

real-time Simulated Open Field Environment

rtSOFE and Convolver User Manual

Version 1.1

Bernhard U. Seeber,

Tong Wang,

Hendrik Noeller, Manuel Hornung, Clara Hollomey and Max Kirchmeier

November 5, 2021

# Contents

<b>1</b>	<b>Introduction</b>	<b>3</b>
<b>2</b>	<b>Architecture Overview</b>	<b>3</b>
<b>3</b>	<b>Execution</b>	<b>4</b>
<b>4</b>	<b>rtsofe.exe</b>	<b>5</b>
4.1	Parallelization Environment Variables . . . . .	5
4.2	Configuration . . . . .	5
4.2.1	Image Source Simulation . . . . .	5
4.2.2	Memory Usage . . . . .	7
4.2.3	Rendering . . . . .	7
4.2.4	Configuration Examples and Templates . . . . .	8
4.3	Coordinate System . . . . .	8
4.4	Valid Roomfiles . . . . .	8
4.5	Troubleshooting . . . . .	9
4.6	Bugs that we live with . . . . .	9
<b>5</b>	<b>convolver.exe</b>	<b>9</b>
5.1	Audio Routing . . . . .	9
5.2	Partitioned FFT Convolution . . . . .	11
5.3	Application States and OSC Control . . . . .	12
5.4	Troubleshooting . . . . .	12
<b>6</b>	<b>OSC Tools – Python and Matlab utilities</b>	<b>12</b>
<b>7</b>	<b>License</b>	<b>13</b>
<b>8</b>	<b>List of Contributors</b>	<b>14</b>
<b>9</b>	<b>Funding Acknowledgment</b>	<b>14</b>
<b>A</b>	<b>Material properties file</b>	<b>15</b>
<b>B</b>	<b>Source directivity file</b>	<b>15</b>
<b>C</b>	<b>Speaker mapping file</b>	<b>16</b>
<b>D</b>	<b>Ambisonics LUT file</b>	<b>16</b>
<b>E</b>	<b>Impulse response file</b>	<b>17</b>
<b>F</b>	<b>Equalization filters file</b>	<b>17</b>
<b>G</b>	<b>Receiver directionality SOFA files</b>	<b>17</b>
<b>H</b>	<b>OSC address patterns and parameters</b>	<b>17</b>
H.1	rtsofe.exe . . . . .	18
H.2	convolver.exe . . . . .	19
	<b>Bibliography</b>	<b>21</b>

# 1 Introduction

rtSOFE.exe and convolver.exe are the two main applications that form the real-time Simulated Open Field Environment (rtSOFE) for real-time auralization of virtual acoustics: rtSOFE.exe for the offline and online simulation of spatial room impulse responses (IRs) and convolver.exe for real-time convolution of a stream of IRs with audio from files or sound card channels, the summation of multiple such convolutions to create early and late reflection parts or multiple simultaneous sound sources, and loudspeaker equalization. They can also be used independently to simulate IRs for other purposes or to convolve signals with an IR from a different source. This manual explains their use starting from an overview of the technology.

## 2 Architecture Overview

The two applications divide the required tasks for (real-time) auralization amongst them:

**rtsofe.exe** creates an IR from a 3D room geometric model using the image source method based on the approach by Borish (1984) and follows the earlier SOFE implementation presented in Hafter and Seeber (2004) and Seeber et al. (2010). It can render image sources into IRs with different approaches: binaural rendering, nearest speaker mapping, 2D-Ambisonics. It can either write that IR to a file and exit (static mode for offline simulation) or receive source/receiver position updates over open sound control (OSC) (e.g. from a motion tracker), render new IRs in real-time and send them to convolver.exe over a TCP connection (real-time mode).

**convolver.exe** manages multiple real-time convolutions with time varying IRs. Each IR is either received from an instance of rtsofe.exe or read from a file. Signals can come from audio files or through a sound card input.

Together they can be connected to create various configurations relevant for hearing research, from high-order offline simulations for pre-determined experiments or auditory modeling to online simulation of source trajectories for real-time auralization or model-in-the-loop approaches. A common configuration is convolving different orders of an IR separately to be able to update them independently. A simulation of the entire IR is usually not feasible in real time because of the high computational cost of high order image source (IS) simulations. Luckily, the most perceptually relevant reflections are the lowest order ones which are relatively cheap to compute. To exploit this situation, a possible setup comprises one instance of rtsofe.exe computing early reflections (e.g., up to order 3) with a high refresh rate and a second instance calculating later reflections (e.g. to order 10) that takes more time to compute. The diffusely reverberant rest of the IR could assumed to be position independent and loaded from a file. The specific orders may vary depending on the application, available computer and used room model. This setup is illustrated in figure 1.

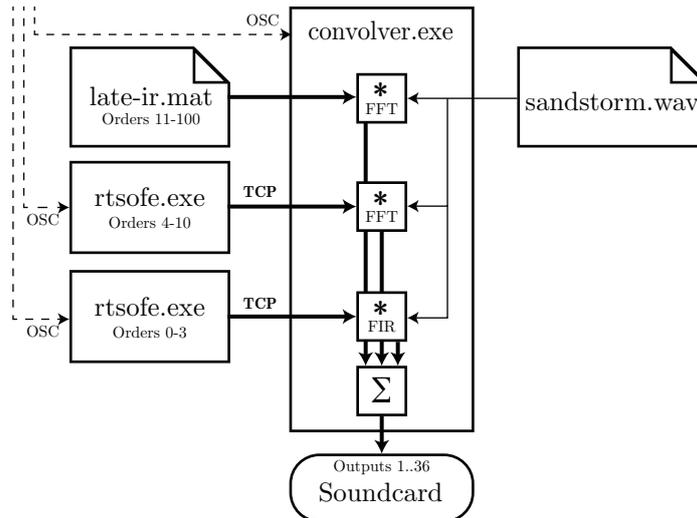


Figure 1: Common setup for real-time auralization with two instances of `rtsofe.exe` and one IR for late, diffuse reverberation loaded from a file. The input signal can come from a file or as real-time input from the sound card.

### 3 Execution

Both applications do not have a graphical user interface (GUI) and need to be started from the command line. Shared `.dll` files are provided in the same folder as the binary files.

