

---

# SoundScape Renderer

*Release 0.4.2-78-g989568c*

**SSR Team**

**Mar 30, 2017**



<b>1</b>	<b>General</b>	<b>1</b>
1.1	Introduction . . . . .	1
1.2	Quick Start . . . . .	1
1.3	Audio Scenes . . . . .	2
1.4	Audio Scene Description Format (ASDF) . . . . .	3
1.5	IP Interface . . . . .	4
1.6	Bug Reports, Feature Requests and Comments . . . . .	4
1.7	Contributors . . . . .	4
1.8	Your Own Contributions . . . . .	5
<b>2</b>	<b>Compiling and Running the SSR</b>	<b>7</b>
2.1	Choosing the Operating System . . . . .	7
2.2	The Debian Package . . . . .	7
2.3	Getting the Source . . . . .	8
2.4	Configuring and Compiling . . . . .	8
2.5	Installing . . . . .	9
2.6	Uninstalling . . . . .	10
2.7	Compiling and Installing from the Git Repository . . . . .	10
2.8	Running the SSR . . . . .	10
2.9	Configuration Files . . . . .	12
2.10	Head Tracking . . . . .	12
2.11	Using the SSR with different audio clients . . . . .	14
2.12	Mac OS X . . . . .	16
2.13	Using the SSR with DAWs . . . . .	23
2.14	Known Issues (Linux & Mac OS X) . . . . .	23
<b>3</b>	<b>The Renderers</b>	<b>29</b>
3.1	General . . . . .	29
3.2	Binaural Renderer . . . . .	34
3.3	Binaural Room Synthesis Renderer . . . . .	36
3.4	Vector Base Amplitude Panning Renderer . . . . .	36
3.5	Wave Field Synthesis Renderer . . . . .	37
3.6	Ambisonics Amplitude Panning Renderer . . . . .	38
3.7	Generic Renderer . . . . .	39
3.8	Summary . . . . .	40
<b>4</b>	<b>Graphical User Interface</b>	<b>41</b>

4.1	General Layout . . . . .	41
4.2	Mouse Actions . . . . .	43
4.3	Keyboard Actions . . . . .	44
<b>5</b>	<b>Network Interface</b>	<b>47</b>
5.1	Scene . . . . .	47
5.2	State . . . . .	47
5.3	Source . . . . .	48
5.4	Reference . . . . .	48
	<b>Bibliography</b>	<b>51</b>

### Introduction

The SoundScape Renderer (SSR) is a software framework for real-time spatial audio reproduction running under GNU/Linux, Mac OS X and possibly some other UNIX variants. The current implementation provides Wave Field Synthesis (WFS), binaural (HRTF-based) reproduction, binaural room (re-)synthesis (BRITF-based reproduction), head-tracked binaural playback, Ambisonics Amplitude Panning (AAP), and Vector Base Amplitude Panning (VBAP). There are also the slightly exotic Generic Renderer. For each rendering algorithm there is a separate executable file. For more details see section *Renderers*.

The SSR is intended as versatile framework for the state-of-the-art implementation of various spatial audio reproduction techniques. You may use it for your own academic research, teaching or demonstration activities or whatever else you like. However, it would be nice if you would mention the use of the SSR by e.g. referencing *[Geier2008a]* or *[Geier2012]*.

Note that so far, the SSR only supports two-dimensional reproduction for any type of renderer. For WFS principally any convex loudspeaker setup (e.g. circles, rectangles) can be used. The loudspeakers should be densely spaced. For VBAP circular setups are highly recommended. APA does require circular setups. The binaural renderer can handle only one listener at a time.

### Quick Start

After downloading the SSR package, open a shell and use following commands:

```
tar xvzf ssr-x.x.x.tar.gz
cd ssr-x.x.x
./configure
make
make install
qjackctl &
ssr my_audio_file.wav
```

You have to replace `x.x.x` with the current version number, e.g. `0.4.0`. With above commands you are performing the following steps:

- Unpack the downloaded tarball containing the source-code.
- Go to the extracted directory<sup>1</sup>.
- Configure the SSR.
- Install the SSR.
- Open the graphical user interface for JACK (`qjackctl`). Please click “Start” to start the server. As alternative you can start JACK with

```
jackd -d alsa -r 44100
```

See section *Running SSR* and `man jackd` for further options.

- Open the SSR with an audio file of your choice. This can be a multichannel file.

This will load the audio file `my_audio_file.wav` and create a virtual sound source for each channel in the audio file. By default, the SSR will start with the binaural renderer. Please use headphones to listen to the generated output!

If you don't need a graphical user interface and you want to dedicate all your resources to audio processing, try

```
ssr --no-gui my_audio_file.wav
```

For further options, see section *Running SSR* and `ssr --help`.

## Audio Scenes

### Format

The SSR can open `.asd` files (refer to section *ASDF*) as well as normal audio files. If an audio file is opened, SSR creates an individual virtual sound source for each channel which the audio file contains. If a two-channel audio file is opened, the resulting virtual sound sources are positioned like a virtual stereo loudspeaker setup with respect to the location of the reference point. For audio files with more (or less) channels, SSR randomly arranges the resulting virtual sound sources. All types that `ecasound` and `libsndfile` can open can be used. In particular this includes `.wav`, `.aiff`, `.flac` and `.ogg` files.

In the case of a scene being loaded from an `.asd` file, all audio files which are associated to virtual sound sources are replayed in parallel and replaying starts at the beginning of the scene. So far, a dynamic handling of audio files has not been implemented.

### Coordinate System

Fig. 1.1 (a) depicts the global coordinate system used in the SSR. Virtual sound sources as well as the reference are positioned and orientated with respect to this coordinate system. For loudspeakers, positioning is a bit more tricky since it is done with respect to a local coordinate system determined by the reference. Refer to Fig. 1.1 (b). The loudspeakers are positioned with respect to the primed coordinates ( $x'$ ,  $y'$ , etc.).

The motivation to do it like this is to have a means to virtually move the entire loudspeaker setup inside a scene by simply moving the reference. This enables arbitrary movement of the listener in a scene independent of the physical setup of the reproduction system.

---

<sup>1</sup> Note that most relative paths which are mentioned in this document are relative to this folder, which is the folder where the SSR tarball was extracted. Therefore, e.g. the `src/` directory could be something like `$HOME/ssr-x.x.x/src/` where “x” stands for the version numbers.

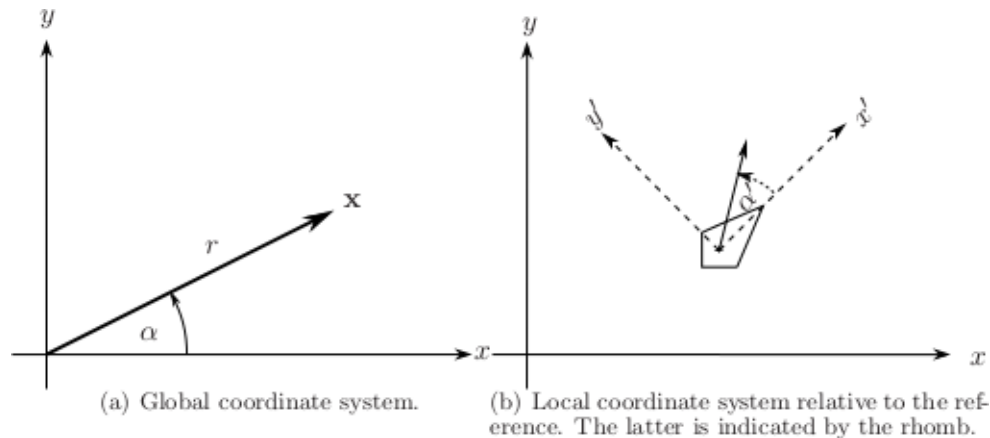


Fig. 1.1: The coordinate system used in the SSR. In ASDF  $\alpha$  and  $\alpha'$  are referred to as azimuth (refer to section *ASDF*).

Please do not confuse the origin of the coordinate system with the reference. The coordinate system is static and specifies absolute positions.

The reference is movable and is always taken with respect to the current reproduction setup. The loudspeaker-based methods do not consider the orientation of the reference point but its location influences the way loudspeakers are driven. E.g., the reference location corresponds to the *sweet spot* in VBAP. It is therefore advisable to put the reference point to your preferred listening position. In the binaural methods the reference point represents the listener and indicates the position and orientation of the latter. It is therefore essential to set it properly in this case.

Note that the reference position and orientation can of course be updated in real-time. For the loudspeaker-based methods this is only useful to a limited extent unless you want to move inside the scene. However, for the binaural methods it is essential that both the reference position and orientation (i.e. the listener's position and orientation) are tracked and updated in real-time. Refer also to Sec. *Head-Tracking*.

## Audio Scene Description Format (ASDF)

Besides pure audio files, SSR can also read the current development version of the *Audio Scene Description Format (ASDF)* [Geier2008b]. Note, however, that so far we have only implemented descriptions of static features. That means in the current state it is not possible to describe e.g. movements of a virtual sound source. As you can see in the example audio scene below, an audio file can be assigned to each virtual sound source. The replay of all involved audio files is synchronized to the replay of the entire scene. That means all audio files start at the beginning of the sound scene. If you fast forward or rewind the scene, all audio files fast forward or rewind. **Note that it is significantly more efficient to read data from an interleaved multichannel file compared to reading all channels from individual files.**

### Syntax

The format syntax is quite self-explanatory. See the examples below. Note that the paths to the audio files can be either absolute (not recommended) or relative to the directory where the scene file is stored. The exact format description of the ASDF can be found in the XML Schema file `asdf.xsd`.

Find below a sample scene description:

```
<?xml version="1.0"?>
<asdf version="0.1">
  <header>
```

```
<name>Simple Example Scene</name>
</header>
<scene_setup>
  <source name="Vocals" model="point">
    <file>audio/demo.wav</file>
    <position x="-2" y="2"/>
  </source>
  <source name="Ambience" model="plane">
    <file channel="2">audio/demo.wav</file>
    <position x="2" y="2"/>
  </source>
</scene_setup>
</asdf>
```

The input channels of a soundcard can be used by specifying the channel number instead of an audio file, e.g. `<port>3</port>` instead of `<file>my_audio.wav</file>`.

## Examples

We provide an audio scene example in ASDF with this release. You find it in `data/scenes/live_input.asd`. If you load this file into the SSR it will create 4 sound sources which will be connected to the first four channels of your sound card. If your sound card happens to have less than four outputs, less sources will be created accordingly. More examples for audio scenes can be downloaded from the SSR website <http://spatialaudio.net/ssr/>.

## IP Interface

One of the key features of the SSR is an interface which lets you remotely control the SSR via a TCP socket using XML messages. This interface enables you to straightforwardly connect any type of interaction tool from any type of operating system. The format of the messages sent over the network is still under development and may very likely change in future versions. Please find some brief information in section *Network*.

## Bug Reports, Feature Requests and Comments

Please report any bugs, feature requests and comments to [ssr@spatialaudio.net](mailto:ssr@spatialaudio.net). We will keep track of them and will try to fix them in a reasonable time. The more bugs you report the more we can fix. Of course, you are welcome to provide bug fixes.

## Contributors

**Written by:** Matthias Geier, Jens Ahrens

**Scientific supervision:** Sascha Spors

**Contributions by:** Peter Bartz, Florian Hinterleitner, Torben Hohn, Lukas Kaser, André Möhl, Till Rettberg, Fiete Winter

**GUI design:** Katharina Bredies, Jonas Loh, Jens Ahrens

**Logo design:** Fabian Hemmert



## Your Own Contributions

The SSR is thought to provide a state of the art implementation of various spatial audio reproduction techniques. We therefore would like to encourage you to contribute to this project since we can not assure to be at the state of the art at all times ourselves. Everybody is welcome to contribute to the development of the SSR. However, if you are planning to do so, we kindly ask you to contact us beforehand (e.g. via [GitHub](#)). The SSR is in a rather temporary state and we might apply some changes to its architecture. We would like to ensure that your own implementations stay compatible with future versions.



---

# Compiling and Running the SSR

---

The following sections describe how to build and/or install the SSR on your computer. The SSR can be used on GNU/Linux and Mac OS X. There is no MS Windows version of the SSR (yet?). If you feel inclined to port the SSR to Windows, let us know!

## Choosing the Operating System

Well, first you'll need a proper operating system. We recommend Debian GNU/Linux (<http://www.debian.org/>) or something Debian-based like Ubuntu (<http://www.ubuntu.com/>). The SSR should work on most modern Linux-distributions, though. You can also try Arch Linux (<http://www.archlinux.org/>), if you want. If you are reluctant to install a new operating system on your computer, you can also install any of the above-mentioned operating systems virtually, for example using <http://www.virtualbox.org/>.

If you happen to have Mac OS X installed on your computer, that's OK, too.

## The Debian Package

Thanks to IOhannes m zmoelnig, the SSR is available as a Debian package!

<http://packages.debian.org/soundscaperenderer>

---

**Note:** The Debian distribution comes in several flavors, such as [stable](#), [testing](#) and [sid](#) (meaning “still in development”). At the time of writing, the *soundscaperenderer* package is part of “testing”, so if you have Debian “stable” installed, you will have to add the package sources for “testing” (you can try the [Debian Sources List Generator](#)). If it still doesn't work, you should consider upgrading your whole system to “testing” or, if that's not feasible, you can still compile and install the SSR from its source (see below for instructions).

---

You can install the *soundscaperenderer* package using your favorite package manager (*apt-get*, *aptitude*, *synaptic*, ...), all dependencies should be installed automatically.

The package may or may not work on other Debian-based distributions like Ubuntu. Feel free to try it and report back to us.

Once the SSR is installed, move on to the section *Running the SSR*.

## Getting the Source

If you didn't receive this manual with the source code of the SSR, you have to download it from the SSR website <http://spatialaudio.net/ssr> or from the git repository. Refer to Section *Compiling and Installing from the Git Repository* in the latter case. Otherwise, continue reading this section.

Once you downloaded the tarball (the file that ends in `.tar.gz`), unpack it and change to the newly created directory:

```
tar xvzf ssr-x.y.z.tar.gz
cd ssr-x.y.z/
```

Of course, `x`, `y` and `z` should have been replaced by the actual version number. If you are new to Unix-like operating systems, you should try *shell completion*: just start typing the beginning of the command (e.g. `tar xvzf ssr`) and hit the Tab-key. Wow, the whole file name gets completed!

