one week here, i.e. 24 to 168 hours. For the case that the sim is really moved, the owner menu has a 'refresh' button. It invalidates all cached data, so next teleports will resolve them.

Message format

```
format = MAPS
```

The param '**format**' defines the format for sent SLURL. Here you can define one of four values, '**MAPS**', '**SLURL**', '**APP**' and a freetext, table 1.5.

value	description
MAPS	Sends a modern MapURL
SLURL	Sends a classical SLURL
APP	Sends a teleport link
freetext	A text with placeholders %s, %x, %y, %z

Table 1.5: SLURL format

The *SLURL*[17] is classical, all viewers understand the link and open the landmark window if it is clicked in chat history. Web browsers open a simple page showing just the location on the map.

The *MapURL*[18] is only familiar to the newer viewers (version 1.23 and newer,) they show a landmark window if the link is clicked in the chat history. Older viewers

[17] A link like http://slurl.com/secondlife/Endora/123/234/345
[18] A link like http://maps.secondlife.com/secondlife/Endora/123/234/345

open the web browser instead, while web browsers display not only the map, but also special informations.

The *teleport link*[19] is a string, not clickable for web browsers, but if the user clicks the link in chat history, nothing opens than, the viewer just starts an immediate teleport to the specified location.

A freetext is a text sent as is. It may have placeholders, replaced by the sim name, and local coordinates on the sim. For example the format string '`%s (%x, %y, %z)`' would produce a text like this: 'Endora (123, 234, 345)'. This string is not clickable in the chat history, but you can use it to define the log-in location.

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).

Message parameters msg:url, msg:lm, msg:return, msg:norelay

```
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, table 1.3.

The param '**msg:url**' saves the text to introduce the target SLURL. The message for traveler 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. Since instant messages cant embed files, the message and the landmark are sent separately.

The param '**msg:return**' saves the text to introduce the return SLURL. Again, only a single message is sent, this text followed by the SLURL.

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

1.2.4 Sensor section

The sensor script is configured in a section with name '**@sensor**'. If more than one sensor script is installed, each of them needs another sensor section. An example of this section shown in the figure 1.9.

[19] A string like secondlife:///app/teleport/Endora/123/234/345

Sensor mode param

The param **mode** chooses the sensor mode. The value can be one of five, table 1.6.

touch	Touch mode. Clicking person is teleported
collide	Collision mode, Teleport of colliding person
scan	Radar mode. Watches the area, teleports intruders
sit	Sit mode, teleports those who sits on the device
chat	Chat mode. Allows limited device control via chat

Table 1.6: Sensor modes

In touch mode the sensor script reacts on touches of system prim and also all prims tagged by '**sensor**' (section 1.3.2.) In collision mode the script makes the device phantom and reacts on collisions with it, e.g. walking into or over it.

In radar mode the sensor reports for teleports the intruders of observed area. In sit mode the sitting avatars are reported, the teleporter works than like common disks placed on floor to sit on, but our teleporters work grid-wide.

The chat mode is special. In this mode, messages sent via chat are turned to button messages controlling the device control messages (section 1.4.2.) This way we can use the chat line to select a target, to trigger teleport and to control other scripts.

Param	touch	collide	scan	sit	chat	Meaning
mode	•	•	•	•	•	Sensor mode
face	•					Touchable prim face
distance	•		•			Distance to avatar
interval		•	•			Scan interval
angle			•			Scan angle
channel					•	Chat channel
strict					•	React on avatars only?
trigger					•	Command list, TriggerTP
repeat					•	Command list, RepeatTP

Table 1.7: Sensor modes and relevant parameters

Note: Depend on the mode some parameters are not used and can be omitted. The table 1.7 states which mode uses what parameters. For example the sit mode needs no parameter at all, hence this definition is complete:

```
[@sensor]
mode = sit
```

Touchable prim face

```
face = all
```

The param '**face**' is used in touch mode only and defines what face is touchable. The value can be either '**all**' or any number between 0 and 8.

Distance setting

```
distance = 0-15
distance = 15   # means thesame as 0-15
```

The param '**distance**' is used in touch and sensor mode and states the observed distance. The value gives the minimum and maximum distance in meters and can use one of two formats: '**min-max**' or '**max**'.

Scan interval

```
interval = 10
```

The '**interval**' param is used in radar and collision mode and gives a value in seconds. In radar mode this is how often the area is scanned. In collision mode the value 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.

Scan angle

```
angle = 90 # The half sphere in front of the device
angle = PI # The full scan
```

The param '**angle**' is used only for scan mode and determines the observed area: The value gives the room angle around the positive X axis in grad, either as number 1 to 180 or '**PI**' which means 180. Only avatars inside that area are noticed.

Chat channel

```
channel = 77
```

The param '**channel**' is meant for chat mode only and gives the channel for listening by the sensor. Commands said on this channel will be taken as button commands.

The value is a number, while 0 should be avoided: This is public channel for avatar communication. This setting would generate unwanted commands and increases lag. Negative numbers are possible but avatars can't chat on negative channels directly.

Strict mode

```
strict = N
```

The param '**strict**' toggles the strict mode on ('**Y**' or '**ON**') or off ('**N**' or '**OFF**'.) If the mode is on, the sensor accepts only commands sent by avatars.

If the mode is off, also objects can send chat commands. In this case, the object owner is taken as command sender – as if they said the command and not he object. However, this works only if the object owner is available on the sim.

The intention of this mode is making a control HUD, which sends commands if the wearer touched buttons.

Command lists: trigger, repeat

```
trigger = .trigger, go, energy
repeat  = -
```

There are two commands to trigger a teleport: '**.trigger**' and '**.repeat**', that teleport the agent to a currently selected or previously used position respectively. The parameters '**trigger**' and '**repeat**' can replace those commands by keywords or block them.

The parameter value can be a list of keywords, separated by comma. The word said in chat is searched in the lists and only if found, accepted as accordant teleport command. Is a dash '**-**' used as value, the keyword list is empty and the teleport command is blocked.

After the configuration in example above, you can use the chat commands '**.trigger**', '**go**' and '**energy**' to start the teleport via TriggerTP.[20] The command '**.trigger**' is still possible, since it is on the list.

[20]The sensor script sends in each case a message as if the command '**.trigger**' was said

The param '**repeat**' has blocked the command '**.repeat**': The list is empty, nothing said on the chat channel would produce this command.

1.2.5 Target sections

A target section has only three entries: The *target name*, the *target image* and the *target position*, figure 1.10.

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

Figure 1.10: A target section

Target name

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 leading underscore denotes the target as default. At last one target must be default. If more than one section name starts with underscore, only one is used.

A default target is selected automatically after each teleport. This keeps for example the order of shown preview pictures. But than you can not follow the telported avatar by touching the target picture – you visit the default location.

Target image

The param '**image**' is used as target picture, and shown also as preview. The param 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.

Target position

The param '**target**' is required to localize the target. The value can be a LM or a SLRL. If this is a LM, it muss be in in the object inventory. If the LM mode is on, section 1.2.3, the LM is given to the traveler so they can find it in the landmark folder.

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

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

Striped SLURL starts with the sim name and thus it is universal.

1.2.6 Configuration examples

Now we start with configuration of each example teleporter. The configuration text is large, hence we show only changes against the base configuration.

TPSign

```
# TPSign configuration
[*MAIN]
title       = %n\n...touch to visit
# --- Section rest like in base configuration ---

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

Figure 1.11: TPSign configuration

The TPSign is a single prim, figure 1.2(a). The device configuration is shown in 1.11, while parts omitted there are shown in the figure 1.9.

The teleporter uses no sensor plugin, thus the senor section is omitted too. Without the scroll possibility only a single target is possible. If we put more targets here, only the first of them will be used.

For the configuration we have to change the notecard text and reset the teleporter via owner menu. To open the menu, we use the long click. After the new configuration is read, we have to start the teleporter via owner menu, too.

```
# TPVendor configuration
[*MAIN]
side        = 4
title       = Destination %p of %c\n%n\n...touch to visit
# --- Section rest like in base configuration ---

# 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

# --- Further nine targets follow ---
```

Figure 1.12: TPVendor configuration, first 2 targets

```
# TPMirror configuration
[*MAIN]
title       = %n\n...step into and visit
# --- Section rest like in base configuration ---

[@sensor]
mode        = collide
interval    = 10

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

Figure 1.13: TPMirror configuration

TPVendor

The TPVendor has besides the picture prim, also 6 preview prims, the scroll button and logo prim for opening the owner menu, figure 1.2(b). The background prim hides the back side of all prims, thus only the front side (face 4) is used for pictures. The configuration is similar to TPSign as the scripts are the same.

Because of scroll possibility the teleporter can handle multiple targets, hence 11 of them are defined in the configuration. The figure 1.12 shows only first two, but it is possible to define up to 80 targets there.

TPMirror

TPMirror is also a single prim for a single target. It reacts on collision and made to put like a picture on the wall or in the middle of a way, or as carpet on the floor. The travel begins by walking into or over it, figure 1.3(a).

The difference to TPSign is in using a sensor plugin in collision mode. The configuration is shown in 1.13, the sensor section has all required params for the mode.

ChatPorter

```
# ChatPorter configuration
[*MAIN]
side        = 4
title       = Destination %p of %c\n%n\n(to use, say "/111 go")
# --- Section rest like in base configuration ---

[@sensor]
mode        = chat
channel     = 111   # Command with: '/111 cmd'
strict      = N     # User's objects can command, too
trigger     = go    # Magic word is: '/111 go'
repeat      = -     # No way to repeat the prevous tp

# --- Target list like for TPVendor ---
```

Figure 1.14: Chatporter configuration

The ChatPorter is build off four prims, figure 1.4(a): The picture prim and prims for scrolling and owner menu. It supports multiple targets and is aimed on control via chat line, hence it uses the sensor plugin in chat mode. The device configuration is shown in figure 1.14. The channels is 111, strictness is off and the trigger command is replaced by '**go**', while the repeat command is blocked.

TPRadar

```
# TPRadar configuration
[*MAIN]
side        = none
title       = Destination %p of %c\n%n
fadeout     = 60
# --- Section rest like in base configuration ---

[@sensor]
mode        = scan
interval    = 10    # Seconds
distance    = 5     # Meters max.
angle       = 90    # Half sphere around the X-Asis

# Target list without images
[Linden Playground (M)]
target      = Da Boom/128/128/35

[Protected Ocean (G)]
target      = Pravatch/221/227/3

# --- Further nine targets follow ---
```

Figure 1.15: TPRadar configuration

The TPRadar is an area scanner, which teleports every avatar entering the area, to the target location. Preconfigured as a halfsphere, so you can install the teleporter on a wall, roof or floor and it will scan only the room in front of it, figure 1.15.

The teleporter is a multitarget one and has scrollprims and also a logoprim for opening the owner menu, figure 1.3(b). The target pictures are deactivated (the side is set to '**none**',) hence the pictures in the target sections are omitted too to make the configuration run faster.

SitPorter

The SitPorter is a single target teleporter. It is made as a sphere with a target picture rotating inside, figure 1.4(b). For teleport, the avatar has to sit on the sphere. The sensor works here in sit mode, figure 1.16.

Displaying of room particles if the teleporter is started, as well rotating the target picture, is done by driver scripts installed into the device. The driver scripts are here out of scope.

```
# SitPorter configuration
[*MAIN]
title       = %n\n...sit here to visit
# --- Section rest like in base configuration ---

[@sensor]
mode        = sit

# --- Single target like in base configuration ---
```

Figure 1.16: SitPorter configuration

HoloPorter

```
# HoloPorter configuration
[*MAIN]
side        = 2
title       = Destination %p of %c\n%n
# --- Section rest like in base configuration ---

[@sensor]
mode        = touch
distance    = 2   # Be not further 2 meters by touching

# --- Target list like for TPVendor ---
```

Figure 1.17: HoloPorter configuration

The HoloPorter is made as a platform with buttons for scrolling and opening menu. The selected target is shown on a holoscreen above the platform, figure 1.5.

The configuration of the device is shown in figure 1.17. The inner side of the screen prim has the number 2. Other faces of the prim use an invisible texture, but restricting the picture to the inner side in the configuration prevents them from being overwritten.

If the teleporter is clicked by a person standing on the platform, that person is teleported. In order to not teleport avatars standing far from the platform, the sensor works in touch mode while the distance is limited by 2 meters. This way only avatars on the platform are accepted.

1.3 System architecture

The architecture of the RTS is shown in figure 1.18.

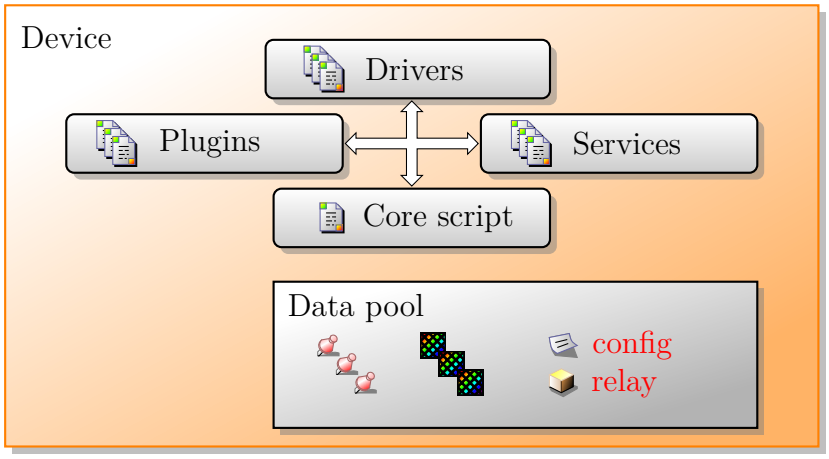


Figure 1.18: RLV Teleporter System architekture

1.3.1 Script classes

Scripts installed in RTS devices are separated in four classes: The *core script*, *service scripts*, *plugin scripts* and *driver scripts*.

The core script and service scripts are device independent and required for every teleport project. Plugin scripts are device-independent, too, but not required for all projects. Driver scripts are for adapting the script system on the device and thus device-dependent and project-dependent.

The Scripts communicate wit each other via linked messages and access to a data pool of landmarks, textures, notecards and other files.

Core script

The core script '**.core**' is required for every teleport project. It performs the teleport, controls other scripts, opens the owner menu and offers core services for special tasks around the device.

The owner menu opens only for the owner and has buttons which reset, start and stop the teleporter and invalidate cached target positions, so they need to be resolved next time.

The core script uses a so-called teleport pipeline, figure 1.19. The pipeline describes chained tasks executed while teleport of a single avatar. By taking the avatar on the pipeline, the teleporter gets ready for teleport next avatar, before teleport of first one is completed.

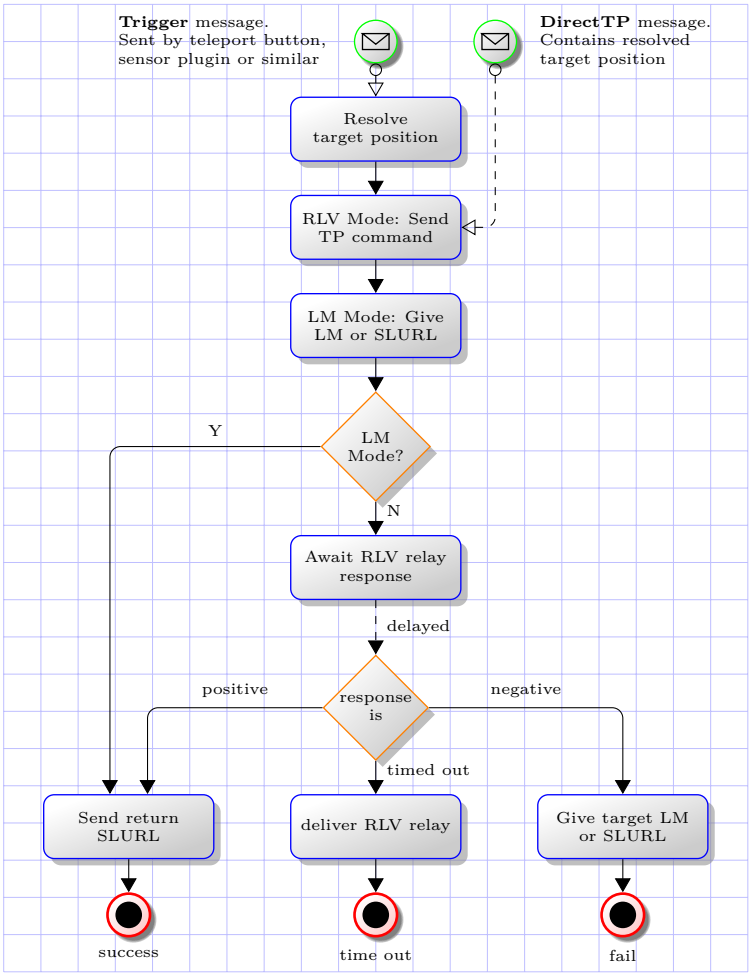


Figure 1.19: Teleport pipeline

Configuration script

The service script '**.config**' is required for all projects. It is the only script with a direct access to the configuration notecard. The task of the script is to read the notecard and to prepare and share the read data via link messages.

This makes the code of other scripts smaller and thus the whole device takes less memory. The configuration data is placed centrally in a single notecard, this simplifies also the maintenance of the device.

The configuration script maintains also a color table, chapter 3, and is able to convert the color names to their vector values: Scripts can't deal with color names.

Sender script

The service script '**.sender**' is required for all projects and is able to send files, instant messages or e-mails on demand.[21] The sender is for example used to send error messages to the owner or landmarks and IMs to the traveller.

The reason to use the script is simple: If a script sends files or IMs, it is delayed by 2 seconds, and by 20 seconds if sending e-mails. By taking this job, the script releases others from that delays.

Sensor script

The plugin script '**.sensor**' is not always required. Its task is to trigger teleport. The core script can do that, too, but only if the device is clicked. The sensor script extends this trigger mode by checking the distance and also adds four further trigger modes: Collision, sit on the device, radar and chat.

In all modes, only *TriggerTP* is triggered. In chat mode the chat commands are taken as button commands, which allows not only to trigger other teleport modes, but also get limited control over the device.

Target list

The plugin script '**.targets**' manages the targets defined in the configuration. It can save up to 80 of them and resolves in runtime their global position. The core scripts needs this data to perform the teleport.

The script reports the global position of the selected target together with images of the target itself and those before and after the selected. This info allows the core script to display thumbnail pictures.

The script allows to select the target in three manners: An absolute position in the target list, a relative to the already selected, and by searching for a target with given name prefix.

Since the script is able to resolve global positions, it takes also another task: Resolving the global coordinate of a position given in DirectTP command. The core script accepts this command only if it gives a global coordinate of the location.

[21] Transmission of e-mails is currently not used.

If the command states local coordinate of the target position, the targetlist script resends the command after resolving the global coordinate of the position.

The script maintains also a so-called sim cache. It saves global positions of 16 sims resolved last time. If the teleporter targets a few sims, the sim cache reduces the server load and time needed to resolve sim positions.

DirectTP Script

The plugin script '**.directtp**' was developed for pure direct teleportrs. Its task is resolving global coordinates for DirectTP commands if given locally. This script replaces the targetlist script, if targets are not set up in the configuration.

Because target management code is cleaned out off the script, this allows to build teleporters that take less script memory compared by those using the targetlist script. The script uses also a sim cache, which takes data of 32 sims.

Driver scripts

Driver scripts are device-depend scrips adapting the script system on the concrete device, or realizing a communication between RTS scripts and the device scripts. Drivers are not always required, but if used, they might be edited for a concrete project.

1.3.2 Core services

The core script offers a number of services: Special tasks around the teleport process and device. This allows the device to run less scripts. However, the manner how the tasks are performed is common and satisfies needs of most projects.

If a concrete project needs to perform this tasks on another wise, this needs a special driver script. With *OptionAPI* offering by the core script, this driver is able to toggle the obsolete core service off, section 1.4.3. Only the unsit service is by default off.

Prim tags

Some services work with certain prims of the device and need to find the prims in link set. Usage of link numbers is not handy, it changes often. Instead, the prims must have special names: The name must start with the '**@**' char, followed by a comma-separated list of prim tags.

For example the text service works with a prim tagged by '**text**'. A prim to display the hover text must have thus the name '**@text**'.[22]

The image service displays target picture on a prim tagged by '**image**'. The prim name is than '**@image**'. Now, if a prim should display both, the target image and hover text, its name must be '**@image,text**'.

You can combine every tag but only tags are compatible which belong to services handling different prim properties. For example the prim name '**@image,thumb**' produces a service conflict: Both services try to set a prim picture. RTS performs no automatical check against service conflicts.

Further scripts can follow this tagging system, too. For example the particle drivers look for pirms tagged by '**psource**' to produce particles on the prim.

Closed tag list

Since version 2.1, RTS allows to close the tag list with the '**@**' char. This way is now possible to tag prims, which names may not start by '**@**' char. For example the name of a menu button must be exactely '**.menu**'. To tag that prim by '**text**', we simply need to use a closed tag list: If the name of the prim is '**@text@.menu**', it is accepted as a menu button and is also used to display the hover text.

To allow this, RTS handles the '**@**' char as a meta separator. This char is used to start and close the tag list, and to separate meta parameters of the prim. Using this char inside a meta parameter (like the tag list or the button name) is no more possible. Also the OptionAPI knows a link message reporting meta parameters of prims, using them. See section 1.4.3 for details and examples.

Unsit service

There is a bug in SL: If a sitting avatar is teleported, the avatar could arrive sitting, the sit animation sticks to the avatar and only a relog can stop it. To prevent this, before teleport the avatar is forced to stand up, by LSL via the '**llUnsit()**' command, as well by a RLV action.

The unsit service is off by default. A script triggering the TP if the traveler is sitting should activate the service; the sensor plugin does it this way if in sit mode.

Text service

If active, the hover text is shown above a prim tagged by '**text**'. If no prim is tagged so, the system prim is used for it. If the service is off, than an other script should show the hover text on its own manner.

[22] Notice please the difference between the tag and prim name.

Message service

This service, if active, relies messages the core script got via NotifyAPI (section 1.4.3) via IMs to the owner. Most of them will be error messages. If another script should take this job, the script should toggle the service off.

Image service

If active, the target picture is shown on a prim tagged by '**image**' on the side as set in the configuration. If more than one prim tagged so, only one of them is used. If no prims are tagged, system prim is used. If the service is off, no image is shown at all.

Thumb service

If active, the core script displays preview pictures on thumbnail prims and also performs clicks on the prims in a way the previewed target is selected and shown by target picture.

Thumbnail prims are those tagged with a *prefix* '**thumb**'. You can tag many prims so, only an even number of them is used. The one half is for thumbnails of targets before the selected one, the other half for targets after it.

The prims are sorted alphabetically by their tag name. This allows to define the order of thumbnail prims while building of the teleporter, you have just to extend the tag prefix by a sortable suffix.

Example: To build a teleporter with 6 thumbnail pictures we can tag all 6 prims with '**thumb**'. But than we can not tell easy in what order they take the preview pictures. To define the order, we just extend the tags by a number: '**thumb 1**', '**thumb 2**' ... '**thumb 6**'. Now the preview order does not depend on the linkset order, which changes every time we link or unlink a prim.

Button service

If active, the sevice turns clicks on certain prims into messages of the ButtonAPI, section 1.4.2. The key param of the messages is the UUID of the clicking person. Prims, the service observes, are called button prims.

A prim is a button prim, if its name starts with a letter or one of three chars '**.**', '*****' or '**_**'. Exceptions: The system prim and a prim with empty name or '**Object**' are no buttons.

Semantics: The period is used for buttons controlling the core script or service scripts. The asterisk for buttons controlling plugins and drivers, the underscore for general control buttons. Buttons with names starting with a letter select a target.

Example: If a button with name '**.menu**' is clicked, the service sends a control message which again opens the owner menu. Buttons with names '**prev**' and '**next**' produce messages, which select previous and next target in the list.

The button name may start by a closed tag list, the tag list is than ignored by the button service. E.g, the prims with names '**@text@.menu**' and '**@psource@prev**' produce the same button message as the prims with names '**.menu**' and '**prev**' but are also used to display the hover text or particle bursts.

However, using a thumb tag on a button prim, e.g. naming a prim '**@thumb 2@.menu**' is not a good idea, the prim with this name is than handled by the thmb service and also by the button service. Both services react on touch. This produces a service conflict and can produce unexpected results.

Touch service

If active, this service converts clicks on the system prim into a '**.trigger**' message. This message is a button message triggering the teleport in TriggerTP mode. The key param of the message is also the UUID of the clicking person.

Clicks on other prims are interpreted, too, but only if the thumb service of button service not interpreted them. The service allows to create a teleporter from a single prim, without to use of sensor plugin.

Menu service

If active a long click on every prim of the device produces a '**.menu**' message, which again opens the owner menu. Long click is if the left mouse button is released more than 1s after it was pushed.

The service is required for a teleporter with a single prim, since there is no other prim opening the menu if clicked: Button service doesn't observes the root prim.

1.4 Interfaces

As stated already, scripts communicate via linked messages, section 1.3. The messages are always limited on the system prim, allowing to install teleporter systems into linked prims of same object. Figure 1.20 displays a map of RTS interfaces.

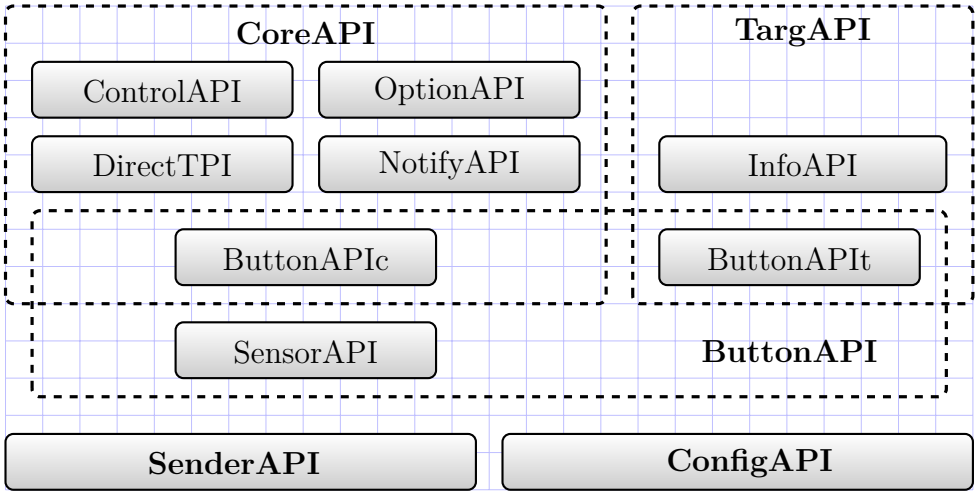


Figure 1.20: API map of the RTS scripts

1.4.1 Format

We use two formats here. In tabular format we introduce each messages of the APIs, and in short format we mention the messages in the text.

Tabular format

The message tables describe each message in detail.

The column '**dir**' denotes the message direction: '**IN**' for pure incoming messages, '**OUT**' for pure outgoing messages and '**BI**' for bidirectional messages (those a script sends and can receive and perform.)

The column '**num**' gives the numeric parameter, the column '**str**' the string param and the column '**key**' the key param of the sent message. If the key is not relevant, '**-**' is used for it. String constants are quoted.

For example the message '**STOP**', table 1.8: **BI**, 0, **"STOP"**, '**-**'. Thus, to stop the teleporter, a script has to execute this LSL command:

```
1 llMessageLinked(LINK_THIS, 0, "STOP", "");
```

The message is bidirectional, it is sent but also executed by the core script, so the teleporter gets stopped correctly. Other for the '`INIT`' command. It is outgoing, the core script sends it but can not perform. Other scripts would reset but not the core script.

Short format

To mention a message inside text, the message is taken into vector brackets: '`<num, str, key>`'. If the key param is not relevant, it is omitted: The message '`STOP`' is referred with `<0, "STOP">`. If the number param is given by context, we can omit it too: `<STOP>`.

An example for that is the Message sequence 1.22. The short format follows also API descriptions headers of supplied driver scripts, while also the message direction is given: '`->[]`' for ingoing, '`[]->`' for outgoing and '`->[]->`' for bidirectional messages.

1.4.2 ButtonAPI

The buttonAPI has a special role. Its messages are user commands, for example clicks on buttons. The messages use the number 1, name the clicked button as string param and the UUID of the clicker as key param.

The messages are also produced by the button core service if buttons are clicked. This allows other scripts to accept control by messages of this sort. This, again allows other scripts emulate such messages to control the first scripts.

For this reason we can not separate the messages of this API completely, neither by the sender, nor by the receiver of the message. Instead, one part of the messages are taken by the CoreAPI and another part by the TargAPI. To denote this, an index '`c`' and '`t`' is added to the API name, figure 1.20.

We just differ between messages for selecting a target (they start by a letter) and control messages (they start by one of control chars '`.`', '`*`' and '`_`'.)

1.4.3 CoreAPI

The CoreAPI keeps messages sent or received by the core script, separated in five subinterfaces, ControlAPI, OptionAPI, DirectTPI, NotifyAPI and ButtonAPIc. They include messages, which are sent or received by core script.

ControlAPI

Messages of this API controls the device or report its status and are sent over the number 0, table 1.8.

dir	num	str	key	Description
OUT	0	"INIT"	-	Resets all scripts
BI	0	"START"	-	Starts the teleporter device
BI	0	"STOP"	-	Stops the teleporter device
BI	0	"TEXT"	Text	Passes over the hover text and displays it via text service
OUT	0	"CMD"	"close"	Owner menu was closed, either via close button or timeout
OUT	0	"TP:INIT"	UUID	Teleport process for an agent with UUID is being initiated
OUT	0	"TP:OK"	UUID	The agent's relay accepted request, the process completed successfully
OUT	0	"TP:KO"	UUID	The relay declined the request, the process is now completed, too
OUT	0	"TP:ABORT"	UUID	Teleport aborted, no response from relay
OUT	0	"TP:ERROR"	UUID	Teleport not started due to errors

Table 1.8: ControlAPI

OptionAPI

This interface is meant for setting up the device and has two messages, `<OPT>` and `<META>` (since RTS 2.1 version), table 1.9.

dir	num	str	key	Description
IN	0	"OPT"	Info	Activates or deactivates core services Info = (+|-)name$_1$,...,(+|-)name$_N$
OUT	0	"META"	Info	Provides a meta parameters of a prim Info = link@meta$_1$@...@meta$_N$

Table 1.9: OptionAPI

The message `<OPT>` toggles core services on and off. The key param provides a comma-separated list of service names with a sign: '`+`' to activate and '`-`' to deacti-

vate the service. If the info string is formatted wrong, an error message with code 10 is generated.

Example: The message **<0, "OPT", "+unsit,-thumbs,-buttons,-text">** toggles the unsit service on and the thumb, button and text services off. Names of flags, their services and default settings gives the table 1.10.

Flag	Core service	By default
unsit	Unsit service	inactive
touch	Touch service	active
text	Text service	active
message	Message service	active
image	Image service	active
thumbs	Thumb service	active
buttons	Button service	active
menu	Menu service	active

Table 1.10: OptionAPI, Option names for core services

The message **<META>** provides meta parameters, set to a prim via the '**@**' char in the prim name. The first meta param is the tag list, the second is the button name. Following parameters are not used by RTS and free to be used by drivers. The message is sent for every prim having '**@**' in name. Examples...

- If a prim with link number 2 has a name '**@text@.menu**', the sent message will be **<0, META, "2@text@.menu">**. Here is a button '**.menu**' tagged by '**text**'.
- If a prim with link number 3 has a name '**@text,psource**', the sent message will be **<0, META, "3@text,psource">**. Only a tag list set.
- If a prim with link number 4 has a name '**text@psource@.menu**', the sent message will be **<0, META, "4@@text@psource@.menu">**. This is a button '**text**' with meta parameters '**psource**' and '**.menu**' and without tags.

ButtonAPIc

The messages of this api start control actions through the user, like opening the owner menu, or starting a teleport, table 1.11. They are performed by core script, thus the index 'c'.

dir	num	str	key	Description
IN	1	".menu"	UUID	Open the owner menu for the owner, if this is their UUID
IN	1	".lm"	UUID	An agent with UUID requests the target LM or SLURL, hand it out
IN	1	".trigger"	UUID	TriggerTP: Teleport the agent with UUID to the selected target
IN	1	".repeat"	UUID	RepeatTP: Teleport the agent with UUID to the target of previous teleport

Table 1.11: ButtonAPIc

DirectTPI

This API has also a single message, table 1.12. The message enforces an immediate teleport and specifies coordinates of the target location. This saves much of management work, since the target is given on-the-fly and not taken from the predefined list.

dir	num	str	key	Description
IN	1	.tptoNXYZ	UUID	*DirectTP* command, either '`<.tpto:X/Y/Z>`' or '`<.tpto:Name/X/Y/Z>`'.

Table 1.12: DirectTPI

The string part of the message must have one of two given formats, while `X`, `Y`, `Z` are global target coordinates on the grid and `Name` is the name of the sim, the given location is.

If X, Y or Z are negative, the command is *invalid* and ignored. The same if X or Y (but not both) are larger 255. If both, X and Y are larger than 255, the position is *global*, and if both, X and Y are between 0 and 255 (inclusive,) the position is *local*. In this case, if the sim name is missing, the actual sim is meant.

The core script executes the command only, if the coordinate is global. If the position is local, it is not executable. But if the targetlist or directtp script is installed, that script becomes active, resolves the global coordinate and resends the – now executable – command than.

Example. We want bring an avatar to the position '`Endora/130/167/45`'. Principally, we could calculate (section 1.1.5) that this position has global coordinates `<146050, 254375, 45>`. Thus, for the teleport we had to send the command `<1, ".tpto:146050/254375/45", UUID>` with the UUID of the avatar.

But we can have it more comfortable and make an installed targetlist script or directtp script to do it. We send this message instead: `<.tpto:Endora/130/167/45>`. For example by using a button or sensor in chat mode. The core script notices, the coordinates are given locally and ignores the command.

Instead, the targetlist or directtp script receives the command, resolves the global position and sends the executable command `<.tpto:Endora/146050/254375/45>`. The core script receives it and starts the teleport.

Why the command has still the sim name? For teleport itself, the sim name is not relevant. But the core script needs it to generate a valid SLURL to the given place, the SLURL can be given to traveler not using RLV so they can visit the place too.

NotifyAPI

dir	num	str	key	Description
BI	-2	Code	Message	NotifyAPI: Internal script message

Table 1.13: NotifyAPI

Scripts produce error, status and other messages as link messages, which again can be forwarded via IM or e-mail. For example by the message service to the owner.

The messages use the number -2. The string param takes the kind of message, e.g. the error code and the key param takes the message content. Both are meant for a human reader and do not use a special semantics, table 1.13.

Example: `<-2, "ERROR 115", "Section already used">`

1.4.4 TargAPI

TargAPI is an interface of the targetlist script. The API has two subinterfaces, *ButtonAPIt* and *InfoAPI*.

ButtonAPIt

This are messages of ButtonAPI, controlling the targetlist script, table 1.14. The key param is not relevant, hence the '`-`' in the specification. But if the messages are sent via button service or sensor plugin, they provide the UUID of the agent touched the button prim or said the command in chat.

dir	num	str	key	Description
IN	1	"first"	-	Select the first target in the list
IN	1	"last"	-	Select the last target in the list
IN	1	"prev"	-	Select the target before the currently selected one. Before the first target, last one comes
IN	1	"next"	-	Select the target after the currently selected one. After the last target, first one comes
IN	1	"prevX"	-	Select the target X positions before the currently selected one. 'prev1' means 'prev'
IN	1	"nextX"	-	Select the target X positions after the currently selected one. 'next1' means 'next'
IN	1	"posX"	-	Select the target on position X in the list, 'pos0' means 'first'
IN	1	"random"	-	Select a target on random position
IN	1	"default"	-	Select a target, specified as default
IN	1	prefix	-	Text search: Select a target, which name in lower case starts with given prefix

Table 1.14: ButtonAPIt

The messages select a target in the predefined target list. If there is no target defined, or a matching target was not found, the messages are ignored. Otherwise the targetlist script sends the `<TARGET>` message of the InfoAPI.

InfoAPI

Messages of this API forward the target data or also control the targetlist, table 1.15.

dir	num	str	key	Description
OUT	2	"LOAD"	Num	Num is the count of found targets
OUT	2	"TARGET"	Data	Transfers the target data as a string
IN	2	"SIZE"	Num	Sets up the number of targets in the window
IN	2	"INVD"	-	Invalidates resolved target position

Table 1.15: InfoAPI

The message `<LOAD>` is sent while configuration after every new target is registered. This visualizes the progress of loading a long target list.

The message `<TARGET>` is sent if a target is selected. The key param of the message takes the data as a list of entries separated by the '`|`' char:

`"tnum|name|spec|lpos|gpos|width|key0|...|keyN"`

The semantics of each value is given in the table 1.16.

index	type	description	
0	integer	tnum	Number of the selected target, first = 0
1	string	name	Target name, e.g. "Home Skybox 2km"
2	string	spec	Name of the LM or sim (from the SLURL)
3	vector	lpos	Local target position (from the SLURL) or `<0.0, 0.0, -1.0>` (if defined via LM)
4	vector	gpos	Global target position (resolved)
5	integer	size	Number of pictures in target window. the window has always a odd number of pictures
6..S+5	key	keyX	Pictures of targets in the shown window, if S the value of size component is. The middle picture is of the selected target

Table 1.16: Target data, resolved

If we got the data in a variable **kData**, we can use this line to extract them:

```
1 list data = llParseStringKeepNulls((string)kData, ["|"], []);
```

We get than a list with content given by table 1.16.[23]

The message **<INVD>** (invalidate) cleans the sim cache and denotes the resolved global position of each target as invalid. After that the targets have to bee resolved again if requested.

The message **<SIZE>** defines the number of displayable pictures. On this way an optimal length of the **<TARGET>** message is set up. The number should be between 1 and 21 and can't be lower than already set one.

Reason: We could use different scripts, which need and set different numbers of preview pictures. Every script can handle too many pictures (and should can) but not too less.

1.4.5 SensorAPI

The sensor plugin '**.sensor**' has no own API, but produces messages of ButtonAPI, section 1.4.2. In the modes click, collision, sit and radar, the script produces only the **<.trigger>** message. In the chat mode, chat commands of the avatar are taken

[23]The data type of each list entry is string and must be casted to shown data type.

as button messages over the avatar. Hence, every message of the ButtonAPI is possible.

1.4.6 SenderAPI

The sender script sends files, instant and e-mail messages to given receivers. The messages of this API use numbers 11, 11 and 12, and command the script to perform sending operation, table 1.17.

dir	num	str	key	Description
IN	10	File	UUID	Gives a file to an agent given by UUID
IN	11	Msg	UUID	Sends message as an IM to the agent
IN	12	Text	e@mail	Sends the text as an e-mail to the address

Table 1.17: SenderAPI

1.4.7 ConfigAPI

The config script has the task to read the config notecard, prepare and pass over the read data. The format of the notecard uses a simple structure: The text consists off empty lines, comments and param-value pairs, grouped by sections, figure 1.21.