The applications can receive their configuration through the command line and through configuration files, with selected parameters also being updatable via OSC messages. Because the configurations are usually extensive but largely reusable, config files are the recommended mechanism. They are expected to be `.ini` files, formatted as follows:

```
# this is a comment
key = value
mode = interactive # this is also a comment
other-key = 42
```

A complete list of available configuration options (including their default values and detailed explanations) can be obtained by running `rtsofe.exe --help / convolver.exe --help`. To execute an application with a given configuration, run:

```
convolver.exe --config myconfig.ini
```

To start (multiple) applications asynchronously in their own windows, start a new command prompt or Powershell window, run:

```
start /high rtsofe.exe --config myconfig1.ini
start /high rtsofe.exe --config myconfig2.ini
start /high rtsofe.exe --config myconfig3.ini
start /high convolver.exe --config myconfig.ini
```

*In the following, when discussing a configuration key it is always written in a typewriter-font with an example value, like this: `sampling-rate = 44100`.*

The first time `rtsofe.exe` or `convolver.exe` runs from a new directory, e.g. through a Windows batch file, both process will evaluate the system it runs on and generate FFTW3 "Wisdom" files and save it in the working directory before exiting. This process takes some extra time during the initialization of `rtsofe.exe` and `convolver.exe`, but it usually runs only once. If the two programs can locate existing wisdom files, they will use the existing files instead.

## 4 rtsofe.exe

### 4.1 Parallelization Environment Variables

rtsofe.exe evaluates a few environment variables to initialize the OpenMP parallelization of the IS computation. They need to be set before launching the application.

- `OMP_NUM_THREADS` sets the number of threads
- `OMP_SCHEDULE` selects the scheduler (we recommend `guided`)
- `KMP_AFFINITY` controls the distribution of threads across the hardware. `balanced` assigns one thread to each physical core and only uses hyper threaded cores if there aren't enough physical ones. `compact` assigns the threads to hardware cores in ascending order, always filling the hyper threaded cores of a physical core together. If the number of threads is equal to the number virtual cores, they should be identical. For lower thread counts, `balanced` usually yields better performance.

For early reflections or if `convolver.exe` runs on the same computer, the best results are achieved when using 2-4 threads in `balanced` mode. For late reflections, one thread per virtual cores should be used.

Note: The observation above may vary for different hardware configurations.

When running the whole rtSOFE system in the interactive mode, including one or more `rtsofe.exe` processes and the `convolver.exe` process, on one computer with hyper-threading enabled, it is recommended to start each process with the `start /affinity [mask]` option to prevent running different processes on two logical cores of a physical core. This is useful to maximize the refresh rate when rtSOFE do not occupy all cores on the computer.

Though the exact hardware requirement for running rtSOFE in the interactive mode depends on the maximal reflection order, it is recommend to run rtSOFE on a computer with a two-core CPU, at least.

### 4.2 Configuration

The first option to set is `mode = static` or `mode = interactive`, which controls whether `rtsofe.exe` will write one IR to a file or compute updated IRs and send them over a TCP connection. In the static mode, `rtsofe.exe` will run once with given parameters and exit, generating a file named `imuplse.responses.mat` containing the multi-channel impulse response, if the program finishes successfully. With `write-sources = 1`, an additional file `image_sources.mat` will be generated, containing all visible and invisible simulated image sources. The default value for `write-sources` is 0 or `false`, because exporting all image sources to the disk is a slow process, especially when simulating high order image sources.

If interactive mode is selected, the application renders the IR with initial parameters and listens to OSC commands for parameter updates to render the next IR. `rtsofe.exe` needs to open two ports for network communication: a UDP port to receive OSC packets and a TCP port to which a convolver connects to receive IRs. They are set using `osc-input-port = 56789` and `output-port = 1234`. The `rtsofe.exe` process will keep running until it receives an OSC message with an address pattern `'/exit'`.

All other parameters configure the simulation. Some of them can be changed over OSC in interactive mode, but they still should be configured with an initial value for the first computation.

#### 4.2.1 Image Source Simulation

To load room geometry, pass an `.obj` file using `room = myroom.obj` and the accompanying material description using `materials = materials.txt`. If your file is rejected because it supposedly contains invalid geometry but you are sure this is a mistake or want to use the file anyway, you can set `room-force-load = true` to circumvent the test. Please refer to section 4.4 for more info on invalid room geometry.

Source- and receiver positions are provided using `source-pos` and `receiver-pos` as xyz coordinates by writing out the argument three times:

```
# arguments that receive a vector of values are filled
# by writing one line for each element.
# For example, to set source-position to [6, 9, 1.5]
source-position = 6
source-position = 9
source-position = 1.5
```

The source can have a direction-dependent frequency response which can be enabled by passing a file describing the characteristics using `source-directivity = directivity-file.txt`. To determine the orientation of the source, three specific parameters and two default behaviours are possible. If multiple ones are provided, a later option overwrites all previous ones.

- If the receiver position uses the default value, the source will look along the Y-Axis
- If a receiver position is given, the source will look at the initial receiver position
- `source-look-at` a point in 3D space the source will look at
- `source-transform` a custom  $3 \times 3$  rotation matrix to transform source directions by pre-multiplication
- `receiver-transform` a custom  $3 \times 3$  rotation matrix to transform receiver directions by pre-multiplication. Example for specifying a source-transform or receiver-transform in the configuration file:

```
# a custom receiver coordinate transformation matrix; can be used to give
# the receiver a custom rotation around all directions

# first column
receiver-transform = 0.42335
receiver-transform = 0.44894
receiver-transform = -0.78691
# second column
receiver-transform = -0.6533
receiver-transform = 0.75305
receiver-transform = 0.078148
# third column
receiver-transform = 0.62767
receiver-transform = 0.48101
receiver-transform = 0.61209
```

- `lock-source-to-receiver` (interactive mode only) the source will follow the receiver to always look at it

To control the depth of the IS simulation, you can combine a variety of termination criteria. Further branching of the IS tree will be stopped once any of the criteria is reached and the simulation finishes if all branches are stopped.

- If no termination criteria are given, `rtsofe.exe` will refuse to run because it's a little afraid of unbounded execution.
- `order = 7` defines the highest order that will be computed (inclusive)
- `max-invis-parents = 3` defines the maximum number of invisible parent sources. This is useful because sources with a high number of invisible parents are more likely to be invisible themselves.
- `max-distance = 500` defines the maximum distance to the receiver in meters.
- `max-sources = 100000` defines a maximum number of sources.
- `attenuation-threshold = -90` defines the maximum attenuation in negative dB. This is approximated using an upper bound, so some sources with stronger attenuation might still be created.

`rtsofe.exe` uses a protrusion check to ensure sound is not obstructed on its path through the room. If you use a simple shoebox room with no obstructions inside, you can save resources by deactivating it using `protrusion-check = false`.

