TASCAR

Toolbox for Acoustic Scene Creation And Rendering

User Manual

G. Grimm, J. Luberadzka, F. Schwark, T. Herzke, V. Hohmann:
TASCAR – User Manual

Carl von Ossietzky Universität Oldenburg
Marie-Curie-Str. 2
D–26129 Oldenburg

TASCAR version 0.234.2.0-58f0188 (February 16, 2025)

Contents

Preface

This user manual is a work in progress, just like the entire TASCAR toolbox. We welcome your feedback: please submit bug reports and suggestions, such as improved documentation for specific features, directly to the TASCAR author through our GitHub issues tracker at https://github.com/gisogrimm/tascar/issues.

1 Introduction

The TASCAR toolbox is designed for the creation and rendering of virtual acoustic environments (Grimm et al., 2015, 2016, 2019). With TASCAR, users have the capacity to construct virtual acoustic 'scenes', which can be rendered in real-time and experienced through almost any sound playback system.

Notably, these acoustic scenes can be manipulated and explored interactively by the user in real-time, such as through the use of headphones and a joystick for directional control within the acoustic space. Both direct sound paths and image sources, created through a geometrical image source model, can be rendered dynamically.

However, it's essential to clarify that TASCAR is not intended to function as a high quality room acoustics simulator. Rather, its aim is to offer a rapid and perceptually credible approach for representing virtual acoustic environments in real-time. TASCAR is adept at creating dynamic, interactive environments suitable for a range of applications, from hearing aid development and assessment, adaptive changes in spatial configuration psychophysics, soundscape simulation, to computer games.

In its simplest form, an acoustic scene consists of three types of objects: Sound sources, a receiver and reflectors. Each of these objects occupies a specific position and orientation within the virtual space at a specific time. To recreate the effect of a moving object, the position or orientation of the object can be changed over time.

The position of the receiver corresponds to the point in the virtual space at which the simulation is rendered. This rendering depends on the direction of incidence relative to the orientation of the receiver. To simulate the sound field, various acoustic phenomena such as reflections, air absorption or diffraction are simulated. A comprehensive discussion of these acoustic simulation methods can be found in the second chapter.

2 General remarks and invocation

TASCAR is primarily developed and tested on Linux. TASCAR is provided as a Debian package for long-term stable versions of Ubuntu Linux. For MacOS, TASCAR can be installed via the 'homebrew' system. A binary version for Microsoft Windows can be found on the github release page. Further information on installation can be found at https://tascar.org/ or on the GitHub wiki pages at https://github.com/gisogrimm/tascar/wiki.

Table 1 provides a list of the most important installation directories of the Linux version.

/usr/share/doc/tascar	documentation and user manual
/usr/share/tascar/examples	example files
/usr/share/tascar/matlab	tools for MATLAB and GNU Octave
/usr/share/tascar/python	tools for python/blender

Table 1: List of relevant TASCAR directories on Linux installations.

After successful installation of the packages, TASCAR is available as the command `tascar` or from the main applications menu, in the “sounds and video” section.

TASCAR relies heavily on the jack audio connection kit (http://jackaudio.org). It is necessary to start jack before loading a session into TASCAR. TASCAR will attempt to start `qjackctl` if the jack server is not running. This behaviour can be disabled by adding an entry to the GUI section of the configuration file (see Section 2.4 for details):

```
<tascar>
  <gui>
    <checkforjack data="0"/>
  </gui>
  ...
</tascar>
```

Jack port names in TASCAR are treated as POSIX regular expressions (Goyvaerts, 2019). If the `^` or `$` anchors are not present at the beginning or end of an expression, they will be added at the beginning and end of the expression to achieve proper full name matching. Please note that some characters such as `.[?()` (and a few others) have special meaning as regular expressions and must be quoted for a correct match.

When using TASCAR with jackd1, memory locking may fail, resulting in the TASCAR process being killed. If you see this behaviour, either run jack without memory locking (using the `-m` flag), or use jackd2 instead.

An acoustic environment can be loaded either from the command line by specifying the filename, or from the File menu in the main window. If loading a session file fails for some reason, it may be helpful to run it from the command line to see additional information.

TASCAR uses SI units unless otherwise specified. Developers of new plugins are encouraged to use SI units for all internal and configuration variables.

2.1 Keyboard shortcuts in the main window

The TASCAR GUI can be controlled using keyboard shortcuts.

Changing the view in the main window between the elements of the menu bar on the left side:

Alt+1	Map
Alt+2	Mixer
Alt+3	XML source
Alt+4	OSC variables
Alt+5	Licenses
Alt+6	Warnings

See Figure 1 for examples of the different views.

Opening the top menu bar elements:

Alt+F	File
Alt+T	Transport
Alt+V	View
Alt+H	Help

Opening and closing TASCAR files:

Ctrl+N	Open new TASCAR scene
Ctrl+O	Open a TASCAR file
Ctrl+X	Open an example TASCAR file
Ctrl+R	Revert to previous scene
Ctrl+W	Close TASCAR scene
Ctrl+Q	Quit the program

Controlling the transport of a TASCAR scene:

Space	Play
Ctrl+Space	Stop
Page Up	Rewind
Page Down	Forward
Home	Previous
End	Next

To show more information about an element of the scene, it can be selected in the Map view (Alt 1) by clicking on the origin of the object. The gain controller and the corresponding part of the XML source code will be displayed on the right side of the window, allowing to switch between the elements by using the drop-down menu in the top right corner. There is also an option to track the selected object in the scene map.

2.2 Network remote control via OSC

The majority of options in TASCAR can be controlled remotely via the Open Sound Control (OSC) protocol. An OSC message comprises one or more numeric values or strings. Each message can be transmitted to an OSC path on an OSC server. TASCAR supports both UDP and TCP transport layers for the receipt of OSC messages. Should the necessity arise for the utilisation of more than one transport layer or multiple ports, the “oscserver” module may be employed (for further details, please refer to section 6.25). Additionally, a MATLAB/GNU Octave remote control tool is available (for further details, please refer to section 11.5).

A list of OSC variables with their type and, in some cases, documentation, typi-

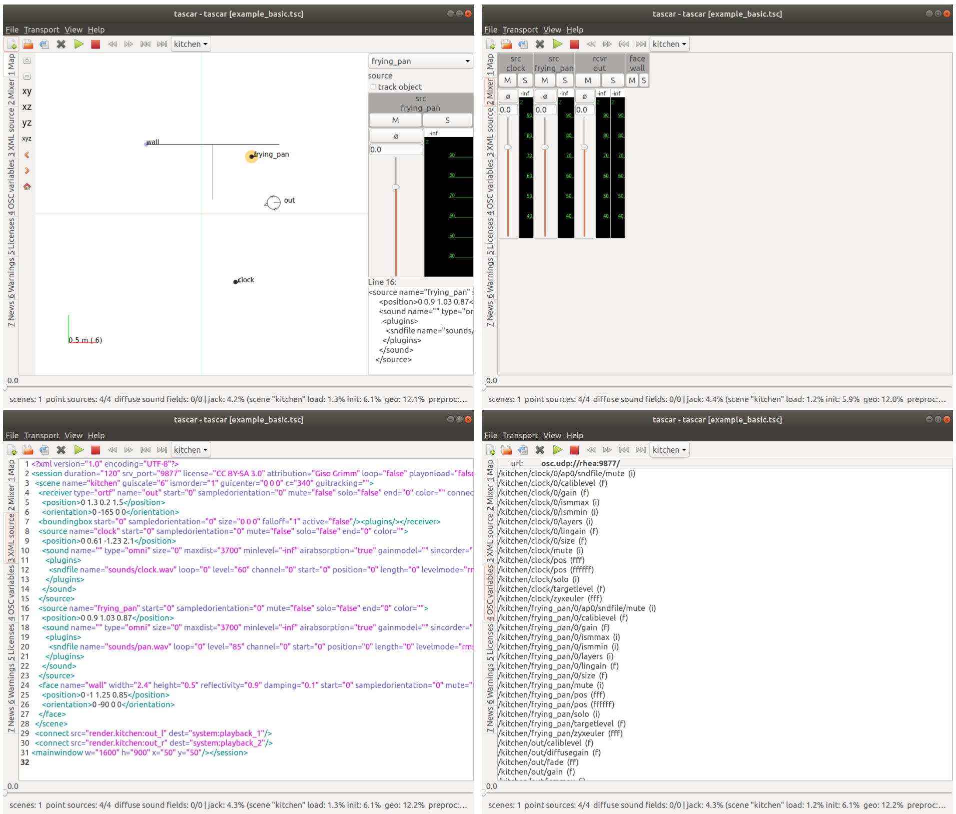


Figure 1: Example of some main window tabs in TASCAR.

cal range and current value can be read using the special variable `/sendvarsto` This OSC variable takes three string parameters: The first is a valid OSC URL, e.g. `osc.udp://localhost:9000/`. The variable list is sent to this address. The second parameter is an OSC path to send the variable list to, e.g. `/getvar`. The third parameter, which is optional, is a prefix. If specified, only variables starting with that prefix will be reported.

When a message to `/sendvarsto` arrives, first an empty message is sent to `<path>/begin`. This is followed by multiple messages of format `ssiss`, one message per matching OSC variable. The parameters are OSC path (`s`), typespec (`s`), indicator if the current value is readable (`i`), a value range hint (`s`) and a help comment (`s`). At the end of the list, an empty message is sent to `<path>/end`.

The current XML configuration can be retrieved by sending a URL and path to the `/sendxmlto` variable. This OSC variable requires two string parameters: The first is a valid OSC URL, e.g. `osc.udp://localhost:9000/`. The XML configuration is sent to this address. The second parameter is an OSC path, e.g. `/xml`. This destination should be able to receive string data (format `s`).

Please note that in some cases the maximum transmission unit of UDP messages is not sufficient to transmit all data. In this case you should use TCP transport.

It is also possible to read OSC variables from a file. This can be achieved either through the XML variable `initoscscript`, or the OSC variable `/runscript`. In this case, every non-empty line that does not begin with `#`, `@`, `<`, or `,`, is interpreted as an OSC message. The initial element of a space-separated list of words or numbers represents the path. All subsequent elements are converted to numeric floats if feasible, whereas those that are not are transmitted as strings. If a line commences with a comma `,`, the number following the comma is interpreted as the time in seconds to await the next line's processing. Lines that commence with a hashtag `#` and those that are empty are disregarded. Lines that commence with the "at" symbol `@` indicate the presence of a "timed message." In this context, the numerical value immediately following the `@` symbol represents the session time at which the subsequent message is dispatched.

In the event that a space-separated list of filenames (optionally quoted to include filenames with spaces) is specified instead of a single filename, all specified files are processed in sequence. It should be noted that if the XML attribute `scriptcancel` is set to "true", the execution of other scripts will be aborted if a script is initiated while other scripts are still being processed. Otherwise, the scripts will be appended. Furthermore, if the filename does not commence with an absolute path, the session attribute `scriptpath` will be used as a prefix. The default file extension for TASCAR is OSC scripts, which are designated by the extension ".tosc". With the session attribute `initoscscript`, an OSC script can be specified which will be run after loading a session. It is important to note that the `/runscript` OSC command cannot be used to read nested OSC script files. Instead, a line containing `<filename` should be written into the script file at the position where the nested file `filename` is to be read. The variables `scriptpath` and `scriptext` will be prefixed and appended to the filename. It is required that there are no leading or trailing spaces in the line that begins with <.

2.3 Optimization of the operating system for audio processing

In multi-user desktop systems, it is important to assign real-time priority to the signal processing threads of audio software. To set up real-time scheduling on the system, users in the 'audio' group must be granted permission to acquire real-time priority and lock memory in RAM. To do this, edit the `/etc/security/limits.conf` file with superuser privileges:

```
sudo gedit /etc/security/limits.conf
```

Add these two lines if they are not already present:

```
@audio - rtprio  99
@audio - memlock unlimited
```

Now add the desired user of TASCAR to the 'audio' group (in this example this user is called 'tascar'):

```
sudo adduser tascar audio
```

Log out and log in again (typically, no re-boot is required).

If you are aiming for low-latency processing, you should further optimize your system by installing a low-latency kernel and an IRQ priority management tool:

```
sudo apt install linux-lowlatency rtirq-init
```

When you start Jack, you should choose a realtime priority that is slightly lower than the priority of the interrupt handler of the selected sound card. This can be checked with the `tascar_testrtprio` tool in the console. A sample output may look like the following:

```
       CPU0   CPU1   CPU2    CPU3
   0:    13      0      0       0   IO-APIC    2-edge      timer
   8:     0      0      0       1   IO-APIC    8-edge      rtc0
   9:     0      0      0       0   IO-APIC    9-fasteoi   acpi
  16:     0      0      0       0   IO-APIC   16-fasteoi   i801_smbus
  18:     0      0      0      54   IO-APIC   18-fasteoi   snd_hdsp
 127:     0      0      0  593043   PCI-MSI 327680-edge    xhci_hcd
 128: 19074      0      0       0   PCI-MSI 376832-edge    ahci[0000:00:17.0]
 129:     0    342      0 2002167   PCI-MSI 4194304-edge   enp8s0
 130:     0     41      0       0   PCI-MSI 360448-edge    mei_me
 131:     0      0      0       0   PCI-MSI 2097152-edge   rtl_pci
 132:     0      0      0       0   PCI-MSI 514048-edge    snd_hda_intel:card3
 133:     0 297116     98       0   PCI-MSI 524288-edge    nvidia
RTPRIO CLS %CPU COMMAND
     5 RR   0.0 /usr/bin/pulseaudio --daemonize=no --log-target=journal
    50 FF   0.0 [idle_inject/0]
    50 FF   0.0 [idle_inject/1]
    50 FF   0.0 [idle_inject/2]
    50 FF   0.0 [idle_inject/3]
    50 FF   0.0 [irq/9-acpi]
    50 FF   0.0 [watchdogd]
    50 FF   0.0 [irq/8-rtc0]
    50 FF   0.0 [irq/16-i801_smb]
    50 FF   0.0 [irq/128-ahci[00]
    50 FF   0.0 [irq/130-mei_me]
    50 FF   0.0 [irq/131-rtl_pci]
    50 FF   0.1 [irq/133-nvidia]
    50 FF   0.2 [irq/133-s-nvidi]
    70 FF   0.1 [irq/127-xhci_hc]
    80 FF   0.0 [irq/132-snd_hda]
    85 FF   2.0 /usr/bin/jackd --sync -P85 -p4096 -m -dalsa -dhw:hdsp -r44100
    -p64 -n2
    90 FF   0.0 [irq/18-snd_hdsp]
    99 FF   0.0 [migration/0]
    99 FF   0.0 [migration/1]
    99 FF   0.0 [migration/2]
    99 FF   0.0 [migration/3]
    99 RR   0.0 /usr/libexec/rtkit-daemon
```

Here you can see that the interrupt handler `irq/18-snd_hdsp` runs with a priority of 90, while the real-time thread of `jackd` has been configured to run with a priority of 85. This prevents the signal processing thread from interrupting communication between the sound card and the operating system. All other devices connected to this computer will run with a lower priority.

CPU frequency scaling

CPU frequency scaling can cause dropouts in audio signal processing when switching between different processor clock speeds. Therefore, disable CPU frequency scaling in the BIOS, or manually switch the CPU to maximum performance after each login, e. g., with

```
for c in {0..11}; do cpufreq-selector -c $c -g performance; done
```

or with the TASCAR provided wrapper `tascar_cpufreq`. If this doesn't work please use the CPU frequency scaling indicators of your desktop manager, or with

```
echo "performance" | sudo tee /sys/devices/system/cpu/cpu*/cpufreq/scaling_gove
```

2.4 Overwriting application default values

Some variables which do not directly affect the acoustic rendering result, e.g., GUI parameters and configuration of the loudspeaker calibration tool, have built-in default values. These values can be overwritten using an external application configuration file in XML format. The files `/etc/tascar/defaults.xml` and `${HOME}/.tascardefaults.xml` are read in this order, i.e., values in the second file overwrite the system defaults. To see the configurable variables, set the environment variable `TASCARSHOWGLOBAL` to "yes" start the application from the command line, e.g.,

```
TASCARSHOWGLOBAL=yes tascar
```

or

```
TASCARSHOWGLOBAL=yes tascar_spkcalib
```

An example application configuration file can look like this:

```xml
<?xml version="1.0"?>
<tascar>
  <spkcalib>
    <inputport data="system:capture_29"/>
    <reflevel data="80"/>
  </spkcalib>
</tascar>
```

This will translate into these variables:

```
tascar.spkcalib.inputport (system:capture_29)
tascar.spkcalib.reflevel (80)
```

2.5 Content ownership rights

Complex virtual acoustic or audiovisual environments in TASCAR often depend on a huge amount of external files, e.g., sound files, trajectory data, 3D models, or texture and material definitions. Keeping track of the ownership of many files can be difficult. Therefore, TASCAR provides methods to facilitate the process of fair and legally correct distribution of TASCAR session files. Part of these methods is the setting of authorship and license abbreviations of session files (see Section 4.1), and a simple text file based way of specifying license conditions of external sound files (see 8.30). A summary of the licenses used by a session is provided in the main window in the “Licenses” tab and with the command line tool `tascar_showlicenses`. Please always check the information provided by those tools carefully before sharing a session file.

Please note that in many cases it is illegal to remove or modify authorship and license information from files originating from other sources.