```
# Comment line
# Another comment line

[*MAIN]
accept  = group # Everyone from the group
color   = white

# Target definition
[Endora]
image   = 8cee492c-917e-341d-b3a9-63d83ed44d34
target  = Endora/127/169/42
```

Figure 1.21: Configuration example

The char '**#**' and the rest of the line is a comment. If the configuration is read, all comments and empty lines are ignored, also lines that became empty after removing comments.

Square brackets enclose section names. The section names are reported with a running number. This allows principally to read each section repeatedly and to

define much more target sections than 80: Is a target is requested and no more in cache, we simply read the defining section again. But because of lag this idea is not used.

Params and values are reported without enclosing spaces, parameter names are lower cased. Furthermore, values of parameters '`color`' and '`colour`' are parsed, the color names are converted to accordant vector values.

The complete API of the configuration script

The messages use the number -1 and are shown in the table 1.18.

The ingoing message `<CONFIG>` starts the configuration over the specified notecard. If '`Note`' is an UUID, the notecard behind it will be used. If its not an UUID, a notecard in object inventory with this name is used. If '`Note`' is an empty string or `NULL_KEY`, a notecard with name '`config`' is used. The configuration process is reported with the outgoing `<CONFIG>` mesage.

The ingoing `<READ>` message starts the repeated reading of a section with given number. In order the script knows where to find this section it has to have completed the configuration once normally. The read process is reported by the outgoing `<READ>` message.

The message `<[Name]>` is sent once per section, but not if a section is read repeatedly. The section data are sent via `<Param>` messages.

Example. Say, we config over a notecard which content is shown in figure 1.21. The config process produces a message sequence as shown in the figure 1.22.

Error messages

The config script sends also error mesages, as introduced in section 1.5.

- If a configuration is started but the notecard is missing, an error message with the code 120 is sent.
- If this is not a notecard, an error message with code 121 is a result.
- A syntax error in the configuration data produces an error message with code 100.

Only lines without syntax errors produce configuration messages.

dir	num	str	key	Description
IN	-1	"CONFIG"	Note	Start configuration using the given notecard
OUT	-1	"CONFIG"	"START"	Configuration started, wait 0.25s before sending further messages
OUT	-1	"CONFIG"	"END"	Configuration complete, wait 0.25s before sending this message
IN	-1	"READ"	Num	Start repeated reading of a section with given number
OUT	-1	"READ"	"START"	Repeated reading of the section begins
OUT	-1	"READ"	"END"	Repeated reading ends up
OUT	-1	[Name]	Num	A section with the name and running number is being read. Waits 0.2s before sending further messages.
OUT	-1	Param	Value	The param-value pair is being read. Spaces around param and value are striped, param name is lower cased

Table 1.18: Configuration script, ConfigAPI

```
        <-1, "CONFIG", "START">
0.25s
        <-1, "[*MAIN]", "0">
0.2s
        <-1, "accept", "group">
        <-1, "color", "<1.00000, 1.00000, 1.00000>">
        <-1, "[Endora]", "1">
0.2s
        <-1, "image", "8cee492c-917e-341d-b3a9-63d83ed44d34">
        <-1, "target", "Endora/127/169/42">
0.25s
        <-1, "CONFIG", "END">
```

Figure 1.22: Message sequence, configuration by 1.21

1.5 Error codes

System scripts produce error messages, which are forwarded to the owner via message service. The table 1.19 displays the error codes and their reasons.

Code	Description
10	Error by setting the options, section 1.4.3: The message to toggle core services has a wrong format.
22	A script has wrong permissions, i.e. copy+trans.
100	Syntax error in the configuration, e.g. a param without a value.
110	A param name is unknown for this section. Typo or wrong section?
111	The param value is wrong. Used wrong flags, or value out of bounds.
112	Violation of data limits, e.g. too many target sections.
114	A param is missing which must be in this section.
115	A target name is used twice in configuration.
116	A script miss a required section.
120	A file missed in inventory. Configuration: A texture or LM is named but not found. Run time: A LM is no more in inventory, eventually deleted.
121	Wrong data type, e.g. picture instead LM used.
122	Wrong permissions, e.g. a picture is not fullperm.
123	No access to the content, e.g. a picture is given by a key but the key is not a key.
133	Target position cant be resolved (any more).

Table 1.19: Error messages

Messages with the code less than 100 are for integration problems and give names of scripts making problems. Messages with code 100 are sent by configuration script and give the line number where error was found.

Messages with larger error code are sent by other scripts. They have no access to the config notecard and give as localization the name of a section where the problem occurs.

2 Deutsch

2.1 Einleitung

Das RLV Teleporter System (kurz *RTS*) ist ein Framework zum Aufbau von Teleportergeräten. In SL existieren grundsätzlich zwei Arten von Teleportsystemen, lokale und globale Teleporter. Lokale Teleporter bewegen den Avatar zur Zielposition, funktionieren deshalb nur wenn der Teleportweg nicht über einen simlosen Weltbereich verläuft.

Bei *globalen* Teleportern meldet sich der Viewer an der aktuellen Sim ab- und an der Zielsim an. Das funktioniert über jede Entfernung. Vor RLV haben nur *Kartenteleporter* so funktioniert. Sie öffnen dem Benutzer die Weltkarte mit der ausgewählten Zielposition und dem 'Teleport' Button. Durch dieses Fenster wird man aus der Umgebung gerissen, die Immersion ist zerstört.

RLV Teleporter übermitteln dagegen den Teleportbefehl über die RLV Schnittstelle an den Vewer. Ist der Befehl akzeptiert, führt der Viewer den Teleport aus, ohne das Kartenfenster zu öffnen. Diese Teleportart ist nur mit einem Viewer möglich, der die RLV Schnittstelle implementiert. Das RTS gehört zu dieser Kategorie. Im Folgenden seien einige wichtige Features des Systems vorgestellt.

RTS funktioniert *gridweit*. Jeder Ort ist erreichbar, für den eine Landmarke oder SLURL existiert. Ist allerdings ein Landepunkt auf dem Zielort aktiv, fängt dieser den Teleport ab. Also genau wie beim Benutzen einer Landmarke. RTS funktioniert allerdings auch lokal, auf dieselbe Weise.

Das System ist *mehrzielfähig* und unterstützt drei *Teleportmodi*: Beim *TriggerTP* Teleportmodus ist das Teleportziel eines aus der Liste von Ziele die in der Gerätekonfiguration angegeben sind. Beim *DirectTP* Teleportmodus wird das Teleportziel direkt im Teleportbefehl selbst angegeben. Schließlich kann ein Zielort mittels *RepeatTP* erneut bereist werden, etwa um einem Avatar zu folgen.

Die *FastTP*-Technik erlaubt den Teleport mit nur einem Klick. Das RTS unterstützt vier weitere *Auslösemodi*: Kollision, Setzen, Raumscan und Chat.

Die Systemarchitektur ist *modular* und *erweiterbar*: Durch Auswahl der Skripte können unterschiedliche Teleporterprojekte realisiert werden. Offene *Schnittstellen* ermöglichen Erweiterung um weitere Skripte.

Linkrobuste Primsemantik: Prims spielen spezielle Rollen, z.B. Buttons, Vorschauprims etc. Skripte arbeiten mit Namen der Prims, anstatt der Linknummer. Dadurch ist das Teleportsystem unempfindlich gegen Veränderungen am Linkset.

Umfangreiche *Konfiguration*: Konfigurationsdaten werden in einer Notekarte festgehalten und in Sektionen unterteilt. Dies erlaubt eine leichtere Handhabung von Daten und Erweiterbarkeit des Systems, wenn verschiedene Sektionen Daten für verschiedene Skripte enthalten.

2.1.1 Zielgruppe

Für wen ist dieses Produkt geeignet? Nun, suchen Sie einen Teleporter, der Sie einfach zum anderen Ort der Sim bringt, etwa ein Aufzug zur nächsten Etage, dann ist RTS wahrscheinlich nicht für Sie. Es gibt viele Teleporterskripte, die diese Arbeit auch tun, vielleicht sogar beser als RTS.

Suchen Sie aber einen Teleporter für Orte auf der ganzen Grid, ob ein Einzelgerät, oder ein Ring von Partnerteleportern, dann könnten Sie RTS nützlich finden, sogar wenn Sie den RLV oder kompatiblen Viewer nicht einsetzen.

Sind Sie mit Skripten und Bauen in SL vertraut? Wenn nicht, kein Problem: Im RTS Paket finden Sie einige Beispielteleporter. Die sind einsatzfertig vorkonfiguriert, Sie brauchen lediglich die Ziele in der Konfiguration durch Erwünschte zu ersetzen. In diesem Fall können Sie auch diese Anleitung nur bis zum Abschnitt 'Konfiguration' (2.2) lesen.

Wissen Sie aber, wie man baut und skriptet, können Sie die Teleportgeräte auch einfach auf Ihre Bedürfnisse anpassen oder mit eigenen Skripten erweitern. Oder aber Teleporter im ganz anderen Style mit RTS als Kernsystem aufbauen. In diesem Fall benötigen Sie die ganze Anleitung und auch das Paket *RTS Extensions*, das kostenlos verfügbar ist.

RTS ist in zwei Versionen verfügbar, die Personal- und Pfrofessionaledition. Wollen Sie die Teleporter selbst aufstellen und Freunde, Gäste und Fremde damit reisen lassen, dann ist die Personaledition ausreichend für Sie. Wollen Sie die Teleporter weitergeben, oder aber verkaufen, dann benötigen Sie die Pfrofessionaledition.

2.1.2 Struktur

RTS gehört zu einem Bundle von zwei Produkten: *RTS* und *RTS Extensions* (Erweiterungspaket.) RTS (dieses Produkt) enthält den Entwicklungskit und Beispielteleporter, die mit diesem Kit aufgebaut wurden. Diese Dokumentation erklärt, wie Teleportersysteme konfiguriert und benutzt werden. Sie spezifiziert auch Schnittstellen, welche die RTS Skripte zur Kontrolle und Interaktion anbieten.

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

Auf die *Konfiguration* des Teleportsystems und der mitgelieferten Beispielteleporter geht ausführlich der Abschnitt 2.2 ein. Im Abschnitt 2.3 wird die allgemeine *Systemarchitektur* vorgestellt, die Systemskripte und wie sie das Teleportsystem bilden.

Die *Schnittstellen* dieser Skripte werden wiederum im Abschnitt 2.4 spezifiziert. Der Abschnitt 2.5 gibt *Fehlermeldungen* wieder, die während der Konfiguration und im Betrieb der Teleportgeräte produziert werden.

Schließlich gibt der Kapitel 3 Farbpaletten an, die dort aufgelistete Farben lassen sich in der Konfiguration auch mit Namen angeben.

Das Paket *RTS Extensions* behandelt Entwicklung von Teleportprojekte mittels des Entwicklungskits und weiteren Skripte. Die Paketdokumentation beginnt wo diese aufhört und enthält eine Schritt-für-Schritt Anleitung, wobei das Paket selbst Resources enthält, die im Tutorial verwendet werden.

2.1.3 Paketinhalt

Das Lieferungspaket besteht neben Begleitdateien aus einem Entwicklungskit, Beispielprojekten und Hilfsdateien. Im Produktpaket selbst sind alle Dateien ungeordnet enthalten. Das Paket kann sich aber automatisch in Ordnern auspacken, wie im Abschnitt 2.1.3 beschrieben.

Das Entwicklungskit enthält in den Ordnern '**System**' und '**Drivers**' die System- und Treiberskripte zum Aufbau von Teleportergeräten. Beispielprojekte sind fertig aufgebaute und vorkonfigurierte Teleporter, Ordner '**> Projects**'.

Das führende Zeichen '>' ist ein Indikator für weitere Unterverzeichnisse. Beim Ausliefern enthällt der Ordner allerdings nur Objekte – die Beispielteleporter. Die Idee hinter dem Ordnernamen ist, Ihnen einen Platz für weitere Teleporterprojekte zu geben, sortiert durch Unterordner.

Hilfsdateien im Ordner '**Resources**' bestehen aus einem Paket mit Sounds und Animationen, einem Bildmanager und einem Farbmanager. Der Bildmanager ist mit einigen Bildern geladen und kann weitere Bilder aufnehmen. Der Farbmanager erleichtert die Farbauswahl und kennt sich mit den Farbnamen aus, Kapitel 3.

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. Das funktioniert zwar überall, der Paketinhalt landet aber in einem Ordner ohne jede Struktur.

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 '**RLV Teleporter System**', so wie vier Weiteren: '**> Projects**', '**Drivers**', '**Resources**' und '**System**'. Leider können Skripte keine verschchtelte Verzeichnisstrukturen übergeben. Bitte verschieben Sie deshalb die vier Ordner ins Produktverzeichnis, so dass es aussieht, wie in 2.1 abgebildet.

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. Ignorieren Sie das Menü, zerstört die Box sich automatisch in 30 Minuten.

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.[1]

Update

Das RTS Produkt 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.[2]

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

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

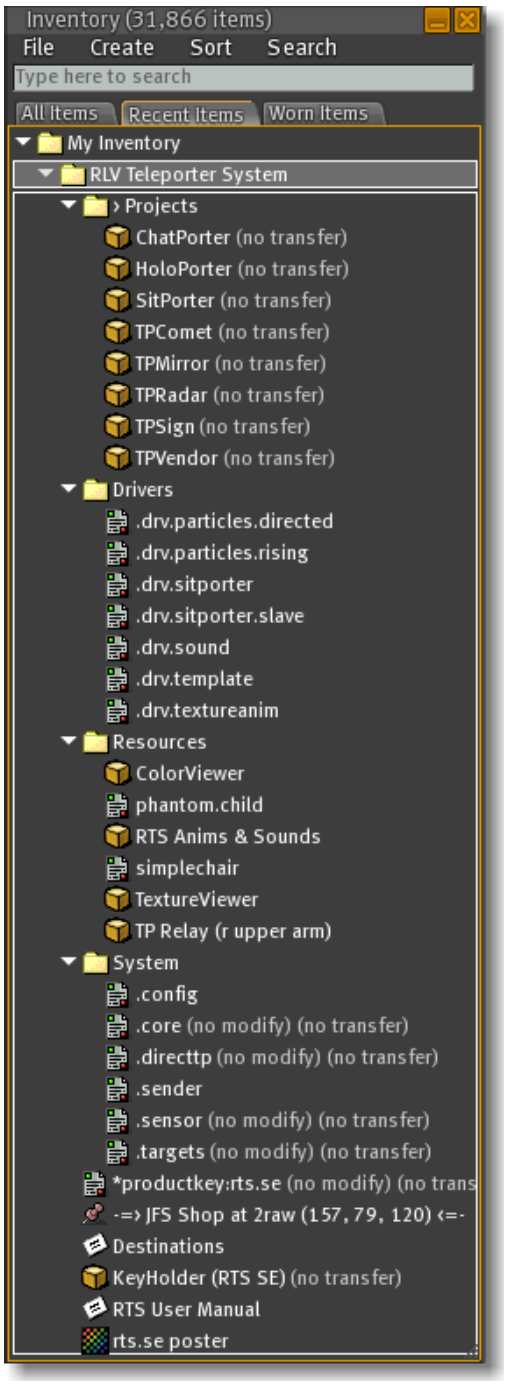


Figure 2.1: Paketinhalt, automatisch ausgepackt

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.

In beiden Fällen ist das Update eine Nachlieferung des aktuellen Produktpakets. Es steht ebenfalls ein Onlinetutorial zum Updaten von JFS Produkten zur Verfügung.[3]

Entwicklungskit

Das Entwicklungskit enthält System- und Treiberskripte zum Aufbau der Teleportgeräte. Die Systemskripte stellen ein 'Betriebssystem' dar. Treiberskripte passen dieses auf das jeweilige Gerät an und sind geräteabhängig, daher auch quelloffen. Per Konvention fangen ihre Skriptnamen mit '**.drv.**' an.

Der *Kernskript* '**.core**' spielt eine zentrale Rolle des Teleportersystems und ist für alle Teleporterprojekte erforderlich. Er führt den eigentlichen Teleport durch und bietet Kernservices um das Teleportergerät an.

Der *Konfigurationsskript* '**.config**' ist für alle Teleporterprojekte erforderlich. Er zentralisiert die Gerätekonfiguration, indem er die Konfigurationsnotekarte ausliest und entsprechende Daten weiterleitet.

Der *Senderskript* '**.sender**' ist für alle Projekte erforderlich. Seine Aufgabe ist es, Dateien und Mitteilungen an Avatare zu versenden, beispielsweise Fehlermeldungen an den Besitzer.

Der *Sensorskript* '**.sensor**' ist nicht immer erforderlich. Er dient dem Auslösen des Teleports, verbessert das Auslösen per Klick (das kann der Kernskript) und fügt Auslösemodi via Kollision, Setzen, Radar und Chat hinzu.

Der *Ziellistenskript* '**.targets**' ist erforderlich, wenn der Teleporter mit einer Liste von in der Konfiguration angegebenen Zielen arbeitet. Der Skript speichert bis zu

[3] http://jennassl.blogspot.com/2011/11/updating-jfs-products.html

80 dieser Zielangaben und ermittelt im Betrieb die für den Teleport erforderliche globale Position des ausgewählten Ziels.

Der Skript löst außerdem die globale Zielposition eines im *DirectTP*-Befehl angegebenen Ziels auf. Das ermöglicht es, Direktteleporter zu bauen, ohne Globalkoordinaten berechnen zu müssen.

Der *DirectTP* Skript '**.directtp**' ist hilfreich, wenn der Teleporter nur via DirectTP eingesetzt wird und nicht mit einer Liste von Ziele konfiguriert wird. Der Skript löst lediglich die globale Zielposition eines im *DirectTP*-Befehl angegebenen Ziels auf.

Der *Soundtreiber* '**.drv.sound**' spielt Geräusche ab, wenn das Teleportergerät gestartet und angehalten wird, oder der Teleport startet und beendet wird.

Die *Partikeltreiber* '**.drv.particles.rising**' und '**.drv.particles.directed**' erzeugen während des Teleports die vom Gerät aufsteigende bzw. zum Reisenden gerichtete Partikel.

Der *Animationstrieber* '**.drv.textureanim**' setzt und animiert die Oberflächentextur entsprechend des Geräte- und Teleportstatus.

Schließlich ist das *Treibertemplate* '**.drv.template**' verfügbar, das die meisten Systemschnittstellen unterstützt und leicht zu einem passiven Treiberskript erweitert werden kann.

Teleportergeräte

Der Projektordner enthält sieben Teleportergeräte. Sie sind so vorbereitet, dass sie sofort nach Auspacken einsetzbar sind. Einige Teleporter haben Scrollprims und ein Logoprim fürs Besitzermenü.

Bei Teleportern ohne Logoprim lässt sich das Menü nur per Langklick öffnen: Den Teleporter mit der linken Maustaste anklicken und die Maustaste erst nach einer Sekunde loslassen. Der Langklick funktioniert bei allen mitgelieferten Teleportern.

TPSign

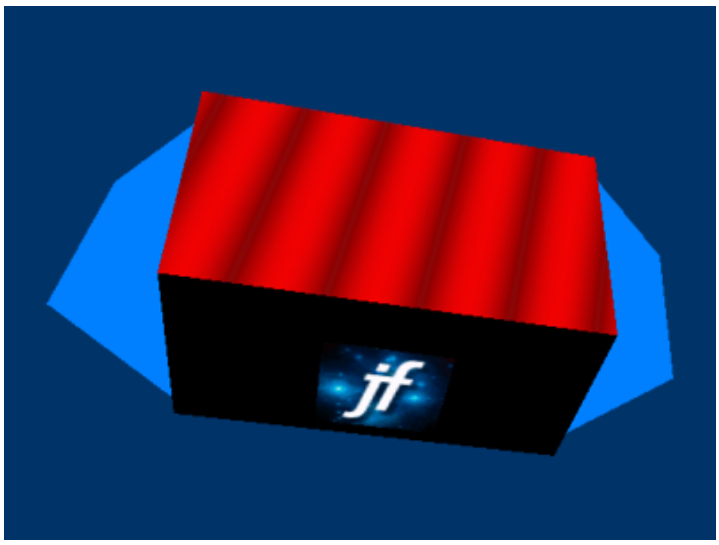
TPSign belegt nur ein Prim, Abbildung 2.2(a) und kommt ohne den Sensorskript aus: Der Teleport startet beim Anklicken des Bildes. Wegen fehlender Scrollmöglichkeit ist das ein Einzelzielteleporter, allerdings wird zum Verwalten dieses Ziels der Ziellistenskript verwendet, der Teleporter ist potenziell mehrzielfähig.

(a) TPSign

(b) TPVendor

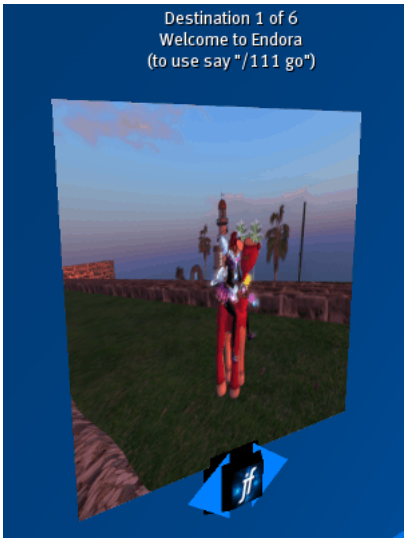
Figure 2.2: Klickteleporter

(a) TPMirror

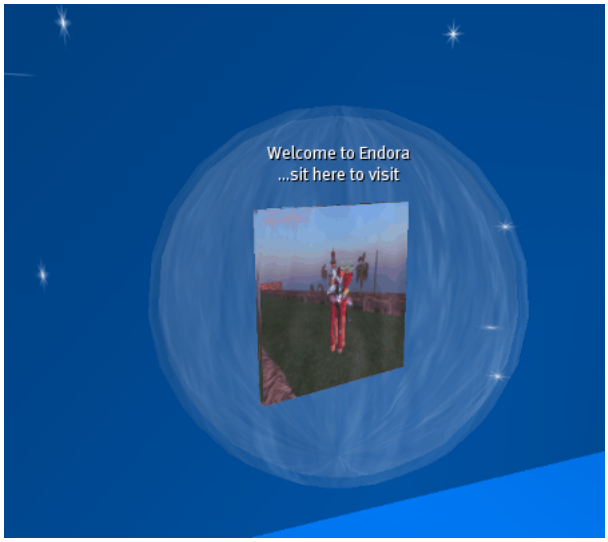


(b) TPRadar

Figure 2.3: Scannende Teleporter



(a) Chatporter



(b) Sitporter

Figure 2.4: Reagierende Teleporter

Figure 2.5: Der Holoporter

TPVendor

TPVendor belegt 11 Prims und erweitert den TPSign um Vorschaubilder und Buttons zum Scrollen und Aufruf des Besitzermenüs, Abbildung 2.2(b). Durch Scrollbuttons ist der Teleporter mehrzielfähig und wird mit elf Zielen vorkonfiguriert.

Das Teleportziel wird durc Ankicken des Vorschaubildes oder Scrollbuttons gewählt. Der Teleport startet beim Anklicken des Zielbildes. Der Teleporter kommt ebenfalls ohne den Sensorskript aus (tatsächlich enthalten TPSign und TPVendor gleiche Skripte.)

TPMirror

Der TPMirror belegt ein Prim, Abbildung 2.3(a) und ist zum Aufhängen auf einer Wand oder als ein Vorhang gebaut. Der Teleport startet beim Hineingehen ins Bild. Die Kollisionserkennung übernimmt der Sensorskript im Kollisionsmodus. Der Teleporter kann auch als Teppich arbeiten, der Avatare beim darüber Laufen teleportiert.

TPRadar

Der TPRadar ist aus vier Prims gebaut um unauffällig auf einer Wand oder am Boden zu befestigen, Abbildung 2.3(b). Bitte vorsicht beim Auspacken, das Gerät ist (5 x 5 x 15) cm^3 klein. Das Zielbild wird gar nicht angezeigt, der Text nicht dauerhaft.

Der Teleporter überwacht den Raum und teleportiert Eindringlinge zum eingestellten Ziel, dafür ist der Sensorskript im Scanmodus verantwortlich. Das Menüprim dient dabei auch als Partikelquelle: Der Treiberskript erzeugt beim Teleport eine Partikelsalve richtung entdeckten Avatare.

Chatporter

Der Chatporter ist als ein Wandbildschirm mit einem Zielbild und Scrollbuttons aufgebaut (4 Prims insgesamt), Abbildung 2.4(a). Der Teleport startet, wenn der Reisende den Befehl '**go**' auf dem Chat-channel **111** gibt, also '**/111 go**'. Dafür ist der Sensorskript im Chatmodus eingesetzt.

Hier können Sie die Stärke dieses Sensormodus kennenlernen: Ist der Teleporter gestartet, funktionieren neben dem Befehl '**go**' (Teleport) auch '**.menu**' zum Aufrufen des Besitzermenüs, '**prev**' und '**next**' zum Blättern durch Ziele. Auch die die Zielsuche ist möglich: Der Befehl '**/111 lind**' wählt das Ziel aus, dessen Name mit 'lind' anfängt, in diesem Fall 'Linden Playground'.

Außerdem können Sie mit '`/111.tpto:HappyMood/179/161/26`' einen (lokalen) DirectTP - Befehl absetzen, der Teleporter ermittelt die globale Zielposition und bringt Sie (als sagender) zum angegebenen Ort.

Sitporter

Der Teleporter ist als eine Kugel ausgeführt, in der sich das Zielbild dreht, Abbildung 2.4(b). Der Teleport startet, wenn der Reisende sich auf das Gerät setzt. Das Sitzen erkennt der Sensorskript im Sitzmodus. Es sind keine Scrollmöglichkeit eingebaut, deshalb verwaltet der Teleporter ebenfalls nur ein Ziel.

Holoporter

Mehrzielfähig, Abbildung 2.5, ist mit 6 Prims aufgebaut – und zwar als eine Plattform, über der das Zielbild holografisch angezeigt wird. Zum Teleportieren muss sich der Reisende auf die Plattform stellen (einfach darauf treten, es ist kein Posestand) und den Teleporter anklicken. Zum Feststellen, dass nur die Person auf der Plattform den Teleporter anklickt, ist der Sensorskript im Klickmodus mit einer Distanzbeschränkung eingesetzt.

TPComet

Im Paket findet sich auch ein Bonusteleporter. Er wurde zum Aufnehmen des Produktbildes erstellt (Titelseite dieses Dokuments), funktioniert aber dennoch. TPComet ist ein Direktteleporter: Zielpositionen sind durch Zielbilder definiert (d.h. Namen der Prims) und nicht in der Konfigurationskarte.

Um den Teleporter zu erklären, braucht es Kenntnisse der Systemarchitectur und der Kernservices. Um Zielpositionen zu ändern, muss man wissen wie man Prims bearbeitet. Deshalb wird der Teleporter nicht hier vorgestellt, sondern erst in der Dokumentation zum Erweiterungspaket.

2.1.4 RLV, RLVa und Immersion

RLV[4] ist ein spezieller Viewer, der mit dem Ziel entwickelt wurde, Immersion (das Gefühl, im Spiel zu sein) des Benutzers zu unterstützen. Um RLV zu verstehen, brauchen wir gar nicht die Szene betreten, in der dieser Viewer entstanden ist. Stellen Sie sich zum Beispiel eine Spielsituation vor, wo man von einem bösen Zauberer zum Stein verwandelt wird.

[4] Ursprünglich 'Restrained Life Viewer', später 'Restrained Love Viewer'

Normalerweise muss man jetzt so tun, als wäre man ein Stein: Nicht bewegen, nichts sagen, und bloß nichts anklicken. Das Gefühl, ein Stein zu sein ist nicht da. Immersion beginnt aber schon, wenn man die Standardmittel von SL benutzt: Ein Attachment, das die Pose einfriert und die Bewegung via Pfeiltasten ausschaltet. Mehr geht hier aber nicht.

Der RLV geht weiter: Er erlaubt diesem Attachment, den Chat zu blockieren, das Anklicken von fernen Gegenständen zu deaktivieren und das Ablegen des Steinattachments wird auch nicht mehr möglich sein. Man tut nicht mehr so, als wäre man versteinert, man fühlt sich auch so.

Der Standardviewer ist spielsicher entwickelt. Seine Fähigkeiten werden zu keinem Zeitpunkt irgendwie eingeschränkt, und es werden nie unerwartete Aktionen ausgeführt. Das ist zwar sicher, ist für die obige Spielszene nicht optimal.

Anders der RLV: Hier können Skripte bestimmte Fähigkeiten ausschalten, wie etwa Chat, Ablegen von Attachments und bestimmte Aktionen ausführen, etwa von einem Objekt aufstehen oder auf *eine Koordinate teleportieren*. Dazu greifen die Skripte auf eine spezielle Schnittstelle des Viewers zu.

Ursprünglich wurde diese Schnittstelle nur durch 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.

Warum benötigen wir aber diese Schnittstelle? LSL (Programmiersprache von SL) bietet keine Möglichkeit, den Avatar gridweit zu bewegen, außer über das geöffnete Kartenfenster. RLV, genauer sein Teleportbefehl ist derzeit der einzige Weg, den Avatar zu versenden, ohne dafür die Spielsituation zu zerstören.

Funktionsweise

Von der ganzen Palette der RLV Befehle werden bei RTS Teleportern nur der Teleportbefehl und der Unsitbefehl verwendet. Was genau passiert, wenn ein (beliebiges) Gerät einen RLV-Befehl absendet, stellt die Abbildung 2.6 dar.

Das Gerät initiiert eine RLV Aktion und versendet dazu einen Chatbefehl mit Angabe des Zielavatars. Der Befehl wird nicht direkt vom Viewer empfangen, sondern von einem Objekt, das seinem Benutzer gehört, das sogenannte *Relay*.

Der Grund ist die im RLV Protokoll eingebaute Sicherheitsschranke: Der Viewer kann keine Befehle von Fremdobjekten ausführen, die Teleporter inworld sind das

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

aber meistens. Deshalb springt das Relay ein, empfängt den Befehl und kann an den eigenen Viewer weiterleiten.

Ob es dies tut, entscheidet seine Sicherheitseinstellung. Fehlt das Relay oder ist aus, erhält das Inworldobjekt keine Antwort und stellt den Aktionsabbruch fest.

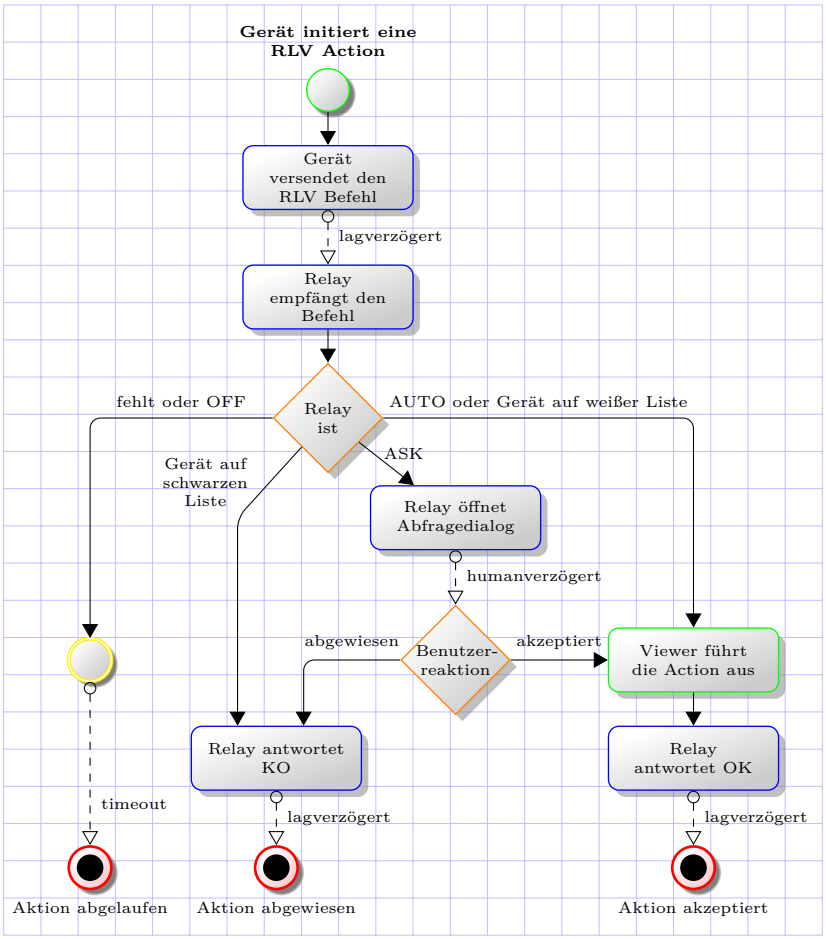


Figure 2.6: Ausführung einer RLV Aktion, clientseitig

Ist das Relay da, das Gerät ist aber auf seiner schwarzen Liste, versendet das Relay sofort `KO` als Rückmeldung. Daran erkennt das Gerät, die Aktion wurde abgewiesen.

Ist das Gerät auf der weißen Liste des Relays oder ist letzteres auf Auto gestellt, akzeptiert es den Befehl sofort, leitet ihn an den Viewer weiter und versendet `OK` als Rückmeldung. Das Gerät erkennt, die Aktion wurde angenommen.

Schließlich kann das Relay auf Ask eingestellt sein, dann fragt es um Benutzererlaubnis. Dazu öffnet es ein Dialogfenster und verfährt je nach der Antwort.

Teleportrelay

Im Produktpaket befindet sich auch ein Objekt namens '**TP Relay**'. Entgegen der Spezifikation[7] leitet dieses Relay nur drei Befehle an den Viewer weiter: Den Teleportbefehl, die Versionsprüfung und den Unsitbefehl. Wird kein RLV-fähiger Viewer benutzt, simuliert das Relay den RLV, indem es eine gestellte Versionsantwort an das Teleportergerät versendet und zum Teleportieren das Kartenfenster öffnet.

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

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[8]
- Cool Viewer[9]
- Firestorm und Phoenix viewers[10]
- Imprudence[11]
- Rainbow[12]
- Singularity[13]

Die Liste dient dem Überblick und wird nicht vollständig sein.

[7] http://wiki.secondlife.com/wiki/LSL_Protocol/Restrained_Love_Relay/Specification
[8] http://www.erestraint.com/realrestraint/, Blog: http://realrestraint.blogspot.com/
[9] http://sldev.free.fr/
[10] http://www.phoenixviewer.com/
[11] http://blog.kokuaviewer.org/
[12] http://my.opera.com/boylane/blog/
[13] http://www.singularityviewer.org/

2.1.5 Die Grid und die Koordinaten

Die Welt von Second Life wird durch sogenannte Sims dargestellt. Eine *Sim* stellt einen Weltabschnitt in Form eines Quaders dar. Der Quader ist 256 Meter in der Breite und Länge und unbeschränkt in der Höhe, wobei höher als 4 km es derzeit nicht gebaut werden kann.

Jede Position auf der Sim kann mit einer *lokalen Koordinate* beschrieben werden, der Ursprung ist die südwestliche Ecke der Sim, die nordöstliche Ecke am Boden hat demnach die Position `<256, 256, 0>` und die Position `<128, 128, 1000>` beschreibt einen Punkt 1000 Meter über die Simmitte.

Lokale Koordinaten sind allerdings nicht eindeutig, es gibt so viele Punkte mit der Lokalkoordinate `<128, 128, 1000>`, wie viele Sims es gibt. Um einen Punkt mit der lokalen Koordinate eindeutig zu beschreiben, muss die Sim angegeben werden, entsprechend ihrem Ursprung kann man den Punkt eindeutig identifizieren.

Sims fügen sich zu einer *Grid* zusammen, das ist die Welt von SL. Einige Sims werden Seite an Seite platziert und bilden größere Simverbunde, die größten davon sind Kontinente, einige Sims sind getrennt stehende Inseln. Sind zwei Sims in einem Simverbund, existiert ein direkter Weg zwischen Positionen auf diesen Sims, den Weg kann man etwa mit einem Fahrzeug oder lokalen Teleporter bereisen, oder mit einem globalen Teleporter überspringen. Siminseln oder Simverbunde sind voneinander getrennt. Zwischen diesen Sims funktionieren nur globale Teleporter wie RTS.

Sims werden immer dicht aneinander platziert, ohne Überlappung oder Abstand. Somit ist der Ursprung zweier an der Seite benachbarten Sim immer 256m voneinander entfernt. Allein dadurch kann man relative Position zweier Punkte bestimmen. Allerdings hat jede Sim eine genau festgelegte Position auf der Grid: Die *globale Koordinate* einer Sim ist die ihres Ursprungs bezüglich eines festgelegten Ursprungs der Grid. Das ist ein dreidimensionaler Vektor, wobei `Z=0` gilt.

Teilt man diese Koordinate durch 256, erhält man die *Gridkoordinate* der Sim. Das ist ein Vektor, der den Abstand vom Ursprung in Sims misst: `X` gibt die Anzahl der Sims vom Gridurspring bis zur untersuchten Sim entlang der west-ösztlichen Richtung und `Y` die Anzahl Sims entlang der süd-nördlcher Richtung. Gridkordinaten zweier benachbarten Sims unterscheiden sich also in `X` oder `Y` genau um 1.

RTS Teleporter arbeiten mit Globalkoordinaten der Ziele. Um die globale Position eines Punkts auf einer Sim zu bestimmen, muss man die globale Koordinate der Sim (ihres Ursprungs) mit der lokalen Koordinate des Punktes auf der Sim vektoriell addieren. Der Ergebnisvektor beschreibt die Position des Punktes eindeutig, kein anderer Punkt auf der Grid wird eine gleiche globale Koordinate haben.

Normalerweise wird man Ziele mit SLURL oder LM angeben, dann braucht man Globalkoordinaten gar nicht, der Teleporter berechnet sie selbst. Wird aber der

Direktteleport ausgelöst, muss die Globalkoordinate übergeben werden. Deshalb rechnen wir exemplarisch die lokale Koordinate eines Ortes in Globale um.

Als Beispielsim nehmen wir '**Da Boom**', die erste Sim in SL überhaupt. Wir wollen die Globalposition des Punktes **<120, 130, 500>** auf dieser Sim bestimmen. Zuerst benötigen wir die Globalposition der Sim selbst. Dazu kann uns etwa der Dienst *Grid Survey*[14] helfen. Konkret rufen wir dazu diese URL ab:

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

Wir erhalten eine Liste von Angaben, zwei davon brauchen wir: **x=1000**, **y=1000**. Das ist die Gridkoordinate der Sim, wobei **z=0** gilt (der Ursprung ist am Boden.) Die Globalposition der Sim bekommen wir, indem wir die Zahlen mit 256 multiplizieren und als Vektor schreiben: **<256000, 256000, 0>**.

Es bleibt uns lediglich die Lokalposition des Zielpunktes an zu addieren und wir bekommen das Ergebnis: **<256120, 256130, 500>**.

Die gute Nachricht ist aber, auch um die Auflösung der globalen Koordinaten im DirectTP Modus brauchen wir uns nicht zu kümmern, Ziellistenskript oder DirecTP Skript können das abnehmen, Abschnitt 2.4.3 zeigt das genauer. Nur wenn wir skriptreduziert arbeiten wollen, müssen wir zum Taschenrechner greifen.

[14] http://api.gridsurvey.com

2.2 Konfiguration

Nachdem die Skripte im Entwicklungskit und die Beispielteleporter (kurz) vorgestellt wurden, gehen wir ihre Konfiguration an, zunächst allgemein, dann die der Beispielteleporter. Die Konfiguration findet immer in einer Notekarte namens '**config**' statt, die im Systemprim vorliegen soll.

2.2.1 Syntax

Das Format der Notekarte sieht eine einfache Strukturierung der Daten vor: Der Inhalt besteht aus leeren Zeilen, Kommentaren, und Parameter-Wert Paaren, die in Sektionen gruppiert werden, Abildung 2.7.[15]