To increase refresh rates in interactive mode, the re-computation of ISs and visibility checks can be skipped for small movements of source or receiver. In such a case, only source positions are recomputed and the new IR is then rendered using the old ISs visibility information. These thresholds are set in meters:

```
source-movement-threshold = 0.05
receiver-movement-threshold = 0.1
```

### 4.2.2 Memory Usage

Memory for all created ISs will be dynamically allocated. Nevertheless, there is an option to pre-allocate memory for a certain number of ISs with the parameter `is-buf-size = 65536` which can be beneficial in real-time mode. If `rtsofe.exe` needs more memory than is physically available, the program will exit with a *bad allocation* error.

The theoretical upper bound of the number of image sources for a model with  $k$  walls up to order  $n$  is  $k * (k - 1)^{(n-1)}$ . However, this only applies if no other termination criteria are given.

### 4.2.3 Rendering

`rtsofe.exe` always renders using 32bit floating point representation. The sampling rate can be set using `sampling-rate = 44100`.

When using multiple instances of `rtsofe.exe` to compute different parts of the same IR, it is crucial to ensure no order is rendered by more than one instance. While the upper limit can be set in the simulation as explained above, lower order sources are inevitably created in the process. They instead have to be discarded during rendering using `min-order = 4` (inclusive: in this example, order 4 will also be rendered).

The frequency characteristics of each reflection are modeled by an inverse FFT creating a finite impulse. It includes reflection coefficients along the reflection path, source directivity, air absorption and receiver directivity when using binaural synthesis. Its length in samples is set using `impulse-length`. Higher values yield a better frequency resolution but reduce IR sparsity and add the latency. The air absorption depends on the humidity that can be set using `humidity = 0.2`.

The delay of an impulse is determined from the speed of sound in meters per second: `speed-sound = 344`. To avoid artificially strong comb filtering, this delay can be randomly jittered for sources starting from a set order. Note that setting the `delay-jitter` parameter too large or the jitter start order too low can cause audible artifacts with moving sound sources or receivers since the jitter is computed anew on every new IR rendering.

```
delay-jitter = 0.005
delay-jitter-start-order = 5
```

To spatially render each IR, i.e. to translate the position of an image source to a placement of the impulse into one or more IR channels, the speaker mapping informs the system how many loudspeakers are available, at what angle they are located and which channel of the matrix holding the spatial impulse response they are linked with. They are assumed to be equidistant to the listener, which can be ensured with loudspeaker equalization filters (see `convolver.exe`). This can be read from a `.mat` file using `speaker-mapping-file` and `speaker-mapping` (see section C).

To render room impulse response (RIR) for loudspeaker array reproduction, set the parameter `render-mode = LS_array`. To render binaural room impulse response (BRIR) with head-related transfer function (HRTF) rendering, set the parameter `render-mode = binaural`.

For playing back on a loudspeaker array, the user can choose between 2D-Ambisonics and 3D-nearest speaker rendering, set `LS-method-1 = nearest` or `LS-method-1 = ambi_2d`.

If you want to use a different method for sources above a certain order:

```
LS-method-1 = ambi_2d
LS-method-2 = nearest
LS-method-switch-order = 3
```

When using Ambisonics rendering, a lookup table (LUT) with precomputed coefficients must be provided: `LS-ambi-lut-file-2D = 2D.lut.basic.mat`. Ambisonics rendering currently only considers azimuth and ignores elevation. The LUT must have an angular grid of `0:0.1:359.9` and contain a copy of the speaker mapping it was generated for. This speaker mapping will be compared to the configured one and must be identical or a subset of it.

When using binaural rendering, the number of channels will be set to 2. The first IR channel will be for the left ear and the second for the right ear. Parameters `speaker-mapping-file` and `speaker-mapping` are overridden and ignored.

The HRTF dataset must be saved in the spatially oriented format for acoustics (SOFA) format, as defined in AES69-2020, by Majdak and Noisternig (2020). `binaural-hrtf-file = C:\file01.sofa` instructs `rtsofe.exe` to load the HRTF dataset and to linearly interpolate the dataset to an internally used

high spatial resolution during the initialization stage. In the rendering stage, the nearest HRTF pair to the sound direction will be selected from that lookup table to map a reflection. Alternatively, the user may use `binaural-hrtf-rt-interp = 1` to instruct `rtSOFE` to skip interpolation in the initialization stage and linearly interpolate the HRTF filters during IR rendering in real-time. Real-time HRTF interpolation takes more time to process each buffer, but uses less memory and provides a spatial resolution as high as linear interpolation and numeric precision can provide.

In addition, please make sure that the SOFA file satisfies the following requirements:

- has the same sampling rate as the `rtsofe.exe` configuration and uses "Hz" as unit of sampling rate
- contains at least 2 HRTF pairs
- filter length of the HRTF filters has to be divisible by 8
- unit for additional delays (if provided) is "samples".

SOFA files are available from online databases.

#### 4.2.4 Configuration Examples and Templates

To give an overview of application areas of `rtSOFE` and their setup, a few example configurations are provided in the directory: `rtSOFE/ExampleConfiguration/Configurations`.

The example use cases demonstrate how to use `rtSOFE` in static and interactive mode, and how to use the Python OSC utilities and the MATLAB multi-channel IR receiver.

A detailed description of use cases regarding each example configuration can be found at `desc.txt` in each subfolder.

### 4.3 Coordinate System

The orientation of an xyz coordinate system in relation to azimuth/elevation angles is not universally standardized. Coupling other systems (such as a motion tracker) to `rtSOFE` may require coordinate transforms. `rtSOFE` accepts positions and rotation matrices of sources and receivers in the following coordinate system (from the viewpoint of the origin):

- positive X points to the right
- positive Y points to the front
- positive Z points upwards
- 0° Azimuth points to the front (along the Positive Y axis)
- 90° Azimuth points to the right (along the Positive X axis) ⇒ Azimuth is right-turning
- 0° Elevation points along the horizon
- 90° Elevation points upwards (along the positive Z axis)

### 4.4 Valid Roomfiles

`rtsofe.exe` can load the room model from an `.obj` file created by e.g. Blender. Please consider the following limitations:

**Walls are flat** A wall is a polygon of at least three points. All these points have to lie within a 2D plane. Curved surfaces are not supported.

**Walls have one side** A wall is only reflective to sound rays impinging from the side of the surface normal. If the corners in the `obj` file are listed in counter-clockwise order from a certain point of view, that point is on the side of the surface normal which is the reflective side. To make a wall reflective on both sides, you need to add a second wall behind it that has its normal pointing in the other direction

**Walls need to be assigned materials** Each wall must be assigned one type of material. rtSOFE needs frequency dependent reflection coefficients of the wall materials to render reflections. Relevant material characteristics, including reflection coefficients at different frequencies, should be provided in a separate file. Please refer to section A for more information on creation of the material definition file.