3 Scene Definition

Virtual Acoustic Scenes are created using the XML scene definition file format. The TASCAR scene definition (`.tsc` file) is a text XML file in which the user specifies all the details about the scene using various commands, XML elements and their attributes. An example of a scene definition is shown below (example file `example_basic.tsc`):

```xml
<?xml version="1.0" encoding="UTF-8"?>
<session duration="120" srv_port="9877" license="CC BY-SA 3.0" attribution="Giso
    Grimm">
  <scene name="kitchen" guiscale="6">
    <receiver type="ortf" name="out">
      <position>0 1.3 0.2 1.5</position>
      <orientation>0 -165 0 0</orientation>
    </receiver>
    <source name="clock">
      <position>0 0.61 -1.23 2.1</position>
      <sound>
        <plugins>
          <sndfile name="sounds/clock.wav" loop="0" level="60" resample="true"/>
        </plugins>
      </sound>
    </source>
    <source name="frying_pan">
      <position>0 0.9 1.03 0.87</position>
      <sound>
        <plugins>
          <sndfile name="sounds/pan.wav" loop="0" level="85" resample="true"/>
        </plugins>
      </sound>
    </source>
    <face name="wall" width="2.4" height="0.5" reflectivity="0.9" damping="0.1">
      <position>0 -1 1.25 0.85</position>
      <orientation>0 -90 0 0</orientation>
    </face>
  </scene>
```

```
  <connect src="render.kitchen:out_l" dest="system:playback_1"/>
  <connect src="render.kitchen:out_r" dest="system:playback_2"/>
</session>
```

Example 1: examples/example_basic.tsc

`<scene/>` , `<receiver/>` , `<face/>` , `<source/>` etc. are the elements and **name**, **loop**, **reflectivity**, **width** etc. are their attributes. Figure 2 shows the representation of this scene definition in TASCAR. The main window contains a toolbar for file interactions, transport and time control, and controls for muting and soloing the different components of the scene (left panel). The scene map window contains a visual representation of the scene. Editing the scene via the graphical user interface is currently not possible.

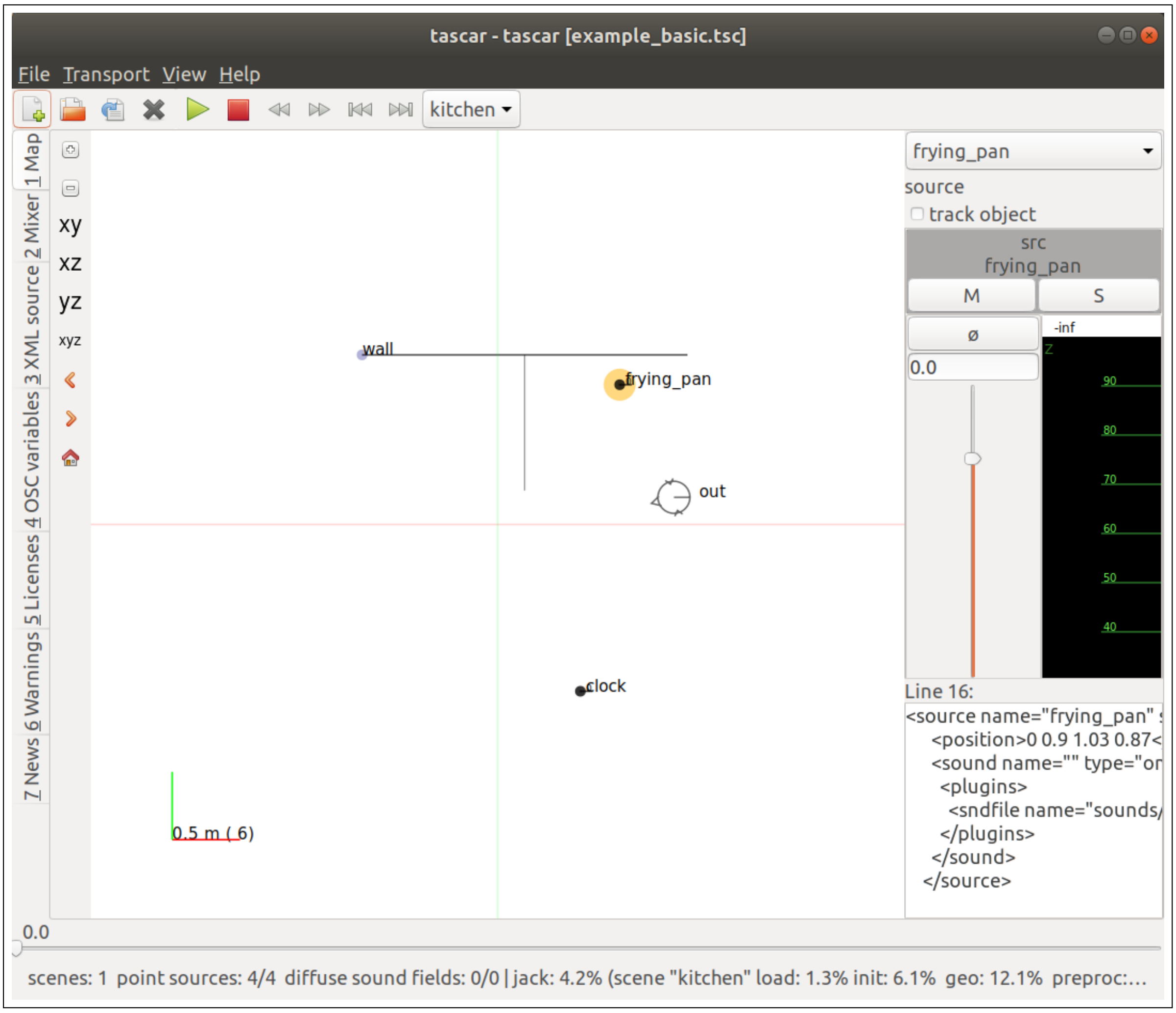


Figure 2: Simple TASCAR scene example. Scene consist of two sources, one reflector and one receiver.

Note: In general, if an attribute or a element is not specified in the scene definition, it is set to default. Therefore, it is not necessary to specify all the recognized attributes and elements.

4 Top level elements

Elements `<session/>` and `<scene/>` are referred to as top-level elements in TASCAR documentation. One `<session/>` element can contain multiple `<scene/>` elements. Together, they form the outermost building blocks of TASCAR scenes.

4.1 The `<session>...</session>` element

`<session/>` is the root element of each scene definition file. It can contain one or more scenes (`<scene/>`), port connections (`<connect/>`), external modules (`<modules/>`) and range definitions (`<range/>`).

Attributes of element **session**

name	description (type, unit)	def.
attribution	attribution of license, if applicable (string)	
duration	session duration (double, s)	60
initcmd	Command to be executed before first connection to jack. Can be used to start jack server. (string)	
initcmdsleep	Time to wait for initcmd to start up, in seconds. (double, s)	0
levelmeter_min	Level meter minimum (double, dB SPL)	30
levelmeter_mode	Level meter mode (rms, rmspeak, percentile) (string)	
levelmeter_range	Level range of level meters (double, dB)	70
levelmeter_tc	level meter time constant (double, s)	2
levelmeter_weight	level meter weighting (f-weight)	Z
license	license type (string)	
loop	loop session at end (bool)	false
name	session name (string)	tascar
playonload	start playing when session is loaded (bool)	false
profilingpath	OSC path to dispatch module profiling information to (string)	
requirefragsize	Session fragment size, stop loading the session if the system fragment size doesn't match (int32)	0
requiresrate	Session sampling rate, stop loading the session if the system sampling rate doesn't match (double, Hz)	0
srv_addr	OSC multicast address in case of UDP transport (string)	
srv_port	OSC port number (string)	9877
srv_proto	OSC protocol, UDP or TCP (string)	UDP
starturl	URL of start page for display (string)	
warnfragsize	Session fragment size, print a warning if the system fragment size doesn't match (int32)	0
warnsrate	Session sampling rate, print a warning if the system sampling rate doesn't match (double, Hz)	0

Attributes of element **connect**

name	description (type, unit)	def.
dest	jack destination port (string)	
failonerror	create an error if connection failed, alternatively just warn (bool)	false
src	jack source port (string)	

Attributes of element **range**

name	description (type, unit)	def.
end	end time (double, s)	0
name	range name (string)	
start	start time (double, s)	0

The sampling rate and fragment size of a session is typically defined by the jack server or the interface of the offline rendering tools. Use the attributes warnrate, requiresrate, warnfragsize and requirefragsize for more control over the audio back-end settings.

A session can have sub-elements **<mainwindow/>** and **<mapwindow/>** to control the window positions. These attributes are allowed:

Attributes:	
x	x-position of window
y	y-position of window
w	Width of window (default: 1600)
h	Height of window (default: 480)

An example of a session with multiple scenes is:

```xml
<?xml version="1.0"?>
<session name="example" duration="120" license="CC 0">
  <scene name="scene1">
    ...
  </scene>
  <scene name="scene2">
    ...
  </scene>
  <scene name="scene3">
    ...
  </scene>
</session>
```

Example 2: examples/example_multiplescenes.tsc

The jack transport can be controlled via the OSC paths /transport/start, /transport/stop and /transport/locate.

OSC variables:

path	fmt.	range	r.	description
/runscript	s	string	no	Name of OSC script file to be loaded.
/scriptpath	s	string	yes	
/sendvarsto	ss		no	
/sendvarsto	sss		no	
/sendxmlto	ss		no	Send session file XML code to an OSC server. First parameter is the URL, the second is the path.
/timedmessages/add	fs		no	
/timedmessages/clear			no	

/transport/addtime	f	no	Move the current transport position by the given number of seconds.
/transport/locate	f	no	Locate the transport to the given second.
/transport/locatei	i	no	Locate the transport to the given audio sample.
/transport/playrange	ff	no	Play the session in the given time interval.
/transport/start		no	Start the playback of the session from the current position
/transport/stop		no	Stop the playback of the session
/transport/unload		no	Unload the scene

A special sub-element `<include/>` can be used to include scenes and other elements from another session file, given by the attribute `name`. Example:

```
<?xml version="1.0"?>
<session>
  <include name="session1.tsc"/>
  <include name="session2.tsc"/>
</session>
```

Attributes:	
name	File name to be included
license	License form of session file
attribution	Attribution of session file, e.g., author name

The `<include/>` element can also be used at other levels; the only limitation is that the root element of the included file needs to match the active element into which the external file is included. In the example above, the root XML element of files `session1.tsc` and `session2.tsc` has to be a `<session/>` element. Any attributes of the root element in the included file are ignored.

The element `<license/>` can be used to specify additional licenses, e.g., for additional visual content. In addition to the licenses, the authors can be specified using the `<author/>` element, and a bibliography can be provided using the `<bibitem/>` elements:

```
<session license="CC BY-SA 3.0" attribution="Author1">
  <license name="visuals" license="CC BY-SA-NC 3.0" attribution="Author2"/>
  <author name="Author1" of="audio"/>
  <author name="Author2" of="visuals"/>
  <bibitem>Grimm, G., Kollmeier, B., &amp; Hohmann, V. (2016). Spatial acoustic
    scenarios in multichannel loudspeaker systems for hearing aid evaluation.
    Journal of the American Academy of Audiology, 27(7), 557-566.</bibitem>
  ...
</session>
```

When at least one author is specified, then this information will be displayed while loading the session. Please note that in many cases it is illegal to remove or modify the authorship information from a work, or change the original license conditions. Therefore it is possible in TASCAR to specify multiple `<author/>` and `<license/>` elements, to correctly attribute your contributions to a session originating from other sources.

The file names provided in the name attribute of the `<include/>` element can be absolute or relative. Relative file names are relative to the directory containing the root `.tsc`-file.

The performance of all loaded modules can be measured by setting the attribute `profilingpath` to an OSC path, which can be added to the datalogging module, see Example 3. In that case, the profiling variable contains the time used by the modules in each processing cycles. The `size` attribute of the OSC variable in the data logging needs to match the total number of modules loaded in a session (multiple `<modules/>` sections will be merged).

```
<?xml version="1.0"?>
<session license="CC0" profilingpath="/profmod">
  <scene>
    <source name="a"/>
    <source name="b"/>
  </scene>
  <modules>
    <route>
      <plugins profilingpath="/prof">
        <timestamp path="/ts1"/>
        <sine/>
        <pink/>
        <filter/>
        <level2osc weights="Z A C" tau="1" threaded="true" url="osc.udp://localhost:9877/"/>
        <lipsync_paper threaded="true" path="/lipsyncp" energypath="/energyp" strmsg="" url="osc.udp://localhost:9877/"/>
        <lipsync threaded="true" path="/lipsync" energypath="/energy" strmsg="" url="osc.udp://localhost:9877/"/>
        <timestamp path="/ts2"/>
      </plugins>
    </route>
    <pos2osc pattern="/*/*"/>
    <datalogging>
      <osc path="/prof" size="8"/>
      <osc path="/level" size="4" ignorefirst="true"/>
      <osc path="/lipsyncp" size="3"/>
      <osc path="/lipsync" size="3"/>
      <osc path="/energyp" size="5"/>
      <osc path="/energy" size="5"/>
      <osc path="/profmod" size="3"/>
      <osc path="/ts1" size="1"/>
      <osc path="/ts2" size="1"/>
    </datalogging>
  </modules>
</session>
```

Example 3: examples/example_profiling.tsc

4.2 The <scene>...</scene> element

Attributes of element **scene**

name	description (type, unit)	def.
active	render scene (bool)	true
c	speed of sound (double, m/s)	340
guicenter	origin of GUI window (pos, m)	0 0 0
guiscale	scale of GUI window of this scene (double, m)	200
guitracking	object name for scene tracking (string)	
id	scene id, or empty to auto-generate id (string)	1
ismorder	order of image source model (uint32)	1
name	scene name (string)	scene

Sub-elements:
<source/> , <receiver/> , <diffuse/> , <face/> , <facegroup/> , <obstacle/> , <description/>, <material/>

<scene/> is a top-level element of a TASCAR scene definition. An example scene definition is given in Example 1.

5 Objects

A scene can be complemented with objects of different types (as it was already shown in the first example of a scene definition). Objects can be any of the following types:

- sources (<source/>), diffuse sound fields (<diffuse/>)
- receivers (<receiver/>)
- reflectors (<facegroup/> , <face/>)
- obstacles (<obstacle/>)
- masks (<mask/>)

There can be many objects of different types in the scene. Each object has position and orientation in space and time, and may also contain different attributes depending on the type.

There are two different ways of defining the position and orientation of an object - “interactive” and “not interactive”. First, we have to specify the “not interactive” position and orientation (it can be also the whole trajectory of an object) in a scene definition file.

As an addition to this predefined geometry, we can steer the object using an external device, for example a joystick, head movement tracking system or by an algorithm which generates a certain type of movement, thus applying “interactive” type of geometry. The resulting position and orientation of an object will be calculated by summing up these two mentioned types of position and orientation. The difference between two types of defining the movement has been depicted in Figure 3.

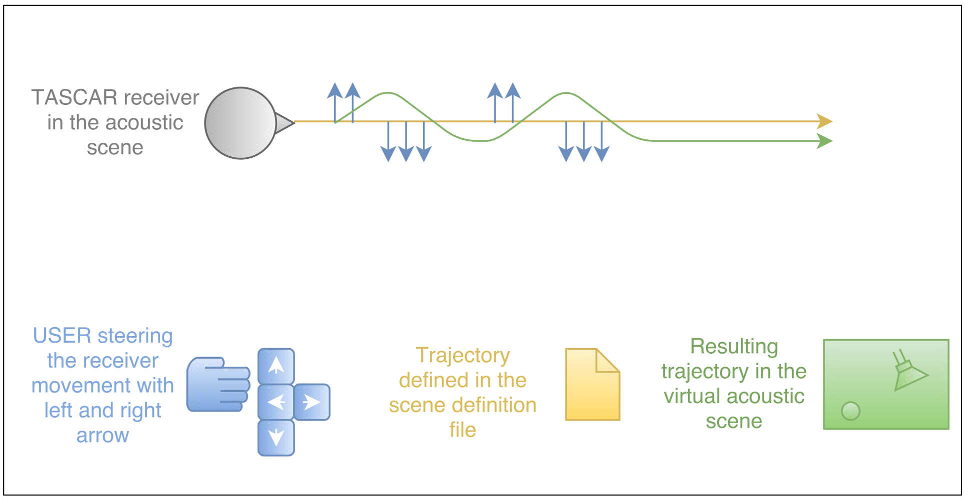


Figure 3: Two different ways of dealing with orientation and position in TASCAR

5.1 Common attributes of objects

The following attributes are common to all scene objects:

Attributes of element **objects** (boundingbox diffuse face facegroup mask obstacle receiver reverb source)

name	description (type, unit)	def.
`dlocation`	delta location (pos, m)	0 0 0
`dorientation`	delta orientation (Euler rot, deg)	0 0 0
`localpos`	local position (pos, m)	0 0 0
`parent`	Name of parent object from same scene (string)	
`sampledorientation`	sample orientation by line fit into curve (double, m)	0
`start`	time when rendering of object starts (double, s)	0

The delta transformation values can be overwritten by actor modules or the OSC interface. The `dorientation` attribute is first rotation around Z axis, then Y axis, followed by X axis.

The render activity is limited to the interval [start,end] only if end $>$ start. All time information of objects, such as dynamic geometry or sound file positions, are relative to the object start time. This *object time* is defined as session time minus object start time.

Muting an object disables it, i. e., muting a `source` will disable the sound, muting a `receiver` will disable the output of the receiver, and muting a `face` or `facegroup` will disable all image sources generated by that reflector.

Attributes of element **ports** (diffuse receiver sound)

name	description (type, unit)	def.
`caliblevel`	calibration level (float, dB SPL)	93.9794
`connect`	Regular expressions of port names for connections (string array)	
`gain`	port gain (float, dB)	0

`inv`	phase invert (bool)	false
`layers`	render layers (bits32)	all

Attributes of element **routes** (diffuse face facegroup mask obstacle receiver reverb source)

name	description (type, unit)	def.
`color`	html color string (string)	
`end`	end of render activity, or 0 to render always (double, s)	0
`id`	Unique route id, empty to autogenerate (string)	22
`mute`	Mute flag of route (bool)	false
`name`	Route name (string)	
`scale`	scale of local coordinates (float)	1
`solo`	Solo flag of route (bool)	false

All objects have these OSC variables:

OSC variables:

path	fmt.	range	r.	description
`/.../pos`	fff		no	XYZ Translation in m
`/.../pos`	ffffff		no	XYZ Translation in m and ZYX Euler angles in degree
`/.../scale`	f		yes	object scale
`/.../zyxeuler`	fff		no	ZYX Euler angles in degree

Objects which represent an audio object have these OSC variables:

OSC variables:

path	fmt.	range	r.	description
`/.../mute`	i	bool	yes	mute flag, 1 = muted, 0 = unmuted
`/.../solo`	i		no	
`/.../targetlevel`	f	dB	yes	Indicator position in level meter display

5.2 Common sub-elements of objects

All scene objects (e.g. instances of **`<source/>`** , **`<receiver/>`** , **`<mask/>`** , **`<facegroup/>`** , **`<face/>`** , etc.) have to define their position and orientation in space and time. The following child elements can be used to specify these parameters (see also Example 1).