## Configuring and Compiling

To build the SSR from source you have to configure first. Open a shell and `cd` to the directory containing the source code of the package and type:

```
./configure
```

This script will check your system for dependencies and prepare the `Makefile` required for compilation. Section *Dependencies* lists the dependencies that must be installed on your system. The `configure` script will signal if dependencies are missing. At successful termination of the `configure` script a summary will show up and you are ready to compile. If dependencies for certain modules of SSR are missing that you are not going to use then you can simply disable the according module by adding the appropriate argument to the call to the “`configure`” script so that you do not need to bother with the dependencies. Examples are

```
./configure --disable-ip-interface
./configure --disable-gui
```

See Section *Hints on Configuration* for details.

If everything went smoothly, you can continue with the next step:

```
make
```

This will take some time (maybe a few minutes). If you have a multi-core or multi-processor computer you can speed up things by specifying the number of processes you want to use with `make -j8` (or any other number that you choose).

## Dependencies

The following list is a list of packages which you should install with the package manager of your choice (`apt-get`, `aptitude`, `synaptic`, ...). However, if you prefer, you can of course also download everything as source code and compile each program yourself.

- **make**
- **g++** (at least version 4.7.3) or **clang**
- **libasio-dev**
- **libqt4-dev** and **libqt4-opengl-dev** (at least version 4.2.2)
- **libecasoundc2.2-dev** or **libecasoundc-dev**
- **ecasound**
- **libxml2-dev**
- **libfft3-dev**
- **libsndfile1-dev**
- **libjack-dev** or **libjack-jackd2-dev**
- **jackd** or **jackd1** or **jackd2**
- **pkg-config**

Extra dependencies for installing from the Git repository:

- **libtool**
- **automake**

Note that there are additional dependencies for some of the tracking systems that SSR supports. Refer to Section [Head Tracking](#) for further information.

To use a different compiler, you can specify it with CXX:

```
./configure CXX=clang++
```

## Hints on Configuration

If you encounter problems configuring the SSR these hints could help:

- Ensure that you really installed all libraries (**lib**) with devel-package (**devel** or **dev**, where available) mentioned in Section [Dependencies](#).
- It may be necessary to run `ldconfig` after installing new libraries.
- Ensure that `/etc/ld.so.conf` or `LD_LIBRARY_PATH` are set properly, and run `ldconfig` after changes.
- If a header is not installed in the standard paths of your system you can pass its location to the configure script using `./configure CPPFLAGS=-Iyourpath`.

Note that with `./configure --help` all configure-options are displayed, e.g. in Section “Optional Features” you will find how to disable compilation of the head trackers and many other things. Setting the influential environment variables with `./configure VARNAME=value` can be useful for debugging dependencies.

## Installing

Until now, everything was happening in the source directory (something like `ssr -x.x.x/`). To be really useful, the SSR should be installed on the system with the command

```
make install
```

On some systems you need root privileges for this step. However, it's easier if you set up your system to be able to do it as a normal user. On Debian, it's enough to add your username to the `staff` group. Just run this command (as root or using `sudo`): `adduser myuser staff` (replacing `myuser` by your actual user name). On other systems you may try something like this (again, as root or using `sudo`): `chown myuser:myuser /usr/local -R` (again, replacing `myuser` by your user name).

## Uninstalling

If the SSR didn't meet your expectations, we are very sorry, and of course you can easily remove it from your system with

```
make uninstall
```

## Compiling and Installing from the Git Repository

In short, in order to install on your machine, open a terminal, create a folder that will host the source code, `cd` to that folder, and type

```
git clone https://github.com/SoundScapeRenderer/ssr.git
```

to download the source code. Then execute

```
./autogen.sh
./configure
make
make install
```

inside the folder in which SSR resides. You'll probably have to have root privileges for the installation. Of course, all dependencies of the compilation process apply. See Section [Dependencies](#).

## Running the SSR

Before you start the SSR, start JACK, e.g. by typing `jackd -d alsa -r 44100` in a shell or using the graphical user interface “`qjackctl`”. Now, the easiest way to get a signal out of the SSR is by passing a sound-file directly:

```
ssr YOUR_AUDIO_FILE
```

By default, the SSR starts with the binaural renderer; please use headphones for listening with this renderer. Type `ssr --help` to get an overview of the command line options and various renderers:

```
USAGE: ssr [OPTIONS] <scene-file>
```

```
The SoundScape Renderer (SSR) is a tool for real-time spatial audio
reproduction providing a variety of rendering algorithms.
```

```
OPTIONS:
```

```
Choose a rendering algorithm:
```

```
--binaural      Binaural (using HRIRs)
--brs           Binaural Room Synthesis (using BRIRs)
--wfs           Wave Field Synthesis
```

```

--aap                Ambisonics Amplitude Panning
--vbap              Stereophonic (Vector Base Amplitude Panning)
--generic           Generic Renderer
--nfc-hoa           Near-field-corrected Higher Order Ambisonics (experimental!)

Renderer-specific options:
--hrirs=FILE        Load the HRIRs for binaural renderer from FILE
--hrir-size=VALUE   Maximum IR length (binaural and BRS renderer)
--prefilter=FILE    Load WFS prefilter from FILE
-o, --ambisonics-order=VALUE Ambisonics order to use (default: maximum)
--in-phase-rendering Use in-phase rendering for Ambisonics

JACK options:
-n, --name=NAME      Set JACK client name to NAME
--input-prefix=PREFIX Input port prefix (default: "system:capture_")
--output-prefix=PREFIX Output port prefix (default: "system:playback_")
-f, --freewheel      Use JACK in freewheeling mode

General options:
-c, --config=FILE    Read configuration from FILE
-s, --setup=FILE     Load reproduction setup from FILE
--threads=N          Number of audio threads (default N=1)
-r, --record=FILE    Record the audio output of the renderer to FILE
--loop               Loop all audio files
--master-volume-correction=VALUE
                    Correction of the master volume in dB (default: 0 dB)
--auto-rotation      Auto-rotate sound sources' orientation toward the reference
--no-auto-rotation   Don't auto-rotate sound sources' orientation toward the
↔reference
-i, --ip-server[=PORT] Start IP server (default on)
                    A port can be specified: --ip-server=5555
-I, --no-ip-server   Don't start IP server
--end-of-message-character=VALUE
                    ASCII code for character to end messages with
                    (default 0 = binary zero)
-g, --gui            Start GUI (default)
-G, --no-gui        Don't start GUI
-t, --tracker=TYPE   Start tracker, possible value(s):
                    fastrak patriot vrpn razor
--tracker-port=PORT
                    A serial port can be specified, e.g. /dev/ttyS1
-T, --no-tracker     Don't start tracker

-h, --help           Show this very help information. You just typed that!
-v, --verbose        Increase verbosity level (up to -vvv)
-V, --version        Show version information and exit

```

Choose the appropriate arguments and make sure that your amplifiers are not turned too loud..

To stop the SSR use either the options provided by the GUI (Section [GUI](#)) or type `Ctrl+c` in the shell in which you started the SSR.

## Keyboard actions in non-GUI mode

If you start SSR without GUI (option `--no-gui`), it starts automatically replaying the scene that you have loaded. You can have some interaction via the shell. Currently implemented actions are (all followed by Return):

- c: calibrate tracker (if available)
- p: start playback
- q: quit application
- r: “rewind”; go back to the beginning of the current scene
- s: stop (pause) playback

Note that in non-GUI mode, audio processing is always taking place. Live inputs are processed even if you pause playback.

## Recording the SSR output

You can record the audio output of the SSR using the `--record=FILE` command line option. All output signals (i.e. the loudspeaker signals) will be recorded to a multichannel wav-file named `FILE`. The order of channels corresponds to the order of loudspeakers specified in the reproduction setup (see Sections *Reproduction Setups* and *ASDF*). The recording can then be used to analyze the SSR output or to replay it without the SSR using a software player like “ecaplay” (<http://eca.cx/ecasound/>).

## Configuration Files

The general configuration of the SSR (whether GUI is enabled, which tracker to use, and most other command line arguments) can be specified in a configuration file (e.g. `ssr.conf`). By specifying your settings in such a file, you avoid having to give explicit command line options every time you start the SSR. We have added the example `data/ssr.conf.example`, which mentions all possible parameters. Take a look inside, it is rather self-explanatory. There are three possibilities to specify a configuration file:

Configuration files are loaded in the following order, if certain options are specified more than once, the last occurrence counts. This means that it is not the last file that is loaded that counts but rather the last occurrence at which a given setting is specified.

1. `/Library/SoundScapeRenderer/ssr.conf`
2. `/etc/ssr.conf`
3. `$HOME/Library/SoundScapeRenderer/ssr.conf`
4. `$HOME/.ssr/ssr.conf`
5. **the path(s) specified with the `--config/-c` option(s) (e.g., `ssr -c my_config.file`)**

We explicitly mention one parameter here that might be of immediate interest for you: `MASTER_VOLUME_CORRECTION`. This a correction in dB (!) that is applied – as you might guess – to the master volume. The motivation is to have means to adopt the general perceived loudness of the reproduction of a given system. Factors like the distance of the loudspeakers to the listener or the typical distance of virtual sound sources influence the resulting loudness, which can be adjusted to the desired level by means of the `MASTER_VOLUME_CORRECTION`. Of course, there’s also a command line alternative (`--master-volume-correction`).

## Head Tracking

We provide integration of the *InterSense InertiaCube3* tracking sensor, the *Polhemus Fastrak* and the *Polhemus Patriot*. They are used to update the orientation of the reference (in binaural reproduction this is the listener) in real-time. Please read Sections *Preparing Intersense* and *Preparing Polhemus* if you want to compile the SSR with the support for these trackers.



Note that on startup, the SSR tries to find the tracker. If it fails, it continues without it. If you use a tracker, make sure that you have the appropriate rights to read from the respective port.

You can calibrate the tracker while the SSR is running by pressing `Return`. The instantaneous orientation will then be interpreted as straight forward, i.e. upwards on the screen ( $\alpha = 90^\circ$ ).

## Preparing InterSense InertiaCube3

If you want to compile the SSR with support for the *InterSense InertiaCube3* tracking sensor, please download the *InterSense Software Development Kit* (SDK) from the InterSense website (<http://www.intersense.com>; Support → Downloads → Development and Troubleshooting Tools). Unpack the archive and place the files

- `isense.h` and `types.h` to `/usr/local/include`, and
- `libisense.so` (the version appropriate for your processor type) to `usr/local/lib`.

The SSR configuration script will automatically detect the presence of the files described above and if they are found, enable the compilation for the support of this tracker. To disable this tracker, use `./configure --disable-intersense` and recompile.

If you encounter an error-message similar to `libisense.so: cannot open shared object file: No such file or directory`, but the file is placed correctly, run `ldconfig`.

Make sure that you have the required access rights to the tracker before starting SSR. For you are using the USB connection type

```
sudo chmod a+rw /dev/ttyUSBX
```

whereby X can be any digit or number. If you are not sure which port is the tracker then unplug the tracker, type

```
ls /dev/ttyUSB*
```

replug the tracker, execute above command again and see which port was added. That one is the tracker. It's likely that it is the one whose name contains the highest number.

## Preparing Polhemus Fastrak/Patriot

For incorporation of the *Polhemus Fastrak/Patriot* with serial connection, no additional libraries are required. Make sure that you have the required access rights to the tracker before starting SSR by typing something like

```
sudo chmod a+rw /dev/ttyS0
```

or

```
sudo chmod a+rw /dev/ttyS1
```

or so.

If you want to disable this tracker, use `./configure --disable-polhemus` and recompile.

## Preparing VRPN

In order to use *Virtual Reality Peripheral Network* (VRPN) compatible trackers create a config file `vrpn.cfg` with one of the following lines (or similar)

```
vrpn_Tracker_Fastrak MyFastrak /dev/ttyUSB0 115200
vrpn_Tracker_Fastrak MyOtherFastrak COM1 115200
vrpn_Tracker_Liberty MyPatriot /dev/ttyUSB1 115200
```

... and start `vrpn_server`. You can choose the name of the Tracker arbitrarily. Then, start the SSR with the given Tracker name, e.g.:

```
ssr-binaural --tracker=vrpn --tracker-port=MyFastrak@localhost
```

If the tracker runs on a different computer, use its hostname (or IP address) instead of localhost. You can of course select your head tracker settings by means of *Configuration Files*.

## Using the SSR with different audio clients

This page contains some short description how to connect your own audio files with the SSR using different audio players.

### VLC Media Player

How to connect the SSR in binaural playback mode with the own audio library using Jack and VLC Media Player:

After installing Jack and the SSR (with all needed components: see :ref: 'Configuring and Compiling <configuring>') it is necessary to install the VLC Media Player with its Jack plugin (for example UBUNTU):

1. `sudo apt-get install vlc vlc-plugin-jack`

(or use the packet manager of your choice instead of the command line and install: `vlc` and `vlc-plugin-jack`)

2. After installing open VLC Media Player and navigate to Tools->Preferences Select "All" on the bottom left corner In the appearing menu on the left navigate to "Audio"->"Output Module" and extend it by using "+"

3. In the submenu of "Output Module" select "JACK" and replace "system" by " Binaural-Renderer" in the "Connect to clients matching"-box. Do not forget to enable "Automatically connect to writable clients" above. (Otherwise you have to connect the audio output of vlc with the SSR input after every played audio file using jack.)