**Parallel walls need distance** Two parallel walls that are too close to each other pose a problem for 3D simulations. In the realm of computer graphics, this phenomenon is known as z-fighting and causes visible flickering or striping since it is left to floating point noise which surface occludes the other. This problem is exacerbated at long distances because absolute resolution diminishes for large floating point values. In rtsofe.exe it can cause sound to get “trapped” between the surfaces. To avoid these issues, “wall-sandwiches” are required to have a small distance that makes it obvious which wall is in front of the other. This applies to scenarios like the double-sided wall described above, as well as putting smaller patches of a different material onto a larger wall to model e.g. a window or an absorber. We recommend a distance of  $1\text{mm} = 0.001\text{m}$  which introduces a negligible acoustical error but is large enough to not disappear into floating point noise even at a distance of 100m. Please check 32 bit floating point precision if you are creating larger models with parallel walls in close proximity.

rtsofe.exe examines .obj files for walls that violate these criteria. The checks are not perfect, so you should inspect your models manually as well. In case the test produces a false positive, you can set `room-force-load = true` to circumvent it.

## 4.5 Troubleshooting

- **When I run two instances of rtsofe.exe on the same computer, one crashes.** Make sure they don't have the same ports configured for receiving OSC messages or sending IRs.
- **rtsofe.exe terminates during initialization or throws a *bad-allocation*.** You have set `is-buf-size` too big. Please refer to section 4.2.2.

## 4.6 Bugs that we live with

- When the line of sight between the receiver and an IS intersects exactly with edges of a wall, the test decides randomly if this IS is considered “visible” or not. However, the point-in-polygon test is so much faster than anything else that we live with it.

# 5 convolver.exe

## 5.1 Audio Routing

convolver.exe is a real-time audio application that requires an ASIO sound card or audio interface to run. Convolutions are always between a multichannel IR and a single channel signal.

**Basics** Set `sample-rate = 44100` and `buffer-size = 128` to match the configurations in the sound card's proprietary settings dialogue.

**Input Signals** Input signals are configured separately and assigned to convolutions later. They can either pull their data from a .wav file or a sound card input, determined by `input-from-file = false`. If a file is requested you need to set `input-file = sandstorm.wav`, otherwise you need to set `input-channel = 7`.

Only single-channel input signals are supported; if the input file has more than one channels, the first channel will be used with the multi-channel IR and other channels will be ignored.

**Convolvers** Convolution is handled in a convolver object that either receives its IR from a file or an instance of rtsofe.exe as determined by `ir-type = File` or `TCPMatrix`. For the former set `ir-file = my_ir.mat` (see section E for format info). If your IR file contains multiple timesteps you can use `repeat-ir = true` to loop back to the beginning when all IRs were consumed. For the latter set `rtsofe-host = 192.169.0.12` and `rtsofe-port = 6500`.

When an IR is updated from rtsofe.exe or by reading the next timestep from a file, the convolver can put a small crossfade to cover up the transition. You can set the length in samples using `intra-fade =`

64. Overlapping crossfades are not supported, so please keep your crossfade length below the shortest expected interval between updates.

The convolution can either be computed in the time- or frequency domain. The former option is set using `part-scheme = FIR`, the latter using a configuration string in the form `part-scheme = 256-4x256-0x1024`. The former is more expensive to compute but handles updates without additional latency or artifacts. The latter is cheaper on the CPU but adds additional delay to IR updates and can introduce update artifacts if set up incorrectly. Refer to section 5.2 for more information on correct Fast Fourier transform (FFT) Convolution setup.

The transition between the end of `FIR` part and the beginning of the `FFT` part is handled with fading to reduce the possible artifacts. You can set the length in samples using `inter-fade = 64`

**Multiple Convolver** `convolver.exe` supports multiple input signals and convolvers in one instance. Each convolver can be assigned one of the input signals, sharing signals between convolvers is expressly supported. All convolvers must have the same number of IR channels.

To set the number of signals, use `num-signals = 3`. The first input signal will use the aforementioned keys, all other ones can be configured by appending the index to the key:

```
input-from-file = false
input-channel = 7

input-from-file-2 = true
input-file-2 = sandstorm.wav

input-from-file-3 = false
input-channel-3 = 1
```

To use multiple convolutions set `num-convolvers = 3`. As with input signals, append numbers to the keys to configure all convolvers:

```
num-convolvers = 2

ir-type = File
ir-file = my_ir.mat
part-scheme = 1024-0x1024
input-signal = 1

ir-type-2 = TCPMatrix
rtsofe-host-2 = 127.0.0.1
rtsofe-port-2 = 6500
part-scheme-2 = FIR
input-signal-2 = 2
```

Please note that `ir-type-1` or `input-from-file-1` are not valid parameters.

Having this layer of abstraction for input signals might seem like an unnecessary detour, but comes in handy when sharing input signals between convolvers. It is especially recommended to share the same input signal between multiple convolvers that cover different segments of the same IR. This reduces the opportunity for user error when changing input channels or files.

`input-signal` and `num-signals` both default to 1. This means that all convolvers use the same input signal unless stated otherwise, making single signal configurations easier to write:

```
input-from-file = true
input-file = sandstorm.wav

num-convolvers = 2
ir-type = TCPMatrix
rtsofe-host = 127.0.0.1
rtsofe-port = 6500

rtsofe-host-2 = 127.0.0.1
rtsofe-port-2 = 6501
```

**Channel Assignments** Just like setting up speaker angles in `rtsofe.exe`, you read assignments from IR channels to hardware outputs by using `channel-assignments-file = mapping.mat` and `channel-assignments-var = mapping_AEC`. Channel Assignment files are the same format as read by `rtsofe.exe` for speaker mappings, see section C for more info.

**EQ Filters** After summing up the result of all convolutions each channel can be equalized individually to calibrate the loudspeakers. These filters consist of a finite impulse response (FIR) filter and an integer delay per channel. The delay is available separately to time align loudspeakers without convolving with leading zeros in the FIR filters. Filters are loaded from a `.mat` file given by `eq-filters-file = filters.mat` as a matrix of filters and a vector of integer delays. The name of the variable containing the matrix is set using `eq-filters-var = filters-ampl` and the delays using `eq-filters-shift-var = index-shift`. Additionally, a copy of the speaker mapping must be contained as a variable named `speaker_mapping` and is checked against the loaded channels assignments. The exact format is described in section F. To enable them, `eq-filters-on = true` must be set.

**Recording** The convolver can record the final output signal as it is sent to the sound card, including EQ filters and channel assignments. The file will contain as many channels as the highest used sound card output, unused outputs in between will be filled with silence. To enable recording, set `record-output = true` and `recording-file = convolution_result.wav`. If you want the convolver to exit after recording a fixed number of seconds, set `recording-length = 120` and `stop-after-record = true`.