```
# Kommentarzeile

[Sektion]
Parameter = Wert # Kommentar
```

Figure 2.7: Konfigurationsbeispiel

Das Zeichen '**#**' und der nachfolgende Zeilenrest gelten als Kommentar und wird ignoriert.[16] Der Kommentar wird mit dem Zeilenende abgeschlossen, einen Blockkommentar gibt es nicht.

Eckige Klammern schließen Sektionsnamen ein. Die Sektionen trennen die Konfigurationsdaten in Abschnitte auf. Die Unterteilung ist einfach, Sektionen sind nicht verschachtelbar. Leerzeichen um Sektionsnamen werden ignoriert. Um die Klammer dürfen nur Leerzeichen stehen:

```
[Sektion]     # Richtig
[ Sektion ]   # Richtig, dasselbe wie [Sektion]
[Sektion].    # Falsch, der Punkt ist zuviel
[Sektion      # Falsch, ']' fehlt
```

Parameter und Werte müssen vorhanden sein, Leerzeichen um Parameter und Wert werden ignoriert. Parameternamen dürfen die Schreibweise ändern:

```
parameter=Wert       # Richtig
PaRAmetER   = Wert   # entspricht 'parameter=Wert'
parameter   =        # Falsch, der Wert fehlt
parameter            # ebenfalls falsch
            = Wert   # Falsch: Parameter fehlt, Zeilenumbruch nicht möglich
```

[15] Die Farbhervorhebung ist ein Service dieses Dokuments. Die Textfarbe des Notekarteneditors in SL ist durchgehend schwarz

[16] Es dürfen daher keine Namen oder Werte dieses Zeichen enthalten.

Hinweis: Skripte können maximal 256 erste Zeichen jeder Zeile lesen. Einige Parameter erfordern Textwerte, beachten Sie bitte diese Einschränkung.

Hinweis: Jede leere Zeile oder reine Kommentarzeile bringt eine Verzögerung von mindestens 0.1s beim Einlesen (bedingt durch LSL.) Damit die Konfiguration schneller abläuft, verzichten Sie bitte auf reine Kommentarzeilen.

Kommentare, die am Ende von Informationszeilen stehen (hinter dem Sektionsnamen oder Parameterwert), haben keine verzögernde Auswirkungen.

2.2.2 Semantik

Der Inhalt der Konfigurationsnote lässt sich in drei Hauptabschnitte unterteilen, Abbildung 2.8.

```
# Hauptsection
[*MAIN]
# Konfigurationsdaten

# Sections für weitere Skripte
[@sensor]
# Konfigurationsdaten

# Zielsections
[Endora]
# Konfigurationsdaten
```

Figure 2.8: Drei Konfigurationsabschnitte

- Der Kernskript benötigt die *Hauptsection*, '**[*MAIN]**' (jede Schreibweise.)
- Andere Skripte, sofern konfigurierbar, könnten eigene Sections benötigen. Wie etwa die *Sensorsektion* '**[@sensor]**' im Beispiel.
- Die Zielliste benötigt mindestens eine *Zielsection*, das sind Sections deren Namen mit einem Buchstaben oder Unterstrich anfangen.

Abbildung 2.9 zeigt einen Konfigurationstext, der als Grundlage für die Konfiguration beiliegender Teleportergeräte dient. Ihre Konfigurationstexte sind umfangreich und werden bis auf die Änderungen verkürzt dargestellt.

```
# RLV Teleporter Konfiguration
[*MAIN]
user        = all

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

title       = »%n«\n.touch to visit.\n(is your relay on?)
color       = lavender
fadeout     = 0

norelay     = TP Relay (r upper arm)

mode        = RLV
fail        = LM
delay       = 15
return      = LM
offset      = -3

fasttp      = OFF
refresh     = 24

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...