(Note: If you want to use another Renderer, e.g. for WFS, you have to enter "WFS-Renderer" in the box)

4. Save your changes.

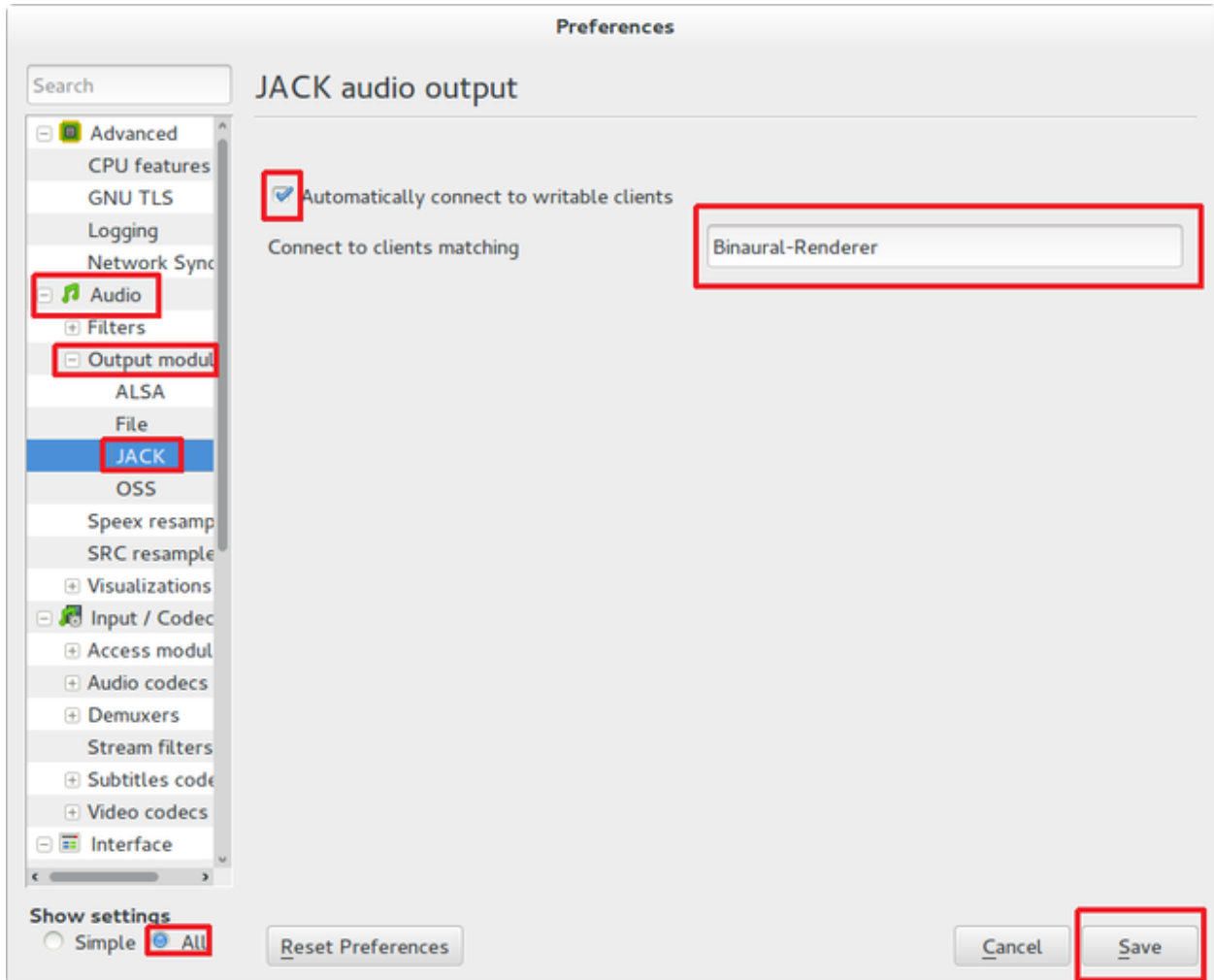
5. Start everything together using the command line:

```
qjackctl -s & vlc & ssr --gui /"path_of_your_scene_file(s)"/stereo.asd &
This will start jack, vlc and the ssr with the GUI and a provided stereo
scene (TODO: LINK) (stereo.asd)
```

6. Open an audio file in vlc and press play

### iTunes

TODO



## mPlayer

TODO

## Mac OS X

### Application Bundle

This assumes that you are using the precompiled SSR application bundle for Mac OS X. If you want to build the SSR yourself, have a look at *Building from Source*.

You can download the application bundle from <http://spatialaudio.net/ssr>. You will need JACK as prerequisite. Refer to *JACK on Mac OS X* for instructions how to obtain and install it.

The installation should be straightforward. Just double-click on the .dmg file and drag the SoundScapeRenderer-x.x.x folder to your Applications folder. Done. When double-clicking the SSR application bundle, it will indicate to you that you should download and install JACK (that's what this JACK.webloc thingy wants). If JACK is installed on your computer, then simply ignore this step and drag the SSR folder to your Applications folder.

The application bundle can be placed anywhere, but spaces in path names might cause trouble, so it's better to avoid them. Another thing is that Mac OS X sort of adds applications placed in the /Applications folder to the environment, so lets assume you put the SSR there (This also works for \$HOME/Applications).

### Building from Source

The following is an overview on how to set up the build environment for SSR on Mac OS X.

#### What to install first?

You can make your life much easier with a decent package manager, name Homebrew (<http://brew.sh/>) or MacPorts (<http://www.macports.org/>). Both greatly simplify the process of installing and managing dependencies.

#### Homebrew (recommended)

After installing homebrew, you can simply run the following line to update homebrew's internal repository, upgrade itself and install all necessary dependencies:

```
brew update && brew upgrade && brew install autoconf fftw libsndfile jack ecasound qt_
↳asio --c++11
```

To build the manual and documentation, you can also install help2man and doxygen:

```
brew install help2man doxygen
```

#### MacPorts

Tested with version 1.9.2

Download here: <http://www.macports.org/>

Then open a terminal and do an initial ports tree update

```
sudo port selfupdate
```

If that is not working it's most likely because the router won't let you use rsync. So we switch to http:

```
sudo nano /opt/local/etc/macports/sources.conf
```

Comment out the rsync entry

```
#rsync://rsync.macports.org/release/ports/ [default]
```

And add a line

```
http://www.macports.org/files/ports.tar.gz [default]
```

Now save the file and try the selfupdate again.

## MacPorts ports

These ports have to be installed (dependencies are installed automatically)

- gcc49 (or some other version of GCC, but at least gcc47)
- pkgconfig
- libsndfile
- libsamplerate
- fftw-3-single
- qt4-mac
- libxml2

If you want, you can also use clang instead of GCC to compile the SSR.

If you want to install the newest SSR development version directly from the Git repository, you'll need those as well:

- autoconf
- automake
- help2man

Ports are installed using

```
sudo port install <portname>
```

Because ports are compiled locally, it may take a long time to install all ports. Issuing one command to install all ports might be more convenient:

```
sudo sh -c "port install gcc49 && port install pkgconfig && port install libsndfile &&
↪ port install libsamplerate && port install fftw-3-single && port install qt4-mac &&
↪ port install libxml2"
```

Lastly, you need to install the asio library if you want to compile with the network interface. You can download it from: <http://think-async.com>

## JACK on Mac OS X

Tested with version 0.87 (64 bit) which includes:

- Jackdmp 1.9.6
- JackRouter 0.9.3
- JackPilot 1.7.0

Note that the site <http://www.jackosx.com/> is outdated. The latest version of JACK is available from <http://jackaudio.org/downloads/>.

Or, you can install JACK using *Homebrew*.

OS X El Capitan is a little trickier as there is no official JACK version for it. When scrolling down [this thread](#) you'll find a link to a beta version of JACK that works on El Capitan. Here's the direct link to this beta version of JACK: [https://dl.dropboxusercontent.com/u/28869550/JackOSX.0.92\\_b3.zip](https://dl.dropboxusercontent.com/u/28869550/JackOSX.0.92_b3.zip)

## Ecasound

Tested with version 2.7.2

If you don't want to get Ecasound from *Homebrew*, then download the source code from <http://www.eca.cx/ecasound/>. (If you choose to use Homebrew and you're experiencing problems, then you might want to take a look at *Ecasound cannot open a JACK port*).

It didn't work with 2.9.0 for us, older versions can be found there: <http://ecasound.seul.org/download/>.

In Terminal go into the unpacked ecasound folder and do:

```
./configure CPPFLAGS=-I/opt/local/include LIBS=-L/opt/local/lib
```

If JACK cannot be found, you can also try this:

```
./configure CPPFLAGS="-I/opt/local/include -I/usr/local/include" LIBS=-L/opt/local/lib
```

When the configure script is finished, check if libsndfile, libsamplerate and JACK are enabled. It should look something like that:

```
...
-----
Following features were selected:
...
Libsndfile:                yes
...
JACK support:              yes
Libsamplerate support     yes
...
-----
...
```

If not, check that JACK and all MacPort packages mentioned above are installed. If everything looks OK, continue with:

```
make
make install
```

For the last step you need write access to `/usr/local`. If it doesn't work, use this instead:

```
sudo make install
```

## Ecasound – git version

Note: if you successfully installed Ecasound 2.7.2, you *don't* need this!

If you want to use the newest Ecasound version from their git repository (<http://ecasound.seul.org/ecasound.git>) with OS X 10.9 (Mavericks), you can try this:

```
./configure CXX="clang++ -stdlib=libc++" CPPFLAGS=-D_DARWIN_C_SOURCE
```

Note, however, that you will have to use the same `-stdlib` options when configuring the SSR.

## Standard build

Get the SSR Source tarball and unpack it where ever you wish, then open a Terminal window and `cd` into the newly created folder.

First of all, you have to issue this command in order for `pkg-config` to find the installed JACK version:

```
export PKG_CONFIG_PATH=/usr/local/lib/pkgconfig
```

You now have two options:

1) If you want to build a clickable application bundle and wrap it into a `.dmg` image, you can build the SSR in *Mac OS X* style:

```
./configure --enable-app-bundle
make
make dmg
```

You can also name the `.dmg`:

```
./configure
--enable-app-bundle="MyVeryOwnSoundScapeRenderer.dmg"
make
make dmg
```

The resulting `.dmg` is output to the SSR source folder. The application bundle inside contains all non-system dynamic libraries needed by the SSR, except the JACK libs. So it should run on every Mac OS X 10.6.x with JACK installed.

2) If you want to build and install in *Linux* style:

```
./configure
make
make install
```

For the last step you need write access in `/usr/local`. If it doesn't work, use this instead:

```
sudo make install
```

The binaries do not get wrapped in an application bundle but they will be installed in `/usr/local/bin` and some files will be installed in `/usr/local/share/ssr`. If you want to remove all these file again, just do:

```
make uninstall
```

or (if you used `sudo` before):

```
sudo make uninstall
```

You can start the SSR and pass arguments to the SSR the same way you would do it on Linux.

### Build with InterSense tracker support

Tested with IntertiaCube3, software version 4.17

Get the SDK from <http://www.intersense.com/>. It should contain a dynamic library called `libisense.dylib` and two header files called `isense.h` and `types.h`, respectively.

Put `libisense.dylib` into `/usr/local/lib` and the header files into `/usr/local/include/intersense`.

Then build like described above, but add `CPPFLAGS=-I/usr/local/include/intersense` to the configure parameters:

```
export PKG_CONFIG_PATH=/usr/local/lib/pkgconfig
./configure CPPFLAGS=-I/usr/local/include/intersense
```

For some strange reason the full path of `libisense.dylib` is not written to the header of the binary. So if you configure with `--enable-app-bundle` and then do `make dmg` to build an application bundle, a tool called `dylibbundler` will ask you to enter its path (`/usr/local/lib`) several times.

### Using the GUI

1. Run the JackPilot and hit start.



2. **Double-click the SoundScape Renderer icon**, **select the renderer type**, and off you go!

When the SSR is started, a Terminal window displaying the standard SSR output is automatically opened. If the SSR closes with an error, it remains open for you to inspect the error messages.

The SSR and JACK must have the same sampling rate, otherwise error will be generated.

More options can be specified by using the config file. The details of using the config file is described on the Section *Configuration Files*.

### Running via the Command Line (Terminal)

The application bundle is more convenient, but brings limitations when using the command line. The included start script changes the working directory to the directory that includes the SSR application bundle, so only absolute paths should be used on the command line.

What can be done using configuration files, should be done using configuration files.

Both JACK and the SSR can be started on the command line by using:

```
open -a JackPilot
open -a SoundScapeRenderer
```



If the SSR is placed in `/Applications` (or `$HOME/Applications`) and was run several times already, OS X should find it. If not, you would have to specify the full path, i.e.

```
open /Applications/SoundScapeRenderer-0.4.0/SoundScapeRenderer.app
```

The SSR starts with a prompt for renderer selection, with the default renderer set on the Binaural renderer. As mentioned above in Section *Configuration Files*, more options at the start up can be specified using the configuration file.

To start the SSR using a different renderer module and skip the prompt for the selection do:

```
open -a SoundScapeRenderer --args --binaural
```

Of course, instead of `--binaural`, you can also use any of the other available renderers (e.g. `--wfs` for Wave Field Synthesis). To see all available command line arguments (specifying a renderer skips the prompt for renderer selection):

```
open -a SoundScapeRenderer --args --binaural "--help"
```

---

**Note:** The arguments other than the renderer selection have to be enclosed in quotation marks ("")!

---

To load a scene do:

```
open -a SoundScapeRenderer --args --binaural "/absolute/path/to/scene.asd"
```

---

**Note:** Paths have to be absolute!

---

To load a scene that has spaces in its path name do:

```
open -a SoundScapeRenderer --args --binaural "/path/to/file\ with\ spaces.asd"
```

---

**Note:** Spaces have to be escaped using backslashes!

---

In addition to the config files in standard locations mentioned above, you can also specify a config file on the command line:

```
open -a SoundScapeRenderer --args --binaural "--config=my_config_file.conf"
```

When you want to record the output of the SSR and play it back again, you should always provide the `.wav` extension of the file name, along with the absolute path of your desired destination location for the file. As an example, if you want to run the SSR and record the output to a file name `output.wav`, you would need the command given below:

```
open -a SoundScapeRenderer --args --binaural "--record=/tmp/output.wav"
```

But if you use a relative path instead of the absolute path, the file that contains the output of the SSR will be saved inside the directory of the SoundScape Renderer in the Applications folder.

If you want to start more than one instance of SSR, then add the `-n` option to the call of `open` for all instances other than the first one:

```
open -n -a SoundScapeRenderer --binaural
```

## Using a Head-Tracker

### Running with InterSense tracker support

Due to copyright reasons, the SSR does not come with a built-in InterSense tracker support. So first you have to build the SSR with *InterSense tracker support* yourself or ask someone to do it for you.