5.2.1 The `<position>...</position>` element

Position is specified by providing Cartesian coordinates (in meters) as well as the time point associated with them (object time in seconds, counted with respect to the time when the object starts to be active, see attribute `start` of parent element):

```
<position>t x y z</position>
```

If we want the object to change its position over the course of the scene, we have to specify more than one point in space and time:

```
<position>
  t_1  x_1  y_1  z_1
  t_2  x_2  y_2  z_2
  t_3  x_3  y_3  z_3
</position>
```

t_n is time and x_n, y_n and z_n are the Cartesian coordinates of an object at time t_n. The object's position will be linearly interpolated between these points. The numbers are separated by white space. The line breaks in this example are solely for human readability, and not required by the TASCAR software.

We can also use an attribute to control the interpolation method:

```
<position interpolation="cartesian">
   0  1  4  0
  10  1 -4  0
</position>
<position interpolation="spherical">
   0  1  4  0
  10  1 -4  0
</position>
```

The first example will interpolate linearly in Cartesian coordinates, i. e., the object will move on a straight line from (1,4,0) to (1,-4,0). The second example will interpolate linearly in spherical coordinates around the origin, i. e., the object will move along an arc from (1,4,0) to (1,-4,0).

The last position of the position track is held until the either the session, or the current position loop iteration (see below), terminates.

Instead of defining the position track in the tsc file it can also be read from a comma-separated file, by setting the attribute `importcsv`. Please note that the file needs to be comma separated, with four numbers t, x, y, z in each row.

```
  <position importcsv="myfile.csv"/>
```

Position tracks and orientation tracks can be looped by adding the attribute `loop` with a number larger than zero.

```
  <position loop="10">0 0 0 0
  6 10 0 0</position>
```

in this case, the position/orientation is sampled with the object time modulo loop time (10 seconds), i. e., the object is moving for 6 seconds, then resting at (10,0,0) for 4 seconds, then again moving for 6 seconds, starting at (0,0,0).

Attributes of element **position**

name	description (type, unit)	def.
importcsv	Read position track from the .csv-file as comma-separated values. The file name can contain absolute or relative path. Relative paths are relative to the session's .tsc-file. Default: position track is contained as space-separated text between opening and closing **<position/>** tags. (string)	
interpolation	Coordinate system in which positions are linearly interpolated between given positions. Possible values are cartesian and spherical. (string)	cartesian
loop	The value, if greater than 0, specifies the time when this position track is repeated from 0 (double, s)	0

5.2.2 The <orientation>...</orientation> element

Orientation is specified in Euler (navigation) angles $R_{z,y,x}$, measured in degrees:

```
<orientation>t Rz Ry Rx</orientation>
```

R_z is the rotation around the z-axis, R_y around the y-axis and R_x around the x-axis. They are applied in z,y,x order, after application of the position. If we would like the orientation of an object to change during the scene, we can specify multiple angles and time points associated with them:

```
<orientation>
  t_1   Rz_1   Ry_1   Rx_1
  t_2   Rz_2   Ry_2   Rx_2
</orientation>
```

The numbers are separated by white space. The line breaks in this example are solely for human readability, and not required by the TASCAR software.

The last orientation of the orientation track is held until the either the session, or the current orientation loop iteration (see below), terminates.

Instead of defining the orientation directly in the tsc file it can also be read from a comma-separated file, by setting the attribute importcsv.

Euler orientation tracks can be looped by adding the attribute loop with a number larger than zero.

Attributes:

importcsv	Read orientation track from the .csv-file as comma-separated values. The file name can contain absolute or relative path. Relative paths are relative to the session's .tsc-file. Default: orientation track is contained as space-separated text between opening and closing **<orientation/>** tags.
loop	The value, if greater than 0, specifies the time in seconds when this orientation track is repeated from 0. Default: 0, no repetition.

5.2.3 The <creator>...</creator> element

Instead of defining the object's movement manually (defining position and orientation for each time point) we can use the creator tool.

```
    <source name="trolley">
      <creator>
        <load format="csv" name="supermarket_trolley.csv"/>
        <velocity const="1.7"/>
      </creator>
      <sound/>
    </source>
```

In this case, the orientation is calculated as a tangent along the given path.

5.2.4 Delta-transformations

In addition to the transformation defined by the <position/>, <orientation/> and <creator/> elements, every object has a delta-transformation which can be controlled via OSC or by actor modules (see section 7).

5.2.5 The <navmesh/> element

The navmesh element can be used to restrict the object motion to a navigation mesh. This is specifically useful when controlling object positions via game controllers.

Attributes of element **navmesh**

name	description (type, unit)	def.
importraw	file name of vertex list (string)	
maxstep	maximum step height of object (double, m)	0.5
zshift	shift object vertically (double, m)	0

Faces can be imported from a text file, containing space-separated lists of polygon coordinates (see section 5.9 on face groups for details), or within the <faces/> sub-element.

5.3 The <source>...</source> element

Recognized attributes:

<source/> supports the attributes common to all scene objects, refer to section 5.1 Common attributes of objects on page 15 for details.

Recognized sub-elements:

<position/> , <orientation/> , <creator/> , <sound/>

<source/> is an element used to create the sound source objects in the scene definition. Since sources are also objects, they can have a trajectory (see 5.1). A source object can

consist of one or more "sound vertices" specified with a sub-element <sound/> . There must also be a sound content, for example from a sound file, assigned to a source. We can assign a sound content to a source using the audio plugin <sndfile/> .

In the box below we can see a definition of a simple point source object (taken from Example 1):

```
<source name="frying_pan">
  <position>0 0.9 1.03 0.87</position>
  <sound>
    <plugins>
      <sndfile name="sounds/pan.wav" loop="0" level="85" resample="true"/>
    </plugins>
  </sound>
</source>
```

5.3.1 The <sound .../> element

Attributes of element **sound** (cardioidmod door farsrc generic1storder omni), inheriting from **ports**

name	description (type, unit)	def.
airabsorption	apply air absorption filter (bool)	true
d	distance to next sound along trajectory, or 0 for normal mode (double, m)	0
delayline	use delayline (bool)	true
gainmodel	gain rule, valid gain models: "1/r", "1" (string)	1/r
id	id of sound vertex (string)	6
ismmax	maximal ISM order to render (uint32)	2147483647
ismmin	minimal ISM order to render (uint32)	0
maxdist	maximum distance to be used in delay lines (float, m)	3700
minlevel	Level threshold for rendering (float, dB SPL)	-inf
name	name of sound vertex (string)	
nearfieldlimit	distance arond 1/r source where the gain is constant (float, m)	0.1
rx	Euler orientation (X) relative to parent (double, deg)	0
ry	Euler orientation (Y) relative to parent (double, deg)	0
rz	Euler orientation (Z) relative to parent (double, deg)	0
sincorder	order of sinc interpolation in delayline (uint32)	0
size	physical size of sound source (effect depends on rendering method) (float, m)	0
type	source directivity type, e.g., omni, cardioid (string)	omni
x	position relative to parent (double, m)	0
y	position relative to parent (double, m)	0
z	position relative to parent (double, m)	0

Another sub-element used in the example is <sound/>. With this sub-element we specify the sound vertices of the source object, i.e., points from which the sound radiates. Properties of the sound vertex define its radiation characteristics (type, gainmodel, size, sincorder), its level calibration and gain characteristics (caliblevel, gain), activity processing (maxdist, minlevel, layers), image source model settings (ismmin, ismmax) and the relative position and orientation (x, y, z, az, el, r, rz, ry, rx, d). Please note that the local position relative to the object origin and orientation can be provided either in Cartesian coordinates (`x`, `y`, `z`) or in spherical coordinates (`az`, `el`, `r`), however, these can not be mixed.

If we want to create a point source, as in the example, we will have one sound vertex exactly at the position of the source object (so at the point specified in the element `<position/>`). When we create a source with more than one sound vertex, the object position specified in sub-element `<position/>` will now be the reference point for all the sound vertices, and changing this position will also change the position of the sound vertices. The same holds for the orientation of a source object consisting of more than one sound vertex. Below we can see how such a source has to be defined:

```
  <source name="piano" color="#101077">
    <position>
      0 -3.2 1.7 1.4
      10 3.2 2.7 1.4
    </position>
    <orientation>0 -24 0 0</orientation>
    <sound name="leftside" x="-0.7">
      <plugins>
        <sndfile name="sounds/jazzclub-piano1.wav" level="75"/>
      </plugins>
    </sound>
    <sound name="rightside" x="0.7">
      <plugins>
        <sndfile name="sounds/jazzclub-piano2.wav" level="75"/>
      </plugins>
    </sound>
  </source>
```

Example 4: examples/example_vertices.tsc

We have a source object called "audience" which is made of four sound vertices called "guy1","guy2" etc. Their position is specified relative to the position in the sub-element `<position/>` using attributes `x`, `y` and `z` – for example the vertex called "guy1" is located -0.7 m from the reference point in x direction and 0.1 m in y direction.

Figure 4 presents an example of a scene containing sound sources consisting of more than one sound vertex.

Source directivity is defined by the source module types. Currently the types “omni”, “cardioidmod” and “door” are supported.

Audio content can be added either from external playback (using jack ports, see the `connect` attribute), or using the audio plugin `<sndfile/>` (see 8.30). When recording new audio material, we recommend to follow the documentation recommendations of the DEGA (Leckschat et al., 2020). A useful source of sound files can be found at https://freesound.org/.

Sounds have these OSC variables:

OSC variables:

path	fmt.	range	r.	description
`/.../caliblevel`	f		yes	calibration level in dB
`/.../gain`	f		no	Gain in dB

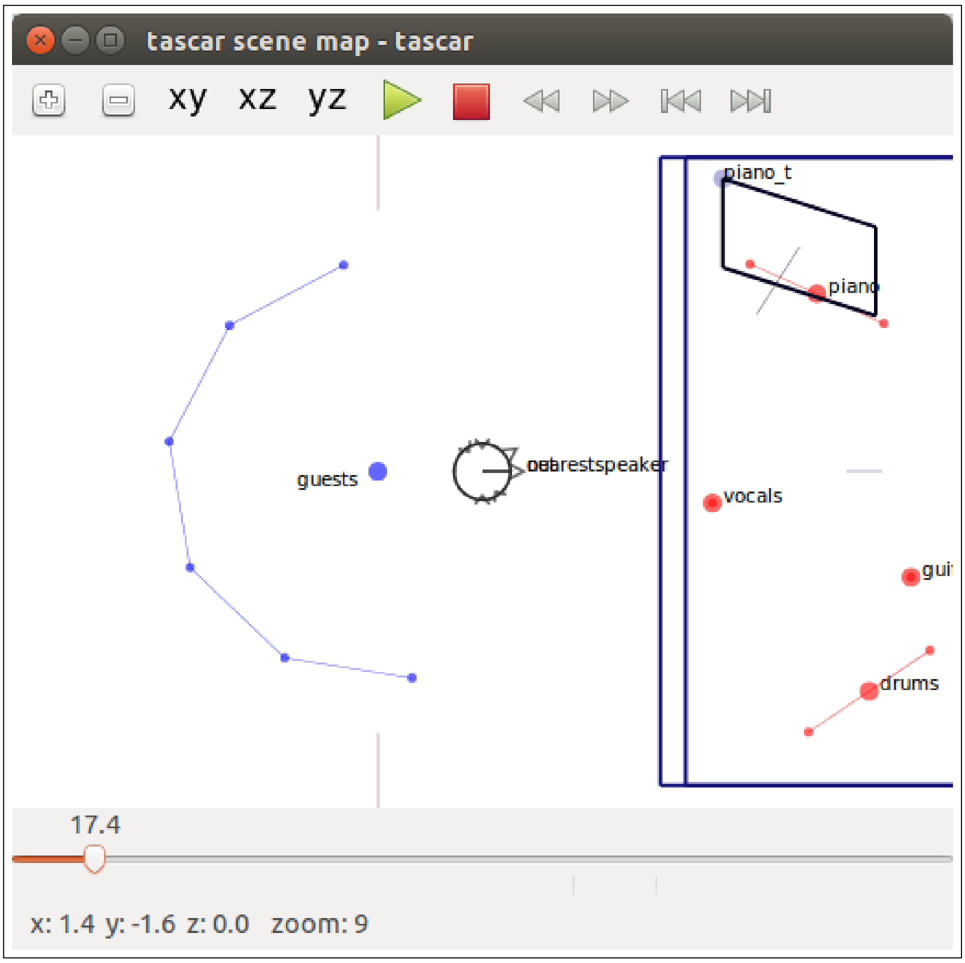


Figure 4: Examples of sound sources and their vertices. In this scene there are point sources like "vocals" or "guitar". There are also sound sources with more than one vertex like "guests" (6 vertices) or "piano" (2 vertices) - the big dot close to the name of the source is the reference point for a given source.

`/.../globalpos`	fff		yes	global position of sound vertex in meters
`/.../ismmax`	i		yes	Maximal Image Source Model order
`/.../ismmin`	i		yes	Minimal Image Source Model order
`/.../layers`	i		yes	Number representing the layers. Each layer is represented by a bit, i.e., for layers 1+3 use 10
`/.../lingain`	f		no	Linear gain
`/.../mute`	i	bool	yes	Mute state of individual sound, independent of parent
`/.../pos`	fff		yes	local position of sound vertex in meters
`/.../size`	f		yes	Object size in meter
`/.../zeuler`	f		no	Z orientation of the sound vertex, in degree
`/.../zyxeuler`	fff		no	ZYX orientation of the sound vertex, in degree

5.3.2 Source directivity "omni"

The "omni" source directivity has no configuration variables. The sound source radiates independently of the direction.

5.3.3 Source directivity "cardioidmod"

The "cardioidmod" source directivity has these attributes:

Attributes of sound element **cardioidmod**, inheriting from **sound**

name	description (type, unit)	def.
`f6db`	Frequency in Hz, at which a 6 dB attenuation at 90 degrees is achieved (double, Hz)	1000
`fmin`	Low-end limit for stabilization (double, Hz)	60

At low frequencies, the source radiates omni-directionally. At higher frequencies, a cardioid-like radiation pattern is achieved.

5.3.4 Source directivity "door"

The "door" source directivity has these attributes:

Attributes of sound element **door**, inheriting from **sound**

name	description (type, unit)	def.
`distance`	Distance by which the source is shifted behind the door (double, m)	1
`falloff`	Distance at which the gain attenuation starts (double, m)	1
`height`	Door height (double, m)	2
`width`	Door width (double, m)	1
`wndsqrt`	Flag to control von-Hann fall-off (false, default) or square-root of von-Hann fall-off (bool)	false

Door sources shift the perceived source position behind a "door" shape, limited by the edges. They are basically designed for interactive transitions between simulated rooms.

The origin of the "door" is in its center, width is measured in the local y direction and height is measured in the local z direction.

5.3.5 Source directivity "farsrc"

The "farsrc" source is the same as "omni" except that sound is attenuated within a spherical volume and faded in with a von-Hann ramp outside the volume, converging to a $\frac{1}{r}$ distance law outside the ramp. It has these attributes:

Attributes of sound element **farsrc**, inheriting from **sound**

name	description (type, unit)	def.
`distance`	Distance at which the fade-in starts (float, m)	1
`falloff`	Length of fade-in area (float, m)	1

5.3.6 Source directivity “generic1storder”

This source type can be controlled to vary between omni-directional and figure-of-eight directivity. See also the description of the receiver type “vmic” (Section 5.6.17 on page 41) for details.

Attributes of sound element **generic1storder**, inheriting from **sound**

name	description (type, unit)	def.
`a`	undocumented (double)	0

5.4 The `<diffuse .../>` element

Attributes of element **diffuse**, inheriting from **objects ports routes**

name	description (type, unit)	def.
`falloff`	falloff ramp length at boundaries (float, m)	1
`size`	size in which sound field is rendered. (pos, m)	1 1 1

Sub-elements:

`<position/>` , `<boundingbox/>`, `<orientation/>` , `<creator/>` , `<plugins/>`

Besides sound sources consisting of one or more vertices, there is also the possibility of creating diffuse sound fields that “fill” the room and are equally loud within a certain volume (e.g. isotropic babble noise in a cafeteria or distant traffic). We can define a diffuse sound field in the following way:

```
  <diffuse name="birds" size="1000 1000 1000">
    <plugins>
      <sndfile name="sounds/birds.wav" loop="0" level="70"
  channelorder="FuMa"/>
    </plugins>
  </diffuse>
```

Example 5: examples/example_diffuse.tsc

Sound files used to create diffuse sound fields must contain 4 channels (B format, FuMa normalisation, ACN channel sequence). The attribute `size="x y z"` defines the dimensions of the box in which the diffuse sound field is audible. To achieve a smooth decay of the diffuse sound field at the edge of this box, there is a von-Hann ramp for the attenuation of the source outside the box. The length of the ramp is determined by the attribute `falloff="..."`. Like all other objects, diffuse sound fields have a position and an orientation that refers to a position and an orientation of the box.

An example on how to add the **<addsndfile/>** audio plugin to a diffuse sound field can be found below:

```
    <diffuse name="birds" size="1000 1000 1000">
      <plugins>
        <sndfile name="sounds/birds.wav" loop="0" level="70"
    channelorder="FuMa"/>
      </plugins>
    </diffuse>
```

Example 6: examples/example_diffuse.tsc

Internally, TASCAR uses FuMa normalization and ACN channel sequence ("wyzx"). At most places, the Ambisonics channel sequence and normalization can be configured. For level metering, the RMS level of the w-channel is taken.