## 5.2 Partitioned FFT Convolution

To be able to use FFT based convolution without adding latency to the signal itself, `convolver.exe` uses partitioned convolution. The IR is sliced into multiple segments, each of which is handled by an FFT that can be at most as long as the position in the IR to ensure that enough time has passed to fill the buffer for the FFT. The first segment of the IR is covered using direct form convolution since there is no time to fill a buffer.

To configure a partitioned convolution you use a string of the format:

```
{Direct Form Length}--{Num FFT Blocks}x{FFT Length}(:{Num Threads})-...
```

For example:

```
256-6x128-6x256-0x1024:3
```

This translates to: *Cover the first 256 samples using direct form convolution, the next 768 samples using 6 blocks of FFT128, the next 1536 samples using 6 blocks of FFT256 and all remaining samples by as many blocks of FFT1024 as required, computed on 3 threads.*

The caveat of partitioned convolution is that the IR is only updated once every FFT blocksize. This means that an update that changes parts of the IR spanning multiple FFT blocks of different blocksize (or an FFT and direct-form block) is not applied synchronously, which can lead to audible artifacts, especially if an impulse moves from one block to the other and is either represented twice or not at all during this transition period. The convolver will print a warning when applying an update of this nature and you should alter your partitioning settings in this case.

We recommend you use partitioning schemes involving multiple FFT blocksizes only with IRs that do not change <sup>1</sup>. When convolving an IR that is computed segment-wise by multiple `rtsofe.exe` instances, we recommend the following setup:

- Early Reflections use `part-scheme = FIR`
- Later Reflections use `part-scheme = 1024-0x1024`. You need to ensure that the first 1024 samples of that IR segment will not contain any information to prevent artifacts due to the different update speeds between the direct segment and the FFT1024. Reduce 1024 until this condition holds.
- Late Reflections with a static IR use whatever partitioned convolution scheme you desire.

---

<sup>1</sup>or in the *highly unlikely* event that you can guarantee that updates will only ever concern areas of the IR covered by a single FFT blocksize

### 5.3 Application States and OSC Control

By default, the convolver will listen to OSC messages on UDP port 7000 for playback commands and configuration changes. The port can be changed by setting `osc-port = 7001`. The convolver behaves as a state machine, as illustrated in figure 2.

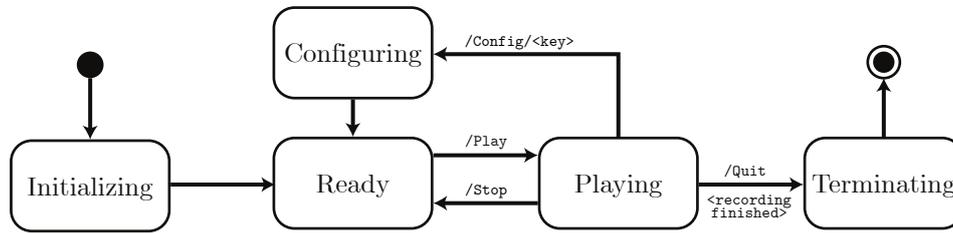


Figure 2: State diagram of the convolver

After initialization, the convolver waits for a `/Play` to start playback, which includes both playback of signals from files as well as inputs from the sound card. Playback can be stopped using `/Stop` or by sending a configuration change using `/Config/ConfigurationKey`. In the latter case, the argument will be used as the new value for the configuration key, the corresponding objects re-initialized using the new settings and the convolver goes back to waiting for a `/Play`. It will quit on `/Stop` or if `stop-after-record = true`.

If `osc-controlled = false` this functionality is not available. The convolver goes directly to the playback state and only quits because of `stop-after-record = true` or a system interrupt.

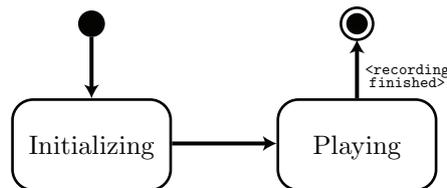


Figure 3: State diagram of the convolver without OSC control

### 5.4 Troubleshooting

- **xyz can't be set twice.** Make sure that you have added numbers (-2, -3) for all keys, e.g.:

```
input-from-file = true
input-file = sandstorm.wav
```

```
input-from-file-2 = true
input-file = the_lick.wav # !!! should be input-file-2 !!!
```

- **the output sounds horribly choppy/distorted and audio files play too fast/slow.** Make sure you have set the same sample rate and buffer size in the convolver config file and your sound card's setup. If this does not help and/or audio is also distorted in other applications, try turning you computer off and on again.

## 6 OSC Tools – Python and Matlab utilities

To obtain real-time receiver position and transformation from a OpenVR compatible VR headset, the script `openvr-poses-osc-send.py` is provided. It has been tested with Steam VR software and an HTC Vive pro eye VR headset. This script also serves as a template for building OSC bundles, which contains one or more OSC commands. All parameter updates in the same OSC bundle will be processed to produce the next IR update.

To send OSC commands to `rtsofe.exe` and `convolver.exe` for quick testing, two python scripts are available in `Y:/resources/rtSOFE/OSC_Python_Utils`. They are based on a script by Niklas Löcherer.

`osc_command_sender.py` sends text from stdin as OSC messages to a single UDP socket. User input is split at spaces, the first word is used as the messages address and all others as arguments. They are converted into float or int if possible. The words 'true' or 'false' (any capitalization) are converted to the integers 1 and 0. For example:

```
python3 osc_command_sender.py --ip 192.168.0.7 --port 54321
>>> \Play
>>> \Config\InputFile sine_1khz.wav
>>> \Example\Command 1.0 5 false sometext
```

`osc_position_sender.py` connects to one or two instances of `rtsofe.exe` and sends source positions to them. If the argument `--manual` is given, they can be entered by the user:

- Three digits are sent as xyz coordinates
- Two digits are interpreted as azimuth and elevation and are sent as a vector of length `--radius` pointing in that direction
- One digit is interpreted as azimuth, elevation defaults to  $0^\circ$ .
- No digits are interpreted as azimuth and elevation =  $0^\circ$ .