If you are using a USB-to-Serial interface with your tracker, you need to install drivers for that. This seems to work fine for the interface made by InterSense: <http://www.ftdichip.com/Drivers/VCP.htm>

To check if the system sees the tracker do:

```
ls -l /dev/tty.usb*
```

On the MacBooks tested, the serial ports were called `/dev/tty.usbserial- 00001004` or `/dev/tty.usbserial-00002006` depending on which USB port was used.

To make the SSR use the InterSense tracker with these ports, you have two options:

Using the command line (only one port can be specified):

```
open -a SoundScapeRenderer --args --binaural "--tracker=intersense  
--tracker-port=/dev/tty.usbserial-XXXXXXXX"
```

... or using config files:

Add these lines to a config file (multiple ports can be specified):

```
TRACKER = intersense  
TRACKER_PORTS = /dev/tty.usbserial-XXXXXXXX /dev/tty.usbserial-YYYYYYYY
```

It's recommended to use the config file approach - best use a global `:ref:' config file<ssr_configuration_file>'`.

### Running with Razor AHRS tracker support

If you happen not to own a Polhemus or InterSense tracker to do your head-tracking, an alternative would be to use our DIY low-cost [Razor AHRS tracker](#).

If you have Arduino installed on you machine, FTDI drivers will be there too. Otherwise get the driver from <http://www.ftdichip.com/Drivers/VCP.htm>.

To check if the system sees the tracker do:

```
ls -l /dev/tty.usb*
```

This should give you something like `/dev/tty.usbserial-A700eEhN`.

To make the SSR use this Razor AHRS tracker, you have two options:

Using the command line:

```
open -a SoundScapeRenderer --args --binaural "--tracker=razor  
--tracker-port=/dev/tty.usbserial-XXXXXXXX"
```

... or using config files:

Add these lines to a config file:

```
TRACKER = intersense  
TRACKER_PORTS = /dev/tty.usbserial-XXXXXXXX
```

It's recommended to use the config file approach - best use a global *config file*.

## Using the SSR with DAWs

As stated before, the SSR is currently not able to dynamically replay audio files (refer to Section *ASDF*). If your audio scenes are complex, you might want to consider using the SSR together with a digital audio work station (DAW). To do so, you simply have to create as many sources in the SSR as you have audio tracks in your respective DAW project and assign live inputs to the sources. Amongst the ASDF examples we provide on SSR website <http://spatialaudio.net/ssr/> you'll find an scene description that does exactly this.

DAWs like Ardour (<http://www.ardour.org>) support JACK and their use is therefore straightforward. DAWs which do not run on Linux or do not support JACK can be connected via the input of the sound card.

In the future we will provide a VST plug-in which will allow you to dynamically operate all virtual source's properties (like e.g. a source's position or level etc.). You will then be able to have the full SSR functionality controlled from your DAW.

## Known Issues (Linux & Mac OS X)

Also visit <https://github.com/SoundScapeRenderer/ssr/wiki/Known-Issues> for updated known issues.

### make dmg on Mac OS X chokes on symbolic links

On some file system (e.g. network shares with ACLs) you might get an error like this:

```
... copy helper ... copy error (canceling): /Volumes/SoundScape Renderer ...
... Operation not supported ... could not access /Volumes/...
hdiutil: create failed - Operation not supported
make[1]: *** [dmg] Error 1
make: *** [dmg] Error 2
```

This has something to do with symbolic links and the way how `hdiutil` handles them. If you get this error, just try to compile the SSR from a different location. You can do this by either moving all the source files somewhere else, or by doing something like this:

```
cd /tmp
mkdir ssr-bundle
cd ssr-bundle
export PKG_CONFIG_PATH=/usr/local/lib/pkgconfig
$PATH_TO_SSR_SOURCE/configure --enable-app-bundle
make
make dmg
```

In this example, `$PATH_TO_SSR_SOURCE` is the directory where you put the SSR source files. Instead of `/tmp` you can of course use something else, but with `/tmp` it should work on most systems out there.

If you don't like this work-around, you may also play around with `fsaclctl`. Only `WAVE_FORMAT_PCM` and `WAVE_FORMAT_IEEE_FLOAT` are supported.

Multi-channel WAV files would normally use the format `WAVE_FORMAT_EXTENSIBLE`, see <http://www-mm.sp.ece.mcgill.ca/documents/audioformats/wave/wave.html>.

However, Ecasound doesn't know this format, that's why we have to use one of the above mentioned formats, although for files with more than 2 channels this is not compliant to the WAV standard.

To check the exact format of your WAV files, you can use `sndfile-info` (Debian package `sndfile-programs`), and to convert your files, you can use, for example, `sox` (Debian package `sox`) with the `wavpcm` option:

```
sox old.wav new.wavpcm
mv new.wavpcm new.wav
```

## SSR crashes with a segmentation fault on Max OS X

If this happens whenever you are opening an audio file or loading a scene that involves opening an audio file, then this might be due to Ecasound. We've seen this with the app bundle. Try the following:

Download the Ecasound source code from <http://nosignal.fi/ecasound/download.php>. Cd into the folder and compile Ecasound with:

```
./configure
make
```

Refer also to *Ecasound* for instructions on how to compile Ecasound. The executable `ecasound/ecasound` will be created.

Finally, replace the Ecasound executable in the SSR bundle with something like this:

```
sudo cp ecasound/ecasound /Applications/SoundScapeRenderer-0.4.2-74-gb99f8b2/
↪SoundScapeRenderer.app/Contents/MacOS/
```

You might have to modify the name of the SSR folder in the above command as you're likely to use a different version.

## A file that can't be loaded results in a connection to a live input

If there is an error in loading a specified audio file, the corresponding source is still created and (unexpectedly) connected to the first soundcard input channel.

We believe this is a bug in the JACK system related to the API function `jack_connect()`: If the `destination_port` argument is an empty string, the function returns (correctly) with an error. However, if the `source_port` argument is an empty string, the port is connected to the first "system" port ( or the first port at all, or who knows ...). And this is the case if the SSR cannot open a specified audio file.

If you also think that's a bug, feel free to report it to the JACK developers.

## Conflicting JACK and Ecasound versions

There is a problem due to a special combination of `ecasound` and JACK versions on 64 bit systems leading to an error (terminating the SSR) similar to this:

```
(ecasoundc_sa) Error='read() error', cmd='cs-connect' last_error='' cmd_cnt=6
last_cnt=5.
```

We experienced this error on 64 bit systems with `ecasound` version 2.4.6.1 and 2.7.0 in combination with JACK version 0.118.0. A similar error occurred on Mac OS X with `ecasound` version 2.7.2 and `JackOSX` version 0.89 (with `Jackdmp` 1.9.8).

Please try to update to the newest `ecasound` and JACK versions.

## Annoying Ecasound message

You may have seen this message:

```
*****
* Message from libecasoundc:
*
* 'ECASOUND' environment variable not set. Using the default value
* value 'ECASOUND=ecasound'.
*****
```

You can totally ignore this, but if it bothers you, you can disable it easily by specifying the following line in `/etc/bash.bashrc` (system-wide setting) or, if you prefer, you can put it into your `$HOME/.bashrc` (just for your user account):

```
export ECASOUND=ecasound
```

## Ecasound cannot open a JACK port

Sometimes, when Ecasound is installed via *Homebrew*, it can have trouble finding JACK. As a result SSR displays the sound source symbols in the GUI, but they don't play audio, and an according error message is posted in the SSR terminal.

Type `ecasound -c` in a terminal to start Ecasound in interactive mode. Then type `aio-register` to list all available outputs that Ecasound has recognized. If JACK is not listed, then download the Ecasound source code from <http://nosignal.fi/ecasound/download.php>, and

```
./configure --enable-jack
make
make install
```

The last line might have to be

```
sudo make install
```

Refer also to *Ecasound* for instructions on how to compile Ecasound.

## Using SSR on Mac OS X El Capitan

SSR works well on El Capitan. JACK is what can cause headache. See *JACK on Mac OS X*.

## Long paths to audio files on Mac OS X

It can happen that SSR displays this error message when loading audio files directly:

```
Error: AudioPlayer::Soundfile: ERROR: Connecting chainsetup failed: "Enabling_
↳chainsetup: AUDIOIO-JACK: Unable to open JACK-client" (audioplayer.cpp:310)
Warning: AudioPlayer: Initialization of soundfile '/Users/YOUR_USERNAME/Documents/
↳audio/YOUR_AUDIO_FILE.wav' failed! (audioplayer.cpp:87)
```

Opening such a file would result in a JACK port name that is too long. You can resolve this limitation by moving the audio file to a location that produces a shorter (full) path name or by wrapping the audio file in an asd-file.

## Segmentation Fault when Opening a Scene

This problem occurred on some old SuSE systems.

When you start the SSR with GUI, everything is alright at first. As soon as you open a scene, a segmentation fault arises. This is a problem in the interaction between Qt and OpenGL. As a workaround, comment the line

```
renderText(0.18f * scale, 0.13f * scale, 0.0f, source->name.c_str(), f);
```

in the file `src/gui/qopenglrenderer.cpp` and recompile the code. The consequence is that the names of the sound sources will not be displayed anymore.

## Choppy Sound on Cheap (On-Board) Sound Cards

Sometimes JACK doesn't play well with those on-board sound cards. One possibility to improve this, is to increase the frames/period setting from the default 2 to a more generous 3. This can be done in the Settings dialog of `qjackctl` or with the command line option `-n`:

```
jackd -n 3
```

## dylibbundler doesn't grok Qt Frameworks

If `make dmg` doesn't copy the Qt `.dylib` files into the application bundle (to `Contents/Libraries`), you might try the following commands (or similar, depending on the exact Qt installation). Go to the [online manual](#) to copy and paste them.

```
install_name_tool -id /opt/local/lib/libQtCore.dylib /opt/local/Library/Frameworks/  
↳QtCore.framework/QtCore  
install_name_tool -id /opt/local/lib/libQtGui.dylib /opt/local/Library/Frameworks/  
↳QtGui.framework/QtGui  
install_name_tool -change /opt/local/Library/Frameworks/QtCore.framework/Versions/4/  
↳QtCore /opt/local/lib/libQtCore.dylib /opt/local/Library/Frameworks/QtGui.framework/  
↳QtGui  
install_name_tool -id /opt/local/lib/libQtOpenGL.dylib /opt/local/Library/Frameworks/  
↳QtOpenGL.framework/QtOpenGL  
install_name_tool -change /opt/local/Library/Frameworks/QtCore.framework/Versions/4/  
↳QtCore /opt/local/lib/libQtCore.dylib /opt/local/Library/Frameworks/QtOpenGL.  
↳framework/QtOpenGL  
install_name_tool -change /opt/local/Library/Frameworks/QtGui.framework/Versions/4/  
↳QtGui /opt/local/lib/libQtGui.dylib /opt/local/Library/Frameworks/QtOpenGL.  
↳framework/QtOpenGL
```

You need the appropriate rights to change the library files, so you probably need to use `sudo` before the commands.

**WARNING:** You can totally ruin your Qt installation with this stuff!

To get some information about a library, you can try something like those:

```
otool -L /opt/local/Library/Frameworks/QtOpenGL.framework/QtOpenGL  
otool -l /opt/local/Library/Frameworks/QtOpenGL.framework/QtOpenGL  
otool -D /opt/local/Library/Frameworks/QtOpenGL.framework/QtOpenGL
```

## SSR for Mac OS X: qt\_menu.nib not found

This was fixed in MacPorts, see <https://trac.macports.org/ticket/37662>. Thanks to Chris Pike!

## Compilation Error on Ubuntu and Archlinux

This issue was resolved in version 0.3.4. Some newer distributions got more picky about the necessary `#include` commands. If the SSR refuses to compile, add this to the file `src/gui/qopenglplotter.h` (somewhere at the beginning):

```
#include <GL/glu.h>
```

On Mac OS X you'll need this instead:

```
#include <OpenGL/glu.h>
```

## Polhemus tracker does not work with SSR

This issue was resolved in version 0.3.3, where we changed the tracker selection. Use `--tracker=fastrak`, `--tracker=patriot` and `--tracker=intersense`, respectively. The serial port can be specified with `--tracker-port=/dev/ttyUSB0` (or similar).

This can happen when both the Intersense tracker as well as the Polhemus tracker are compiled and the file `isports.ini` is present. The latter tells the Intersense tracker which port to use instead of the standard serial port `/dev/ttyS0`. If the `isports.ini` file lists the port to which the Polhemus tracker is connected, it can happen that something that we have not fully understood goes wrong and the Polhemus data can not be read. In this case you can either rename the file `isports.ini` or change its content.

It might be necessary to execute `echo C > /dev/ttyS0` several times in order to make the Polhemus tracker operational again. You can check with `cat /dev/ttyS0` if it delivers data.

## Missing GUI Buttons and Timeline

This issue was resolved in version 0.3.2, the default setting for `--enable-floating-control-panel` is chosen depending on the installed Qt version.

Different versions of Qt show different behaviour regarding OpenGL Overlays and as a result, the GUI buttons are not shown in newer Qt versions.

To overcome this limitation, we provide two GUI variants:

- Traditional GUI, can be used up to Qt 4.6.x
- Floating control panel, should be used with Qt 4.7 and above

The floating control panel is the default setting on Mac OS X, for Linux it can be activated with:

```
./configure --enable-floating-control-panel
```

## OpenGL Linker Error

This issue was resolved in version 0.3.2.

On some systems, after running `make`, you'll get an error mentioning "glSelectBuffer".

For now, this is the solution (see also the issue below):

```
./configure LIBS=-lGL
```

## IP interface isn't selected although boost libraries are installed

This issue was resolved with dropping boost::asio for asio in version 0.5.0.

For older builds, you might need to add the `-lpthread` flag:

```
./configure LIBS=-lpthread
```

## Second instance of SSR crashes

This happens when two or more instances of the SSR are started with the IP server enabled. Start all (or at least all instances higher than 1) with the `-I` flag to disable the IP interface.

## Audio files with spaces

This issue was resolved in version 0.3.2.

Please do not use audio files with spaces for scenes. Neither the filename nor the directory referenced in the scene (asd-file) should contain spaces.