5.5 The <receiver .../> element

A receiver object can be thought of as a virtual microphone that captures sound in virtual space and serves as the output of the virtual acoustic environment. The choice of the receiver type depends on the playback system and the desired rendering method. It captures the signal of all sound sources in the scene and computes them according to their type. The output signals of a receiver are sent to the playback system, e.g. loudspeakers or headphones.

Attributes of element **receiver** (amb1h0v amb1h1v amb3h0v amb3h3v cardioid chmap debugpos fakebf hann hoa2d hoa2d_fuma hoa3d hoa3d_enc hrtf intensityvector itu51 micarray nsp omni ortf vbap vbap3d vmic wfs), inheriting from **objects ports routes**

name	description (type, unit)	def.
`avgdist`	Average distance which is assumed inside receiver boxes, or 0 to use $(\frac{1}{8}V)^{1/3}$ (float, m)	0
`delaycomp`	subtract this value from delay in delay lines (float, s)	0
`diffuse`	render diffuse sources (bool)	true
`diffusegain`	gain of diffuse sources (float, dB)	0
`fade_gain`	linear fade gain (float)	1
`falloff`	Length of von-Hann ramp at volume boundaries, or -1 for normal distance model (float, m)	-1
`globalmask`	use global mask (bool)	true
`image`	render image sources (bool)	true
`ismmax`	maximal ISM order to render (uint32)	2147483647
`ismmin`	minimal ISM order to render (uint32)	0
`layerfadelen`	duration of fades between layers (float, s)	1
`muteonstop`	mute when transport stopped to prevent playback of sounds from delaylines and reverb (bool)	false
`point`	render point sources (bool)	true

proxy_airabsorption	Use proxy position for air absorption (bool)	false
proxy_delay	Use proxy position for delay (bool)	false
proxy_direction	Use proxy position for direction (bool)	false
proxy_gain	Use proxy position for gain (bool)	false
proxy_is_relative	Proxy is relative to receiver (true) or in absolute coordinates (false) (bool)	false
proxy_position	Proxy position (pos, m)	0 0 0
scatterdamping	damping of scatter reflection filter (float)	0
scatterreflections	Number of reflections created by scattering filter (uint32)	0
scatterspread	Spatial spread of scattering (float, deg)	22.5
scatterstructuresize	size of scatter structure (float, m)	1
type	receiver type (string)	omni
volumetric	volume in which receiver does not apply distance based gain model (pos, m)	0 0 0
volumetricgainwithdistance	For volumetric receivers, increase gain with distance (bool)	false

Sub-elements:

<speaker/> , <boundingbox/>, <position/> , <orientation/> , <creator/>

Attributes of element **boundingbox**, inheriting from **objects**

name	description (type, unit)	def.
active	use bounding box (bool)	false
falloff	fade-out ramp length at boundaries (float, m)	1
size	dimension of bounding box (pos, m)	0 0 0

A receiver encodes the signals of primary sources, image sources and diffuse sound fields into a receiver type specific output format. Each receiver owns one jack output port for each output channel n; the number of channels N depends on the receiver type and configuration. The output signal of a receiver is $\mathbf{z}(t) = (z_1(t), z_2(t), \dots, z_N(t))$.

The receiver functionality can be split into a *panning* or directional encoding of primary and image sources, and a *decoding* of first order Ambisonics diffuse signals:

$$\mathbf{z}(t) = \underbrace{\sum_{k=1}^{K} \mathbf{w}(\mathbf{p}_{k,rel}) y_k(t)}_{\text{panning}} + \underbrace{\sum_{l=1}^{L} \mathbf{D}\hat{\mathbf{O}}_{rec}\hat{\mathbf{O}}_{src}^{-1}\mathbf{f}_l(t)^T}_{\text{diffuse decoding}} \tag{1}$$

In the panning part, the driving weights $\mathbf{w} = (w_1, w_2, \dots, w_N)$ depend on the relative source position in the receiver coordinate system, $\mathbf{p}_{rel} = \mathbf{O}_{rec}^{-1}(\mathbf{p}_{src} - \mathbf{p}_{rec})$. For the definition of the receiver orientation matrix $\mathbf{O}_{rec}$ see Eq. 21 on page 149. $y_k(t)$ is the output signal of the acoustic model, i.e., distance-dependent gain and air absorption, for the k-th source; K is the number of all primary and image sources. In the diffuse decoding part, $\mathbf{D}$ is the receiver type specific first order Ambisonics decoding matrix,

$$\mathbf{D} = \begin{pmatrix} d_{1,w} & d_{1,x} & d_{1,y} & d_{1,z} \\ \vdots & \vdots & \vdots & \vdots \\ d_{n,w} & d_{n,x} & d_{n,y} & d_{n,z} \end{pmatrix},$$

and $\hat{\mathbf{O}}_{rec}$is the rotation matrix for first order Ambisonics signals, to compensate the receiver orientation (see Eq. 25, page 149). $\mathbf{f}_l$ is the first order Ambisonics signal of the l-th diffuse sound field; L is the number of all diffuse sound fields, including diffuse reverberation inputs.

For all loudspeaker-based receiver types, D is a first order Ambisonics decoder matrix, with optional loudspeaker density compensation and decorrelation filters. By default, a $\max rE$-decoder is used. The order gain g_{xyz} is set according to Table 3.10 of Daniel (2001). A loudspeaker layout is assumed to reproduce in 3D when at least one loudspeaker has non-zero elevation. For Ambisonics based receiver types, D is a diagonal matrix. By default, the decoded output of the first order Ambisonics rendering is de-correlated using FIR all-pass filters to achieve diffuse sound fields and avoid coloration artifacts (see `decorr` and `decorr_length` for details).

Figure 5 presents the typical connections in TASCAR and may help to visualize the role of the receiver.

If the `volumetric` attribute defines a non-zero volume, then all sources within the receiver volume box will be rendered with the same gain (volumetric rendering). An average distance of $(\frac{1}{8}V)^{1/3}$ with volume V is assumed, or if `avgdist` is given, the given value is used. Outside the box, either a von-Hann ramp is applied (`falloff` > 0), or the standard distance model is applied. With volumetric receiver settings, the delay depends on the relative distance between the receiver origin and the source position.

OSC control

Receivers can be controlled via OSC similar to other objects (position, zyx Euler rotation, gain). They also support fade commands:

```
/<scene>/<name>/fade <gain> <duration> [ <starttime> ]
```

Here `gain` is the linear target gain, `duration` is the length of the fade, and the optional third parameter `starttime` is the start time, at which the fade is applied. If the current time is later than `starttime` then the fade is applied immediately. The fade is always calculated using a raised cosine ramp. A new fade event will overwrite any currently ongoing or scheduled fade events.

OSC variables:

path	fmt.	range	r.	description
`/.../caliblevel`	f	[0,120]	yes	
`/.../diffusegain`	f	[-30,30]	yes	relative gain of diffuse sound field model
`/.../fade`	ff		no	
`/.../fade`	fff		no	

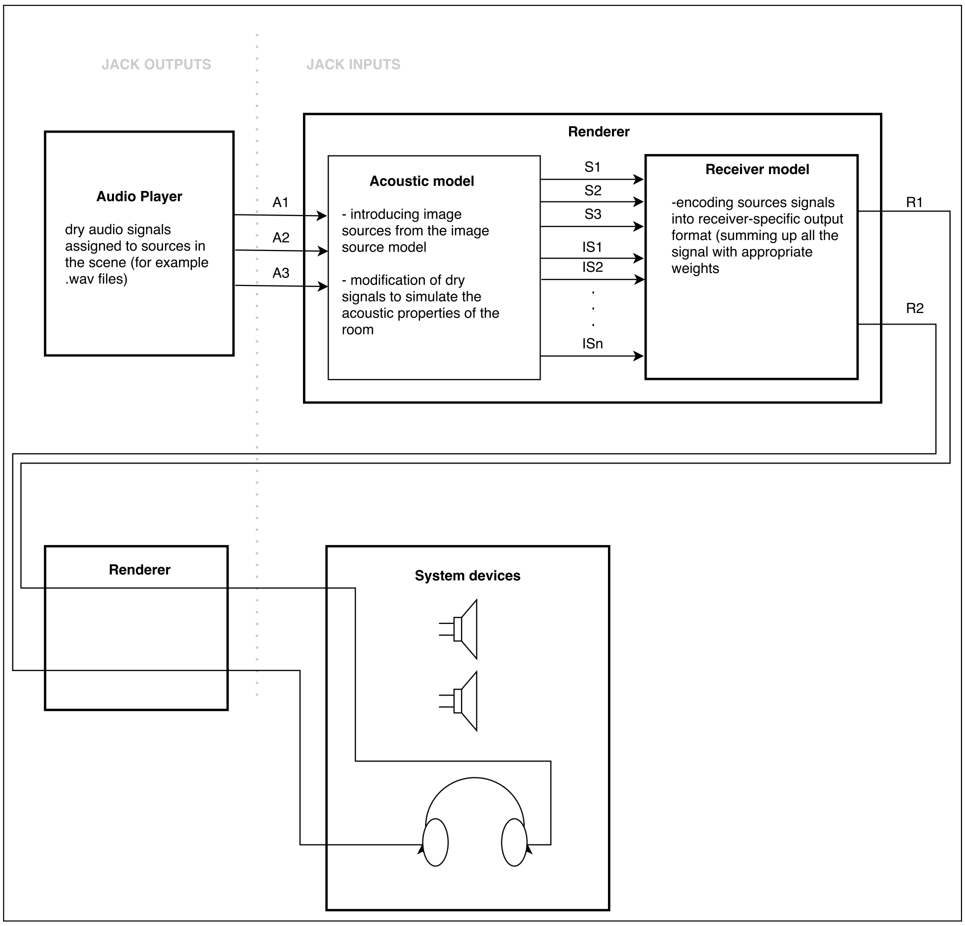


Figure 5: Typical structure of connections in TASCAR.

`/.../gain`	f		no	
`/.../ismmax`	i		yes	
`/.../ismmin`	i		yes	
`/.../layers`	i		yes	
`/.../lingain`	f		no	
`/.../proxy/airabsorption`	i	bool	yes	Use proxy position for air absorption
`/.../proxy/delay`	i	bool	yes	Use proxy position for delay
`/.../proxy/direction`	i	bool	yes	Use proxy position for direction
`/.../proxy/gain`	i	bool	yes	Use proxy position for gain
`/.../proxy/is_relative`	i	bool	yes	Proxy is relative to receiver (true) or in absolute coordinates (false)
`/.../proxy/position`	fff		yes	Proxy position in m
`/.../scatterdamping`	f	[0,1]	yes	damping of scatter reflection filter
`/.../scatterspread`	f		yes	Spatial spread of scattering
`/.../scatterstructuresize`	f	[0,10]	yes	size of scatter structure in m

Proxy position

Receivers can replace the source position with a proxy position. The properties of air absorption, delay, gain, and direction can be replaced separately. Proxy position and property selection can be controlled in the XML file or via OSC.

5.6 Receiver types

The following types of generic receivers (see Table 195 for an overview) can be used in TASCAR:

List of generic receiver types:

- **amb1h0v**
- **amb1h1v**
- **amb3h0v**
- **amb3h3v**
- **cardioid**
- **chmap**
- **debugpos**
- **fakebf**
- **hoa2d_fuma**
- **hoa3d_enc**
- **hrtf**
- **intensityvector**
- **itu51**
- **micarray**
- **omni**
- **ortf**
- **vmic**

5.6.1 amb1h0v

First order horizontal Ambisonics encoder, B-format (FuMa channel sequence “wxy” and normalization).

```
<receiver type="amb1h0v"/>
```

The normalization attributes `normalization="FuMa"` (default) or `normalization="SN3D"` are supported.

5.6.2 amb1h1v

First order Ambisonics encoder, B-format (FuMa channel sequence “wxyz”).

```
<receiver type="amb1h1v"/>
```

The normalization attributes `normalization="FuMa"` (default) or `normalization="SN3D"` are supported.

Attributes of receiver element **amb1h1v**, inheriting from **receiver**

name	description (type, unit)	def.
`channelorder`	Channel order, either “ACN” (wyzx) or “FuMa” (wxyz) (string)	ACN
`normalization`	Normalization, either “FuMa” or “SN3D” (string)	FuMa

5.6.3 amb3h0v

Third order horizontal Ambisonics encoder, B-format (FuMa channel sequence “wyxvuqp” and normalization).

Horizontal HOA:

```
<receiver type="amb3h0v"/>
```

$$N = 7,\, w_k = \begin{cases} \sqrt{2} & k = 0 \\ \cos(\frac{k+1}{2}\alpha) & k \,\text{odd} \\ \sin(\frac{k}{2}\alpha) & k \,\text{even} \end{cases}$$

5.6.4 amb3h3v

Third order Ambisonics encoder, B-format (FuMa channel sequence “wyzxvtrsuqomklnp” and normalization).

```
<receiver type="amb3h3v"/>
```

$N = 16$

To play back the content of a virtual scene on an arbitrary playback device, we have to use an external tool to decode the ambisonics format (a tool which will mix the ambisonics channels signals in an appropriate way in order to get the signals for channels of our playback system). To achieve this, we can make a jack connection between the output of the ambisonics receiver (`<scenename>.render:<receivername>.<channel>`) and ambisonics decoder "ambdec":

```
  <receiver type="amb3h0v" name="receiver 1" connect="ambdec:in">
    <position>0 1.3 0.2 1.5</position>
```

```
  <orientation>0 -165 0 0</orientation>
</receiver>
```

If we use connect="ambdec:in", then the connections will be done, so that the channels have the same name in both receiver output and ambdec input, as shown in the Figure 6. We can then go to the settings of the ambdec device and find a type of output corresponding to our playback set up (Config>>Load>>usr/share/ambdec/presets). For example if we choose a preset *octagon-3h0v*, the appropriate output ports will appear (Figure 7), which can be then connected with the system playback devices.

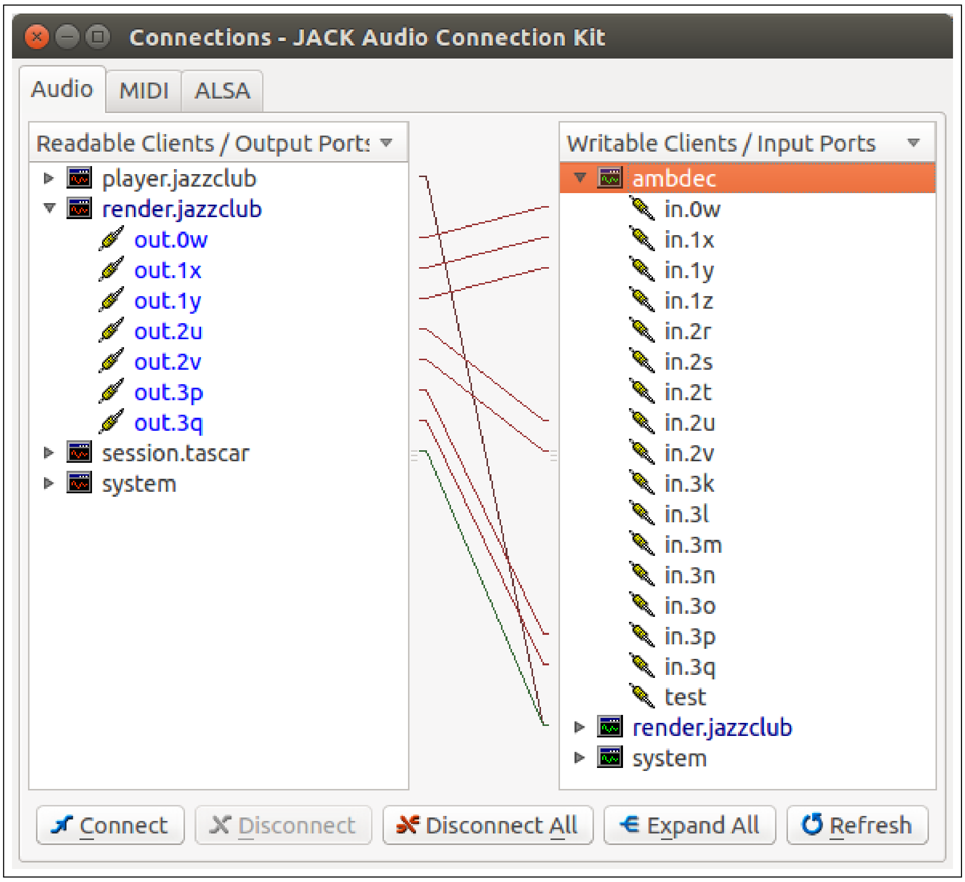


Figure 6: Connections, which are created, when using attribute `connect` in the element `<sound/>`

5.6.5 cardioid

Cardioid microphone simulation.

```
<receiver type="cardioid"/>
```

If we use a cardioidal receiver, then sources are multiplied with a different weight (depending on the direction of arrival, according to cardioidal directivity pattern) and at the output of the

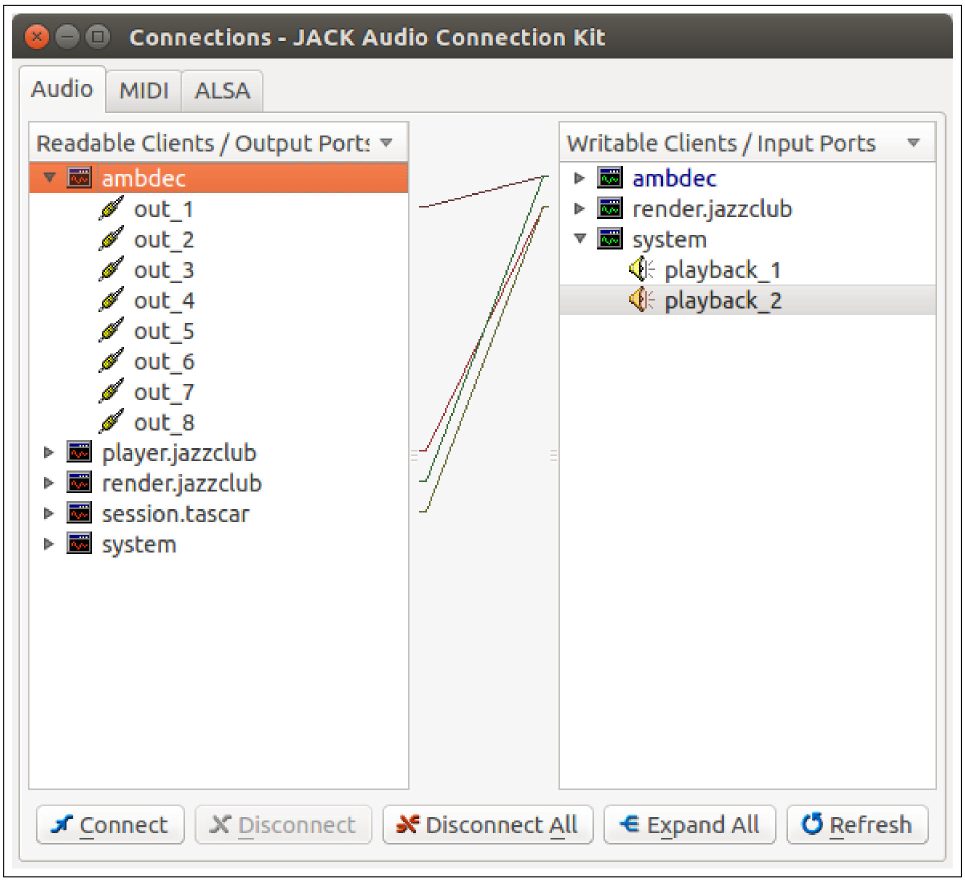


Figure 7: Ambdec output ports for a horizontal octagon

renderer we will also get just one channel - a summation of sources coming from different directions multiplied with different weights.