# Sensorsektion
[@sensor]
mode        = touch
distance    = 5-10   # Berührung nur aus Entfernung von 5 bis 10 Meter

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

Figure 2.9: Grundkonfiguration

2.2.3 Hauptsection

Die Hauptsektion hat den Namen '***main**' (jede Schreibweise), Abbildung 2.9.

Benutzerparameter: user

```
user = group+!owner
```

Der Parameter '**user**' gibt an, wer den Teleporter benutzen kann, d.h. damit reisen darf. Der Wert ist eine Liste von Stringflags, verbunden via '**+**'. Die Stringflags sind in 2.1 angegeben.

Flag	Bedeutung
owner	Der Besitzer
!owner	Der Besitzer nicht
group	Jeder aus der Objektgruppe
!group	Niemand aus der Objektgruppe
all	Jeder
!all	Niemand

Table 2.1: Flagamen, Parameter '**user**'

Sind die Flags '**all**', '**!all**' angegeben, werden andere ignoriert. Im ersten Fall kann jeder den Teleporter nutzen, im zweiten – keiner. Die Flags '**owner**', '**group**' erweitern den Nutzerkreis. Die Flags '**!owner**', '**!group**' reduzieren ihn. Zum Beispiel die Definition am Anfang bedeutet: Jeder aus der Objektgruppe darf mit dem Teleporter reisen, der Besitzer aber nicht.

Bildparameter: image, side

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

Der Parameter '**image**' gibt das Defaultbild an, dieses wird angezeigt, wenn das Ziel kein eigenes 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: Das Zielbild wird auf dem Bildprim angezeigt, Vorschaubilder auf den Vorschauprims. Mögliche Einstellungen gibt die Tabelle 2.2 an.

Ausnahme: Ist die Einstellung '**none**', wird das Zielbild unterdrückt, Vorschaubilder aber auf allen Seiten der Vorschauprims angezeigt.

Flag	Bedeutung
all	Das Bild auf allen Primseiten auftragen
none	Keine, das Bild gar nicht anzeigen
X	Auf der Primseite mit der LSL-Nummer X, wobei X eine Zahl im Bereich 0 bis 8 ist.

Table 2.2: Bildseite, Parameter '**side**'

Titelparameter: title, color, colour, fadeout

```
title   = »%n«\n.touch to visit.\n(is your relay on?)
color   = <1.000000, 0.980390, 0.803920> # 'LemonChiffon'
fadeout = 0
```

Der Parameter '**title**' legt den Titel fest (den Hovertext.) Der Wert kann Platzhalter enthalten, Tabelle 2.3.

\n	Erzeugt einen Zeilenumbruch an dieser Stelle
\t	Das Tabulatorzeichen
%n	Name des aktuell angezeigten Ziels
%s	Name der LM oder der Zielsim, falls per SLURL festgelegt
%p	Nummer des Ziels in der Liste, 1 fürs erste Ziel
%c	Anzahl der Ziele in der Liste

Table 2.3: Textplatzhalter

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 Vektor, wie im obigen Beispiel. Einige gebräuchliche Farben können aber auch mittels ihren Namen angegeben werden. Die erkennbaren Farbnamen sind im Kapitel 3 aufgelistet. Der Farbname beachtet die Schreibweise nicht, '**White**' bedeutet dasselbe wie '**WHITE**' oder '**white**'. In diesem Dokument wird die Kleinschreibweise bevorzugt.

Beim Erkennen der Farbnamen werden nur so viele Zeichen am Wortanfang beachtet, wie es erforderlich ist, um den Namen von anderen zu unterscheiden. Das signifikante Namenspräfix wird in den Tabellen in rot hervorgehoben. Beispielweise hat diese Zeile dieselbe Wirkung:

```
colour = LemonCh # 'LemonChiffon'
```

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

Teleportparameter: mode, fail, delay, return, offset

```
mode   = RLV
fail   = LM
delay  = 15
return = LM
offset = -3
```

Der Wert von Parametern '**mode**', '**fail**' und '**return**' ist eine Kombination von Stringflags. Die Tabelle 2.4 gibt Flagnamen an und Parameter, für die sie gelten.

Flag	mode	fail	return	Bedeutung
RLV	•			RLV des Benutzers ansprechen
LM	•	•	•	Übergebe die LM oder SLURL zum Ziel
OFF	•	•	•	Tue nichts
N	•	•	•	Das selbe wie 'OFF'

Table 2.4: Teleportflags, Teleportparameter

Die Flags werden via '**+**' konkateniert und wirken additiv: Der Wert '**LM+OFF**' bedeutet dasselbe wie '**LM**' oder '**lm**': Flagnamen beachten die Schreibweise nicht.

Der Parameter '**mode**' bestimmt, ob der Teleportprozess via RLV oder mittels Übergabe der LM bzw. SLURL zum Ziel auszuführen ist. 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.

Der Parameter '**return**' gibt an, ob eine SLURL zum Standort des Teleporters an den Agenten übergeben wird, falls er teleportiert wurde.

Der Parameter '**delay**' bestimmt, wie lange auf die Rückmeldung des Realay gewartet wird, bis das Timeout festgestellt ist. Die Zeit ist in Sekunden gegeben, das Minimum ist 5 Sekunden.

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, wobei die Angabe per Einzelzahl einer via Vektor entspricht, wo die Zahl als x Komponente gilt. Diese beide Definitionen sind identisch:

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

Was bedeutet dieser Wert? Ist der Offset null, die Rückkehrposition ist die Position des Teleporters selbst, dann werden Avatare, die die Rückkehr SLURL benutzen auf seiner Position gerezzt. Manchmal ist das ungünstig:

Rückkehrende Avatare würden mit dem Teleporter kollidieren, und sofern er auf Kollision reagiert, teleporterier er sie erneut, was nicht erwünscht ist. Ist der Teleporter auf der Decke montiert, sehen sich die zurückkehrende Avatare auf den Boden fallen. Die Einstellung kann den Rückkehrpunkt zur Seite verschieben oder gleich auf den Boden stellen.

Hinweis: Die Position eines Avatars ist Zentrum seiner Boundingbox, dieser Punkt ist in etwa 1 Meter über dem Boden, deshalb muss die Rückkehrposition 1m über dem Boden sein.

Der Offsetvektor ist immer ein Vektor in der globalen xy Ebene um den Systemprim. Ist es nicht rotiert, zielt der Vetor `<3, 0, 0>` auf den Punkt 3m richtung 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.

Parameter norelay

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

Der Parameter '**norelay**' gibt die Datei an, die übergeben wird, falls keine Antwort vom Relay des Benutzers kam. Auf diese Weise kann man eine Notekarte oder ein Hilfsrelay übergeben, das die Teleporternutzung vereinfacht.

Parameter fasttp

```
fasttp = OFF
```

Der Parameter '**fasttp**' erwartet einen der Werte '**N**' bzw. '**OFF**' zum Deaktivieren und '**Y**' bzw. '**ON**' zum Aktivieren. Wenn aktiv, findet der Teleport statt, sobald ein Ziel ausgewählt wird.

Ist der Modus off, kann man durch Ziele browsen, den Teleport muss man gesondert starten. FastTP erlaubt es Teleporter zu bauen, die nur einen Klick brauchen.

Parameter refresh

```
refresh = 24
```

Zum Teleportieren benötigt man die globale Koordinate der Zielposition. Da Sims ihre Position manchmal ändern, stimmen Zielkoordinaten nach der Simverschiebung nicht mehr und müssen erneut aufgelöst (bei SL Servern abgefragt) werden.

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. Da Sims seltener wesentlich, empfielt sich ein größerer Wert, etwa 24 Stunden.

Was bedeutet der Wert genau? Es wird kein Timer gestartet, der permanent die Zielangaben wieder auflöst, das Wiederauflösen findet auf Anfrage, beim Teleport auf die veraltete Position. Kleinerer Wert bedeutet nicht automatisch mehr Lag. Allerdings werden Zielangaben häufiger wieder aufgelöst.

Das Wiederauflösen ist mit einer Verzögerung von 1 bis 2 Sekunden verbunden, der Teleporter scheint dann etwas träge. Bei einem größeren Wert reagiert der Teleporter flüssiger, kann aber die Simverschiebung wahrscheinlicher übersehen: Durchschnittlich die Hälfte der angegebenen Zeit wird der falsche Wert gespeichert.

Es empfiehlt sich deshalb ein Wert von einem Tag bis einer Woche zu nehmen (24 bis 168 Stunden.) Für den Fall, dass eine Sim sich tatsächlich verschoben haben sollte, hat das Besitzermenü den 'refresh' Button. Er setzt alle gespeicherten Daten als ungültig, erzwingt damit das Wiederauflösen beim nächsten Teleport.

Nachrichtformat

```
format = MAPS
```

Der Parameter '**format**' legt das Format von übergebenen SLURLs fest. Hier kann man einen der vier Werte angeben: '**MAPS**', '**SLURL**', '**APP**' und Freitext, Tabelle 2.5.

Die *SLURL*[17] ist klassisch, alle Viewer verstehen den Link und öffnen das Landmarkenfenster, wenn der Link im Chatverlauf angeklickt ist. Web Browser zeigen lediglich die Zielposition auf der Karte.

[17] Ein Link wie http://slurl.com/secondlife/Endora/123/234/345

Wert	Beschreibung
`MAPS`	Versendet eine moderne MapURL
`SLURL`	Versendet eine klassische SLURL
`APP`	Versendet einen Teleportlink
Freitext	Ein Text mit Platzhaltern `%s`, `%x`, `%y`, `%z`

Table 2.5: SLURL-Format

Die *MapURL*[18] ist nur den neueren Viewern bekannt (seit Version 1.23,) sie öffnen das Landmarkenfenster, wenn der Link angeklickt wird. Ältere Viewer öffnen den Webbrowser, wobei dieser nicht nur die Position auf der Karte zeigt, sondern auch zusätzliche Informationen.

Der *Teleportlink*[19] ist ein String, der nicht in Webbrowsern anklickbar ist, wird er aber im Chatverlauf angeklickt, teleportiert der Viewer sofort zum Zielort ohne etwas zu öffnen.

Der Freitext ist wie angegeben versendet, wobei der Text Platzhalter enthalten darf, 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)'. Dieser ist nicht anklickbar, kann aber zum Festlegen der Loginposition benutzt werden.

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

Nachrichtparameter: msg:url, msg:lm, msg:return, msg:norelay

```
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, Tabelle 2.3 und speichern Textnachrichten die beim Übergeben von Items oder SLURLs mitversendet werden, und zwar...

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 (es wird nur eine IM versendet.)

[18] Ein Link wie http://maps.secondlife.com/secondlife/Endora/123/234/345
[19] Ein String wie secondlife:///app/teleport/Endora/123/234/345

Der Parameter '**msg:lm**' gibt den Text an, der beim Übergeben der Ziel-LM als IM versendet wird. Man kann keine Dateien in eine IM einbetten, daher bekommt der Reisende die LM und die IM getrennt.

Der Parameter '**msg:return**' gibt den Text an, der die Rückkehr-SLURL einleitet. Es wird ebenfalls eine IM versendet, kombiniert aus dem Nachrichttext und der SLURL.

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

2.2.4 Sensorsection

Der Sensorskript wird in der Sensorsection konfiguriert. Werden mehrere Sensorskripte installiert, muss pro Sensor eine Section in der Konfiguration vorhanden sein. Ein Beispiel gibt die Grundkonfiguration 2.9 an.

Sensormodus, Parameter mode

Der Parameter **mode** wählt den Sensormodus aus. Es kann einer der fünf Namen angegeben werden, Tabelle 2.6.

touch	Touchmodus. Klickende Person wird teleportiert
collide	Kollisionsmodus. Kollidierende Person wird teleportiert
scan	Radarmodus. Raumüberwachung. Teleportiert Eindringlinge
sit	Sitzmodus, wer sich aufs Gerät hinsetzt, wird teleportiert
chat	Chatmodus. Erlaubt eingeschränkte Gerätekontrolle via Chat

Table 2.6: Sensormodi

Im Touchmodus wird das Anklicken des Systemprims überwacht, so wie aller Prims mit der Kennung '**sensor**', Abschnitt 2.3.2. Im Kollisionsmodus macht der Skript das Gerät phantom und überwacht Kollisionen damit, etwa hineintreten oder darüber laufen.

Im Radarmodus werden Eindringlinge eines überwachten Bereichs wegteleportiert. Im Sitzmodus funktioniert der Teleporter ähnlich den üblichen Teleportsystemen, wo man sich zum Teleportieren auf eine Scheibe am Boden setzt, nur eben gridweit.

Der Chatmodus ist speziell. Hier werden Nachrichten, die auf dem eingestellten Channel ausgesprochen werden, in Buttonnachrichten umgesetzt, die das Gerät steuern können, Abschnitt 2.4.2. Auf diese Weise lassen sich über die Chatzeile Ziele auswählen, Teleports auslösen, und weitere Skripte ansteuern.

Param	touch	collide	scan	sit	chat	Bedeutung
mode	-	-	-	-	-	Sensormodus
face	•					Anklickbare Primseite
distance	•		•			Distanz zum Avatar
interval		•	•			Scaninterval
angle			•			Scanwinkel
channel					•	Chatchannel
strict					•	Nur auf Avatare reagieren?
trigger					•	Befehlsliste, TriggerTP
repeat					•	Befehlsliste, RepeatTP

Table 2.7: Sensormodi und relevante Parameter

Hinweis: Je nach Modus werden bestimmte Parameter nicht beachtet und können ausgelassen werden. Die Tabelle 2.7 gibt an, im welchen Modus welche Parameter beachtet werden.

Beispielsweise benötigt der Sitzmodus lediglich die Modusangabe:

```
[@sensor]
mode = sit
```

Klickseite

```
face = all
```

Der Parameter '**face**' wird nur im Touchmodus benutzt und gibt die Primseite an, deren Klicks beachtet werden. Als Wert kann entweder '**all**' angegeben werden oder eine Zahl zwischen 0 und 8.

Avatarabstand

```
distance = 0-15
distance = 15    # Dasselbe wie 0-15
```

Abstandsparameter wird im Touchmodus und Sensormodus benutzt und gibt den Distanzbereich an, als minimale und maximale beachtete Distanz in Meter. Der Wert kann in Form '**min-max**' oder '**max**' angegeben werden. Beispielsweise sind beide Angaben äquivalent.

Scaninterval

```
interval = 10
```

Scaninterval wird im Radar- und Kollisionsmodus benötigt und gibt den Wert in Sekunden an. Im Radarmodus ist es die Zeit, wie oft Avatare im Raumbereich gescannt werden.

Im Kollisionsmodus gibt der Wert an, 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.

Scanwinkel

```
angle = 90 # Nur die Halbsphere vor dem Gerät
angle = PI # Vollscan.
```

Scanwinkel ist für Scanmodus interessant und bestimmt den überwachten Bereich. Der Wert gibt den Raumwinkel in Grad um die positive lokale X-Achse des Systemprims an, entweder als Zahl im Bereich 1 bis 180, oder als '`PI`', was 180 meint.

Chatchannel

```
channel = 77
```

Chatchannel bestimmt für den Chatmodus den Channel, auf dem der Scanner lauscht. Nur auf diesem Channel können Befehle abgesetzt werden, die zu Buttonnachrichten werden.

Der Wert ist eine Zahl, wobei 0 lieber zu vermeiden ist: Da alle Avatare darauf sprechen, können ungewollte Befehle produziert werden, auch ist das eine garantierte Lagquelle. Negative Zahlen sind auch möglich, Avatare können jedoch nicht direkt im negativen Channel reden.

Striktheitsmodus

```
strict = N
```

Schaltet den Striktheitsmodus ein oder aus. Der Wert ist entweder '`Y`' bzw. '`ON`' zum Einschalten oder '`N`' bzw. '`OFF`' zum Ausschalten. Wenn eingeschaltet, akzeptiert der Sensor nur Befehle von Avataren.

Ist der Modus ausgeschaltet, werden Befehle auch von Objekten akzeptiert. Dann wird der Objektbesitzer als Befehlssender angegeben – als ob er den Befehl gesagt hat und nicht das Objekt. Das funktioniert allerdings nur, wenn der Objektbesitzer auf der Sim anwesend ist.

Der Modus erlaubt es, Huds zu bauen, welche Kontrolbefehle versenden, sobald der Träger die Buttons anklikt.

Befehlslisten: trigger, repeat

```
trigger = .trigger, go, energy
repeat  = -
```

Es gibt zwei Befehle zum Auslösen des Teleports: '**.trigger**' und '**.repeat**' teleportieren den Agenten aufs aktuelle und vorige Ziel. Parameter '**trigger**' und '**repeat**' können diese Befehle durch andere Stichwörter ersetzen oder blockieren.

Als Parameterwerte werden kommaseparierte Listen von Stichwörtern angegeben. Der empfangene Befehl wird in diesen Listen aufgesucht und falls gefunden, durch den entsprechenden Befehl ersetzt. Ist als Wert '**-**' angegeben, kann der Befehl gar nicht mehr via Chat eingegeben werden.

Nach dem Konfigurationsbeispiel kann man neben '**.trigger**' auch '**go**' und '**energy**' auf dem konfigurierten Channel sagen, es wird in allen Fällen zu '**.trigger**' übersetzt.[20] Dadurch, dass '**.trigger**' auf der Liste bleibt, bleibt der Befehl wirksam.

Der Parameter '**repeat**' hat den Befehl '**.repeat**' blokiert: Auch wenn der Befehl empfangen wird, wird er nicht weitergeleitet.

2.2.5 Zielsections

Eine Zielsektion besteht aus nur drei einträgen: Der *Zielname*, das *Zielbild* und die *Zielangabe*, Abbildung 2.10.

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

Figure 2.10: Zielsection

[20] Der Sensor übermittelt an den Kernskript eine Nachricht, als hätte der Benutzer den Befehl '**.trigger**' gegeben

Zielname

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.

Mit einem führenden Unterstrich werden sogenannte Defaultziele ausgezeichnet. Es kann nur eines der Ziele als Default ausgezeichnet werden, fangen mehrere Sectionsnamen mit Unterstrich an, wird eines davon als Default angenommen.

Das Defaultziel wird automatisch nach dem erfolgten Teleport ausgewählt. Auf diese Weise wird etwa die Reihenfolge der angezeigten Vorschaubilder beibehalten. Allerdings kann man dann dem teleportierten Avatar nicht mehr per Klick auf das Zielbild folgen (man erreicht dann das bereits ausgewählte Defaultziel.)

Zielbild

Das Zielbild wird stellvertretend für das Ziel angezeigt (als Vorschau- und Zielbild.) 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 (sie wird benötigt, um andere Prims damit zu texturieren).

Zielangabe

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

Ist das Ziel als LM angegeben, muss sie im Objektinventar vorliegen. In diesem Fall wird die LM an den Reisenden übergeben, falls LM-Übergabe aktiv ist, Abschnitt 2.2.3. 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:

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

Striped SLURL begint mit dem Simnamen und ist daher universell.

2.2.6 Konfigurationsbeispiele

Jetzt ist die Konfiguration der Beispielteleporter an der Reihe. Der Konfigurationstext ist umfangreich, deshab werden nur Änderungen gegenüber der Grundkonfiguration angegeben.

TPSign

Der TPSign besteht aus einem Prim, Abbildung 2.2(a). Seine Konfiguration ist in 2.11 abgebildet. Der Text stützt sich auf die Grundkonfiguration, 2.9.

Der Teleporter setzt den Sensorplugin nicht ein (dadurch entfällt auch die Sensorsektion) und kann ohne Skrollhilfe nur ein Ziel behandeln. Weitere Ziele können in der Konfiguration angegeben werden, nur das Erste wird aber beachtet.

Haben Sie die Konfiguration geändert, müssen Sie nach Abspeichern den Teleporter via Besitzermenü zurücksetzen, das Besitzermenü erhalten Sie via langen Klick. Gestartet wird der Teleporter, nachdem die Konfiguration eingelesen wurde, ebenfalls via Besitzermenü.

TPVendor

Der TPVendor bietet neben dem Zielbild auch 6 Vorschaubilder an, so wie Scrollbuttons und das Logoprim zum Aufruf des Besitzermenüs, Abbildung 2.2(b). Von den Skripten her, gibt es keinen Unterschied zu TPSign.

Das Hintergrundprim verdeckt die Hinterseite aller Prims, sodass hier nur die Vorderseite (Seite 4) für die Bildanzeige vorgesehen ist.

Durch die Scrollhilfe ist der Teleporter mehrzielfähig und ist bereits mit 11 Zielen vorkonfiguriert, die Abbildung 2.12 zeigt nur die ersten 2 davon. Es ist aber möglich, bis zu 80 Ziele einzutragen.

TPMirror

TPMirror ist ebenfalls ein Einzielteleporter, der auf Kollision reagiert. Geformt als ein Bild zum Aufhängen an der Wand oder Auflegen auf dem Boden. Die Reise beginnt beim Hineinlaufen ins Bild oder laufen darüber, Abbildung 2.3(a).