## Error ValueError: unknown locale: UTF-8 when building the manual

This can happen on non-US Macs. Go to your home folder `/Users/YOUR_USER_NAME`, open (or create) the file `.bash_profile` and add the following to this file:

```
export LC_ALL=en_US.UTF-8
export LANG=en_US.UTF-8
export LANGUAGE=en_US.UTF-8
export LC_CTYPE=en_US.UTF-8
```

You might have to re-open the terminal or log out and in again to see the effect.



## General

### Reproduction Setups

The geometry of the actual reproduction setup is specified in `.asd` files, just like sound scenes. By default, it is loaded from the file `/usr/local/share/ssr/default_setup.asd`. Use the `--setup` command line option to load another reproduction setup file. Note that the loudspeaker setups have to be convex. This is not checked by the SSR. The loudspeakers appear at the outputs of your sound card in the same order as they are specified in the `.asd` file, starting with channel 1.

A sample reproduction setup description:

```
<?xml version="1.0"?>
<asdf version="0.1">
  <header>
    <name>Circular Loudspeaker Array</name>
  </header>
  <reproduction_setup>
    <circular_array number="56">
      <first>
        <position x="1.5" y="0"/>
        <orientation azimuth="-180"/>
      </first>
    </circular_array>
  </reproduction_setup>
</asdf>
```

We provide the following setups in the directory `data/reproduction_setups/`:

- `2.0.asd`: standard stereo setup at 1.5 mtrs distance
- `2.1.asd`: standard stereo setup at 1.5 mtrs distance plus subwoofer
- `5.1.asd`: standard 5.1 setup on circle with a diameter of 3 mtrs

- `rounded_rectangle.asd`: Demonstrates how to combine circular arcs and linear array segments.
- `circle.asd`: This is a circular array of 3 mtrs diameter composed of 56 loudspeakers.
- `loudspeaker_setup_with_nearly_all_features.asd`: This setup describes all supported options, open it with your favorite text editor and have a look inside.

There is some limited freedom in assigning channels to loudspeakers: If you insert the element `<skip number="5"/>`, the specified number of output channels are skipped and the following loudspeakers get higher channel numbers accordingly.

Of course, the binaural and BRS renderers do not load a loudspeaker setup. By default, they assume the listener to reside in the coordinate origin looking straight forward.

## A Note on the Timing of the Audio Signals

The WFS renderer is the only renderer in which the timing of the audio signals is somewhat peculiar. None of the other renderers imposes any algorithmic delay on individual source signals. Of course, if you use a renderer that is convolution based such as the BRS renderer, the employed HRIRs do alter the timing of the signals due to their inherent properties.

This is different with the WFS renderer. Here, also the propagation duration of sound from the position of the virtual source to the loudspeaker array is taken into account. This means that the farther a virtual source is located, the longer is the delay imposed on its input signal. This also holds true for plane waves: Theoretically, plane waves do originate from infinity. Though, the SSR does consider the origin point of the plane wave that is specified in ASDF. This origin point also specifies the location of the symbol that represents the respective plane wave in the GUI.

We are aware that this procedure can cause confusion and reduces the ability of a given scene of translating well between different types of renderers. In the upcoming version 0.4 of the SSR we will implement an option that will allow you specifying for each individual source whether the propagation duration of sound shall be considered by a renderer or not.

## Subwoofers

All loudspeaker-based renderers support the use of subwoofers. Outputs of the SSR that are assigned to subwoofers receive a signal having full bandwidth. So, you will have to make sure yourself that your system lowpasses these signals appropriately before they are emitted by the subwoofers.

You might need to adjust the level of your subwoofer(s) depending on the renderers that you are using as the overall radiated power of the normal speakers cannot be predicted easily so that we cannot adjust for it automatically. For example, no matter of how many loudspeakers your setup is composed of the VBAP renderer will only use two loudspeakers at a time to present a given virtual sound source. The WFS renderer on the other hand might use 10 or 20 loudspeakers, which can clearly lead to a different sound pressure level at a given receiver location.

For convenience, ASDF allows for specifying permanent weight for loudspeakers and subwoofers using the `weight` attribute:

```
<loudspeaker model="subwoofer" weight="0.5">
  <position x="0" y="0"/>
  <orientation azimuth="0"/>
</loudspeaker>
```

`weight` is a linear factor that is always applied to the signal of this speaker. Above example will obviously attenuate the signal by approx. 6 dB. You can use two ASDF description for the same reproduction setup that differ only with respect to the subwoofer weights if you're using different renderers on the same loudspeaker system.

## Distance Attenuation

Note that in all renderers – except for the BRS and generic renderers –, the distance attenuation in the virtual space is  $\frac{1}{r}$  with respect to the distance  $r$  of the respective virtual point source to the reference position. Point sources closer than 0.5 m to the reference position do not experience any increase of amplitude. Virtual plane waves do not experience any algorithmic distance attenuation in any renderer.

You can specify your own preferred distance attenuation exponent  $exp$  (in  $\frac{1}{r^{exp}}$ ) either via the command line argument `--decay-exponent=VALUE` or the configuration option `DECAY_EXPONENT` (see the file `data/ssr.conf.example`). The higher the exponent, the faster is the amplitude decay over distance. The default exponent is  $exp = 1$ <sup>1</sup>. Fig. 3.1 illustrates the effect of different choices of the exponent. In simple words, the smaller the exponent the slower is the amplitude decay over distance. Note that the default decay of  $\frac{1}{r}$  is theoretically correct only for infinitesimally small sound sources. Spatially extended sources, like most real world sources, exhibit a slower decay. So you might want to choose the exponent to be somewhere between 0.5 and 1. You can completely suppress any sort of distance attenuation by setting the decay exponent to 0.

The amplitude reference distance, i.e. the distance from the reference at which plane waves are as loud as the other source types (like point sources), can be set in the SSR configuration file (Section *Configuration File*). The desired amplitude reference distance for a given sound scene can be specified in the scene description (Section *ASDF*). The default value is 3 m.

The overall amplitude normalization is such that plane waves always exhibit the same amplitude independent of what amplitude reference distance and what decay exponent have been chosen. Consequently, also virtual point source always exhibit the same amplitude at amplitude reference distance, whatever it has been set to.

## Doppler Effect

In the current version of the SSR the Doppler Effect in moving sources is not supported by any of the renderers.

## Signal Processing

All rendering algorithms are implemented on a frame-wise basis with an internal precision of 32 bit floating point. The signal processing is illustrated in Fig. 3.2.

The input signal is divided into individual frames of size  $nframes$ , whereby  $nframes$  is the frame size with which JACK is running. Then e.g. frame number  $n + 1$  is processed both with previous rendering parameters  $n$  as well as with current parameters  $n + 1$ . It is then crossfaded between both processed frames with cosine-shaped slopes. In other words the effective frame size of the signal processing is  $2 \cdot nframes$  with 50% overlap. Due to the fade-in of the frame processed with the current parameters  $n + 1$ , the algorithmic latency is slightly higher than for processing done with frames purely of size  $nframes$  and no crossfade.

The implementation approach described above is one version of the standard way of implementing time-varying audio processing. Note however that this means that with *all* renderers, moving sources are not physically correctly reproduced. The physically correct reproduction of moving virtual sources as in [Ahrens2008a] and [Ahrens2008b] requires a different implementation approach which is computationally significantly more costly.

<sup>1</sup> A note regarding previous versions of the WFS renderer: In the present SSR version, the amplitude decay is handled centrally and equally for all renderers that take distance attenuation into account (see Table 2). Previously, the WFS renderer relied on the distance attenuation that was inherent to the WFS driving function. This amplitude decay is very similar to an exponent of 0.5 (instead of the current default exponent of 1.0). So you might want to set the decay exponent to 0.5 in WFS to make your scenes sound like they used to do previously.

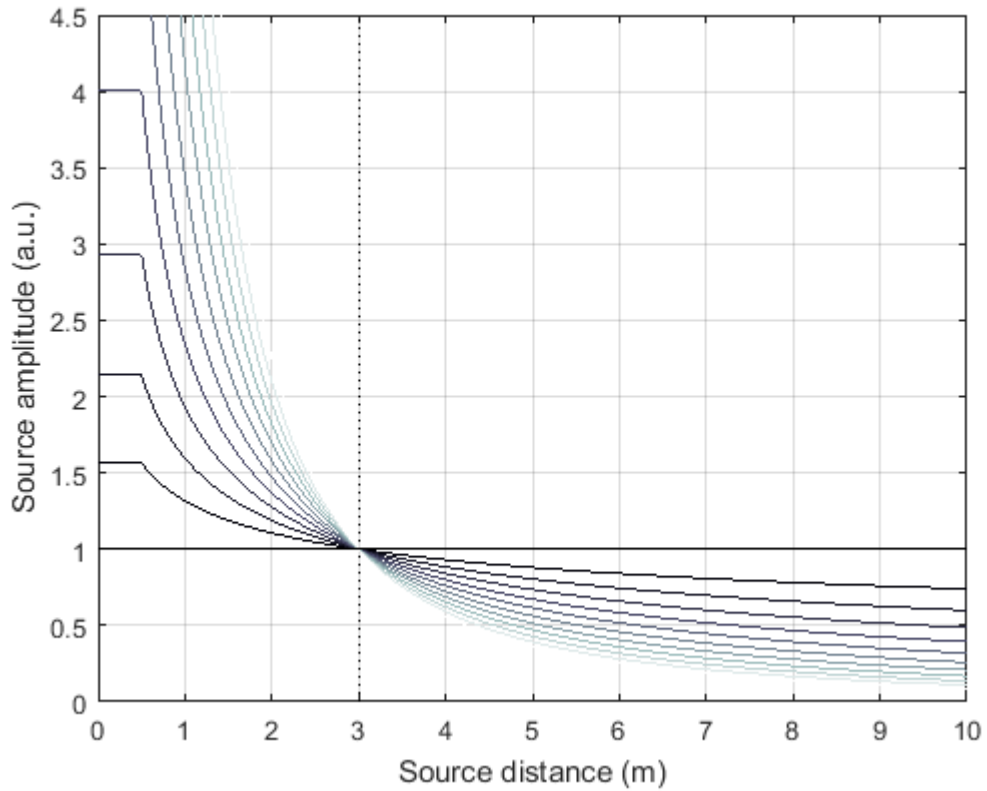


Fig. 3.1: Illustration of the amplitude of virtual point sources as a function of source distance from the reference point for different exponents  $exp$ . The exponents range from 0 to 2 (black color to gray color). The amplitude reference distance is set to 3 m. Recall that sources closer than 0.5 m to the reference position do not experience any further increase of amplitude.

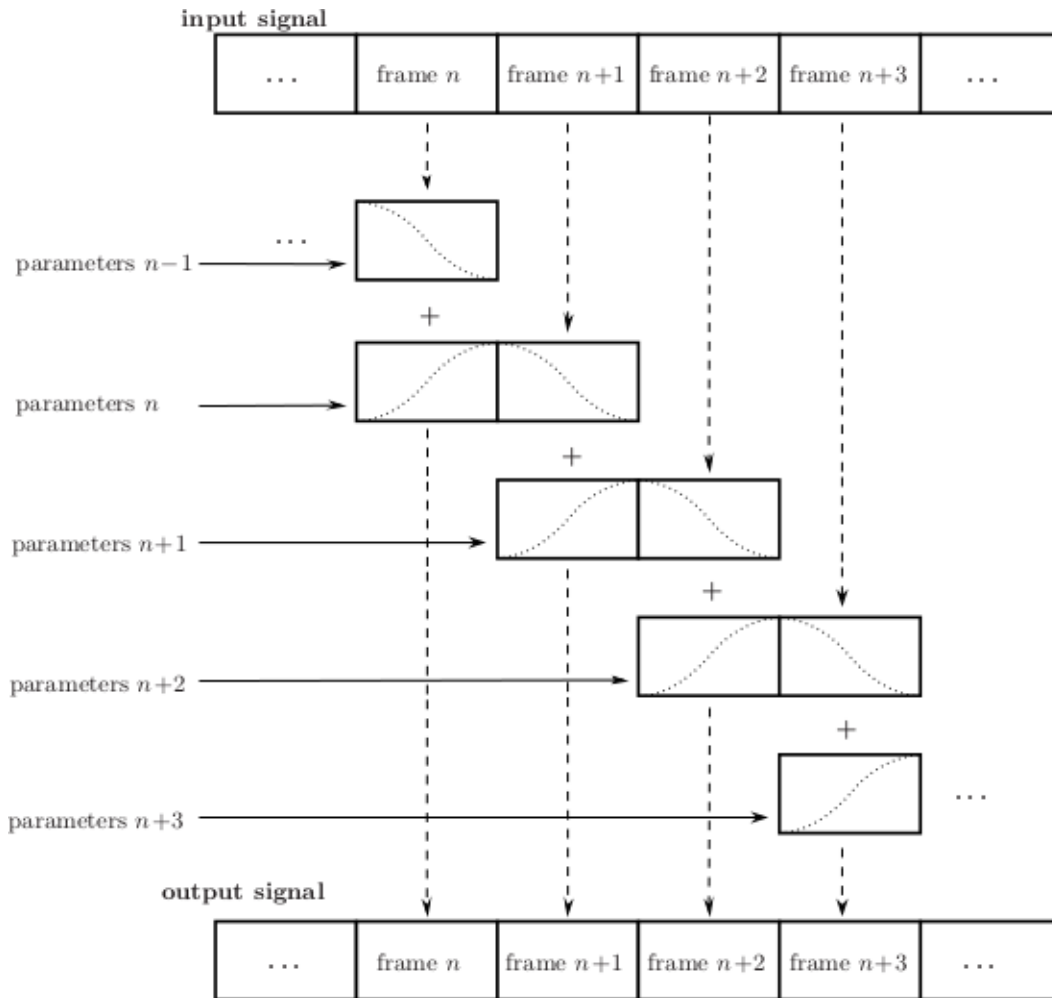


Fig. 3.2: Illustration of the frame-wise signal processing as implemented in the SSR renderers (see text)

## Binaural Renderer

Binaural rendering is an approach where the acoustical influence of the human head is electronically simulated to position virtual sound sources in space. **Be sure that you are using headphones to listen.**

The acoustical influence of the human head is coded in so-called head-related impulse responses (HRIRs) or equivalently by head-related transfer functions. The HRIRs are loaded from the file `/usr/local/share/ssr/default_hrirs.wav`. If you want to use different HRIRs then use the `--hrirs=FILE` command line option or the SSR configuration file (Section *Configuration File*) to specify your custom location. The SSR connects its outputs automatically to outputs 1 and 2 of your sound card.