If `--manual` is not given, positions are automatically sent along a geometrical path. `--rate` sets the updates per second, `--velocity` the speed in meters per second. `--trajectory` selects between `line` and `circle`. The line goes from `--point-a` to `--point-b`, the circle is centered around `--point-a` and has radius `--radius`. The circle repeats `--revolutions` times. To pass vectors, write all components into the command line without brackets or commas. Scientific notation is supported:

```
python3 osc_position_sender.py --point-a 6.0 0.9e1 1.8
```

MATLAB scripts are provided for interaction with `rtSOFE` running in interactive mode. The scripts contains MATLAB functions that performs the following tasks:

- `mk_osc_msg.m`: Make OSC messages
- `mk_osc_bundle.m`: Make OSC bundles from messages and bundles
- `read_mcIR_from_stream.m`: Read the multi-channel IR from bitstream

MATLAB scripts provided under e.g. `rtSOFE_1.1/ExampleConfiguration/Configurations/config03` demonstrates how to use the functions above. A list of OSC address patterns is provided in the appendix.

## 7 License

Copyright (c) 2015-2021, Bernhard Seeber and the `rtSOFE` development team at the Professorship for Audio Information Processing (AIP), Technical University of Munich, Germany.

`rtSOFE.exe v1.1` and `convolver.exe v1.1` is free software: you can redistribute it and/or modify it under the terms of the GNU General Public License as published by the Free Software Foundation, version 3 of the License. You should have received a copy of the GNU General Public License along with this program. If not, see <https://www.gnu.org/licenses/>. The source code of v1.1 is available on written request to [aip@ei.tum.de](mailto:aip@ei.tum.de).

Auxiliary files developed at the Professorship for Audio Information Processing, like Python and Matlab tools to control `rtSOFE`, the configuration files for `rtSOFE`, and the example files demonstrating different use cases of `rtSOFE.exe` and `convolver.exe`, are licensed under a Creative Commons Attribution 4.0 International (CC BY 4.0) License, see <https://creativecommons.org/licenses/by/4.0/> for more information. This allows you to freely adapt them to your application.

Libraries re-distributed in the `rtSOFE` package have their own license terms which are reproduced in the respective `*.rights-file`.

This program is distributed in the hope that it will be useful, but WITHOUT ANY WARRANTY; without even the implied warranty of MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE. See the GNU General Public License for more details.

IN NO EVENT SHALL THE COPYRIGHT OWNER OR CONTRIBUTORS BE LIABLE FOR ANY DIRECT, INDIRECT, INCIDENTAL, SPECIAL, EXEMPLARY, OR CONSEQUENTIAL DAMAGES (INCLUDING, BUT NOT LIMITED TO, PROCUREMENT OF SUBSTITUTE GOODS OR SERVICES; LOSS OF USE, DATA, OR PROFITS; OR BUSINESS INTERRUPTION) HOWEVER CAUSED AND ON ANY THEORY OF LIABILITY, WHETHER IN CONTRACT, STRICT LIABILITY, OR TORT (INCLUDING NEGLIGENCE OR OTHERWISE) ARISING IN ANY WAY OUT OF THE USE OF THIS SOFTWARE, EVEN IF ADVISED OF THE POSSIBILITY OF SUCH DAMAGE.

When publishing work using rtSOFE, please cite:

Seeber, B.U., Kerber, S., and Hafter, E.R. (2010). *A System to Simulate and Reproduce Audio-Visual Environments for Spatial Hearing Research*, *Hearing Research* 260, 1-10.

...or a subsequent technical publication about rtSOFE (currently in preparation). Thanks!

## 8 List of Contributors

rtSOFE development is based on the Matlab version of the room acoustic simulation SOFE by Bernhard Seeber (Hafter and Seeber, 2004; Seeber et al., 2010), which is conceptually identical to rtSOFE in that it follows the geometrical acoustics approach with backtracking and visibility checking Borish (1984). rtSOFE is the C++ implementation of SOFE for real-time simulation and rendering. Many students and team members have worked with Bernhard to develop rtSOFE. All contributed in one way or another to this version, through helping our understanding or contributing code. The following is intended to be a complete list of contributors and contributions, but please bear with me if I have overlooked someone or some contributions and let me know.

### AIP team members

Bernhard U. Seeber	Simulation and programming concepts, project management, supervision, development, documentation, debugging and system test pipeline, creation of the Matlab version SOFE
Tong Wang	Development incl. binaural rendering, use and test cases, documentation
Clara Hollomey	Documentation, development, supervision
Samuel W. Clapp	Initial Ambisonics rendering, supervision

### AIP students

Manuel Hornung	Development first rtSOFE version, real-time code and modularization concepts, documentation
Niklas Löcherer	Development testing pipeline, test cases, debugging
Hendrik Nöller	Verification and debugging, data exchange formats, testing pipeline, test cases, documentation
Max Kirchmeier	Development modularization of convolver, extension to multiple sound sources and overlapping impulse responses
Sebastian Pods	Development initial version real-time convolver
Giorgio Fabbro	Code review
Felix Enghofer	Parallel rendering concept
Julian Fährmann	Convolver approaches
Markus Grabichler	Backtracking aspects.

## 9 Funding Acknowledgment

The development of rtSOFE was funded by the BMBF Bernstein Center for Computational Neuroscience, BMBF 01 GQ 1004B, and the Deutsche Forschungsgemeinschaft (DFG, German Research Foundation) – Projektnummer 352015383 – SFB 1330 C5.

# Appendices

## A Material properties file

A material properties file contains a list of frequency dependent reflection factors, each one associated with a material name. The reflection factors are used by `rtsofe.exe` to compute the reflection spectrum. Please note that the reflection factors should be given in the amplitude domain, while reflection, absorption and scattering coefficients are usually given in the energy domain.

The frequency grid points are given inside the file.

Materials files are text files containing comments, one or more lines defining the frequency grid points, and one or more lines containing one material each.

Comments are prefixed with `#` and span until the next newline. Lines beginning with `#` are ignored entirely, lines containing a `#` in the middle will only be evaluated up to the `#`.

Lines starting with `frequencies` define the frequency grid points for all following materials until the next line beginning with `frequencies`. They contain a list of frequencies in increasing order separated by one or multiple space and/or tab characters. For example:

```
frequencies 125 250 500 1000 2000 4000 8000 20000
```

It is recommended to define frequencies up to 20 kHz; the lowest and highest defined value is repeated to zero and Nyquist frequency, respectively. Lines starting with anything else than `#` or `frequencies` define a material. The text until the first tab or space character is interpreted as the name which is followed by the reflection factor at the frequencies given by the most recent `frequencies` line. The number of reflection coefficients must match the number of frequencies. For example:

```
plywood 0.84 0.88 0.91 0.95 0.94 0.94 0.94 0.94 # 3/8" Plywood paneling
```

The following is an example of a valid materials file:

```
# Example materials file
# All numbers are REFLECTION FACTORS
# They are also purely fictional, please don't use them

frequencies      2000  5000  10000  20000

fresh-bread      0.99  0.98  0.81  0.74
stale-bread      0.99  0.98  0.85  0.83 #five days old

frequencies      1000  2000  5000  10000  20000
vegetable-lasagna 0.67  0.52  0.42  0.24  0.25
```

## B Source directivity file

Directivity files contain a number of frequency dependent attenuations in dB each associated with a direction given in spherical coordinates. They are used to inform `rtsofe.exe` on how to attenuate sound radiating from the source to different directions. The frequency grid points are given inside the file. Modeled to the data gathered in Flanagan (1960), the format only allows for directions that move along azimuth or elevation with the other fixed at 0°. The format is similar to the materials files explained in appendix A.