$N = 1$, $w_n = \frac{1}{2}(cos(\alpha) + 1)$,
where $w(n)$ depends on the angle between the source and the receiver, α (direction from which the source is coming).

5.6.6 chmap

Channel mapping receiver type. Each (primary or image) sound source is rendered to a different channel. If more sources than channels are active, then the channels are wrapped around.

Attributes of receiver element **chmap**, inheriting from **receiver**

name	description (type, unit)	def.
`channels`	number of output channels (uint32)	1

5.6.7 debugpos

Instead of the audio signal, the relative source position in cartesian coordinates is returned.

Attributes of receiver element **debugpos**, inheriting from **receiver**

name	description (type, unit)	def.
`sources`	number of sources to output (uint32)	1

5.6.8 fakebf

Beam former simulating receiver type, to simulate the directional effects of a beamformer on the rendering side.

Attributes of receiver element **fakebf**, inheriting from **receiver**

name	description (type, unit)	def.
`angle`	Angular distance between microphone axes (double, deg)	110
`c`	Speed of sound (double, m/s)	340
`distance`	Microphone distance (double, m)	0.17
`sincorder`	Sinc interpolation order of ITD delay line (uint32)	0
`start_angle`	Angle at which attenutation ramp starts (double, deg)	0
`stop_angle`	Angle at which full attenutation is reached (double, deg)	90

Attributes:

`distance`	Microphone distance in meter (0.17)
`angle`	Angular distance between microphone axes in degrees (110)
`start_angle`	Angle at which attenutation ramp starts, in degrees (0)
`stop_angle`	Angle at which full attenutation is reached, in degrees (90)
`sincorder`	Sinc interpolation order of ITD delay line (0)
`c`	Speed of sound in m/s (340)

5.6.9 hoa2d_fuma

Horizontal higher order ambisonics encoder with FuMa normalization and ACN channel sequence.

Attributes of receiver element **hoa2d_fuma**, inheriting from **receiver**

name	description (type, unit)	def.
`diffup`	Use diffuse upsampling similar to Zotter et al. (2014) (bool)	false
`diffup_delay`	Decorrelation delay (double, s)	0.01
`diffup_maxorder`	Maximum order of diffuse sound fields (uint32)	100
`diffup_rot`	Decorrelation rotation (double, deg)	45

`filterperiod`	Filter period for source width encoding (double, s)	0.005
`filtershape`	De-correlation filter shape for source width encoding, one of “none”, “notch”, “sine”, “tria”, “triald” (string)	none
`order`	Ambisonics order; 0: use maximum possible (uint32)	0

5.6.10 hoa3d_enc

Higher order Ambisonics encoder (3D) with SN3D normalization and ACN channel sequence.

Attributes of receiver element **hoa3d_enc**, inheriting from **receiver**

name	description (type, unit)	def.
`order`	Ambisonics order (int32)	3

Attributes:	
`order`	Ambisonics order

5.6.11 hrtf

HRTF simulation.

This receiver describes the main features of measured head related transfer functions by using a few low-order digital filters. The parametrization is based on the Spherical Head Model (SHM) by Brown and Duda (1998) and includes three further low-order filters.

The SHM introduces an approach to model the head as a rigid sphere. It includes a model for the head shadow effect as well as a method to compute the interaural time difference. The head shadow effect is approximated by a first-order high-shelf filter which depth varies depending on the incident angle. The high-shelf can be described by means of three parameters: The cut-off frequency `omega`, the angle `thetamin` at which the maximal depth of the high-shelf is reached and the parameter `alphamin` which influences the maximal reached depth of the high-shelf.

The Duda SHM was extended by O. Buttler and S.D. Ewert in the context of room acoustics simulator RAZR (Wendt et al., 2014; Ewert, 2018) in Buttler (2018) to improve left-right, front-back, and elevation perception:

i) a pre-warping of the azimuth angles is introduced to better match experimentally observed interaural level differences as a function of azimuth, particularly in the frontal region.

ii) Two further first-order high-shelf filters similarly to that which realizes the SHM are used to model pinna - respectively torso - shadow. These filters are as well described by three parameters. The two parameters `alphamin_front` and `omega_front` – respectively `alphamin_up` and `omega_up` – are used in the same way as described for the SHM. However, the third parameter `startangle_front` – respectively `startangle_up` –, which is defined with respect to a certain reference direction (front [1 0 0] – respectively up [0 0 1]), is used in order to define a region of incident directions in which these filters are applied. The maximal depth is reached at 180 degrees with respect to the reference direction.

iii) Furthermore, a notch filter is used in order to reproduce the concha notch which provides an important feature in order to distinguish between elevation angles. This filter is applied in the upper hemisphere for angles smaller than `startangle_notch`. In order to have a smooth transition, the gain of the notch increases linearly from 0 dB at `startangle_notch` to the `maxgain` for an incidence direction directly above the head. Moreover, the center frequency is chosen to vary linearly over the range as well. At `startangle_notch` the center frequency is equal to `freq_start` as changes linearly to `freq_end` for incidence direction right above the head. Furthermore, the notch is described by the quality factor `Q_notch`.

In order to optimize the values for the filter parameters of the original RAZR SHM-Model, the frequency response of the receiver has been fitted to measured HRTFs of the KEMAR dummy head (Schwark, 2020) provided by the OlHeaD-HRTF database (Denk and Kollmeier, 2020).

Attributes of receiver element **hrtf**, inheriting from **receiver**

name	description (type, unit)	def.
`Q_notch`	quality factor of the notch filter (float)	2.3
`alphamin`	parameter which determines the depth of the high-shelf realizing the SHM (float)	0.14
`alphamin_front`	parameter which determines the depth of the second high-shelf (float)	0.39
`alphamin_up`	parameter which determines the depth of the second high-shelf (float)	0.1
`angle`	Position of the ears on the sphere (float, deg)	90
`c`	Speed of sound (float, m/s)	340
`decorr`	Flag to use decorrelation of diffuse sounds (bool)	false
`decorr_length`	Decorrelation length (float, s)	0.05
`diffuse_hrtf`	apply hrtf model also to diffuse rendering (bool)	false
`freq_end`	notch center frequency at [0 0 1] (float, Hz)	650
`freq_start`	notch center frequency at `startangle_notch` (float, Hz)	1300
`gaincorr`	channel-wise gain correction (float array, dB)	0 0
`maxgain`	gain applied at [0 0 1] – gain is 0 dB at `startangle_notch` and increases linearly (float, dB)	-5.4
`omega`	cut-off frequency of the high-self realizing the SHM (float, Hz)	3100
`omega_front`	cut-off frequency of the second high-self (float, Hz)	11200
`omega_up`	cut-off frequency of the second high-shelf in Hz (float, Hz)	2125
`prewarpingmode`	Azimuth pre-warping mode, 0 = original, 1 = none, 2 = corrected (uint32)	0
`radius`	Radius of sphere modeling the head (float, m)	0.08
`sincorder`	Sinc interpolation order of ITD delay line (uint32)	0
`sincsampling`	Sinc table sampling of ITD delay line, or 0 for no table. (uint32)	64
`startangle_front`	the second high-shelf, e.g. to model pinna shadow effect, is applied when the angle with respect to front direction [1 0 0] is larger than `startangle_front` (float, deg)	0
`startangle_notch`	notch filter to model concha notch is applied if angle with respect to up direction [0 0 1] is smaller than `startangle_notch` (float, deg)	102
`startangle_up`	the third high-shelf which models the shadow effect of the torso is applied when the angle with respect to up direction [0 0 1] is larger than `startangle_up` (float, deg)	135

`thetamin`	angle with respect to the position of the ears at which the maximum depth of the high-shelf realizing the SHM is reached (float, deg)	160

OSC variables:

path	fmt.	range	r.	description
`/.../Q_notch`	f		yes	
`/.../alphamin_front`	f		yes	
`/.../alphamin_up`	f		yes	
`/.../alphamin`	f		yes	
`/.../angle`	f		yes	
`/.../decorr`	i	bool	yes	
`/.../diffuse_...`	i	bool	yes	
`/.../freq_end`	f		yes	
`/.../freq_start`	f		yes	
`/.../gaincorr`	ff		no	channel-wise gain correction
`/.../maxgain`	f		yes	
`/.../omega_front`	f		yes	
`/.../omega_up`	f		yes	
`/.../omega`	f		yes	
`/.../prewarpingmode`	i	[0,1,2]	yes	pre-warping mode, 0 = original, 1 = none, 2 = corrected
`/.../radius`	f		yes	
`/.../startangle_front`	f		yes	
`/.../startangle_notch`	f		yes	
`/.../startangle_up`	f		yes	
`/.../thetamin`	f		yes	

5.6.12 intensityvector

This specialized receiver type accumulates the sound intensity weighted direction. This receiver type is used only for analysis and characterization of acoustic scene properties. Its only attribute is the intensity integration time constant `tau`, measured in seconds, with the default value of 0.125.

Attributes of receiver element **intensityvector**, inheriting from **receiver**

name	description (type, unit)	def.
`tau`	intensity integration time constant (double, s)	0.125

5.6.13 itu51

This receiver renders for ITU 5.1 loudspeaker layouts. Point sources are panned using VBAP on the C, L, R, Ls and Rs speakers. A warped space is used (0° mapped to C, ±45° mapped to L and R, ±135° mapped to Ls and Rs) to achieve a stable image in the frontal speaker set and to avoid excess intensities on the rear speakers. Diffuse sounds are rendered to L, R, Ls and Rs speakers, without de-correlation of the speaker signals. The LFE channel is created using an omni-directional characteristics (both, point sources and diffuse sound fields), and low pass filtered.

Attributes of receiver element **itu51**, inheriting from **receiver**

name	description (type, unit)	def.
`diffusegainfront`	Diffuse gain for frontal speakers (double, dB)	-6.0206
`diffusegainrear`	Diffuse gain for rear speakers (double, dB)	0
`fc`	LFE cut off frequency (double, Hz)	80

5.6.14 micarray

Microphone array simulation.

This receiver implements a hierarchic parametric multi-microphone (head-)model. The (relative) transfer functions are parameterized by a filter and a delay model. For each node of the hierarchic structure a delay model needs to be chosen (default freefield). A filter model can be defined by setting a single or multiple filter models. Multiple filter models are applied in a cascade. If no filter model is set, the transfer functions corresponds to a pure delay component.

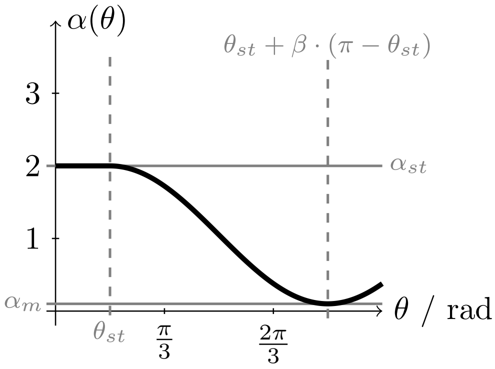
At the top level, only a single microphone can be added, typically representing the origin. This signal may need to be discarded later.

Two filter types are implemented:

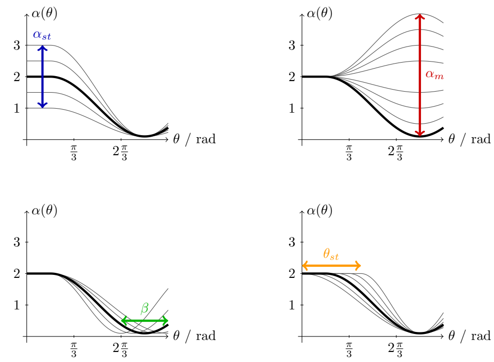
i) A High-Shelf Filter (`highshelf`)

The spatial design of this filter is an adapted version of the Spherical Head Model by Brown and Duda (1998). As proposed by Brown and Duda, a first order high-shelf is created by the single pole-zero pair $s_p = -2\omega$ and $s_z = \frac{-2\omega}{\alpha(\theta)}$. However, the design function $\alpha(\theta)$ is adopted and additional parameters are added to allow more flexibility in the filter design. Adaptation of the design function results in the following:

$$\alpha(\theta) = \left(\frac{\alpha_{st}}{2} + \frac{\alpha_m}{2}\right) + \left(\frac{\alpha_{st}}{2} - \frac{\alpha_m}{2}\right) \cdot \cos\left(\frac{\theta - \theta_{st}}{\beta \cdot (\pi - \theta_{st})} \cdot \pi\right) \forall\, \theta \leq \theta_{st} \tag{2}$$



(a) Exemplary course of $\alpha(\theta)$.



(b) Influence of each parameter on $\alpha(\theta)$.

Table 34: Relation between design parameters and course of the function $\alpha(\theta)$.

Table 34 shows how the four parameters `alpha_st`, `alpha_m`, `theta_st` and `beta` of this filter type can be used to vary the course of the design function and thus the spatial design of the filter. Furthermore, the frequency `omega` is an additional parameter of this filter type. By varying the frequency `omega` the position of the pole and the zero are varied and the range in which the high-shelf is applied is adjusted. Moreover, the orientation `axis` of the filter can be chosen freely. The angle θ is then computed with respect to the specified orientation `axis`.

Attributes of filter element **highshelf**

name	description (type, unit)	def.
`alpha_m`	alpha at theta = beta*(pi-theta_st) (double)	nan
`alpha_st`	alpha for all theta < theta_st (double)	nan
`axis`	orientation axis for filter parameter variation relative to receiver orientation (pos)	0 0 0
`beta`	parameter to determine angle at which alpha = alpha_m (double)	nan
`omega`	cut-off frequency of high-shelf (double, Hz)	nan
`theta_st`	angle at which the zero position starts to vary (double, rad)	nan
`type`	filter model type (string)	

ii) A Parametric Equalizer (`equalizer`)

With the aid of a second-order parametric equalizer a cut or boost can be created around a certain center frequency. The spatial design of the parametric equalizer is a continuous variation in center frequency and gain. The design is defined with respect to a freely selectable orientation `axis`. The gain `gain_st` is applied in the direction of this orientation `axis`. Moreover, the gain of the parametric equalizer is equal to `gain_end` at the angle `theta_end`. The gain is continuously varied in between. The center frequency of the parametric equalizer is continuously varied between the starting value `omega_st` at the orientation `axis` and the end value `omega_end` at the angle `theta_end`.

Attributes of filter element **equalizer**

name	description (type, unit)	def.
`Q`	quality factor (double)	nan
`axis`	orientation axis for filter parameter variation relative to receiver orientation (pos)	0 0 0
`gain_end`	gain applied for all theta >= theta_end (double, dB)	nan
`gain_st`	gain applied at theta = 0 rad (double, dB)	nan
`omega_end`	center frequency for theta >= theta_end (double, Hz)	nan
`omega_st`	center frequency at theta = 0 rad (double, Hz)	nan
`theta_end`	angle until which the gain is varied (double, rad)	nan
`type`	filter model type (string)	

It can be chosen between two delay models:

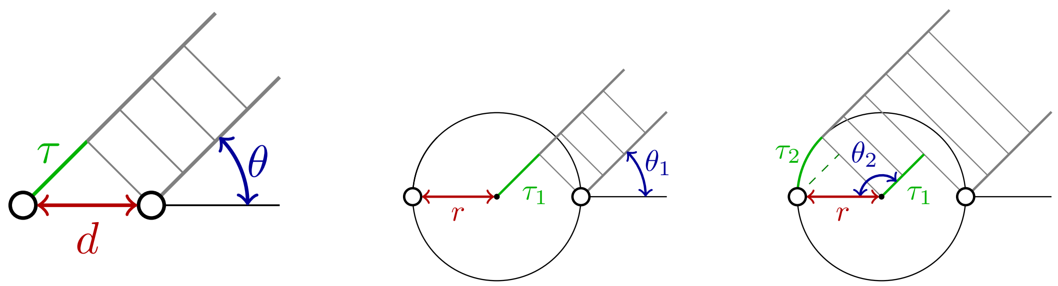
i) Free-Field (`freefield`)

This delay model determines the delay between two microphones in the free field.

ii) Sphere (`sphere`)

This delay models the delay of a microphone positioned on a sphere. The used formula is the model proposed by Brown and Duda (1998) for modeling the interaural time delay for the Spherical Head Model.

$$\tau = \frac{d \cdot \cos\theta}{c} f_s \;\forall\, \theta \qquad \tau_1 = \frac{r \cdot \cos\theta}{c} f_s \;\forall\, \theta < \frac{\pi}{2} \qquad \tau_2 = \frac{r \cdot (\theta - \frac{\pi}{2})}{c} f_s \;\forall\, \theta \geq \frac{\pi}{2}$$



(a) Free-field (b) Sphere

Table 37: Used formulas and graphical representation of the delay models.