For virtual sound sources that are closer to the reference position (= the listener position) than 0.5 m, the HRTFs are interpolated with a Dirac impulse. This ensures a smooth transition of virtual sources from the outside of the listener's head to the inside.

SSR uses HRIRs with an angular resolution of  $1^\circ$ . Thus, the HRIR file contains 720 impulse responses (360 for each ear) stored as a 720-channel .wav-file. The HRIRs all have to be of equal length and have to be arranged in the following order:

- 1st channel: left ear, virtual source position  $0^\circ$
- 2nd channel: right ear, virtual source position  $0^\circ$
- 3rd channel: left ear, virtual source position  $1^\circ$
- 4th channel: right ear, virtual source position  $1^\circ$
- ...
- 720th channel: right ear, virtual source position  $359^\circ$

If your HRIRs have lower angular resolution you have to interpolate them to the target resolution or use the same HRIR for several adjacent directions in order to fulfill the format requirements. Higher resolution is not supported. Make sure that the sampling rate of the HRIRs matches that of JACK. So far, we know that both 16bit and 24bit word lengths work.

The SSR automatically loads and uses all HRIR coefficients it finds in the specified file. You can use the `--hrir-size=VALUE` command line option in order to limit the number of HRIR coefficients read and used to VALUE. You don't need to worry if your specified HRIR length VALUE exceeds the one stored in the file. You will receive a warning telling you what the score is. The SSR will render the audio in any case.

The actual size of the HRIRs is not restricted (apart from processing power). The SSR cuts them into partitions of size equal to the JACK frame buffer size and zero-pads the last partition if necessary.

Note that there's some potential to optimize the performance of the SSR by adjusting the JACK frame size and accordingly the number of partitions when a specific number of HRIR taps are desired. The least computational load arises when the audio frames have the same size like the HRIRs. By choosing shorter frames and thus using partitioned convolution the system latency is reduced but computational load is increased.

## The HRIR sets shipped with SSR

SSR comes with two different HRIR sets: FABIAN and KEMAR (QU). They differ with respect to the manikin that was used in the measurement (FABIAN vs. KEMAR). The reference for the FABIAN measurement is [*Lindau2007*], and the reference for the KEMAR (QU) is [*Wierstorf2011*]. The low-frequency extension from [*SpatialAudio*] has been applied to the KEMAR (QU) HRTFs.

You will find all sets in the folder `data/impulse_responses/hrirs/`. The suffix `_eq` in the file name indicates the equalized data. The unequalized data is of course also there. See the file `data/impulse_responses/hrirs/hrirs_fabian_documentation.pdf` for a few more details on the FABIAN measurement.

Starting with SSR release 0.5.0, the default HRIR set that is loaded is headphone compensated, i.e., we equalized the HRIRs a bit in order to compensate for the alterations that a typical pair of headphones would apply to the ear signals. Note that by design, headphones do not have a flat transfer function. However, when performing binaural rendering, we need the headphones to be transparent. Our equalization may not be perfect for all headphones or earbuds as these can exhibit very different properties between different models.

We chose a frequency sampling-based minimum-phase filter design. The transfer functions and impulse responses of the two compensation filters are depicted in Fig. 3.3. The impulse responses themselves can be found in the same folder like the HRIRs (see above). The length is 513 taps so that the unequalized HRIRs are 512 taps long, the equalized ones are 1024 taps long.

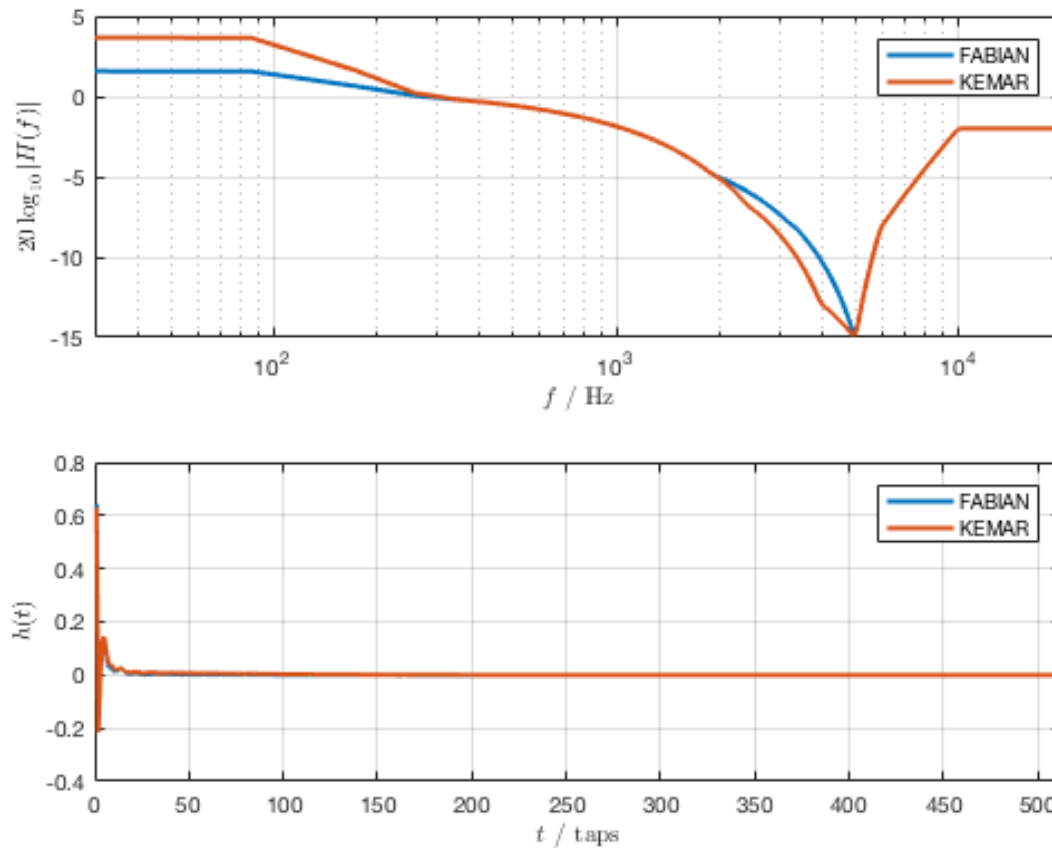


Fig. 3.3: Magnitude transfer functions and impulse responses of the headphone compensation / equalization filters

Recall that there are several ways of defining which HRIR set is loaded, for example the `HRIR_FILE_NAME` in the *SSR configuration files* property, or the command line option `--hrirs=FILE`.

## Preparing HRIR sets

You can easily prepare your own HRIR sets for use with the SSR by adopting the MATLAB script `data/matlab_scripts/prepare_hrirs_cipic.m` to your needs. This script converts the HRIRs of the KEMAR manikin included in the CIPIC database [AlgaziCIPIC] to the format that the SSR expects. See the script for further information and how to obtain the raw HRIRs. Note that the KEMAR (CIPIC) HRIRs are not identical to the KEMAR (QU) ones.

## Binaural Room Synthesis Renderer

The Binaural Room Synthesis (BRS) renderer is a binaural renderer (refer to Section *Binaural Renderer*) which uses one dedicated HRIR set of each individual sound source. The motivation is to have more realistic reproduction than in simple binaural rendering. In this context HRIRs are typically referred to as binaural room impulse responses (BRIRs).

Note that the BRS renderer does not consider any specification of a virtual source's position. The positions of the virtual sources (including their distance) are exclusively coded in the BRIRs. Consequently, the BRS renderer does not apply any distance attenuation. It only applies the respective source's gain and the master volume. No interpolation with a Dirac as in the binaural renderer is performed for very close virtual sources. The only quantity which is explicitly considered is the orientation of the receiver, i.e. the reference. Therefore, specification of meaningful source and receiver positions is only necessary when a correct graphical illustration is desired.

The BRIRs are stored in the a format similar to the one for the HRIRs for the binaural renderer (refer to Section *Binaural Renderer*). However, there is a fundamental difference: In order to be consequent, the different channels do not hold the data for different positions of the virtual sound source but they hold the information for different head orientations. Explicitely,

- 1st channel: left ear, head orientation 0°
- 2nd channel: right ear, head orientation 0°
- 3rd channel: left ear, head orientation 1°
- 4th channel: right ear, head orientation 1°
- ...
- 720th channel: right ear, head orientation 359°

In order to assign a set of BRIRs to a given sound source an appropriate scene description in `.asd`-format has to be prepared (refer also to Section *Audio Scenes*). As shown in `brs_example.asd` (from the example scenes), a virtual source has the optional property `properties_file` which holds the location of the file containing the desired BRIR set. The location to be specified is relative to the folder of the scene file. Note that – as described above – specification of the virtual source's position does not affect the audio processing. If you do not specify a BRIR set for each virtual source, then the renderer will complain and refuse processing the respective source.

We have measured the BRIRs of the FABIAN manikin in one of our mid-size meeting rooms called Sputnik with 8 different source positions. Due to the file size, we have not included them in the release. You can obtain the data from [\[BRIRs\]](#).

## Vector Base Amplitude Panning Renderer

The Vector Base Amplitude Panning (VBAP) renderer uses the algorithm described in [\[Pulkki1997\]](#). It tries to find a loudspeaker pair between which the phantom source is located (in VBAP you speak of a phantom source rather than a virtual one). If it does find a loudspeaker pair whose angle is smaller than 180° then it calculates the weights  $g_l$  and  $g_r$  for the left and right loudspeaker as

$$g_{l,r} = \frac{\cos \phi \sin \phi_0 \pm \sin \phi \cos \phi_0}{2 \cos \phi_0 \sin \phi_0}.$$

$\phi_0$  is half the angle between the two loudspeakers with respect to the listening position,  $\phi$  is the angle between the position of the phantom source and the direction “between the loudspeakers”.

If the VBAP renderer can not find a loudspeaker pair whose angle is smaller than 180° then it uses the closest loudspeaker provided that the latter is situated within 30°. If not, then it does not render the source. If you are in verbosity level 2 (i.e. start the SSR with the `-vv` option) you'll see a notification about what's happening.



Note that all virtual source types (i.e. point and plane sources) are rendered as phantom sources.

Contrary to WFS, non-uniform distributions of loudspeakers are ok here. Ideally, the loudspeakers should be placed on a circle around the reference position. You can optionally specify a delay for each loudspeakers in order to compensate some amount of misplacement. In the ASDF (refer to Section *ASDF*), each loudspeaker has the optional attribute `delay` which determines the delay in seconds to be applied to the respective loudspeaker. Note that the specified delay will be rounded to an integer factor of the temporal sampling period. With 44.1 kHz sampling frequency this corresponds to an accuracy of 22.676  $\mu$ s, respectively an accuracy of 7.78 mm in terms of loudspeaker placement. Additionally, you can specify a weight for each loudspeaker in order to compensate for irregular setups. In the ASDF format (refer to Section *ASDF*), each loudspeaker has the optional attribute `weight` which determines the linear (!) weight to be applied to the respective loudspeaker. An example would be

```
<loudspeaker delay="0.005" weight="1.1">
  <position x="1.0" y="-2.0"/>
  <orientation azimuth="-30"/>
</loudspeaker>
```

Delay defaults to 0 if not specified, weight defaults to 1.

Although principally suitable, we do not recommend to use our amplitude panning algorithm for dedicated 5.1 (or comparable) mixdowns. Our VBAP renderer only uses adjacent loudspeaker pairs for panning which does not exploit all potentials of such a loudspeaker setup. For the mentioned formats specialized panning processes have been developed also employing non-adjacent loudspeaker pairs if desired.

The VBAP renderer is rather meant to be used with non-standardized setups.

## Wave Field Synthesis Renderer

The Wave Field Synthesis (WFS) renderer is the only renderer so far that discriminates between virtual point sources and plane waves. It implements the simple (far-field) driving function given in [Spors2008]. Note that we have only implemented a temporary solution to reduce artifacts when virtual sound sources are moved. This topic is subject to ongoing research. We will work on that in the future. In the SSR configuration file (Section *Configuration File*) you can specify an overall predelay (this is necessary to render focused sources) and the overall length of the involved delay lines. Both values are given in samples.

### Prefiltering

As you might know, WFS requires a spectral correction additionally to the delay and weighting of the input signal. Since this spectral correction is equal for all loudspeakers, it needs to be performed only once on the input. We are working on an automatic generation of the required filter. Until then, we load the impulse response of the desired filter from a .wav-file which is specified via the `--prefilter=FILE` command line option (see Section *Running SSR*) or in the SSR configuration file (Section *Configuration File*). Make sure that the specified audio file contains only one channel. Files with a differing number of channels will not be loaded. Of course, the sampling rate of the file also has to match that of the JACK server.

Note that the filter will be zero-padded to the next highest power of 2. If the resulting filter is then shorter than the current JACK frame size, each incoming audio frame will be divided into subframes for prefiltering. That means, if you load a filter of 100 taps and JACK frame size is 1024, the filter will be padded to 128 taps and prefiltering will be done in 8 cycles. This is done in order to save processing power since typical prefilters are much shorter than typical JACK frame sizes. Zero-padding the prefilter to the JACK frame size usually produces large overhead. If the prefilter is longer than the JACK frame buffer size, the filter will be divided into partitions whose length is equal to the JACK frame buffer size.

If you do not specify a filter, then no prefiltering is performed. This results in a boost of bass frequencies in the reproduced sound field.