Directivity files are text files containing comments, one or more lines defining the frequency grid points, and one or more lines containing samples along azimuth and elevation.

Comments and frequency line work in exactly the same way as materials files.

Lines starting with `az` or `e1` define a sample in a direction. `az` or `e1` defines whether the angle is along azimuth or elevation, followed by an angle and the attenuations at the frequencies given by the most recent `frequencies` line. The number of attenuations must of course match the number of frequencies. For example:

```
az 42 0.0 0.5 0.0 -0.5 0.25
```

The following is an example of a valid directivity file:

```
# Example directivity
# All numbers are GAINS IN DB (AMPLITUDE)
# They are also PURLEY FICTIONAL

frequencies 250 500 1000 2000
az 0 0.0 0.0 0.0 0.0
az 90 0.0 1.2 0.5 0.5
az 180 1.0 0.0 -0.5 -1.0
az 270 0.0 1.2 0.5 0.5
el 0 0.0 0.0 0.0 0.0
el 90 0.0 -0.5 -0.5 -0.5
el 180 1.0 0.0 -0.5 -1.0
```

It is advisable to ensure azimuth and elevation have a point at  $0^\circ$  and  $180^\circ$  each with identical values since the axes intersect at those angles. If not, interpolation might produce unpredictable results around those points. Values at  $0^\circ$  are usually unity gain. Levels are formally defined at 1 m distance (unity gain for 1/r-law).

## C Speaker mapping file

Speaker mappings contain a list of loudspeaker positions and soundcard output channels. They are stored as an  $n \times 4$  matrix inside a MAT-file. MAT-files are binary files containing MATLAB workspace variables (MathWorks, 2021). The rtSOFE applications use a shared library included with MATLAB to read these files.

Since the mapping is contained in a single variable, one MAT-file may contain multiple speaker mapping variables and each application therefore needs to be provided a file name and a variable name to read a speaker mapping. This enables the use of a single MAT-file as a collection of speaker mappings or the inclusion of a speaker mapping in other files (such as ambisonics LUTs, see appendix D) for cross-verification purposes.

Each speaker mapping is a  $n \times 4$  matrix containing four columns and as many rows as loudspeakers. The columns contain *azimuth*, *elevation*, *soundcard output* and *index*. *Index* always needs to equal the index of the row and is used to provide some protection against accidentally or ineptly altered speaker mappings. The applications will refuse to load a file if the index does not correspond to the row. An example is given in table C.

Azimuth	Elevation	Output	Index
-110	0	5	1
-30	0	1	2
30	0	2	3
110	0	6	4
0	0	3	5

Table 1: An exemplary speaker mapping. Note how soundcard outputs are shuffled and are non-contiguous. Column headings are not part of the MAT-file.

## D Ambisonics LUT file

Ambisonics LUTs contain the pre computed weighting coefficients necessary to perform 2D Ambisonics panning along the horizontal plane. There is one coefficient per channel for each angle in the range of  $[0, 360^\circ[$  with a step size of  $0.1^\circ$ . The coefficients are stored in a MAT-file, a binary file containing MATLAB workspace variables (MathWorks, 2021).

The MAT-File contains three variables:

- `speaker_mapping` contains the speaker mapping the LUT was computed for. This variable needs to follow the format described in appendix C.

- `LUT` is an  $n \times 3600$  matrix containing the weighting coefficients. Lines represent IR channels and are ordered according to `speaker_mapping`.
- `azim_s_degree` is a  $1 \times 3600$  vector containing the angles in degrees that the columns of `LUT` correspond to. This vector is currently required to be exactly equal to `0:0.1:359.9`, no other grids are supported.

## E Impulse response file

IR files contain one or several pre-computed multichannel IRs for use with `convolver.exe`. They are useful for rendering measured IRs, for hybrid rendering of `rtSOFE`-simulated (early) reflections with offline-simulated or measured late reverberation or for rendering IRs transitions at pre-defined times to simulate trajectories or room changes. If multiple IRs are given, a time interval needs to be supplied that dictates how often a new IR is loaded. They are stored in a MAT-File, a binary file containing MATLAB workspace variables (MathWorks, 2021).

Data is stored in a variable named `impulse_response` that either has a shape of `points`  $\times$  `channels` if a single IR is given or `points`  $\times$  `channels`  $\times$  `steps` if multiple IRs are given. If the latter is the case, a second variable `update_interval` of type `uint64` should be supplied that contains the interval between IR updates in microseconds ( $10^{-6}$  seconds).

## F Equalization filters file

Equalization (EQ) filters files contain a set of FIR filters for equalizing individual loudspeaker or head-phone channels with a fixed filter. The convolver output of each channel is filtered with its respective filter and an integer delay is applied to allow for amplitude spectrum, phase and delay correction, the latter without the computational burden that convolving leading zeros in the filters would cause. The filters are stored in a MAT-File (MathWorks, 2021).

The MAT-File must contains three variables:

- `speaker_mapping` contains the speaker mapping the filters were measured for. This variable exhibits the format described in appendix C.
- `eq_filters` is an  $n \times L$  matrix containing the  $L$  filters of length  $n$ . The columns are ordered according to the lines in `speaker_mapping`, NOT the hardware outputs. The variable name may vary (option: `--eq-filters-var`).
- `index_shift` is a  $1 \times L$  vector containing the integer delays. The columns are ordered according to the lines in `speaker_mapping`, NOT the hardware outputs. The variable name may vary (option: `--eq-filters-shift-var`).

## G Receiver directionality SOFA files

SOFA-files are used to define receiver directionality, i.e. the head-related transfer functions needed for binaural rendering. `rtSOFE` uses the FIR type of a SOFA file as defined by AES69-2020 (Majdak and Noisternig, 2020). Additional requirements can be found in section 4.2.

## H OSC address patterns and parameters

This section lists parameter names (as in configuration files), OSC address patterns, number and type of arguments for `rtsofe.exe` and `convolver.exe` that can be changed through OSC messages.