Table 37 shows the graphical representation as well as provides the used formulas for the computation of the delay models. The delay model is always applied with respect to the parent microphone.

Attributes of receiver element **micarray**, inheriting from **receiver**

name	description (type, unit)	def.
`c`	speed of sound (double, m/s)	340

Attributes of element **mic**

name	description (type, unit)	def.
`delay`	delay line model type, "freefield" or "sphere" (string)	freefield
`name`	microphone label (string)	
`position`	microphone position relative to receiver origin (pos, m)	0 0 0
`sincorder`	Sinc interpolation order of delay line (double)	0
`sincsampling`	Sampling of sinc table, or 0 for direct calculation (uint32)	64

An example of a binaural microphone array is shown below. Note that the first microphone definition (line 2) serves only as a reference microphone whose signal is discarded. On each side of the head, one microphone is selected as the reference of a local microphone array (lines 3 and 8), which uses a spherical head model and head shadow filters. The other microphones (lines 5, 6, 10 and 11) are calculated relative to the left and right reference microphones, using only a free field delay for the relative transfer function.

```
1  <receiver type="micarray" name="out">
2    <mic delay="freefield" position="0 0 0">
3      <mic name="left middle" delay="sphere" sincorder="1" position="0.0 0.083
     0.0">
4        <filter type="highshelf" axis="-0.14 0.95 0.29" theta_st="0.59"
     beta="0.98" omega="2725.0" alpha_st="1.53" alpha_m="0.07"/>
5        <mic name="left front" delay="freefield" sincorder="1" position="0.0076
     0.083 0.0"/>
6        <mic name="left rear" delay="freefield" sincorder="1" position="-0.0073
     0.083 0.0"/>
7      </mic>
8      <mic name="right middle" delay="sphere" sincorder="1" position="0.0 -0.083
     0.0">
9        <filter type="highshelf" axis="-0.14 -0.95 0.29" theta_st="0.59"
     beta="0.98" omega="2725.0" alpha_st="1.53" alpha_m="0.07"/>
10       <mic name="right front" delay="freefield" sincorder="1" position="0.0076
     -0.083 0.0"/>
11       <mic name="right rear" delay="freefield" sincorder="1" position="-0.0073
     -0.083 0.0"/>
12     </mic>
13   </mic>
14 </receiver>
```

5.6.15 omni

Omnidirectional microphone.

```
<receiver type="omni"/>
```

If we use the simple omni-directional receiver, then sources coming from all directions are rendered with the same weight $w = 1$ and at the output of the renderer we will get just one channel, $N = 1$ - the summation of sources from all directions:

$N = 1$, $w_n = 1$

5.6.16 ortf

This receiver implements a classic ORTF stereo microphone technique. The cardioid microphone characteristic is frequency dependent; the 6 dB cut-off frequency for 90 degrees is specified by the attribute `f6db`. The attribute `fmin` defines the cut-off frequency for sources from 180 degrees angle of incidence. To disable the frequency dependence and use a broadband cardioid polar pattern instead, use the attribute `broadband="tue"`. The attributes `distance` and `angle` control the microphone geometry.

Typical values for small diaphragm microphones are `f6db="3000"` and `fmin="800"` (these are the default values since version 0.172.2); for higher directivity use `f6db="1000"` and `fmin="60"` (default values for earlier versions).

Attributes of receiver element **ortf**, inheriting from **receiver**

name	description (type, unit)	def.
`angle`	Angular distance between microphone axes (double, deg)	110
`attscale`	Scaling factor for cosine attenuation function (double)	1
`broadband`	Use broadband cardioid characteristics (bool)	false
`c`	Speed of sound (double, m/s)	340
`decorr`	Flag to use decorrelatin of diffuse sounds (bool)	false
`decorr_length`	Decorrelation length (double, s)	0.05
`distance`	Microphone distance (double, m)	0.17
`f6db`	6 dB cutoff frequency for 90 degrees (double, Hz)	3000
`fmin`	Cutoff frequency for 180 degrees sounds (double, Hz)	800
`sincorder`	Sinc interpolation order of ITD delay line (uint32)	0
`sincsampling`	Sinc table sampling of ITD delay line, or 0 for no table. (uint32)	64

OSC variables:

path	fmt.	range	r.	description
`/.../angle`	f		yes	Angular distance between microphone axes, in degree
`/.../attscale`	f		yes	Scaling factor for cosine attenuation function
`/.../decorr`	i	bool	yes	Flag to use decorrelatin of diffuse sounds
`/.../distance`	f		yes	Microphone distance, in m

Example:

```
<receiver type="ortf" f6db="3000" fmin="80" distance="0.17" angle="110"/>
```

5.6.17 vmic

Generic first-order microphone, directivity can be controlled between omni and figure-of-eight.

```
<receiver type="vmic" a="0"/>
```

The virtual microphone receiver type has a single output channel. The driving weight is

$$w = 1 + a(\tilde{p}_{rel,x} - 1). \tag{3}$$

Its directivity pattern can be controlled between omni-directional and figure-of-eight with the directivity coefficient a; with $a = 0$ this is an omni-directional microphone, with $a = \frac{1}{2}$ a standard cardioid, and with $a = 1$ a figure-of-eight. The diffuse decoding matrix is

$$\mathbf{D} = \begin{pmatrix} \sqrt{2}(1-a) & a & 0 & 0 \end{pmatrix}. \tag{4}$$

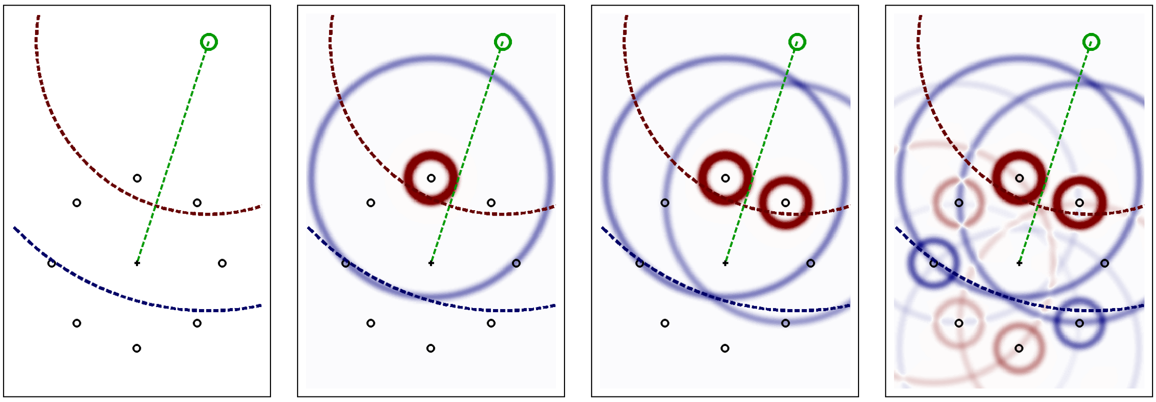
The factor $\sqrt{2}$ of the w-channel is needed to account for the Furse-Malham normalization of the diffuse signals.

Attributes of receiver element **vmic**, inheriting from **receiver**

name	description (type, unit)	def.
a	directivity coefficient (double)	0

5.7 Loudspeaker-based receiver types

In addition to the generic receiver types, there are also loudspeaker-based decoding methods (VBAP, Ambisonics Panning and Nearest Speaker Panning). These require the specification of the loudspeaker layout, i.e. their positions and, optionally, calibration data.



(a) original (b) nsp (c) vbap (d) hoa2d

Figure 8: Schematic representation of the reproduced sound fields with different reproduction methods.

The loudspeaker layout of loudspeaker-based receiver types can be defined in a separate layout file specified in the `layout` attribute, or in a list of `<speaker/>` elements within the receiver definition. Each of the `<speaker/>` entries has the following attributes:

Attributes of element **speaker**

name	description (type, unit)	def.
`az`	Azimuth (double, deg)	0
`calibrate`	Use this loudspeaker during calibration (bool)	true
`compB`	FIR filter coefficients for speaker calibration (double array)	
`connect`	Connection to jack port (string)	
`conv`	Name of impulse response for convolution (string)	
`delay`	Static delay (double, s)	0
`el`	Elevation (double, deg)	0
`eqfreq`	Frequencies for IIR filter design (float array, Hz)	
`eqgain`	Gains for IIR filter design (float array, dB)	
`eqstages`	Number of biquad-stages in IIR frequency correction (0 = disable) (uint32)	0
`gain`	Broadband gain correction (double, dB)	0
`label`	Additional port label (string)	
`r`	Distance (double, m)	1

In addition to regular broadband loudspeakers, a number of subwoofers can be defined using `<sub/>` elements with the same attributes as in the `<speaker/>` element. When subwoofers are defined, an IIR crossover filter with 24 dB/octave is applied to all signals. The subwoofer signals are spatially mapped from the broadband loudspeaker positions to the subwoofer positions using a modified DBAP (Lossius et al., 2009) method.

To enable the FIR loudspeaker correction filter, provide the FIR filter coefficients in the `compB` attribute. Note that the filter coefficients are sample rate specific and are not automatically recalculated when the sample rate is changed. The maximum length of the correction filter is the size of the audio fragment plus one.

The top-level element of a layout file, `<layout/>`, can be configured with these attributes:

Attributes of element **layout**

name	description (type, unit)	def.
`addring`	Create a circular layout with this number of speakers (uint32)	0
`addsphere`	Create a spherical layout with at least this number of speakers by barycentric subdivision (uint32)	0
`calibdate`	Calibration date in format YYYY-MM-DD (string)	
`calibfor`	Summary of receiver parameters (string)	
`caliblevel`	Calibration level (double, dB SPL)	93.9794
`checksum`	autogenerated value for validation of calibration (uint64)	0
`convlabels`	Space-separated list of labels of convolution output channels (string array)	
`convprecalib`	Apply convolution before calibration (true) or after (false). (bool)	true
`decorr`	Decorrelate speaker signals in diffuse sound field rendering (bool)	true
`decorr_length`	Length of decorrelation filter (double, s)	0.05
`densitycorr`	In diffuse rendering, correct gains locally for loudspeaker density (bool)	true
`diffusedecoder`	Diffuse-decoder method (string, basic|maxre|inphase)	maxre
`diffusegain`	Calibration gain of diffuse sound fields (double, dB)	0
`fcsub`	Cross-over frequency, used only if subwoofers are defined (double, Hz)	80
`name`	Name of layout, for documentation only (string)	
`onload`	system command to be executed when layout is loaded (string)	
`onunload`	system command to be executed when layout is unloaded (string)	
`sofa_file`	SOFA convolution file (string)	

Changing any of these attributes (except `calibdate`) may affect the output calibration and require re-calibration.

If the caliblevel is provided in the receiver element and in the layout file, a warning is issued and the value from the layout file is used. If a calibration date is provided and the calibration is older than 30 days, a warning is displayed.

As the calibration values may depend on the rendering method and its parameters, the `calibfor` attribute must be set to the correct value for the session file in which the loudspeaker layout is used. These values can be queried using the command line tool `tascar_getcalibfor` which prints out the correct value for each loudspeaker-based receiver type used in a session file.

A simple example of a loudspeaker layout file is shown in Example 7.

```
<?xml version="1.0"?>
<layout>
  <speaker el="30" az="45" r="12"/>
  <speaker el="0" az="40" r="3"/>
  <speaker el="60" az="4" r="1"/>
  <speaker el="0" az="-40" r="0"/>
</layout>
```

Example 7: examples/nsp.spk

Attributes common to all loudspeaker-based layouts are:

Attributes of element **speakerbased** (hann hoa2d hoa3d nsp vbap vbap3d wfs)

name	description (type, unit)	def.
`layout`	name of speaker layout file (string)	
`showspatialerror`	show absolute and angular error for rE and rV for 2D and 3D rendering, given the actual speaker layout and settings (bool)	false
`spatialerrorpos`	Additional point list in Cartesian coordinates for testing spatial error (pos array, m)	

For all receiver types that utilize loudspeakers, an impulse response can be designated for convolution for each loudspeaker channel, as indicated by the `conv` attribute of the `<speaker/>` element in the layout definition. If an impulse response is assigned to one channel, a corresponding impulse response with the same channel count must also be specified for all other channels.

The convolution's output will be available in supplementary output channels; you can assign the names of these channels using the `convlabels` attribute. The convolution may be carried out either prior to or following the compensation for loudspeaker gain and delay.

Bear in mind, this method is currently not compatible with layouts that include subwoofer definitions. If you wish to utilize HRTF databases in SOFA format, use the `sofa_file` attribute. At present, only binaural SOFA databases are supported. Here is an example:

```
  <scene>
    ...
    <receiver type="hoa2d">
      <layout addring="16" sofa_file="MIT_KEMAR_normal_pinna.sofa"/>
```

```
  </receiver>
</scene>
```

OSC variables:

path	fmt.	range	r.	description
/.../decorr	i	bool	yes	
/.../densitycorr	i	bool	yes	

List of speaker based receiver types:

- **hann**
- **hoa2d**
- **hoa3d**
- **nsp**
- **stereo**
- **vbap**
- **vbap3d**
- **wfs**

5.7.1 hann

Panning of audio between two best-matching speakers with von-Hann ramps.

```
<receiver type="hann" wexp="0.5">...</receiver>
```

If N speakers are defined, α is the angle between a speaker k and the virtual sound source, and γ is the window exponent (`wexp`), then the speaker gain g_k is

$$w_k = \left(\frac{1}{2} + \frac{1}{2}\cos\left(\frac{N}{2}\alpha\right)\right)^\gamma \quad (5)$$

Attributes of receiver element **hann**, inheriting from **receiver speakerbased**

name	description (type, unit)	def.
wexp	window exponent γ (double)	0.5

5.7.2 hoa2d

Horizontal higher-order Ambisonics with embedded decoder, for regular loudspeaker layouts.

```
<receiver type="hoa2d" order="3" maxre="true">...</receiver>
```

This receiver type provides horizontal higher order ambisonics with basic or $\max r_E$ decoding. If `order` is zero or unset, then the maximum possible order for the given number of loudspeakers is used.

Attributes of receiver element **hoa2d**, inheriting from **receiver speakerbased**

name	description (type, unit)	def.
`diffup`	Use diffuse upsampling similar to Zotter et al. (2014) (bool)	false
`diffup_delay`	Decorrelation delay (double, s)	0.01
`diffup_maxorder`	Maximum order of diffuse sound fields (uint32)	100
`diffup_rot`	Decorrelation rotation (double, deg)	45
`filterperiod`	Filter period for source width encoding (double, s)	0.005
`filtershape`	De-correlation filter shape for source width encoding, one of “none”, “notch”, “sine”, “tria”, “triald” (string)	none
`maxre`	Use $\max r\_E$ decoder (true) or basic decoder (false) (bool)	false
`order`	Ambisonics order; 0: use maximum possible (uint32)	0

OSC variables:

path	fmt.	range	r.	description
`/.../diffup_delay`	f		yes	
`/.../diffup_maxorder`	i		yes	
`/.../diffup_rot`	f	[0,360]	yes	
`/.../diffup`	i	bool	yes	

Note:

Only regular speaker arrays can be used. Explicit speaker distributions are ignored, and a regular speaker distribution with counter-clockwise azimuths is assumed, with the first speaker starting at the value provided in the `rotation` attribute. If the `rotation` attribute is not given, then the average difference between a regular layout and the explicit speaker azimuth is taken as `rotation`.

If `diffup` is set to “true”, diffuse-decoding is using the internal decoder, which is also used for decoding of panned sources. If `diffup` is set to “false”, the standard speaker-based diffuse render method is applied. Source-width encoding splits the signal into two uncorrelated signals and creates virtual sound sources separated by the source width.

An alternative receiver type `hoa2d_fuma` can be used to return the encoded signal in FuMa normalization and ACN channel sequence.

5.7.3 hoa3d

Higher order Ambisonics receiver (3D) with embedded decoder, for arbitrary 3D speaker layouts.

Attributes of receiver element **hoa3d**, inheriting from **receiver speakerbased**

name	description (type, unit)	def.
`dectype`	Decoder type, “basic”, “maxre” or “inphase” (string)	maxre
`decwarnthreshold`	Warning threshold for decoder matrix abs/rms ratio (double)	8
`method`	Decoder generation method, “pinv” or “allrad” (string)	pinv
`order`	Ambisonics order (int32)	3
`savedec`	Save Octave/Matlab script for decoder matrix debugging (bool)	false

Either the Ambisonics mode matching method using the pseudo-inverse of the encoding matrix can be used, `method="pinv"`, or the ALLRAD method via regular virtual speakers rendered with VBAP, `method="allrad"`. See Daniel (2001) and Heller et al. (2012); Heller and Benjamin (2014) for details; the decoding methods have been validated against the Ambisonics Decoder Toolbox (Heller and Benjamin, 2014). Except for minor differences in the underlying triangulation method the results are comparable.

Note:

No automatic order calculation from based on the loudspeaker layout is applied, thus it is always required to configure the correct Ambisonics order.

Note:

With AllRAD decoder, the triangulation of the speaker layout may differ depending on the operating system and version due to different numerical resolutions. This can lead to different speaker channel signals, but the effects on perception should be negligible.