In order to assist you in the design of an appropriate prefilter, we have included the MATLAB script `data/matlab_scripts/make_wfs_prefilter.m` which does the job. In the very top of the file, you can specify the sampling frequency, the desired length of the filter as well as the lower and upper frequency limits of the spectral correction. The lower limit should be chosen such that the subwoofer of your system receives a signal which is not spectrally altered. This is due to the fact that only loudspeakers which are part of an array of loudspeakers need to be corrected. The lower limit is typically around 100 Hz. The upper limit is given by the spatial aliasing frequency. The spatial aliasing is dependent on the mutual distance of the loudspeakers, the distance of the considered listening position to the loudspeakers, and the array geometry. See [Spors2006] \_ for detailed information on how to determine the spatial aliasing frequency of a given loudspeaker setup. The spatial aliasing frequency is typically between 1000 Hz and 2000 Hz. For a theoretical treatment of WFS in general and also the prefiltering, see [Spors2008].

The script `make_wfs_prefilter.m` will save the impulse response of the designed filter in a file like `wfs_prefilter_120_1500_44100.wav`. From the file name you can extract that the spectral correction starts at 120 Hz and goes up to 1500 Hz at a sampling frequency of 44100 Hz. Check the folder `data/impules_responses/wfs_prefilters` for a small selection of prefilters.

## Tapering

When the listening area is not enclosed by the loudspeaker setup, artifacts arise in the reproduced sound field due to the limited aperture. This problem of spatial truncation can be reduced by so-called tapering. Tapering is essentially an attenuation of the loudspeakers towards the ends of the setup. As a consequence, the boundaries of the aperture become smoother which reduces the artifacts. Of course, no benefit comes without a cost. In this case the cost is amplitude errors for which the human ear fortunately does not seem to be too sensitive.

In order to taper, you can assign the optional attribute `weight` to each loudspeaker in ASDF format (refer to Section [sec:asdf]). The `weight` determines the linear (!) weight to be applied to the respective loudspeaker. It defaults to 1 if it is not specified.

## Ambisonics Amplitude Panning Renderer

The Ambisonics Amplitude Panning (AAP) renderer does very simple Ambisonics rendering. It does amplitude panning by simultaneously using all loudspeakers that are not subwoofers to reproduce a virtual source (contrary to the VBAP renderer which uses only two loudspeakers at a time). Note that the loudspeakers should ideally be arranged on a circle and the reference should be the center of the circle. The renderer checks for that and applies delays and amplitude corrections to all loudspeakers that are closer to the reference than the farthest. This also includes subwoofers. If you do not want close loudspeakers to be delayed, then simply specify their location in the same direction like its actual position but at a larger distance from the reference. Then the graphical illustration will not be perfectly aligned with the real setup, but the audio processing will take place as intended. Note that the AAP renderer ignores delays assigned to an individual loudspeaker in ASDF. On the other hand, it does consider weights assigned to the loudspeakers. This allows you to compensate for irregular loudspeaker placement.

Note finally that AAP does not allow to encode the distance of a virtual sound source since it is a simple panning renderer. All sources will appear at the distance of the loudspeakers.

If you do not explicitly specify an Ambisonics order, then the maximum order which makes sense on the given loudspeaker setup will be used. The automatically chosen order will be one of  $(L - 1)/2$  for an odd number  $L$  of loudspeakers and accordingly for even numbers.

You can manually set the order via a command line option (Section *Running SSR*) or the SSR configuration file (Section *Configuration File*). We therefore do not explicitly discriminate between “higher order” and “lower order” Ambisonics since this is not a fundamental property. And where does “lower order” end and “higher order” start anyway?

Note that the graphical user interface will not indicate the activity of the loudspeakers since theoretically all loudspeakers contribute to the sound field of a virtual source at any time.

## Conventional driving function

By default we use the standard Ambisonics panning function presented, for example, in [Neukom2007]. It reads

$$d(\alpha_0) = \frac{\sin\left(\frac{2M+1}{2}(\alpha_0 - \alpha_s)\right)}{(2M+1)\sin\left(\frac{\alpha_0 - \alpha_s}{2}\right)},$$

whereby  $\alpha_0$  is the azimuth angle of the position of the considered secondary source,  $\alpha_s$  is the azimuth angle of the position of the virtual source, both in radians, and  $M$  is the Ambisonics order.

## In-phase driving function

The conventional driving function leads to both positive and negative weights for individual loudspeakers. An object (e.g. a listener) introduced into the listening area can lead to an imperfect interference of the wave fields of the individual loudspeakers and therefore to an inconsistent perception. Furthermore, conventional Ambisonics panning can lead to audible artifacts for fast source motions since it can happen that the weights of two adjacent audio frames have a different algebraic sign.

These problems can be worked around when only positive weights are applied on the input signal (*in-phase* rendering). This can be accomplished via the in-phase driving function given e.g. in [Neukom2007] reading

$$d(\alpha_0) = \cos^{2M}\left(\frac{\alpha_0 - \alpha_s}{2}\right).$$

Note that in-phase rendering leads to a less precise localization of the virtual source and other unwanted perceptions. You can enable in-phase rendering via the according command-line option or you can set the `IN_PHASE_RENDERING` property in the SSR configuration file (see section *Configuration File*) to be `TRUE` or `true`.

## Generic Renderer

The generic renderer turns the SSR into a multiple-input-multiple-output convolution engine. You have to use an ASDF file in which the attribute `properties_file` of the individual sound source has to be set properly. That means that the indicated file has to be a multichannel file with the same number of channels like loudspeakers in the setup. The impulse response in the file at channel 1 represents the driving function for loudspeaker 1 and so on.

Be sure that you load a reproduction setup with the corresponding number of loudspeakers.

It is obviously not possible to move virtual sound sources since the loaded impulse responses are static. We use this renderer in order to test advanced methods before implementing them in real-time or to compare two different rendering methods by having one sound source in one method and another sound source in the other method.

Download the ASDF examples from <http://spatialaudio.net/ssr/> and check out the file `generic_renderer_example.asd` which comes with all required data.

	individual delay	weight
binaural renderer	-	-
BRS renderer	-	-
VBAP renderer	+	+
WFS renderer	-	+
AAP renderer	autom.	+
generic renderer	-	-

Table 1: Loudspeaker properties considered by the different renderers.

	gain	mute	position	orientation <sup>2</sup>	distance attenuation	model
binaural renderer	+	+	+	+	+	only w.r.t. ampl.
BRS renderer	+	+	-	-	-	-
VBAP renderer	+	+	+	+	+	only w.r.t. ampl.
WFS renderer	+	+	+	+	+	+
AAP renderer	+	+	+	-	+	only w.r.t. ampl.
generic renderer	+	+	-	-	-	-

Table 2: Virtual source's properties considered by the different renderers.

## Summary

Tables 1 and 2 summarize the functionality of the SSR renderers.

---

<sup>2</sup> So far, only planar sources have a defined orientation. By default, their orientation is always pointing from their nominal position to the reference point no matter where you move them. Any other information or updates on the orientation are ignored. You can change this behavior by using either the command line option `--no-auto-rotation`, using the `AUTO_ROTATION` configuration parameter, or hitting `r` in the GUI.

---

## Graphical User Interface

---

Our graphical user interface (GUI) is quite costly in terms of computation. So we emphatically recommend that you **properly configure the hardware acceleration of your graphics card**. If you still have performance issues make the window as small as possible. The smaller the window is the less is the processing cost.

The SSR GUI tries to enable samplebuffer support to enable anti-aliasing of the screen output. It will tell you if it didn't work out. Check Fig. 4.1 to get an idea of the influence of anti-aliasing. One day we will also implement a variable frequency for the screen update so that you can slow it down if CPU load is too high. Of course it won't look as nice then.

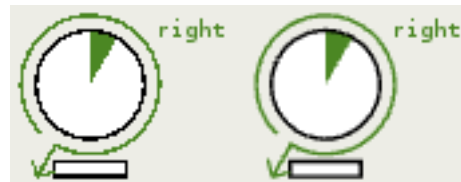


Fig. 4.1: No anti-aliasing on the left image.

### General Layout

The graphical user interface (GUI) consists mainly of an illustration of the scene that you are hearing and some interaction tools. The renderer type is indicated in the window title. See a screen shot in Fig. 4.2.

On the top left you will find the file menu where you can open files, save scenes, and quit the application. So far only the *save scene as...* option is available. That means every time to save the current scene you will be asked to specify the file name. This will be made more convenient in the future.

Next to the file menu, there is a button which lets you activate and deactivate the audio processing. Deactivating the audio processing does not necessarily lower the CPU load. It means rather that the SSR won't give any audio output, neither for involved audio files nor for live inputs.

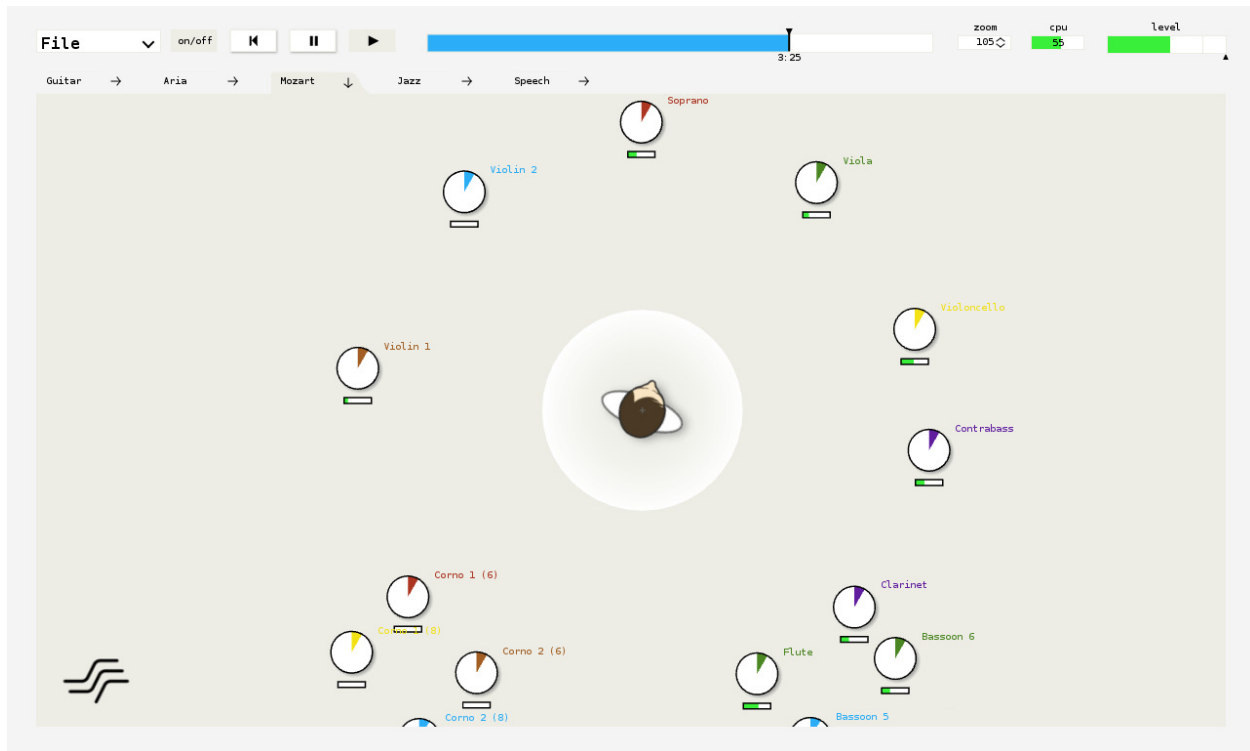


Fig. 4.2: Screen shot of the SSR GUI.

Next to the processing button, you find the transport section with buttons to skip back to the beginning of a scene, pause replaying, and continue/start playing. Note that pausing a scene does not prevent live inputs from being processed. To prevent audio output switch off processing (see above). You may also replay while processing is switched off to navigate to a certain point in time in the respective scene.

In the top middle section of the GUI there is the audio scene time line. By default, it shows a time interval of two minutes duration. Whenever the progress exceeds the displayed time interval the latter is shifted such that the progress is always properly indicated. Below the handle, there is a numerical indication of the elapsed time with respect to the beginning of the scene. See Sec. *Mouse Actions* for information on how to operate on the time line.

To the right of the time line there's the CPU load gauge. It displays the average CPU load as estimated by the JACK audio server on a block-wise basis. Further right there's the label to indicate the current zoom factor in percent.

And finally, on the top right you find the master level meter combined with the master volume fader. The colored bar indicates an estimation of the relative maximum audio level in dB, also updated block-wise. The left boundary of the meter is at -50 dB; the right boundary is at +12 dB. The black triangle below the colored bar indicates the master volume in dB. Click somewhere into the widget and the master volume gets additionally displayed as a number. Note that this meter displays full scale, i.e. above 0 dB clipping and thus distortion of the output signal occurs! 0 dB is indicated by a thin vertical line.

In the row below the transport section, you occasionally find some tabs giving fast access to a number of scenes. These tabs can be defined in a file. By default, the file `scene_menu.conf` in the current working directory is assumed; there is also an option to specify the file name in the SSR configuration file. Refer to Sec. *Configuration-File*. The configuration file for the tabs may contain something like the following:

```
# This file configures the menu for the scene selection.
#
scenes/dual_mono.asd Guitar##### comments are possible
scenes/jazz.asd Jazz
```

```
scenes/rock.asd Rock
#scenes/speech.asd Speech
scenes/live_conference.xml live conference
```

The syntax is as follows:

- Everything after a hash symbol (#) in a line is ignored.
- A valid entry consists of the path (relative or absolute) to ASDF file (or pure audio file) followed by space and a short keyword that will be displayed on the respective tab on the screen.

Of course, also audio files can be specified instead of .asd files. Note that so far, no syntax validation is performed, so watch your typing. We furthermore recommend that you keep the keywords short because space on the screen is limited. Note also that only those tabs are displayed which fit on the screen.

The SSR always tries to find the file `scene_menu.conf` in its current working directory (or at the location specified in the SSR configuration file). If it does not find it no tabs will be displayed in the GUI. So you can have several of such files at different locations. We have added an example in folder `data/`.

The main part of the screen is occupied by the graphical illustration of the scene that you are hearing. The orientation of the coordinate system is exactly like depicted in Fig. 1.1. I.e., the  $x$ -axis points to the right of the screen, the  $y$ -axis points to the top of the screen. The origin of the coordinate system is marked by a cross, the reference is marked by a rhomb. The direction “straight in front” is typically assumed to be vertically upwards on the screen, especially for binaural techniques. We do so as well. Note that in this case “straight in front” means  $\alpha = 90^\circ$  and NOT  $\alpha = 0^\circ$ .