```
# TPSign Konfiguration
[*MAIN]
title       = %n\n...touch to visit
# --- Sektionsrest wie in der Grundkonfiguration ---

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

Figure 2.11: TPSign Konfiguration

```
# TPVendor Konfiguration
[*MAIN]
side        = 4
title       = Destination %p of %c\n%n\n...touch to visit
# --- Sektionsrest wie in der Grundkonfiguration ---

# 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

# --- Neun weitere Zieldefinitionen folgen ---
```

Figure 2.12: TPVendor Konfiguration, erste 2 Ziele

```
# TPMirror Konfiguration
[*MAIN]
title       = %n\n...step into and visit
# --- Sektionsrest wie in der Grundkonfiguration ---

[@sensor]
mode        = collide
interval    = 10

# --- Einzelziel wie in der Grundkonfiguration ---
```

Figure 2.13: TPMirror Konfiguration

Der Unterschied zu TPSign besteht lediglich darin, das hier der Sensorplugin im Kollisionsmodus verwendet wird. Die Konfiguration ist in 2.13 abgebildet, die Sensorsection enthält alle benötigten Parameter für den Modus.

```
# ChatPorter Konfiguration
[*MAIN]
side        = 4
title       = Destination %p of %c\n%n\n(to use, say "/111 go")
# --- Sektionsrest wie in Grundkonfiguration ---

[@sensor]
mode        = chat
channel     = 111   # Befehlen mit: '/111 cmd'
strict      = N     # Benutzer's Objekte können auch befehligen
trigger     = go    # Zauberwort ist: '/111 go'
repeat      = -     # Kein weg, voriges TP zu wiederholen

# --- Zielliste wie beim TPVendor ---
```

Figure 2.14: Chatporter Konfiguration

ChatPorter

Der ChatPorter ist aus vier Prims aufgebaut, das Bildprim, Scrollprims und Logoprim, Abbildung 2.4(a). Mehrzielfähig, der Sensor ist auf Chatmodus eingestellt: Der Channel ist 111, die Striktcheit ist ausgeschaltet. Der Triggerbefehl wurde durch '**go**' ersetzt und der Repeatbefehl deaktiviert, Abbildung 2.14.

TPRadar

Beim TPRadar handelt es sich um einen Raumscanner, der Avatare, die den überwachten Bereich betreten, auf die Zielposition teleportiert. Vorkonfiguriert ist der Sensor auf eine Halbkugel, damit der Teleporter auf einer Wand oder auch Boden bzw. Decke befestigt werden kann und nur den Raum davor scannt, Abb. 2.15.

Der Teleporter ist mehrzielfähig und erhält zusätzlich Scrollprims und das Menüprim, Abbildung 2.3(b). Die Anzeige der Zielbilder ist ausgeschaltet (Primseite ist '**none**') deshalb konnten die Bilder auch in Zielangaben entfernt werden.

```
# TPRadar Konfiguration
[*MAIN]
side        = none
title       = Destination %p of %c\n%n
fadeout     = 60
# --- Sektionsrest wie in Grundkonfiguration ---

[@sensor]
mode        = scan
interval    = 10    # Sekunden
distance    = 5     # Meter max.
angle       = 90    # Halbkugel um die X-Achse

# Zielliste ohne Bildangaben
[Linden Playground (M)]
target      = http://slurl.com/secondlife/Da%20Boom/128/128/35

[Protected Ocean (G)]
target      = http://slurl.com/secondlife/Pravatch/221/227/3

# --- Neun weitere Zieldefinitionen folgen ---
```

Figure 2.15: TPRadar Konfiguration

```
# Sitporter Konfiguration
[*MAIN]
title       = %n\n...sit here to visit
# --- Sektionsrest wie in Grundkonfiguration ---

[@sensor]
mode        = sit

# --- Einzelziel wie in der Grundkonfiguration ---
```

Figure 2.16: Sitporter Konfiguration

```
# Holoporter Konfiguration
[*MAIN]
side        = 2
title       = Destination %p of %c\n%n
# --- Sektionsrest wie in Grundkonfiguration ---

[@sensor]
mode        = touch
distance    = 2   # Beim Klicken nicht weiter als 2m entfernt stehen

# --- Zielliste wie beim TPVendor ---
```

Figure 2.17: Holoporter Configuration

SitPorter

Der Sitporter ist ein Einzelziel-Teleporter. Ausgeführt als eine Kugel, in der sich das Zielbild dreht, Abbildung 2.4(b). Beim Hinsetzen auf die Kugel wird der sitzende Avatar teleportiert. Der Sensor arbeitet hier im Sitzmodus, Abbildung 2.16.

Die Anzeige von Raumpartikel beim gestarteten Teleporter, wie auch die Rotation des Zielbildes übernehmen Gerätetreiber, sie werden hier aber nicht beschrieben.

HoloPorter

Der HoloPorter ist in Form einer Plattform aufgebaut, die zusätzlich Buttons für die Zielauswahl und Menuaufruf enthält. Das ausgewählte Ziel erscheint auf einem holografisch anmutenden Schirm, Abbildung 2.5.

Klickt eine auf der Plattform stehende Person die Plattform selbst oder den Schirm an, wird sie teleportiert. Die Konfiguration des Holoporters ist in 2.17 abgebildet.

Teleport startet, wenn der Reisende das Gerät anklickt. Damit keine Avatare teleportiert werden, die weit von der Plattform entfernt sind, wird der Sensor im Klickmodus verwendet, wobei die Distand auf 2m beschränkt ist. So wird nur der Avatar auf der Plattform beachtet.

2.3 Architektur

Die Architektur von RTS ist in 2.18 abgebildet.

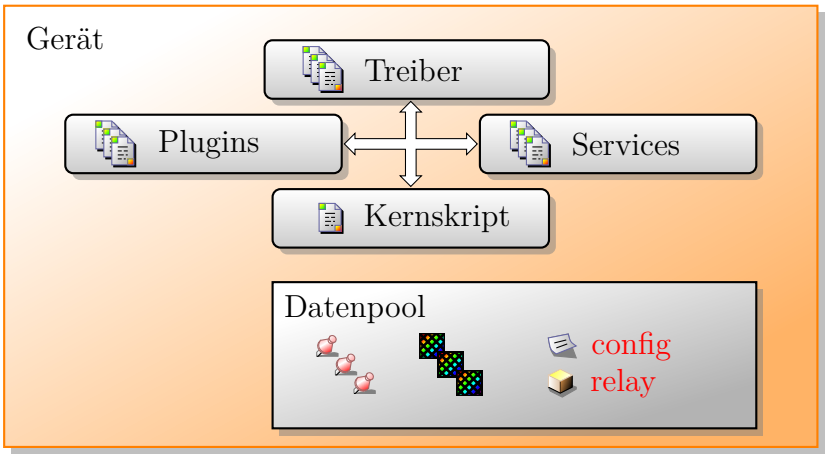


Figure 2.18: Systemarchitektur der Teleportergeräte

2.3.1 Skriptklassen

Die im RTS installierbare Skripte werden in vier Klassen unterteilt: Der *Kernskript*, *Serviceskripte*, *Plugins* und *Treiber*.

Der Kernskript und Serviceskripte sind geräteunabhängig und für alle Projekte erforderlich. Plugins sind ebenfalls geräteunabhängig, aber nicht immer erforderlich. Treibergeräte sind dazu da, das System an das jeweilige Gerät anzupassen. Sie sind geräteabhängig und je nach Projekt erforderlich.

Die Skripte kommunizieren miteinander via Linknachrichten und greifen auf den Datenbestand zu, bestehend aus Landmarken, Texturen, Notekarten und anderen Dateien.

Kernskript

Der Kernskript '**.core**' ist erforderlich für alle Teleporterprojekte. Er führt den eigentlichen Teleportvorgang aus, kontrolliert andere Skripte, präsentiert das Besitzermenü und übernimmt mit den Kernserivces eine Reihe von Begleitaufgaben.

Das Besitzermenü wird nur dem Besitzer geöffnet und enthält Buttons zum Resetten, Starten oder Anhalten des Teleporters, so wie zum Setzen von gespeicherten Zielpositionen als ungültig, damit sie erneut aufgelöst werden müssen.

Der Skript setzt eine sogenannte Teleportpipeline ein, Abbildung 2.19. Sie beschreibt verkettete Aufgaben, die zum Teleport eines Avatars ausgeführt werden. Durch die Aufnahme eines Avatars auf die Pipeline wird der Teleporter bereit zum Teleportieren des nächsten Avatars, noch bevor der erste Teleport abgeschlossen ist.

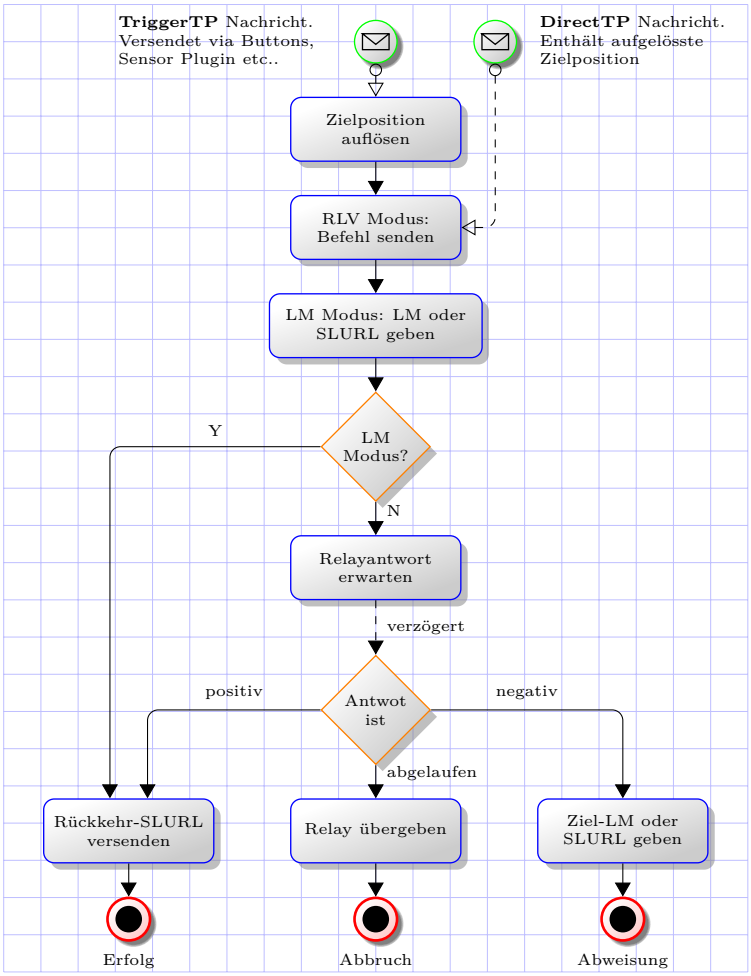


Figure 2.19: Teleportpipeline

Konfigurationsskript

Der Servicescript '**.config**' ist für alle Projekte erforderlich und der einzige Skript mit einem direkten Zugriff auf die Konfigurationsnotekarte. Seine Aufgabe ist es, sie auszulesen, Daten aufzubereiten und via Linknachrichten zu versenden.

Dadurch vereinfacht sich der Code anderer Skripte und daher verkleinert sich die gesamte Skriptgröße. Die Konfigurationsdaten liegen zentral in einer Notekarte vor, was wiederum die Pflege vereinfacht.

Der Konfigurationsskript verwaltet außerdem eine Farbtabelle, Kapitel 3, und ist in der Lage, Farbwerte, die als Farbnamen angegeben sind, in Vektorwerte zu übersetzen, da mit den Farbnamen wissen die Skripte nichts anzufangen.

Senderskript

Der Serviceskript '**.sender**' ist für alle Projekte erforderlich. Er ist in der Lage, Dateien auf Anfrage zu senden, aber auch Mitteilungen via IM oder e-Mail.[21] Der Service wird z.B. benötigt, um Fehlermeldungen an den Besitzer zu übermitteln, oder Landmarken und Mitteilungen an den Reisenden zu übergeben.

Der Grund für der Skript ist einfach: Das Versenden von IMs und Elemente ist LSL-bedingt immer mit einer Verzögerung von 2 Sekunden verbunden, bei einer e-Mail sogar 20 Sekunden. Durch Übernahme von Versandaufgaben befreit der Senderskript andere Skripte von diesen Aussetzern.

Sensorskript

Der Pluginskript '**.sensor**' ist optional. Er ist fürs Auslösen des Teleports zuständig. Der Kernskript kann den Teleport auch auslösen, aber nur wenn der Teleporter angeklickt wird.

Der Sensorplugin erweitert diesen Modus um die Distanzprüfung und kann etwa Klicks ausschließen, wenn der Avatar zu weit entfernt ist. Ferner kann der Skript auf vier weitere Ereignisse reagieren: Kollision mit dem Gerät, Setzen darauf, Radarmodus und Chatbefehle.

In allen Modi wird nur der *TriggerTP* ausgelöst. Im Chatmodus kann der Skript auch beide anderen Teleportarten auslösen, so wie Chatbefehle in (einige) Steuerbefehle des Geräts konvertieren.

Ziellistenskript

Der Pluginskript '**.targets**' wird zum Verwalten der Zielangaben aus der Konfiguration benötigt. Er speichert bis zu 80 dieser Ziele und ermittelt im Betrieb ihre globale Position. Diese benötigt der Kernskript um den Teleport auszuführen.

Die Zielposition meldet der Skript zusammen mit dem Zielbild und Bildern der Ziele vor und nach dem Ausgewählten in der Zielliste. Erst dadurch ist der Kernskript in der Lage, Vorschaubilder dieser Ziele anzuzeigen.

[21] Versand von e-Mails wird aktuell nicht verwendet.

Der Skript unterstützt die Zielauswahl auf drei Arten: Eine absolute Zielnummer in der Liste, eine relative (Iteration durch die Liste) und Textsuche, indem das Ziel mit dem angegebenen Namenspräfix ausgewählt wird.

Da der Skript die Globalposition von Ziele auflösen kann, übernimmt er auch eine andere Aufgabe: Auflösen der globalen Position eines Ziels, das via DirectTP-Befehl angegeben ist. Der Kernskript akzeptiert diesen Befehl nur dann, wenn die Zielposition in Globalkoordinaten angegeben ist. Der Ziellistenskript vereinfacht die Handhabung von Koordinaten, indem er diesen Befehl mit lokal angegebener Position erkennt und zu einem Befehl mit globalen Koordinaten umrechnet.

Der Skript unterhält einen sogenannten Simcache. Dort wird die globale Position von 16 zuletzt abgefragten Sims gespeichert. Finden Teleports auf wenige Sims statt, beschleunigt der Cache den Auflösungsvorgang und entlastet so die Server.

DirectTP Skript

Der Pluginskript '**.directtp**' wurde für reine Direktteleporter entwickelt. Er dient lediglich dem Auflösen von Globalkoordinaten in DirectTP-Befehlen mit lokal angegebenen Positionen.

Dieser Skript dient als Ersatz für den Ziellistenskript. Durch Auslassung der Zielverwaltung wurde die Skriptgröße erheblich reduziert, was etwas schlankere Teleportersysteme erlaubt.

Der Skript verwendet ebenfalls den Simcache, der auf 32 Sims vergrößert ist.

Treiberskripte

Treiberskripte sind geräteabhängige Skripte, die das Skriptsystem an das jeweilige Gerät anpassen oder eine Kommunikation mit dem Teleportergerät ermöglichen. Sie sind nicht immer erforderlich, wenn sie aber benutzt werden, müssen sie in vielen Fällen bearbeitet werden. Deshalb liegen mitgelieferte Treiber quelloffen vor.

2.3.2 Kernservices

Der Kernskript bietet eine Reihe von Kernservices an, das sind spezielle Aufgaben rund ums Teleportprozess. Dadurch reduziert sich die Menge benötigten Skripte. Die Art, in der diese Aufgaben ausgeführt werden ist allgemein und für die meisten Projekte ausreichend.

Sollte in einem konkreten Projekt diese Aufgaben auf eine andere Weise erledigt werden, bedarf dies eines Treibers. Mit *OptionAPI* bietet der Kernskript eine

Schnittstelle zum Ausschalten des betreffenden Kernservice an, Abschnitt 2.4.3. Nur der Unsitservice ist standardmäßig aus.

Primkennung

Einige Services arbeiten mit bestimmten Prims der Geräte und müssen die Prims im Linkset erkennen. Die Linknummern sind dafür unhandlich, deshalb werden die betreffende Prims per Name ausgezeichnet: Der Primname muss mit dem Zeichen '**@**' anfangen, dem eine kommaseparierte Liste von Kennungen folgt.

Zum Beispiel arbeitet der Textservice mit einem Prim mit Kennung '**text**', das Prim über dem der Hovertext erscheinen soll, muss also den Namen '**@text**' haben.[22]

Der Imageservice zeigt das Zielbild auf dem Prim mit Kennung '**image**' an. Dazu muss das Prim den Namen '**@image**' haben. Soll nun ein Prim sowohl das Zielbild, als auch den Hovertext anzeigen, kann man es mit '**@image,text**' benennen.

Die Kennungen sollten nur dann kombiniert werden, wenn die entsprechende Services unterschiedliche Primeigenschaften verändern. Z.B. Erzeugt der Primname '**@image,thumb**' einen Servicekonflikt: Beide Services müssen ein Bild auf dem Prim anzeigen. Auf solche Konflikte wird aber nicht automatisch geprüft.

Weitere Skripte können in dieses Kennungssystem ebenfalls einklinken. Z.B. sucht ein Partikeltreiber nach einem Prim mit Kennung '**psource**', um es als Partikelquelle zu nutzen.

Geschlossene Kennlisten

Seit der version 2.1, erlaubt RTS die Liste von Kennungen mit dem Zeichen '**@**' zu schließen. Auf diese Weise ist es möglich, auch Prims zu kennzeichnen, deren Namen nicht mir '**@**' anfangen dürfen.

Beispielweise muss der Name des Menübuttons genau '**.menu**' sein. Um dieses Prim mit '**text**' zu kennzeichnen, braucht man einfach die Kennliste zu schließen: Ist der Primname '**@text@.menu**', funktioniert es als Menübutton und kann auch den Hovertext darstellen.

Um das zu erlauben, behandelt RTS das Zeichen '**@**' als Metatrennzeichen. Das Zeichen wird zum Öffnen und Schließen der Kennliste verwendet und zum Trennen von Metaparameter des Prims. Innerhalb eines Metaparameters (wie Kennliste oder Buttonname) ist es nicht zulässig.

Außerdem kennt die OptionAPI eine Linknachricht, welche die Metaparameter von Prims übermittelt. Einzelheiten und Beispiele sind im Abschnitt 2.4.3 zu finden.

[22] Achten Sie bitte auf den Unterschied zwischen Kennung und Primname.

Unsitservice

Wird der sitzende Avatar wegteleportiert, kann es dank einem Bug in SL dazu führen, dass die Sitzanimation nicht beendet wird, der Avatar sitzend ankommt und sitzen bleibt. Die Animation kann dann u.U. nur durch den Relog beendet werden.

Um das zu verhindern, wird der Avatar vor dem TP aufgestanden, sowohl via LSL (der '`llUnsit()`' Befehl), als auch einer entsprechenden RLV-Aktion. Der Service ist standardmäßig ausgeschaltet, und sollte vom Sensor, der aufs Sitzen reagiert, eingeschaltet werden (der Sensorplugin im Sitzmodus tut das.)

Textservice

Wenn eingeschaltet, wird der Hovertext über dem mit '`text`' gekennzeichneten Prim dargestellt. Ist kein Prim so gekennzeichnet, wird der Systemprim genommen. Sind mehrere Prims so gekennzeichnet, wird nur eines davon berücksichtigt.

Nachrichtenservice

Wenn ein, werden interne Meldungen des NotifyAPI, Abschnitt 2.4.3, das sind vor allem Fehlermeldungen, via IM an den Besitzer weitergeleitet. Soll dies ein anderer Skript besser erledigen, muss er den Service deaktivieren.

Imageservice

Wenn ein, wird das Zielbild auf dem mit '`image`' gekennzeichneten Prim und entsprechend der Konfiguration angezeigt. Sind mehrere Prims so gekennzeichnet, wird nur eines davon berücksichtigt. Ist keines gekennzeichnet, wird das Systemprim verwendet.

Thumbservice

Falls ein, übernimmt der Kernskript das Anzeigen von Vorschaubildern und verarbeitet auch Klicks auf die Vorschauprims derart, dass das auf dem Vorschauprim angezeigte Ziel ausgewählt ist.