## H.1 rtsofe.exe

Parameter Name	Address Pattern	Arguments	Note
source-pos	/source-pos	3, float	in x,y,z order
receiver-pos	/receiver-pos	3, float	in x,y,z order
source-transform	/source-transform	9, float	rotation matrix, col. major order
receiver-transform	/receiver-transform	9, float	rotation matrix, col. major order
order	/order	1, float	
max-invis-parents	/max-invis-parents	1, float	
max-distance	/max-distance	1, float	
attenuation-threshold	/attenuation-threshold	1, float	
max-sources	/max-sources	1, float	
(none)	/exit	(none)	

## H.2 convolver.exe

Parameter	Address Pattern	Default	Description
(none)	/Play	(none)	start playing
(none)	/Stop	(none)	stop playing
(none)	/Quit	(none)	stop the convolver process
buffer-size	/Config/Init/BufferSize	32	Buffer size (samples)
channel-assignments	/Config/Init/ChannelAssignments		Specifies output channels of the sound card to be used. Add multiple of these in order of the IR channels. Overwrites channel-assignments-file and channel-assignments-name
channel-assignments-file	/Config/Init /ChannelAssignmentsFile		.mat file to read channel assignments from. Looks for a matrix with the name given in channel-assignments-var which represents each speaker as a row. The columns are azimuth, elevation, playback channel, id. Channels can be selected freely with arbitrary gaps, as long as they are physically present on the soundcard. This determines the number of channels the convolver is operating on. You should always load the same file as speaker_mapping in rtSOFE. Can be overwritten with by channel-assignments.
channel-assignments-var	/Config/Init/ChannelAssignmentsVar		Name of the variable to read from ChannelAssignmentsFile. For more info see channel-assignments-file. Can be overwritten by channel-assignments
collect-durations	/Config/Init/CollectDurations	0	Whether to collect measured durations; necessary for 99th-percentile calculation but causes memory growth and (miniscule) slowdown
device	/Config/Init/Device	(None)	Device name
eq-filters-file	/Config/Init/EQFiltersFile		MAT-file holding loudspeaker equalization filters.
eq-filters-on	/Config/Init/EQFiltersOn	0	Whether to apply loudspeaker-equalization
eq-filters-shift-var	/Config/Init/EQFiltersShiftVar	index_shift	Variable containing the additional integer delays for the filters. Their order is assumed to correspond to eq-filters-filter-var
eq-filters-var	/Config/Init/EQFiltersVar	eq_filters_ampl	Variable containing the filters. Their order is assumed to correspond to IR Channels (LS IDs), NOT hardware channels
eval	/Config/Init/EvalMode	0	Whether to run convolver for 'eval-time' and quit
eval-time	/Config/Init/EvalTime	10	How long to run the convolver in eval-mode for.
exp-update-freq	/Config/Init/ExpectedUpdateFreq	120	Expected frequency of IR updates, for timing measurement

Parameter	Address Pattern	Default	Description
log-level	/Config/Init/LogLevel	info	Log level
num-convolvers	/Config/Init/NumConvolvers	1	Number of separately run convolver-instances (for early/late reflections, or multiple sound sources)
num-signals	/Config/Init/NumSignals	1	Number of audio sources
record-output	/Config/Init/RecordOutput	0	Whether to record the output. Channels in the output file will be ordered according to ChannelAssignments and include silent channels if there are unused soundcard channels in between.
recording-file	/Config/Init/RecordingFile	out.wav	Filename for recording output
recording-length	/Config/Init/RecordingLength	0	How much of the output should be recorded (in seconds)
sample-rate	/Config/Init/SampleRate	44100	Sampling frequency (Hz)
stop-after-record	/Config/Init/StopAfterRecord	1	Whether to stop the convolver after recording is finished
warn-on-wait	/Config/Init/WarnOnWait	0	Whether to print a warning when one of the FFT-pipelines has to wait for input or a free output slot.
const-ir-offset	/Config/Init/ConstIROffset	0	Start Const IR at offset x
input-signal	/Config/Init/InputSignal	1	Index of the input signal the convolver is connected to (one based)
inter-fade	/Config/Init/InterFade	0	Overlap-fading <i>between</i> segments to touch up segments updating asynchronously
intra-fade	/Config/Init/IntraFade	32	Fade length <i>within</i> each segment whenever an IR changes. When using a partitioning scheme (preset) that involves FFTs this cannot exceed the smallest FFT length.
ir-file	/Config/Init/IRFile		IR file name (.mat). Contains a 3D matrix impulse_response of shape length x channel x timestep and a uint64 called update_interval which dictates the microseconds (1/1000000 second) until the ir for the next timestep is loaded
ir-length	/Config/Init/IRLength	0	IR length; determines length of File/Const IRs and truncates IRs received over network
ir-type	/Config/Init/IRType	0	Type of IR-source: TCPMatrix, File
pad-channels	/Config/Init/PadChannels	0	Whether to repeat a lower-channel File-IR to ChannelAssignments.size()
part-scheme	/Config/Init/PartScheme	FIR	Partitioning scheme. Overwritten by part-scheme-preset
part-scheme-preset	/Config/Init/PartSchemePreset	-1	Partitioning scheme index (1-3: schemes from old convolver). If in valid range overwrites part-scheme
repeat-ir	/Config/Init/RepeatIR	0	Whether to repeat Const/File IRs after they're finished
rtsofe-host	/Config/Init/RTSoFeHost		Hostname for rt_soFe
rtsofe-port	/Config/Init/RTSoFePort	0	First listening port for TCP connections to receive impulse responses from rt_soFe

## References

- Borish, J. (1984). Extension to the image model to arbitrary polyhedra. *The Journal of the Acoustical Society of America*, 75(6):1827–1836.
- Flanagan, J. (1960). Analog measurements of sound radiation from the mouth. *The Journal of the Acoustical Society of America*, 32(12):1613–1620.
- Hafter, E. and Seeber, B. (2004). The Simulated Open Field Environment for auditory localization research. In *Proc. ICA 2004, 18th Int. Congress on Acoustics, Kyoto, Japan, 4.-9.04.2004*, volume V, pages 3751–3754. Int. Commission on Acoustics.
- Majdak, P. and Noisternig, M. (2020). *AES69-2020: AES standard for file exchange – Spatial acoustic data file format*. Audio Engineering Society, Inc.
- MathWorks (2021). MAT-file Versions. [https://www.mathworks.com/help/matlab/import\\_export/mat-file-versions.html](https://www.mathworks.com/help/matlab/import_export/mat-file-versions.html). Accessed on 2021-11-04.
- Seeber, B., Kerber, S., and Hafter, E. (2010). A system to simulate and reproduce audio-visual environments for spatial hearing research. *Hearing Research*, 260(1-2):1–10.