In Fig. 4.2 you see a number of sound sources with their individual audio level meters (combined with their individual volume sliders) underneath. The left hand boundary of the level meter is at -50 dB; the right hand boundary is at 0 dB. Spherical sources don’t have any additional decoration. The wave front and propagation direction of plane waves are indicated.

You also see icons for the loudspeakers of the current rendering setup (if the currently applied technique employs any).

## Mouse Actions

The GUI is designed such that the most important functionalities can be accessed via a touch screen. Thus, it mostly employs ‘left clicks’ with the mouse.

The use of the file and transport section is rather intuitive so we won’t further explain it here. The time line can be used to jump to a certain position within the sound scene and it also shows the progress of the scene. Click into the white/blue area of the time line in order to jump to a specific point in time, or drag the handle to fast forward or rewind. Left-clicking to the right of the time line skips forward by 5 seconds, left-clicking to the left of the time line skips back by 5 seconds. Double-clicking on the time line skips back to the beginning of the scene. Right-clicking on the time line opens an input window in order that you can numerically specify the time instant to jump to (refer to Sec. *Keyboard Actions*).

You can change the zoom either by clicking into the zoom label and dragging up or down for zooming in or out. Alternatively, you can use the mouse wheel<sup>1</sup>. Clicking and dragging on the background of the screen lets you move inside the scene. A double-click brings you back to the default position and also defaults the zoom.

Clicking and dragging on a sound source lets you select and move it. Note that you cannot directly manipulate the propagation direction of plane waves. It’s rather such that plane sources always face the reference point. To change their direction of incidence move the plane wave’s origin point to the appropriate position. Right clicking<sup>2</sup> on a sound source opens a window that lists the properties of the source such as position, volume, etc. Refer to Fig. 4.3 and Sec. *Source Properties Dialog*.

<sup>1</sup> On a touchpad on Mac OS, swipe up and down with two fingers to zoom in and out (don’t click!).

<sup>2</sup> On a touchpad on Mac OS, click with two fingers simultaneously.

A right mouse click on the scene background<sup>3</sup> lets you select multiple sound sources via a rubber band.

If you hold the `Ctrl` key pressed during any mouse action then you operate on all selected sound sources at the same time (i.e. mute, move, etc. them).

Click on the SSR logo and you'll see the *About the SSR* information.

## Source Properties Dialog

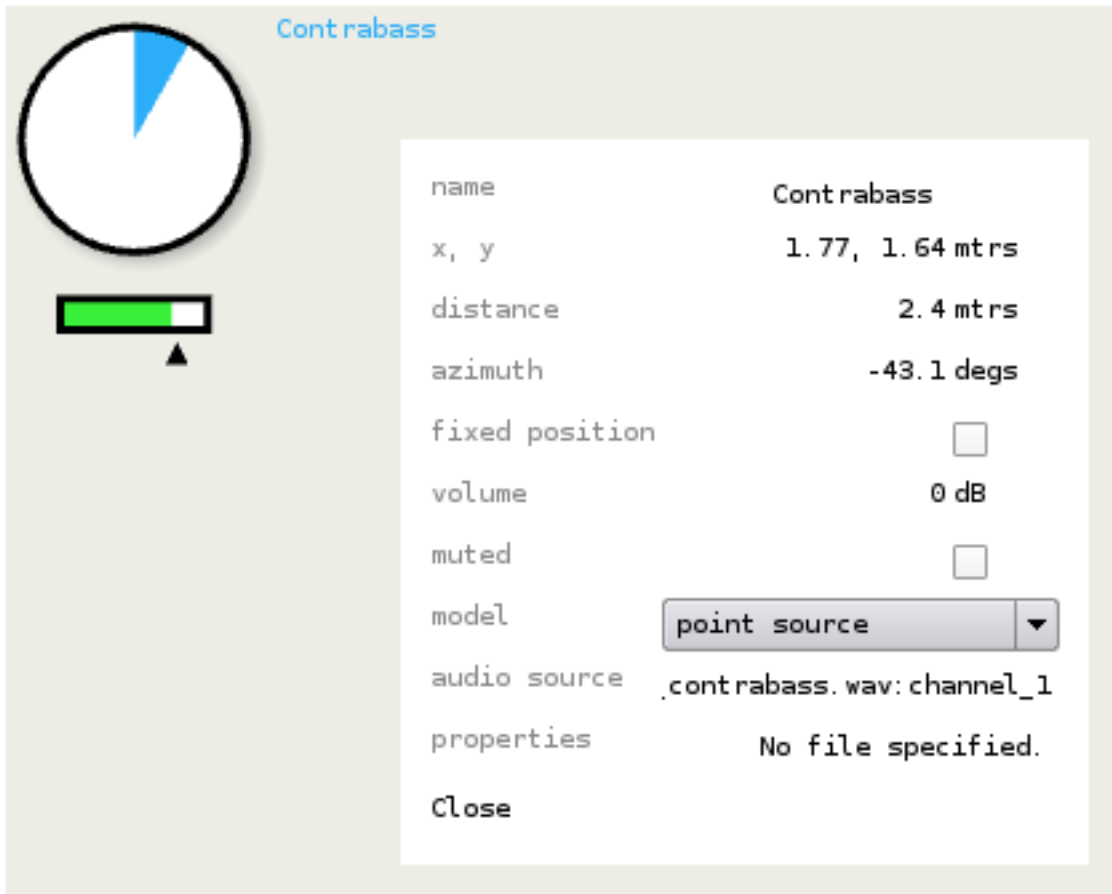


Fig. 4.3: Source properties dialog

The source properties dialog can be accessed via a right click on a source and shows information about the actual state of the selected source. Its main purpose is to provide the possibility of an exact positioning of sources. The properties `fixed position`, `muted` and `model` can be changed. Refer to Fig. 4.3 to see the complete list of properties this dialog shows.

## Keyboard Actions

A number of keyboard actions have been implemented as listed below. Recall that also some keyboard actions are available when the SSR is run without GUI (refer to Sec. *Running SSR*).

<sup>3</sup> On a touchpad on Mac OS, click with two fingers simultaneously, hold the click and move one finger, or both fingers simultaneously, or use a third finger to operate the rubber band.



- +/-: if no sound source is selected: raise/lower master volume by 1dB, otherwise raise/lower the selected sources' volume by 1dB
- Arrow up/down/left/right: navigate in scene
- Space: toggles the play/pause state
- Backspace: skip to beginning of scene
- Return: calibrate tracker (if present). When pressed, the instantaneous orientation is assumed to be straight forward (i.e.  $90^{\circ}$  azimuth)
- Ctrl: when pressed, multiple sound sources can be selected via mouse clicks or operations can be performed on multiple sources simultaneously
- Ctrl+Alt: individual sound sources can be deselected from a larger selection via a mouse click or the rubber band
- Ctrl+a: select all sources
- f: toggles the position-fix-state of all selected sound sources (sources which can not be moved are marked with a little cross)
- m: toggles the mute state of all selected sound sources (muted sources are displayed with a grey frame instead of a black one)
- p: toggles the source model between *plane wave* and *point source*
- r: toggles whether or not all sources are always automatically oriented toward the reference.
- s: if no source selected: unsolos all potentially soloed sources, otherwise: solos selected sound sources.
- Ctrl+s: opens the *save scene as...* dialog
- F11: toggles window fullscreen state
- 1-9: select source no. 1-9
- 0: deselect all sources
- Ctrl+c: quit
- Ctrl+t: open text edit for time line. The format is hours:mins (2digits) :secs (2digits) whereby hours: and hours:mins (2digits) : can be omitted if desired.
- Esc: quit



---

## Network Interface

---

This is just a short overview about the XML messages which can be sent to the SSR via TCP/IP. By default, messages have to be terminated with a binary zero (`\0`). This can be changed to, for example, a newline / line feed (`\n`) or to a carriage return (`\r`) using one of the following ways:

- Command line: Use the command line option `--end-of-message-character=VALUE` VALUE is the ASCII code for the desired character (binary zero: 0; line feed: 10; carriage return: 13).
- Configuration file: Here, there is the option `END_OF_MESSAGE_CHARACTER`. See the example `data/ssr.conf.example`. Use ASCII codes here as well.

The choice of delimiter applies to, of course, both sent and received messages.

**WARNING:** We did not evaluate the network interface in terms of security. So please be sure that you are in a safe network when using it.

### Scene

- Load Scene: `<request><scene load="path/to/scene.asd"/></request>`
- Clear Scene (remove all sources): `<request><scene clear="true"/></request>`
- Set Master Volume (in dB): `<request><scene volume="6"/></request>`

### State

- Start processing: `<request><state processing="start"/></request>`
- Stop processing: `<request><state processing="stop"/></request>`
- Transport Start (Play): `<request><state transport="start"/></request>`
- Transport Stop (Pause): `<request><state transport="stop"/></request>`
- Transport Rewind: `<request><state transport="rewind"/></request>`

- Transport Locate: `<request><state seek="4:33"/></request>` `<request><state seek="1.5 h"/></request>` `<request><state seek="42"/></request>` (*seconds*)  
`<request><state seek="4:23:12.322"/></request>`
- Reset/Calibrate Head-Tracker: `<request><state tracker="reset"/></request>`

## Source

- Set Source Position (in meters): `<request><source id="42"><position x="1.2" y="-2"/></source></request>`
- Fixed Position (true/false): `<request><source id="42"><position fixed="true"/></source></request>`

```
<request><source id="42">
  <position x="1.2" y="-2" fixed="true"/>
</source></request>
```

- Set Source Orientation (in degrees, zero in positive x-direction): `<request><source id="42"><orientation azimuth="93"/></source></request>`
- Set Source Gain (Volume in dB): `<request><source id="42" volume="-2"/></request>`
- Set Source Mute (true/false): `<request><source id="42" mute="true"/></request>`
- Set Source Name: `<request><source id="42" name="My first source" /></request>`
- Set Source Model (point/plane): `<request><source id="42" model="point"/></request>`
- Set Source Port Name (any JACK port): `<request><source id="42" port="system:capture_3"/></request>`
- New Source (some of the parameters are optional):

```
<request>
  <source new="true" name="a new source"
    file="path/to/audio.wav" channel="2">
    <position x="-0.3" y="1" fixed="true"/>
    <orientation azimuth="99"/>
  </source>
</request>
```

```
<request>
  <source new="true" name="a source from pd"
    port="pure_data_0:output0" volume="-6">
    <position x="0.7" y="2.3"/>
  </source>
</request>
```

- Delete Source: `<request><delete><source id="42"/></delete></request>`

## Reference

- Set Reference Position (in meters): `<request><reference><position x="-0.3" y="1.1"/></reference></request>`

- Set Reference Orientation (in degrees, zero in positive x-direction):  
`<request><reference><orientation azimuth="90"/></reference></request>`



---

## Bibliography

---

- [Geier2008a] Matthias Geier, Jens Ahrens, and Sascha Spors. The SoundScape Renderer: A unified spatial audio reproduction framework for arbitrary rendering methods. In 124th AES Convention, Amsterdam, The Netherlands, May 2008 Audio Engineering Society (AES).
- [Geier2012] Matthias Geier and Sascha Spors. Spatial audio reproduction with the SoundScape Renderer. In 27th Tonmeistertagung – VDT International Convention, 2012.
- [Geier2008b] Matthias Geier, Jens Ahrens, and Sascha Spors. ASDF: Ein XML Format zur Beschreibung von virtuellen 3D-Audioszenen. In 34rd German Annual Conference on Acoustics (DAGA), Dresden, Germany, March 2008.
- [Ahrens2008a] Jens Ahrens and Sascha Spors. Reproduction of moving virtual sound sources with special attention to the doppler effect. In 124th Convention of the AES, Amsterdam, The Netherlands, May 17–20, 2008.
- [Ahrens2008b] Jens Ahrens and Sascha Spors. Reproduction of virtual sound sources moving at supersonic speeds in Wave Field Synthesis. In 125th Convention of the AES, San Francisco, CA, Oct. 2–5, 2008.
- [Lindau2007] Alexander Lindau and Stefan Weinzierl. FABIAN - Schnelle Erfassung binauraler Raumimpulsantworten in mehreren Freiheitsgraden. In Fortschritte der Akustik, DAGA Stuttgart, 2007.
- [Wierstorf2011] Hagen Wierstorf, Matthias Geier, Alexander Raake, and Sascha Spors. A Free Database of Head-Related Impulse Response Measurements in the Horizontal Plane with Multiple Distances. In 130th Convention of the Audio Engineering Society (AES), May 2011.
- [SpatialAudio] <https://github.com/spatialaudio/lf-corrected-kemar-hrtfs> (commit 5b5ec8)
- [AlgaziCIPIC] V. Ralph Algazi. The CIPIC HRTF database. [http://interface.cipic.ucdavis.edu/CIL\\_html/CIL\\_HRTF\\_database.htm](http://interface.cipic.ucdavis.edu/CIL_html/CIL_HRTF_database.htm).
- [BRIRs] The Sputnik BRIRs can be obtained from here: [https://dev.qu.tu-berlin.de/projects/measurements/wiki/Impulse\\_Response\\_Measurements](https://dev.qu.tu-berlin.de/projects/measurements/wiki/Impulse_Response_Measurements). More BRIR repositories are compiled here: <http://www.soundfieldsynthesis.org/other-resources/#impulse-responses>.
- [Pulkki1997] Ville Pulkki. Virtual sound source positioning using Vector Base Amplitude Panning. In Journal of the Audio Engineering Society (JAES), Vol.45(6), June 1997.
- [Spors2008] Sascha Spors, Rudolf Rabenstein, and Jens Ahrens. The theory of Wave Field Synthesis revisited. In 124th Convention of the AES, Amsterdam, The Netherlands, May 17–20, 2008.
- [Spors2006] Sascha Spors and Rudolf Rabenstein. Spatial aliasing artifacts produced by linear and circular loudspeaker arrays used for Wave Field Synthesis. In 120th Convention of the AES, Paris, France, May 20–23, 2006.

[Neukom2007] Martin Neukom. Ambisonic panning. In 123th Convention of the AES, New York, NY, USA, Oct. 5–8, 2007.