Als Vorschauprims dienen Prims mit *Kennpräfix* '`thumb`'. Es können mehrere Prims so gekennzeichnet werden, jedoch nur eine gerade Primzahl wird berücksichtigt: Die eine Hälfte zeigt Ziele vor dem Ausgewählten, die andere danach.

Die Prims werden entsprechend der alphabetischen Reihenfolge der Kennungen angeordnet. Damit lassen sich die Prims entsprechend der Reihenfolge der dargestellten Ziele anordnen: Man muss das Präfix '`thumb`' ums sortierbare Suffix erweitern.

Beispiel: Um einen Teleporter mit 6 Vorschaubildern aufzubauen, muss man 6 Prims als Vorschaubilder ausweisen. Man kann sie alle zwar mit '**thumb**' kennzeichnen, dann können wir aber nicht einfach bestimmen, in welcher Reihenfolge sie die Ziele anzeigen.

Um die Reihenfolge bestimmen zu können, erweitern wir das Präfix mit einer Zahl: '**thumb 1**', '**thumb 2**' ... '**thumb 6**', jetzt hängt die Anzeigereihenfolge nicht mit dem Linkset zusammen.

Buttonservice

Wenn ein, erzeugt der Kernskript beim Anklicken bestimmter Prims die Buttonnachrichten (ButtonAPI, Abschnitt 2.4.2). Der Schlüsselparameter der erzegten Nachricht überträgt UUID des klickenden Avatars.

Als Button dient jedes Prim, dessen Name mit einem Buchstaben oder einem der Zeichen '**.**', '*****' oder '**_**' beginnt. Ausnahme: Der Systemprim oder Prims mit einem leeren Namen oder '**Object**' als Name sind keine Buttons.

Der Punkt ist für Nachrichten gedacht, die den Kernskript oder Serviceskripte steuern. Der Stern für Steuernachrichten von Plugins und Treiber, der Unterstrich ist für allgemeine Steuernachrichten reserviert.

Beispielsweise führt ein Klick auf ein Prim namens '**.menu**' zu einer Nachricht, die das Besitzermenü anzeigen lässt. Ebenfalls können die Scrollbuttons durch skriptlose Prims mit Namen '**prev**' und '**next**' realisiert werden.

Der Buttonname kann auch mit einer geschlossenen Kennliste anfangen, die Kennliste wird durch den Buttonservice ignoriert. Beispielweise erzeugen die Prims mit Namen '**@text@.menu**' und '**@psource@prev**' dieselben Buttonnachrichten, wie die Prims mit Namen '**.menu**' und '**prev**', werden aber auch zum Anzeigen von Hovertext und Partikelsalven verwendet.

Wie auch immer, einem Buttonprim eine Thumbkennung zu geben, etwa durch den Namen '**@thumb 2@.menu**' ist keine gute Idee: Das Prim wird durch den Buttonservice und den Thumbservice behandelt. Beise Services reagieren aufs Klicken. Das erzeugt ein Servicekonflikt und kann zum unerwarteten Ergebnis führen.

Touchservice

Wenn ein, interpretiert der Kernskript das Ankliken des Systemprims durchs Versenden der Buttonnachricht '**.trigger**'. Diese Nachricht löst wiederum den Teleport zum angezeigten Ziel aus.

Klicks auf andere Prims werden auf die gleiche Weise interpretiert, jedoch nur, wenn sie nicht bereits durch Thumbservice oder Buttonservice abgefangen wurden.

Der Service erlaubt es, auf den Sensorskript zu verzichten, falls der Teleporter aufs Anklicken reagieren soll und die Distanz zur anklickenden Person unwichtig ist.

Menüservice

Wenn ein, wird ein Langklick auf jeden Prim des Teleporters als '`.menu`' Nachricht interpretiert, also das Besitzermenü öffnet. Langer Klik ist, wenn die Maustaste länger als 1s gehalten wurde.

Der Service erlaubt es, einen Teleporter mit nur einem Prim aufzubauen, ohne dass dabei ein Skript verwendet wird, der das Besitzermenü öffnen lässt.

2.4 Schnittstellen

Wie bereits erklärt, kommunizieren alle Skripte via Linknachrichten, Abschnitt 2.3. Diese sind immer auf das Prim selbst beschränkt. Dies erlaubt es, Teleportsysteme in verlinkten Prims ohne Übersprechen zu installieren. Die Abbildung 2.20 zeigt die API-Karte des Teleportersystems.

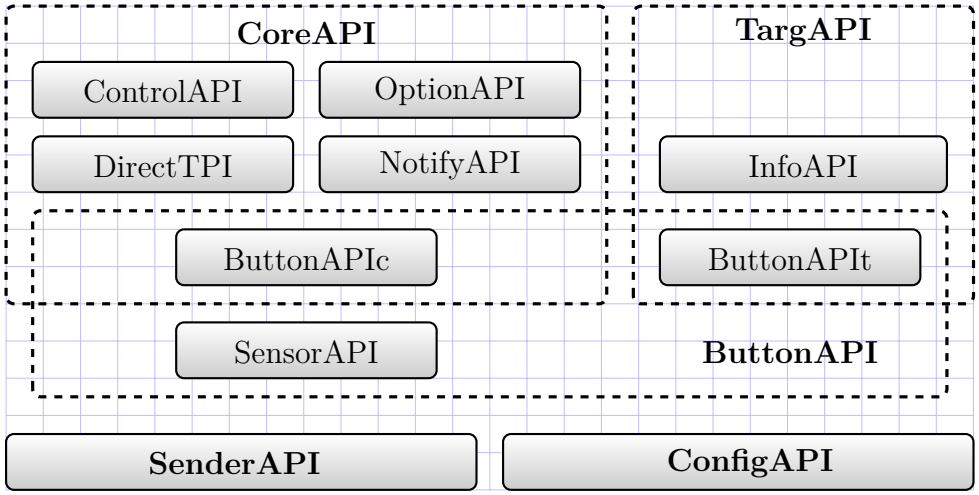


Figure 2.20: API-Karte der RTS Skripte

2.4.1 Format

Es werden zwei Formate verwendet. Im tabellarischen Format werden Nachrichten des API dargestellt. Im Kurzformat Bezug auf einzelne Nachrichten genommen.

Tabellarisches Format

Die Tabellen geben eine detaillierte Nachrichtbeschreibung wieder.

Die Spalte '`dir`' gibt die Senderichtung an: '`IN`' für rein eingehende Nachrichten, '`OUT`' für rein ausgehende Nachrichten und '`BI`' für Bidirektionale (Nachrichten, die der Skript versendet und empfangen kann.)

Die Spalte '`num`' gibt den Zahlenparameter der Nachricht an, die Spalte '`str`' den Stringparameter und die Spalte '`key`' den Schlüsselparameter oder '`-`', falls nicht relevant. Stringkonstanten werden in Anführungszeichen genommen.

Beispielsweise die Nachricht '`STOP`', Tabelle 2.8: `BI`, 0, `"STOP"`, '`-`'. Um den Teleporter anzuhalten, kann ein Skript also diesen Befehl ausführen:

```
1 llMessageLinked(LINK_THIS, 0, "START", "");
```

Da diese Nachricht bidirektional ist, kann sie auch der Kernskript empfangen und den Teleporter richtig anhalten. Anders die Nachricht '`INIT`', sie wird vom Kernskript nur versendet, nicht empfangen, andere Skripte reagieren darauf zwar, der Kernskript selbst aber nicht.

Kurzformat

Wird auf eine Nachricht Bezug genommen, wird sie im Text verkürzt als Vektor angegeben: '`<num, str, key>`'. Ist der Schlüsselparameter nicht relevant, wird er weggelassen. Die Nachricht '`STOP`' kann also als `<0, "STOP">` angegeben werden. Ist der Zahlenparameter aus Kontext klar, kann er ebenfalls ausgelassen werden: `<STOP>` in diesem Fall.

Ein Beispiel für diese Angabe stellt die Nachrichtenfolge 2.22 dar. Dem Kurzformat folgt auch die API-Beschreibung in beiliegenden Skripten, wobei dort auch die Richtung angegeben wird: '`->[]`' für eingehende, '`[]->`' für ausgehende und '`->[]->`' für bidirektionale Nachrichten.

2.4.2 ButtonAPI

Die ButtonAPI hat eine Sonderstellung. Ihre Nachrichten übermitteln Benutzerbefehle, z.B. Klicks auf Buttons. Die Nachrichten benutzen die Nummer 1, als Stringparameter den Buttonnamen und als Schlüsselparameter den die UUID des Agenten der den Button angeklickt haben soll.

Die Nachrichten können via Buttonservice durch den Kernskript erzeugt werden, das erlaubt es anderen Skripten, die Steuerung über solche Nachrichten zu realisieren, und das führt wiederum dazu, dass Skripte Buttonnachrichten emulieren können, um andere Skripte anzusteuern.

Deshalb können die Nachrichten weder nach Sender noch Empfänger vollständig einer API zugeordnet werden. Stattdessen werden die Nachrichten der ButtonAPI teilweise unter CoreAPI, TargAPI und SensorAPI untergebracht, die APIs schneiden sich in der Abbildung 2.20.

Es wird lediglich unterschieden zwischen Nachrichten, die einer Zielauswahl dienen (sie fangen mit einem Buchstaben an) und Steuernachrichten, diese fangen mit einem der Symbolzeichen '`.`', '`*`' und '`_`' an.

2.4.3 CoreAPI

Die CoreAPI besteht aus fünf Schnittstellen: ControlAPI, OptionAPI, DirectTPI, NotifyAPI und ButtonAPIc. Sie erfassen Nachrichten, die vom Kernskript versendet oder empfangen werden.

ControlAPI

Nachrichten dieser Schnittstelle dienen der Gerätesteuerung oder Statusmeldung und werden über die Nummer 0 versendet, Tabelle 2.8.

dir	num	str	key	Beschreibung
OUT	0	"INIT"	-	Setzt alle Skripte zurück
BI	0	"START"	-	Startet den Teleporter (das Gerät)
BI	0	"STOP"	-	Hält den Teleporter an
OUT	0	"TAG"	"link:tags"	Prim mit der Linknumber nutzt diese Kennungen
BI	0	"TEXT"	Text	Übermittelt den Hovertext und zeigt ihn via Textservice auch an
OUT	0	"CMD"	"close"	Besitzermenü wurde geschlossen über den close-Button oder Timeout
OUT	0	"TP:INIT"	UUID	Teleportvorgang für den Agenten mit der UUID wird initiiert
OUT	0	"TP:OK"	UUID	Das Relay des Agenten mit UUID hat die Anfrage akzeptiert, der Vorgang ist erfolgreich abgeschlossen
OUT	0	"TP:KO"	UUID	Das Relay hat die Anfrage abgewiesen, der Vorgang ist ebenfalls abgeschlossen
OUT	0	"TP:ABORT"	UUID	Teleport-Abbruch: Keine Antwort vom Relay
OUT	0	"TP:ERROR"	UUID	Teleport aus Fehlergründen nicht gestartet

Table 2.8: ControlAPI

OptionAPI

Dieses Interface wird zum Einstellen des Geräts verwendet. Zwei Nachrichten gehören dazu, `<OPT>` und `<META>` (seit RTS 2.1), Tabelle 2.9.

dir	num	str	key	Beschreibung
IN	0	"OPT"	Info	Aktiviert oder deaktiviert Kernservices Info = (+|-)name$_1$,...,(+|-)name$_N$
OUT	0	"META"	Info	Übermittelt Metaparameter eines Prims Info = link@meta$_1$@...@meta$_N$

Table 2.9: OptionAPI

Die Nachricht **<OPT>** schaltet Kernservices ein oder aus. Der Schlüsselparameter enthält dabei eine kommaseparierte Liste von Servicenamen, mit einem Vorzeichen versehen, '**+**' fürs Einschalten und '**-**' fürs Ausschalten des Service. Sollte der Infostring falsch formatiert sein oder unbekannten Service angeben, wird die Fehlermeldung mit dem Code 10 produziert.

Beispiel: Die Nachricht **<0, "OPT", "+unsit,-thumbs,-buttons,-text">** schaltet den Unsitservice ein und die Thumbs-, Button- und Textservices aus, Tabelle 2.10.

Flag	Kernservice	standardmäßig
unsit	Unsitservice	inaktiv
touch	Touchservice	aktiv
text	Textservice	aktiv
message	Nachrichtenservice	aktiv
image	Imageservice	aktiv
thumbs	Thumbsservice	aktiv
buttons	Buttonservice	aktiv
menu	Menüservice	aktiv

Table 2.10: OptionAPI, Optionennamen der Kernservices

Die Nachricht **<META>** übermittelt Metaparameter, die einem Prim via Zeichen '**@**' im Namen zugewiesen wurden. Der erste Metaparameter ist die Kennliste, der zweite Metaparameter – der Buttonname. Weitere Parameter werden durch RTS nicht verwendet, dürfen aber von den Treiberskripten verwendet werden. Die Nachricht ist für jedes Prim versendet, welches das Zeichen '**@**' im Namen enthällt. Beispiele...

- Wenn ein Prim mit der Linknummer 2 den Namen '**@text@.menu**' hat, wird die Nachricht **<0, META, "2@text@.menu">** versendet. Hier ist der Button namens '**.menu**' mit '**text**' gekennzeichnet.
- Hat ein Prim mit der Linknummer 3 den Namen '**@text,psource**', wird die Nachricht **<0, META, "3@text,psource">** versendet. Das Prim hat nur die

Kennliste zugewiesen.

- Hat ein Prim mit der Linknummer 3 den Namen '`text@psource@.menu`', wird die Nachricht `<0, META, "4@@text@psource@.menu">` versendet. Dies ist ein Button '`text`' mit Metaparametern '`psource`' und '`.menu`' und ohne die Kennliste.

ButtonAPIc

Die Nachrichten dieser API starten einige Steueraktionen seitens des Benutzers, wie Menüaufruf oder auslösen von Teleports. Sie werden durch den Kernskript ausgeführt und werden unter CoreAPI aufgeführt. Das API erhält daher den Suuffix 'c', Tabelle 2.11.

dir	num	str	key	Beschreibung
IN	1	".menu"	UUID	Ist UUID die des Besitzers, zeige ihm das Besitzermenü
IN	1	".lm"	UUID	Der Agent mit UUID fordert die LM bzw. SLURL zum ausgewählten Ziel an
IN	1	".trigger"	UUID	TriggerTP: Teleport des Agenten mit der UUID aufs ausgewählte Ziel
IN	1	".repeat"	UUID	RepeatTP: Teleport des Agenten aufs vorher benutzte Ziel

Table 2.11: ButtonAPIc

DirectTPI

dir	num	str	key	Beschreibung
IN	1	.tptoNXYZ	UUID	*DirectTP*-Befehl, eines zwei Formate: '`<.tpto:X/Y/Z>`' oder '`<.tpto:Name/X/Y/Z>`'.

Table 2.12: DirectTPI

Diese API hat ebenfalls nur eine Nachricht, Tabelle 2.12. Sie übermittelt Koordinaten der Zielposition, auf die der Teleport sofort starten soll. Das erspart einen Großteil von Verwaltungsaufgaben, da die Zielposition nicht auf der Zielliste steht.

Der Stringteil der Nachricht muss einen der beiden angegebenen Formate besitzen, wobei `X`, `Y`, `Z` die globale Zielkoordinaten auf der Grid sind und `Name` ist Name der Sim auf welche die Koordinaten verweisen.

Ist X, Y oder Z negativ, gilt der Befehl als *ungültig* und wird ignoriert. Dasselbe wenn X oder Y (aber nicht beide) größer als 255 ist. Sind beide, X und Y größer als 255, die Koordinate ist *global*. Sind X und Y zwischen 0 und 255 (inklusiv,) gilt die Koordinate als lokal. Ist in diesem Fall der Simname nicht angegeben, wird die aktuelle Sim vorausgesetzt.

Der Kernskript führt den Befehl aber nur dann aus, wenn die Koordinate global ist. Ist dagegen die position lokal, ignoriert der Kernskript den Befehl als nicht ausführbar. Allerdings schaltet sich der Ziellistenskript oder DirektTP Skript ein, löst die globale Position auf und versendet den Befehl erneut, dann mit der globalen Zielkoordinate.

Beispiel: Man möchte den Avatar auf die Position '`Endora/130/167/45`' bringen. Prinzipiell könnte man ausrechnen (Abschnitt 2.1.5), dass diese Position in globalen Koordinaten `<146050, 254375, 45>` wäre. Dann müsste man für den TP diesen Befehl versenden: `<1, ".tpto:146050/254375/45", UUID>` mit der UUID des reisenden Avatars.

Um es aber komfortabler zu haben, kann man das Auflösen den Skripten überlassen. Dazu versendet man die lokale Position: `<.tpto:Endora/130/167/45>`. Zum Beispiel mittels Buttonservice oder Sensorplugins im Chatmodus. Der Kernskript erkennt, dass die Koordinaten lokal sind und ignorert die Anfrage.

Stattdessen erkennt der Ziellistenskript oder DirectTP Skript, dass er gebraucht wird, lösst die Koordinate auf und versendet die Nachricht mit den globalen Koordinaten: `<.tpto:Endora/146050/254375/45>`. Der Kernskript akzeptiert den Befehl und führt den Teleport aus.

Warum ist der Simname bei globalen Koordinaten geblieben? Für den Teleport selbst ist die Sim irrelevant. Jedoch kann der Kernskript damit eine gültige SLURL bilden, die etwa übermittelt wird, wenn der Teleporter im LM Modus arbeitet. Benutzer ohne RLV können so immer noch das Ziel besuchen.

NotifyAPI

dir	num	str	key	Beschreibung
BI	-2	Code	Message	NotifyAPI: Interne Skriptmeldung

Table 2.13: NotifyAPI

Skripte produzieren Fehler-, Status- und weitere Meldungen als Linknachrichten, die dann etwa per IM oder Mail weitergeleitet werden können. Der Meldungsservice leitet etwa diese Nachrichten via SenderAPI direkt als IM an den Besitzer.

Die Nachrichten benutzen die Nummer -2. Der Stringparameter gibt die Art der Meldung an, der Schlüsselparameter ihren Inhalt. Beide haben keine maschinell erfassbare Semantik, die Meldung ist nur für den Benutzer bestimmt, Tabelle 2.13.

Beispiel: `<-2, "ERROR 115", "Section already used">`

dir	num	str	key	Beschreibung
IN	1	"first"	-	Zeige das erste Ziel in der Liste an
IN	1	"last"	-	Zeige das letzte Ziel in der Liste an
IN	1	"prev"	-	Zeige das Ziel vor dem Aktuellen an, vor dem Ersten liegt das letzte Ziel
IN	1	"next"	-	Zeige das Ziel nach dem Aktuellen an, nach dem letzten Ziel liegt das Erste
IN	1	"prevX"	-	Zeige das Ziel X Positionen vor dem Aktuellen an, X ist eine Zahl. 'prev1' entspricht 'prev'
IN	1	"nextX"	-	Zeige das Ziel X Positionen nach dem Aktuellen an. 'next1' entspricht 'next'
IN	1	"posX"	-	Zeige das Ziel auf der Position X, 'pos0' entspricht 'first'
IN	1	"random"	-	Das Ziel auf einer zufälligen Listenposition
IN	1	"default"	-	Zeige das als Default gekennzeichnete Ziel an
IN	1	prefix	-	Textsuche: Es wird nach einem Ziel gesucht, dessen Name in der Kleinschreibung mit Präfix anfängt

Table 2.14: ButtonAPIt

2.4.4 TargAPI

TargAPI ist die API des Ziellistenskripts. Sie besteht aus zwei Teilschnittstellen, *ButtonAPIt* und *InfoAPI*.

ButtonAPIt

Das sind Nachrichten des ButtonAPI, die den Ziellistenskript ansteuern, Tabelle 2.14. Der Keyparameter ist für die Zielliste selbst irrelevant, daher ein '`-`' in der Spezifikation. Werden die Nachrichten aber durch den Buttonservice oder Sensorplugin versendet, übermitteln sie auch die UUID des Avatars, der den Buttonprim anklickte oder den Chatbefehl ausgesprochen hat.

Die Nachrichten führen dazu, dass ein Ziel in der Liste ausgewählt wird: Ist die Zielliste leer oder wurde kein Ziel mit dem angeforderten Präfix gefunden, wird die Anfrage ignoriert, ansonsten die Nachricht `<TARGET>` der InfoAPI versendet.

InfoAPI

Diese API übermittelt die Zieldaten oder stellt das Plugin ein, Tabelle 2.15.

dir	num	str	key	Beschreibung
OUT	2	"LOAD"	Num	Num ist die Anzahl geladener Ziele
OUT	2	"TARGET"	Data	Übermittelt Zieldaten als String
IN	2	"SIZE"	Num	Setzt Anzahl der Bilder im Zielfenster
IN	2	"INVD"	-	Markiert alle Zielpositionen als ungültig

Table 2.15: InfoAPI

Die Nachricht `<LOAD>` wird während der Konfiguration beim Registrieren jedes Ziels versendet. Dadurch kann etwa der Fortschritt beim Laden einer langen Zieliste dargestellt werden.