5.7.4 nsp

Nearest speaker selection, i.e., always a single speaker is used to render a virtual sound source. In case of moving sources or receivers, the transition between two speakers will be linearly interpolated within one audio block.

```
<receiver type="nsp"><speaker az="0"/>...</receiver>
```

This receiver also requires defining the position of the playback channels and we can do it in the following way (see `example_nearest.tsc`):

```
    <receiver name="nearestspeaker" type="nsp" layout="nsp.spk">
      <position>0 0 0 1.6</position>
      <orientation>0 34 0 0</orientation>
    </receiver>
```

Example 8: examples/example_nearest.tsc

```
<layout>
  <speaker el="30" az="45" r="12"/>
  <speaker el="0" az="40" r="3"/>
  <speaker el="60" az="4" r="1"/>
  <speaker el="0" az="-40" r="0"/>
</layout>
```

Example 9: examples/nsp.spk

If we load a scene with such a receiver in TASCAR, we will see all the specified channels as an output of the rendering stage in the Jack Audio. However, this time, for each source there is only one channel which is active, i.e. the one for which there is the lowest angular distance from the loudspeaker to the source.

The attribute `useall` activates all speakers independent of the source position.

Attributes of receiver element **nsp**, inheriting from **receiver speakerbased**

name	description (type, unit)	def.
`useall`	activate all speakers independent of source position (bool)	false

OSC variables:

path	fmt.	range	r.	description
`/.../useall`	i	bool	yes	

5.7.5 stereo

Simple stereo receiver based on VBAP

```
<receiver type="stereo" layout="stereo.spk"/>
```

This module is inheriting from speaker based receiver methods and has no specific attributes.

5.7.6 vbap

2-dimensional VBAP.

```
<receiver type="vbap" layout="spkeaker.spk"/>
```

This module inherits from speaker based receiver methods and has no specific attributes. Note that 2-dimensional VBAP only works with flat layout files, i.e. all elevation angles must be zero, which is the default.

5.7.7 vbap3d

3-dimensional VBAP Pulkki (1997).

```
<receiver type="vbap3d" layout="spkeaker3d.spk"/>
```

This module inherits from speaker based receiver methods and has no specific attributes. Note that 3-dimensional VBAP only works with non-flat layout files, i.e. the convex hull must cover the origin.

Note:

The triangulation of the speaker layout may differ depending on the operating system and version due to different numerical resolutions. This can lead to different speaker channel signals, but the effects on perception should be negligible.

5.7.8 wfs

This receiver defines a very simple WFS renderer. Loudspeaker distance is compensated for planar source wave fronts. The gain is proportional to the cosine of the angle between source and speaker, for angles smaller than 90 degrees, and zero otherwise.

Attributes of receiver element **wfs**, inheriting from **receiver speakerbased**

name	description (type, unit)	def.
`c`	Speed of sound (float, m/s)	340
`planewave`	Simlate always plane waves independent of distance (bool)	true

OSC variables:

path	fmt.	range	r.	description
`/.../planewave`	i	bool	yes	

5.8 Adding diffuse reverberation: `<reverb .../>`

To generate diffuse reverberation in TASCAR, signal components from the image source model must be transferred to the diffuse sound field model. There are two options for this: Either external reverberation generators can be used, which receive their input signals via JACK and also reproduce the reverberation signal in Ambisonics format via JACK. For this, a `<receiver/>` must be used to transmit from the image source model to the external reverberation module, and a diffuse sound field (`<diffuse/>`) must be used to transmit from the external reverberation module to TASCAR. Another option is to use the TASCAR internal reverberation generators. In this method, the `<reverb/>` element combines both the receiver and the diffuse sound field into a single object. The type of the reverb plugin is specified with the `type` attribute, which can be any of the types listed below. Diffuse sources do not contribute to diffuse reverberation.

Both methods have in common that the receiver does not follow the normal distance laws, but renders all sources within a given volume with equal gains and delays. This is achieved by the attribute `volumetric`; this attribute defines the shoebox-shaped volume in which sound sources contribute to the reverberation.

Under the hood, the **<reverb/>** element combines a dedicated receiver with a diffuse sound field. Therefore, most common attributes of receivers (see 5.5) can be used here as well. To create diffuse reverberation with external convolution or algorithmic reverb tools connected via jack, you may use a receiver plugin, set the `volumetric` attribute, and add a diffuse sound field using the same position, orientation and dimension as the volumetric receiver.

A very basic FDN and a partitioned convolution module are provided as part of TASCAR. An example of diffuse reverberation using the “simplefdn” plugin looks like this:

```
    <reverb type="simplefdn" volumetric="3 3 3" image="false">
      <position>0 0 0 1.5</position>
    </reverb>
```

Example 10: examples/example_diffreverbnew.tsc

The attribute “volumetric” defines the shoe-box shaped volume in which sound sources are rendered.

Reverb receivers have the attribute `layers`, which defines the layers in which the receiver receives sound, and `outputlayers`, which defines the layers in which the diffuse sound field is reproduced.

Attributes of element **reverb** (foaconv simplefdn), inheriting from **objects routes**

name	description (type, unit)	def.
`avgdist`	Average distance which is assumed inside receiver boxes, or 0 to use $(\frac{1}{8}V)^{1/3}$ (float, m)	0
`caliblevel`	calibration level (float, dB SPL)	93.9794
`connect`	Regular expressions of port names for connections (string array)	
`delaycomp`	subtract this value from delay in delay lines (float, s)	0
`diffuse`	render diffuse input sound fields (bool)	false
`fade_gain`	linear fade gain (float)	1
`falloff`	Length of von-Hann ramp at volume boundaries, or -1 for normal distance model (float, m)	-1
`gain`	port gain (float, dB)	0
`globalmask`	use global mask (bool)	true
`image`	render image sources (bool)	true
`inv`	phase invert (bool)	false
`ismmax`	maximal ISM order to render (uint32)	2147483647
`ismmin`	minimal ISM order to render (uint32)	0
`layerfadelen`	duration of fades between layers (float, s)	1
`layers`	render layers (bits32)	all
`muteonstop`	mute when transport stopped to prevent playback of sounds from delaylines and reverb (bool)	false
`outputlayers`	output layers (bits32)	all

`proxy_airabsorption`	Use proxy position for air absorption (bool)	false
`proxy_delay`	Use proxy position for delay (bool)	false
`proxy_direction`	Use proxy position for direction (bool)	false
`proxy_gain`	Use proxy position for gain (bool)	false
`proxy_is_relative`	Proxy is relative to receiver (true) or in absolute coordinates (false) (bool)	false
`proxy_position`	Proxy position (pos, m)	0 0 0
`scatterdamping`	damping of scatter reflection filter (float)	0
`scatterreflections`	Number of reflections created by scattering filter (uint32)	0
`scatterspread`	Spatial spread of scattering (float, deg)	22.5
`scatterstructuresize`	size of scatter structure (float, m)	1
`type`	receiver type (string)	omni
`volumetric`	volume in which receiver does not apply distance based gain model (pos, m)	0 0 0
`volumetricgainwithdistance`	For volumetric receivers, increase gain with distance (bool)	false

List of reverb receiver types:

- **foaconv**
- **simplefdn**

5.8.1 foaconv

This receiver implements a partitioned convolution with First Order Ambisonics (FOA) impulse responses.

Attributes of reverb element **foaconv**, inheriting from **reverb**

name	description (type, unit)	def.
`channelorder`	Channel order of FOA response, either “FuMa” (wxyz) or “ACN” (wyzx) (string)	ACN
`irsname`	Name of IRS sound file (string)	
`maxlen`	Maximum length of IRS, or 0 to use full sound file (uint32, samples)	0
`normalization`	Normalization of FOA response, either “FuMa” or “SN3D” (string)	FuMa
`offset`	Offset of IR in sound file (uint32, samples)	0

5.8.2 simplefdn

This receiver implements a simple Feedback Delay Network (FDN) based on Schroeder (1962) and Rocchesso and Smith (1997). It uses a first order Ambisonics sound field for each audio sample, and applies a rotation at each reflection.

To set the room dimensions, use the `volumetric` attribute. By default, the T_{60} is calculated using the Sabine's formula, see `absorption`. If an explicit T_{60} is provided, this is used and the `absorption` attribute is ignored.

If the variables `vcf` and `vt60` are specified, an iterative optimization process will be started. Resulting optimized parameters will be printed at the console and can be used for further usage as long as the sampling rate or other parameters of the plugin are not altered.

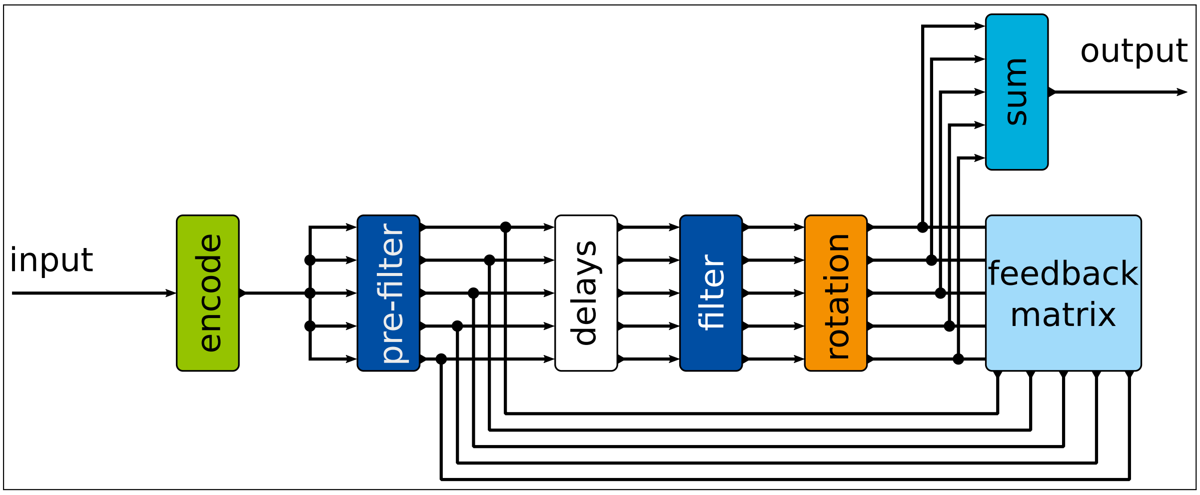


Figure 9: Signal flow in the FDN module. Each line corresponds to a First Order Ambisonics signal.

Attributes of reverb element **simplefdn**, inheriting from **reverb**

name	description (type, unit)	def.
`absorption`	Absorption used in Sabine's equation (float)	0.6
`c`	Speed of sound (float, m/s)	340
`damping`	Damping (first order lowpass) coefficient to control spectral tilt of T60 (float)	0.3
`dw`	Spatial spread of rotation (float, rounds/s)	60
`fdnorder`	Order of FDN (number of recursive paths) (uint32)	5
`fixcirculantmat`	Apply fix to correctly initialize circulant feedback matrix (bool)	false
`forwardstages`	Number of feed forward stages (uint32)	0
`gainmethod`	Gain calculation method (string, original mean schroeder)	original
`lowcut`	low cut off frequency, or zero for no low cut (float, Hz)	0
`numiter`	Number of iterations in T60 optimization (uint32)	100
`prefilt`	Apply additional filter before inserting audio into FDN (bool)	true
`rallpass`	Allpass filter radius vector (requires four entries) (float array, [0,1])	0.96 0.95 0.951 0.93
`t60`	T60, or zero to use Sabine's equation (float, s)	0
`truncate_forward`	Truncate delays of feed forward path (bool)	false
`use_biquad_allpass`	Use biquad allpass filters instead of first order filters (bool)	false
`vcf`	Center frequencies for T60 optimization, or empty for no optimization (float array, Hz)	
`vt60`	T60 at specified center frequencies (float array, s)	

OSC variables:

path	fmt.	range	r.	description
/.../dim_damp_absorption	fffff		no	Set dimension (x,y,z in m), damping and absorption coefficient
/.../fixcirculantmat	i	bool	no	Fix a neglegible bug in the feedback matrix design
/.../usebiquad	i	bool	yes	Use biquad allpass filters instead of first order

The output signal has ACN channel order and FuMa normalization.

5.9 Reflectors: <face .../> and <facegroup .../> elements

TASCAR uses a geometric source model to generate early reflections. Sound sources are reflected from surfaces if they meet the visibility criteria: The primary sound source must be in the direction of the face normal of the reflector, and the closest point between the plane defined by the reflector and the sound source must be within the surface boundary, or edge diffraction must be enabled for the reflectors.

The audio signal from the image sound source is a filtered copy of the signal from the primary sound source. The reflection filters are determined by the material properties and can be specified either as filter parameters or as a material definition with frequency-dependent absorption coefficients, see below. If material definitions are used to define the reflection properties, numerical optimisation is used to find optimal filter parameters for the given sampling rate. The parameters attenuation δ and reflectivity ρ are used to define a first order recursive reflection filter[1]:

$$y(t) = \delta\, y(t - f_s^{-1}) + \rho\,(1 - \delta)\, x(t) \tag{6}$$

The **<face/>** element defines a single reflecting surface.

Attributes of element **face**, inheriting from **objects routes**

name	description (type, unit)	def.
damping	Damping coefficient (float)	0
edgereflection	Apply edge reflection in case of not directly visible image source (bool)	true
height	Height of reflector (double, m)	1
layers	render layers (bits32)	all
material	Material name, or empty to use coefficients (string)	
reflectivity	Reflectivity coefficient (float)	1
scattering	Relative amount of scattering (float)	0
vertices	List of Cartesian coordinates to define polygon surface (pos array, m)	
width	Width of reflector (double, m)	1

If the attribute vertices contains at least three coordinates then a polygon surface is constructed using the vertices list. Otherwise, a rectangular surface with the given width and

[1] Unfortunately the Equation (3) in Grimm et al. (2019) containes an error; this is the correct equation.

height is created. The vertices of the reflector are at $(0,0,0)$, $(0,w,0)$, $(0,w,h)$ and $(0,0,h)$. The face normal, i.e., the reflecting side of the surface, is pointing in positive x-axis.

In `example_reflectors.tsc` both cases are shown:

```
    <face name="triangle" vertices="0 0 0 1 2 0 2 0 0" color="#ffd500">
      <position>0 0 1.25 0.85</position>
      <orientation>0 -30 0 0</orientation>
    </face>
    <face name="rectange" width="2.4" height="0.5" color="#00ff80">
      <position>0 -0.5 0 0.85</position>
      <orientation>0 5 0 0</orientation>
    </face>
```

Example 11: examples/example_reflectors.tsc

The `<facegroup/>` element creates a group of polygon reflectors, with common reflection properties.

Attributes of element **facegroup**, inheriting from **objects routes**

name	description (type, unit)	def.
`damping`	Damping coefficient (float)	0
`edgereflection`	Apply edge reflection in case of not directly visible image source (bool)	true
`importraw`	File name of raw file containing list of polygon surfaces (string)	
`layers`	render layers (bits32)	all
`material`	Material name, or empty to use coefficients (string)	
`reflectivity`	Reflectivity coefficient (float)	1
`scattering`	Relative amount of scattering (float)	0
`shoebox`	Generate a shoebox room of these dimensions (pos, m)	0 0 0
`shoeboxceiling`	generate shoebox room with only the ceiling surface (pos, m)	0 0 0
`shoeboxfloor`	generate shoebox room with only the floor surface (pos, m)	0 0 0
`shoeboxwalls`	generate shoebox room without floor and ceiling (pos, m)	0 0 0

Reflection properties can be defined either by explicitely setting the reflection filter coefficients `damping` and `reflectivity`, or by selecting a material, previously defined within in the scene using the `<material/>` element:

Attributes of element **material**

name	description (type, unit)	def.
`alpha`	Absorption coefficients (float array)	0.013 0.015 0.02 0.03 0.04 0.05
`f`	Frequencies at which alpha is provided (float array, Hz)	125 250 500 1000 2000 4000
`name`	Name of material (string)	plaster

Some basic material definitions are built into TASCAR, see Table 62.

Some properties can be changed via OSC messages:

OSC variables:

Table 62: Absorption coefficients of built-in material definitions.

name	125 Hz	250 Hz	500 Hz	1 kHz	2 kHz	4 kHz
parquet	0.04	0.04	0.07	0.06	0.06	0.07
window	0.35	0.25	0.18	0.12	0.07	0.04
concrete	0.36	0.44	0.31	0.29	0.39	0.25
acoustic_tiles	0.05	0.22	0.52	0.56	0.45	0.32
plaster	0.013	0.015	0.02	0.03	0.04	0.05
carpet_on_concrete	0.02	0.06	0.14	0.37	0.60	0.65
metal_8mm	0.50	0.35	0.15	0.05	0.05	0.00

path	fmt.	range	r.	description
/.../damping	f	[0,1[	yes	Damping coefficient
/.../layers	i		yes	Number representing the layers. Each layer is represented by a bit, i.e., for layers 1+3 use 10
/.../reflectivity	f	[0,1]	yes	Reflectivity of object
/.../scattering	f	[0,1]	yes	Scattering coefficient

Element <facegroup/> behaves also as an object, since it also has a position and orientation in space. So if we change the position or orientation of the whole <facegroup/>, it will also relatively change for all the planes included in the <facegroup/>.

We can define a <facegroup/> in the following way (see example example_reflectors.tsc):

```
    <facegroup name="mirrors" color="#2a00ff">
      <position>0 -1 -1.25 0.85</position>
      <orientation>0 -90 0 0</orientation>
      <faces>0 0 0  1 2 0  2 0 0
      -4 -2 0  -4 0 0  -1 0 0  -1 -2 0</faces>
    </facegroup>
```

Example 12: examples/example_reflectors.tsc

First, we define the facegroup with the name, reflectivity as well as the position and orientation of the whole facegroup. Then, we use a sub-element <faces/> (not the same as <face/> !) to define the surfaces which will be included in the group. Each line has to contain the coordinates x y z for at least three vertices. Each surface is defined in one line (by specifying coordinates of the vertices of a surface). At this point in the code we shouldn't leave empty lines.