Die Nachricht `<TARGET>` wird beim Anzeigen eines Ziels versendet und übermittelt Zieldaten als Schlüsselparameter, die via '`|`' zu einem String konkateniert werden:

```
"tnum|name|spec|lpos|gpos|width|key0|...|keyN"
```

Die Semantik des Strings gibt die Tabelle 2.16 an.

index	typ	Beschreibung	
0	integer	tnum	Nummer des ausgewählten Ziels, erstes = 0
1	string	name	Sein Name, z.B. "Home Skybox 2km"
2	string	spec	Name der LM oder Sim (aus der SLURL)
3	vector	lpos	Lokale Zielkoordinate (aus der SLURL) oder `<0.0, 0.0, -1.0>` (Definition via LM)
4	vector	gpos	Globale Zielkoordinate (aufgelöst)
5	integer	size	Anzahl Bilder im angezeigten Zielfenster. Das Fenster hat immer eine ungerade Anzahl Bilder
6..S+5	key	keyX	Bilder der im Fenster angezeigen Ziele, falls S die Anzahl von Bilder im Zielfenster ist. Das mittlere Bild ist gerade ausgewählt

Table 2.16: Zielinformation, aufgelöst

Um die Daten aus dem String als eine Liste zurückzugewinnen, bitte diesen Befehl ausführen:

```
1 list data = llParseStringKeepNulls((string)kData, ["|"], []);
```

Sie erhalten eine Liste mit dem Inhalt genau wie in der Tabelle 2.16 angegeben.[23]

Die Nachricht **<INVD>** (invalidate) leert den Simcache und markiert alle Zielpositionen als ungültig. Dadurch müssen alle Ziele wieder aufgelöst werden, wenn sie angefordert werden.

Die Nachricht **<SIZE>** übermittelt die Anzahl von anzeigbaren Zielbildern. Auf diese Weise wird eine optimale Länge der **<TARGET>** Nachricht eingestellt. Die Zahl muss zwischen 1 und 21 liegen. Die bereits eingestellte Bildzahl kann sich nur vergrößern.

Grund: Es können verschiedene Skripte installiert werden, die eine verschiedene Anzahl Bildern erwarten und einstellen. Mit zu viel Bildern können die Skripte umgehen (und müssen auch können), mit zu wenig Bildern jedoch nicht.

2.4.5 SensorAPI

Der Sensorplugin '**.sensor**' besitzt kein eigenes API. Er erzeugt Nachrichten der ButtonAPI, Abschnitt 2.4.2. In den Modi Klick, Kollision, Sit und Scan versendet der Skript nur die **<.trigger>** Nachricht. Im Chatmodus werden Chatbefehle eines Avatars in Buttonnachrichten über diesen Avatar umgewandelt. So können alle Nachrichten der ButtonAPI auf der Chatzeile abgesetzt werden.

2.4.6 SenderAPI

Der Senderskript versendet Dateien, IMs und eMails an angegebene Empfänger und vermeidet Verzögerungen beim Skript, der diese Aufgabe ausführen würde. Die Nachrichten belegen die Nummern 10, 11 und 12, Tabelle 2.17.

dir	num	str	key	Beschreibung
IN	10	File	UUID	Übergibt die Datei an den Agenten mit UUID
IN	11	Msg	UUID	Versendet die Msg als eine IM an den Agenten
IN	12	Text	e@mail	Versendet den Text als e-Mail an die Addi

Table 2.17: SenderAPI

[23] Der Datentyp jedes Eintrags ist string, und muss zum angegebenen Typ gecastet werden.

2.4.7 ConfigAPI

Der Konfigurationsskript übernimmt das lesen der Konfigurationsnotekarte und Aufbereiten der gelesenen Daten. Das Format der Notekarte sieht eine einfache Strukturierung der Daten vor: Der Inhalt besteht aus leeren Zeilen, Kommentaren, und Parameter-Wert Paaren, die in Sektionen gruppiert werden, Abbildung 2.21.

```
# Kommentarzeile
# Noch eine Kommentarzeile

[*MAIN]
accept  = group # Jeder aus der Gruppe
color   = white

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

Figure 2.21: Konfigurationsbeispiel

Das Zeichen '**#**' und der nachfolgende Zeilenrest gelten als Kommentar. Beim Einlesen der Konfiguration werden Kommentare entfernt und Zeilen ausgelassen, die leer waren oder nach Entfernen von Kommentaren leer werden.

Eckige Klammern schließen Sektionsnamen ein. Sie werden mit der laufenden Sektionsnummer gemeldet: Das erlaubt gezieltes Wiedereinlesen jeder Sektion im Betrieb. Dadurch könnte theoretisch eine viel größere Anzahl von Zielen festgelegt werden, als 80. Sobald ein Ziel angefordert wird, das nicht mehr im Speicher vorhanden ist, müsste man nur die entsprechende Sektion erneut einlesen. Aus Laggründen wird das jedoch nicht verwendet.

Parameter und Werte werden ohne umschließende Leerzeichen gemeldet. Parameternamen werden zuvor in Kleinbuchstaben konvertiert. Außerdem wird der Wert der Parameter '**color**' und '**colour**' geparst und der Farbname in den Vektorwert übersetzt.

Die vollständige API des Konfigurationsskripts

Die Nachrichten belegen die Nummer -1, Tabelle 2.18.

Die eingehende Nachricht **<CONFIG>** startet die Konfiguration über die übergebene Notekarte. Ist '**Note**' eine UUID, wird aus einer Notekarte mit dieser UUID gelesen, sonst aus Notekarte mit diesem Namen. Ist '**Note**' ein Leerstring oder **<NULL_KEY>**,

dir	num	str	key	Beschreibung
IN	-1	"CONFIG"	Note	Starte Konfiguration über die Note
OUT	-1	"CONFIG"	"START"	Konfiguration beginnt, wartet 0.25s bevor weitere Nachrichten folgen
OUT	-1	"CONFIG"	"END"	Konfiguration abgeschlossen, wartet 0.25s mit dieser Nachricht
IN	-1	"READ"	Num	Wiedereinlesen einer Sektion mit der gegebenen Laufnummer starten
OUT	-1	"READ"	"START"	Wiedereinlesen der angeforderten Sektion beginnt
OUT	-1	"READ"	"END"	Wiedereinlesen der angeforderten Sektion abgeschlossen
OUT	-1	[Name]	Num	Eine Section mit dem Namen und Laufzeitnummer wird gelesen. Wartet 0.2s mit Lesen weiteren Sektionszeilen
OUT	-1	Param	Value	Das Parameter-Wert Paar eingelesen. Leerzeichen um das Parameter und Wert werden ausgelassen, der Parametername in Kleinbuchstaben konvertiert

Table 2.18: Konfigurationsskript, API

```
        <-1, "CONFIG", "START">
0.25s
        <-1, "[*MAIN]", "0">
0.2s
        <-1, "accept", "group">
        <-1, "color", "<1.00000, 1.00000, 1.00000>">
        <-1, "[Endora]", "1">
0.2s
        <-1, "image", "8cee492c-917e-341d-b3a9-63d83ed44d34">
        <-1, "target", "Endora/127/169/42">
0.25s
        <-1, "CONFIG", "END">
```

Figure 2.22: Nachrichtfolge, Konfiguration 2.21

wird aus der Notekarte namens '**config**' gelesen. Der Konfigurationsprozess wird mit den ausgehenden **<CONFIG>** Nachrichten gemeldet.

Die eingehende Nachricht **<READ>** startet das Wiedereinlesen einer Sektion. Dazu muss die Konfiguration abgeschlossen sein, damit Laufzeitnummern von Sektionen gespeichert sind. Das Wiedereinlesen wird mit den ausgehenden **<READ>** Nachrichten gemeldet.

Die Nachricht **<[Name]>** wird einmal pro Sektion versendet, jedoch nicht beim Wiedereinlesen der Sektion. Danach werden Sektionsdaten via **<Param>** Nachrichten gemeldet.

Beispiel. Angenommen es wird über eine Notekarte konfiguriert, mit dem Inhalt aus 2.21. Dann entsteht der Nachrichtenfluss wie in Abbildung 2.22.

Fehlermeldungen

Der Konfigurationsskript versendet Fehlermeldungen, wie Abschnitt 2.5 vorgestellt:

- Wird Konfiguration über eine fehlende Notekarte gestartet, wird eine Fehlermeldung mit dem Code 120 gesendet.
- Ist es keine Notekarte, gibt das einen Fehler mit dem Code 121.
- Beim Syntaxfehler im Konfigurationstext – ein Fehler mit dem Code 100.

Nur Zeilen, die keine Syntaxfehler enthalten, produzieren Konfigurationsnachrichten.

2.5 Fehlercodes

Systemskripte produzieren Fehlermeldungen, die via Nachrichtservice an den Besitzer weitergeleitet werden. Die Fehlercodes werden in 2.19 angegeben.

Code	Beschreibung
10	Fehler beim Setzen von Optionen, Abschnitt 2.4.3: Die Nachricht zum Umschalten von Kernservices ist falsch formatiert.
22	Ein Skript hat falsche Besitzerrechte, i.e. copy+trans.
100	Syntaxfehler in der Konfiguration, z.B. ein Parameter ohne Wert angegeben.
110	Unbekannter Parametername für gegebene Sektion. Tippfehler oder in der falschen Section notiert.
111	Der Parameterwert ist falsch. Falsche Flags eingesetzt oder der Wert ist außerhalb zulässiger Grenzen.
112	Verletzung des Datenlimits, zB. zuviele Ziele in der Konfiguration angegeben.
114	Ein obligatorischer Parameter fehlt in der Section.
115	Ein Zielname in der Konfiguration doppelt verwendet.
116	Ein Skript findet die Sektion nicht, die er benötigt.
120	Eine Datei im Inventar verloren. Konfiguration: Ein Bild oder LM ist angegeben, aber im Objektinventar nicht gefunden. Betrieb: Eine LM ist nicht mehr im Objektinventar vorhanden, evtl. zufällig gelöscht.
121	Falscher Typ, z.B. Bild anstelle LM angegeben.
122	Falsche Rechte, z.B. ein Bild ist nicht fullperm.
123	Kein Zugriff auf den Inhalt, zB. ein Bild ist via Key angegeben, der vermeintliche Key ist aber kein Key.
133	Zielposition läst sich nicht (mehr) ermitteln.

Table 2.19: Fehlermeldungen

Nachrichten mit dem Code unter 100 sind für Integritätsprobleme reserviert und geben betreffende Scriptnamen an. Nachrichten mit dem Code 100 werden durch den Konfigurationsskript versendet und geben die Zeilennummer der Fehlerposition an. Nachrichten mit der höheren Codenummer versenden andere Skripte. Sie haben keinen direkten Zugriff auf die Konfigurationskarte und geben den Sektionsnamen als Orientierungshilfe an.

3 Color palettes

name	preview	LSL vector
BlanchedAlmond		<1.000000, 0.921570, 0.803920>
AntiqueWhite		<0.980390, 0.921570, 0.843140>
PapayaWhip		<1.000000, 0.937250, 0.835290>
MistyRose		<1.000000, 0.894120, 0.882350>
LemonChiffon		<1.000000, 0.980390, 0.803920>
LightGoldenrodYellow		<0.980390, 0.980390, 0.823530>
Beige		<0.960780, 0.960780, 0.862750>
Lavender		<0.901960, 0.901960, 0.980390>
Linen		<0.980390, 0.941180, 0.901960>
Cornsilk		<1.000000, 0.972550, 0.862750>
NavajLace		<0.992160, 0.960780, 0.901960>
LightCyan		<0.878430, 1.000000, 1.000000>
LightYellow		<1.000000, 1.000000, 0.878430>
Honeydew		<0.941180, 1.000000, 0.941180>
SeaShell		<1.000000, 0.960780, 0.933330>
LavenderBlush		<1.000000, 0.941180, 0.960780>
AliceBlue		<0.941180, 0.972550, 1.000000>
FloralWhite		<1.000000, 0.980390, 0.941180>
MintCream		<0.960780, 1.000000, 0.980390>
Azure		<0.941180, 1.000000, 1.000000>
Ivory		<1.000000, 1.000000, 0.941180>
GhostWhite		<0.972550, 0.972550, 1.000000>
Snow		<1.000000, 0.980390, 0.980390>
White		<1.000000, 1.000000, 1.000000>

Table 3.1: White colors

name	preview	LSL vector
ShadowMaroon		<0.501960, 0.000000, 0.000000>
DarkRed		<0.545100, 0.000000, 0.000000>
Firebrick		<0.698040, 0.133330, 0.133330>
Brown		<0.647060, 0.164710, 0.164710>
Red		<1.000000, 0.000000, 0.000000>
IndianRed		<0.803920, 0.360780, 0.360780>
RosyBrown		<0.737250, 0.560780, 0.560780>
LightCoral		<0.941180, 0.501960, 0.501960>
Salmon		<0.980390, 0.501960, 0.447060>
Tomato		<1.000000, 0.388240, 0.278430>
DarkSalmon		<0.913730, 0.588240, 0.478430>
Coral		<1.000000, 0.498040, 0.313730>
OrangeRed		<1.000000, 0.270590, 0.000000>
LightSalmon		<1.000000, 0.627450, 0.478430>

Table 3.2: Red colors

name	preview	LSL vector
Sienna		<0.627450, 0.321570, 0.176470>
Chocolate		<0.823530, 0.411760, 0.117650>
SaddleBrown		<0.545100, 0.270590, 0.074510>
SandyBrown		<0.956860, 0.643140, 0.376470>
PeachPuff		<1.000000, 0.854900, 0.725490>
Peru		<0.803920, 0.521570, 0.247060>
Bisque		<1.000000, 0.894120, 0.768630>
DarkOrange		<1.000000, 0.549020, 0.000000>
Burlywood		<0.870590, 0.721570, 0.529410>
Tan		<0.823530, 0.705880, 0.549020>
NavajoWhite		<1.000000, 0.870590, 0.678430>
Moccasin		<1.000000, 0.894120, 0.709800>
Orange		<1.000000, 0.647060, 0.000000>
Wheat		<0.960780, 0.870590, 0.701960>
DarkGoldenrod		<0.721570, 0.525490, 0.043140>
Goldenrod		<0.854900, 0.647060, 0.125490>

Table 3.3: Brown colors

name	preview	LSL vector
Olive		<0.501960, 0.501960, 0.000000>
DarkKhaki		<0.741180, 0.717650, 0.419610>
Yellow		<1.000000, 1.000000, 0.000000>
Gold		<1.000000, 0.843140, 0.000000>
LightGoldenrod		<0.933330, 0.866670, 0.509800>
Khaki		<0.941180, 0.901960, 0.549020>
PaleGoldenrod		<0.933330, 0.909800, 0.666670>

Table 3.4: Yellow colors

name	preview	LSL vector
OliveDrab		<0.419610, 0.556860, 0.137250>
YellowGreen		<0.603920, 0.803920, 0.196080>
DarkOliveGreen		<0.333330, 0.419610, 0.184310>
OliveGreen		<0.568630, 0.709800, 0.333330>
GreenYellow		<0.678430, 1.000000, 0.184310>
Chartreuse		<0.498040, 1.000000, 0.000000>
LawnGreen		<0.486270, 0.988240, 0.000000>
DarkGreen		<0.000000, 0.392160, 0.000000>
ShadowGreen		<0.000000, 0.501960, 0.000000>
ForestGreen		<0.133330, 0.545100, 0.133330>
Green		<0.000000, 1.000000, 0.000000>
Lime		<0.000000, 1.000000, 0.000000>
LimeGreen		<0.196080, 0.803920, 0.196080>
DarkSeaGreen		<0.560780, 0.737250, 0.560780>
LightGreen		<0.564710, 0.933330, 0.564710>
PaleGreen		<0.596080, 0.984310, 0.596080>
SeaGreen		<0.180390, 0.545100, 0.341180>
MediumSeaGreen		<0.235290, 0.701960, 0.443140>
SpringGreen		<0.000000, 1.000000, 0.498040>
MediumSpringGreen		<0.000000, 0.980390, 0.603920>

Table 3.5: Green colors

name	preview	LSL vector
MediumAquamarine		<0.400000, 0.803920, 0.666670>
Aquamarine		<0.498040, 1.000000, 0.831370>
Turquoise		<0.250980, 0.878430, 0.815690>
LightSeaGreen		<0.125490, 0.698040, 0.666670>
MediumTurquoise		<0.282350, 0.819610, 0.800000>
DarkSlateGray		<0.184310, 0.309800, 0.309800>
Teal		<0.000000, 0.501960, 0.501960>
DarkCyan		<0.000000, 0.545100, 0.545100>
Cyan		<0.000000, 1.000000, 1.000000>
Aqua		<0.000000, 1.000000, 1.000000>
PaleTurquoise		<0.686270, 0.933330, 0.933330>
DarkTurquoise		<0.000000, 0.807840, 0.819610>
CadetBlue		<0.372550, 0.619610, 0.627450>
PowderBlue		<0.690200, 0.878430, 0.901960>
LightBlue		<0.678430, 0.847060, 0.901960>
DeepSkyBlue		<0.000000, 0.749020, 1.000000>
SkyBlue		<0.529410, 0.807840, 0.921570>
LightSkyBlue		<0.529410, 0.807840, 0.980390>
SteelBlue		<0.274510, 0.509800, 0.705880>
DodgerBlue		<0.117650, 0.564710, 1.000000>
LightSlateGray		<0.466670, 0.533330, 0.600000>
SlateGray		<0.439220, 0.501960, 0.564710>
LightSteelBlue		<0.690200, 0.768630, 0.870590>
CornflowerBlue		<0.392160, 0.584310, 0.929410>
RoyalBlue		<0.254900, 0.411760, 0.882350>
NavyBlue		<0.000000, 0.000000, 0.501960>
Navy		<0.000000, 0.000000, 0.501960>
DarkBlue		<0.000000, 0.000000, 0.545100>
MidnightBlue		<0.098040, 0.098040, 0.439220>
MediumBlue		<0.000000, 0.000000, 0.803920>
Blue		<0.000000, 0.000000, 1.000000>
SlateBlue		<0.415690, 0.352940, 0.803920>
LightSlateBlue		<0.517650, 0.439220, 1.000000>
DarkSlateBlue		<0.282350, 0.239220, 0.545100>
MediumSlateBlue		<0.482350, 0.407840, 0.933330>
Indigo		<0.294120, 0.000000, 0.509800>

Table 3.6: Blue colors

name	preview	LSL vector
MediumPurple		<0.576470, 0.439220, 0.858820>
BlueViolet		<0.541180, 0.168630, 0.886270>
Purple		<0.627450, 0.125490, 0.941180>
DarkOrchid		<0.600000, 0.196080, 0.800000>
DarkViolet		<0.580390, 0.000000, 0.827450>
MediumOrchid		<0.729410, 0.333330, 0.827450>
ShadowPurple		<0.501960, 0.000000, 0.501960>
DarkMagenta		<0.545100, 0.000000, 0.545100>
Magenta		<1.000000, 0.000000, 1.000000>
Fuchsia		<1.000000, 0.000000, 1.000000>
Plum		<0.866670, 0.627450, 0.866670>
Violet		<0.933330, 0.509800, 0.933330>
Thistle		<0.847060, 0.749020, 0.847060>
Orchid		<0.854900, 0.439220, 0.839220>
VioletRed		<0.815690, 0.125490, 0.564710>
MediumVioletRed		<0.780390, 0.082350, 0.521570>
DeepPink		<1.000000, 0.078430, 0.576470>
HotPink		<1.000000, 0.411760, 0.705880>
Maroon		<0.690200, 0.188240, 0.376470>
PaleVioletRed		<0.858820, 0.439220, 0.576470>
Crimson		<0.862750, 0.078430, 0.235290>
Pink		<1.000000, 0.713730, 0.756860>
LightPink		<1.000000, 0.752940, 0.796080>

Table 3.7: Violett colors

name	preview	LSL vector
Black		<0.000000, 0.000000, 0.000000>
DimGray		<0.411760, 0.411760, 0.411760>
ShadowGray		<0.501960, 0.501960, 0.501960>
DarkGray		<0.662750, 0.662750, 0.662750>
Gray		<0.745100, 0.745100, 0.745100>
Silver		<0.752940, 0.752940, 0.752940>
LightGray		<0.827450, 0.827450, 0.827450>
Gainsboro		<0.862750, 0.862750, 0.862750>
WhiteSmoke		<0.960780, 0.960780, 0.960780>
White		<1.000000, 1.000000, 1.000000>

Table 3.8: Grayscale