Instead of defining all the surfaces manually, they can be modeled in blender (blender 2.79 – the scripts do not yet work with blender 2.80). The meshes can be exported with:

```
tascar_blenderexport blendfile.blend
```

This will export the meshes of all blender mesh objects of the currently selected scene, or if available, from the scene named “tascar”, to the file <blendfilename>_<objectname>.raw and all curve objects to a TASCAR track file <blendfilename>_<objectname>.csv. Curve objects may set the custom property “speed” (see “Custom properties” in the “Object” tab in the blender property view), to set the speed in m/s.

It is recommended to minimize the number of faces, e.g. by using polygon faces instead of triangulated faces. Also only the acoustically relevant surfaces should be modeled, e.g. typically only the top of a table is acoustically important, but a table modeled for visualization also contains the bottom, legs and side surfaces. If all these surfaces were imported into TASCAR, one image source would be created for each of the reflectors and primary sources, which would lead to a waste of computational performance. Small structures can be better modeled using the `scattering` attribute of reflectors. Late reverberation can be modeled independently of the image source model (see Section 5.8).

When we already have a text file, where the coordinates for all the vertices are already specified, we can import it to TASCAR scene definition using an attribute `importraw`:

```
<facegroup name="mirrors" reflectivity="1" damping="0" importraw="filename.raw"/>
```

A simple shoebox shaped room can be created by setting the attribute `shoebox="x y z"` to a finite size. The size is given in x, y, z dimensions. All faces are pointing inwards.

The normal of faces, i.e., the face orientation, is relevant for the acoustics simulation: Image sources are only active if the primary source is in the direction of the face normal.

Polygons meshes are flattened by a projection on a plane which is orthogonal to the polygon normal vector.

5.9.1 Scattering

TASCAR contains a very simple scattering model. For each reflector the amount of scattering can be controlled using the `scattering` attribute. This is added to the diffuse sound field model path. By default, the spatial dispersion of the scattering is reproduced by the receiver's decorrelation stage, if enabled. Optionally, the scattering can be rendered explicitly using additional virtual sound sources added to the diffuse sound field model. This can be enabled in each receiver by setting the attribute `scatterreflections` to a number greater than zero. Reasonable values with a reasonable trade-off between computational effort and spatial dispersion are 4 to 8.

5.10 Obstacles: `<obstacle .../>` element

Obstacles are polygon meshes which can absorb sound and create diffraction at their boundaries. The diffraction pattern is only a rough approximation.

Attributes of element **obstacle**, inheriting from **objects routes**

name	description (type, unit)	def.
`aperture`	Override aperture of airy disk calculation, zero for calculation from area (float, m)	0
`importraw`	file name of vertex list (string)	
`ishole`	Simulate infinite plane with hole instead of finite surface (bool)	false
`transmission`	transmission coefficient (float)	0

An example configuration file can be found in the file `example_obstacle.tsc`. For an exact definition of the frequency response, see Equation (10) in Grimm et al. (2019). The aperture is $a = 2\sqrt{A/\pi}$, e.g., in case of an obstacle of 1 times 1 meter the aperture is 1.1284 meter.

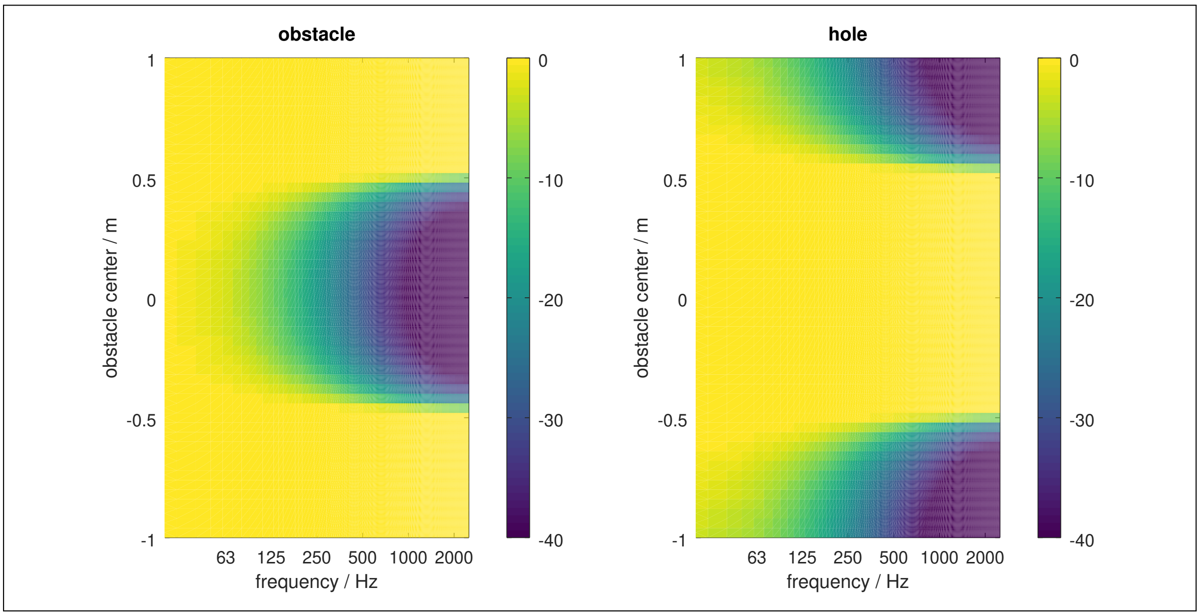


Figure 10: Frequency attenuation of a limited obstacle (left) and a hole (right) of 1 square meter, as a function of obstacle distance.

5.11 Masks: `<mask ../>` element

Global masks affect the attenuation in a receiver, based on the receiver position. See Figure 11.

Attributes of element **mask**, inheriting from **objects routes**

name	description (type, unit)	def.
`falloff`	ramp length at boundaries (double, m)	1
`inside`	mask inner objects (bool)	false
`size`	dimension of mask (pos, m)	0 0 0

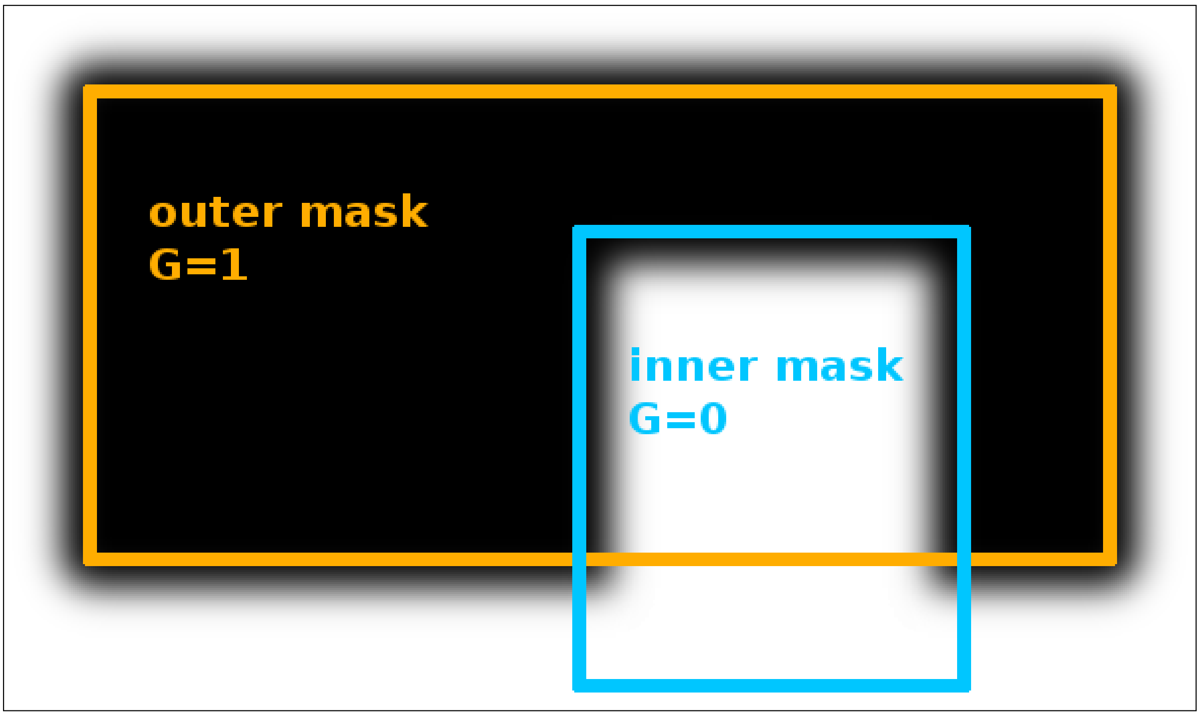


Figure 11: Global masks example.

6 General purpose modules

External modules which do not directly interact with the acoustic model of the virtual acoustic environment can be loaded as dynamic libraries. These modules may analyse or modify the session data, or simply provide some additional functionality. Modules can be added to a session file within the `modules`, e.g.,

```
<modules>
  <simplecontroller actor="/*/out" ... />
</modules>
```

List of general purpose modules:

- **datalogging**
- **dirgain**
- **echoc**
- **glabsensors**
- **granularsynth**
- **hoafdnrot**
- **hossustain**
- **hrirconv**
- **jackrec**
- **levels2osc**
- **lightcolorpicker**
- **lightctl**
- **lsl2osc**
- **lsljacktime**
- **ltcgen**
- **matrix**
- **midicc2osc**
- **midictl**
- **mididispatch**
- **osc2lsl**

- **osceog**
- **oscevents**
- **oscjacktime**
- **oscrelay**
- **oscserver**
- **route**
- **sampler**
- **savegains**
- **sleep**
- **system**
- **systime**
- **timedisplay**
- **touchosc**
- **transportgui**
- **waitforjackport**
- **waitforlslstream**

6.1 datalogging

The data logging module allows logging OSC messages and LSL data streams together with the timeline of TASCAR. Application examples are external sensors such as motion capture or bio-physical sensors such as EEG, but also control data, e.g., send from measurement applications.

Example:

```
<datalogging port="9998" multicast="" fileformat="matcell" outputdir="${HOME}">
  <osc path="/sensor1/pos" size="3"/>
  <osc path="/sensor1/rot" size="3"/>
  <osc path="/sensor3" size="4"/>
  <oscs path="/msg"/>
  <lsl predicate="name='EEGamp'"/>
</datalogging>
```

To record the data sent by a device as a series of OSC messages, the message path `path` and dimension `size` must be specified:

```
<osc path="/sensor1/pos" size="3"/>
```

The `ignorefirst` attribute can be used to hide the first channel in the display, which can be useful if the first channel contains time values or other control data. This will not affect the recording of the data.

To record the data sent by a device as an LSL stream, the LSL stream must be selected. This is done via the attribute `predicate`:

```
<lsl predicate="name='EEGamp'" tctimeout="2"/>
```

The `tctimeout` attribute is the maximum time used to measure the time correction values between sender and receiver. The `required` attribute can be set to “false” to allow TASCAR to start without requiring all LSL streams to be available. The stream will then not be restored later during the session.

Text data (e.g., trigger messages) can be recorded from LSL, or from osc with the `<oscs/>` element:

```
<oscs path="/msg"/>
```

Attributes of element **datalogging**

name	description (type, unit)	def.
`controltransport`	Control transport with recording session control (bool)	true
`displaydc`	Display DC components (bool)	true
`fileformat`	File format, can be either “mat”, “matcell” or “txt” (string)	matcell
`headless`	Use without GUI (bool)	false
`lsltimeout`	Number of seconds to scan for LSL streams (double, s)	10
`multicast`	OSC multicasting address (string)	
`outputdir`	Data output directory (string)	
`port`	OSC port, or empty to use session server (string)	
`srv_proto`	Server protocol, UDP or TCP (string)	UDP
`usetransport`	Record only while transport is rolling (bool)	false

Attributes of element **osc**

name	description (type, unit)	def.
`ignorefirst`	Ignore first value in visualization. (bool)	false
`path`	OSC path name, expecting messages with 'd' format (usedouble=true) or 'f' format. (string)	
`size`	Numer of double/float values per sample. (uint32)	1
`usedouble`	Use double precision OSC variable instead of single precision. (bool)	true

Attributes of element **oscs**

name	description (type, unit)	def.
ds_format	Use ds format, i.e., a double in addition to the string. (bool)	false
path	OSC path name, expecting messages with 's' format (string)	

Attributes of element **lsl**

name	description (type, unit)	def.
predicate	LSL stream resolving predicate, e.g., "name='EEG'" (string)	
required	Require this stream. If true, then loading will fail if stream is not available. (bool)	true
tctimeout	Time correction timeout (double, s)	2

The window size and position of the datalogging GUI can be controlled with the attributes `x`, `y`, `w` and `h`. Within the GUI, continuous data arrival is indicated with a green dot for each variable.

Depending on the content of the `fileformat` variable, the storage format differs: In the `mat` file format, each variable is stored as a matrix under the variable name. This means that it is not possible to record two streams with the same variable name. To work around this problem, the `matcell` file format can be used. Here the data is stored in a cell array, with one entry for each variable. Each entry contains a structure, with a `name` field, a `data` field and for LSL variables some additional stream information.

OSC control

Data recording can be started and stopped via OSC messages by sending a message to `/session_start` and `/session_end` respectively. The trial ID can be set via `/session_trialid`; a new trial ID will be used at the next `/session_start` event.

The output directory can be set with `/session_outputdir`. This is possible up to the `/session_stop` event.

OSC variables:

path	fmt.	range	r.	description
/session_outputdir	s	string	yes	Set the output directory
/session_start			no	Start the recording of a session
/session_stop			no	Stop the recording of a session and save data to the file
/session_trialid	s	string	no	Set the new trial ID

Timeline control and data logging

With the default settings the datalogging will start the timeline transport from zero upon `/session_start`, and will stop the transport upon `/session_stop`. This can be changed by setting the attribute `controltransport="false"`. In that case the transport will not be started or stopped upon any `/session_start` or `/session_stop` event.

To record data only while the transport is rolling, the attribute `usetransport="true"` can be used.

Data logging, session time and lab streaming layer

The data logging can record two types of streams: OSC based floating point values (`<osc/>`), and LSL based floating point streams (`<lsl/>`). For OSC messages, the first row of the data matrix contains the session time t_{session} at which the data packet arrived. The underlying function from the jack audio connection kit, `jack_get_current_transport_frame`, is used to get a high resolution estimate of the current session time. For LSL streams, the situation is more complex, since LSL provides an own method of time stamping. Here, the second row in the data matrix contains the original LSL time stamps of the remote sender, $t_{\text{lsl,remote}}$. Since the data is processed in chunks, it is not possible to use the arrival time as a session time stamp. Instead, the clock difference between the local LSL clock and the remote LSL clock Δ_{stream} is measured at the beginning and also at the end of each recording session, using the LSL function `lsl_time_correction`, i.e., the local LSL clock minus the remote clock, $\Delta_{\text{stream}} = t_{\text{lsl,local}} - t_{\text{lsl,remote}}$. Additionally, upon each update of the local session time, i.e., upon each processing cycle, the difference between the session time and the local LSL time, $\Delta_{\text{session}} = t_{\text{session}} - t_{\text{lsl,local}}$ is measured. The combination of Δ_{session} and Δ_{stream} is used to convert remote LSL time stamps into session time stamps: the estimated session time at time of sending the sample, $\tilde{t}_{\text{session}}$ is

$$\tilde{t}_{\text{session}} = t_{\text{lsl,remote}} + \Delta_{\text{stream}} + \Delta_{\text{session}} \tag{7}$$

Δ_{stream} is the value which was measured at the beginning of a recording session. $\tilde{t}_{\text{session}}$ is the time stamp which is stored in the first row of the LSL data matrix.

Clock drift may occur between clocks. The drift between the local LSL clock $t_{\text{lsl,local}}$ and the audio clock (basis of t_{session}) is continuously compensated by the measures of Δ_{session}. The drift between the local LSL time $t_{\text{lsl,local}}$ and the remote LSL time $t_{\text{lsl,remote}}$ can be compensated offline by taking the difference between Δ_{stream} at the beginning and the end of a recording session, which are both stored in the datalogging file for each LSL stream. Thus the drift-compensated estimated session time $\hat{t}_{\text{session}}$ is

$$\hat{t}_{\text{session}} = \tilde{t}_{\text{session}} + \frac{t_{\text{lsl,local}} - t_{\text{lsl,local,start}}}{t_{\text{lsl,local,end}} - t_{\text{lsl,local,start}}}(\Delta_{\text{stream,end}} - \Delta_{\text{stream,start}}). \tag{8}$$

Some sensors (e.g., the ESP-based IMU/EOG sensor of the Gesture lab in University of Oldenburg), synchronize the sensor clock with the (remote) LSL clock only upon initialization. This causes the problem, that the clock drift reported by Δ_{stream} is not related to the clock drift between the sensor and the session time. To overcome this problem, the `<espheadtracker/>` glabsensor submodule (see section 6.4) sends a local difference $\Delta_{\text{sensor}} = t_{\text{lsl,remote}} - t_{\text{sensor}}$ as an LSL stream. This data contains drift as well as jitter caused by the WiFi transmission. The sensor drift can be estimated by a linear fit to this data. The linear fit of Δ_{sensor} needs to be added to $\hat{t}_{\text{session}}$ of the data of the LSL streams corresponding to this sensor.

6.2 dirgain

The **dirgain** module optionally applies channel direction dependent low-pass filtering (i.e., directional filtering) to signals. Typical application is to apply beamformer/cardioid simulation on a regular circular loudspeaker system.

Attributes:

`id`	Plugin ID, used in jack name and OSC path
`channels`	Number of channels (default: 1)
`az`	Steering azimuth in degrees (default: 0)
`az0`	Azimuth of first channel (default: 0)
`f6db`	Frequency in Hz, at which a 6 dB attenuation at 90 degrees is achieved (default: 1000)
`fmin`	Low-end limit for stabilization (default: 60)
`active`	Boolean to control start-up activity (default: true)

All variables except for `id` and `channels` can be controlled via OSC.

6.3 echoc

The *echoc* module provides echo cancellation. As a non-adaptive method, it operates in two phases: In the measurement phase, a test signal is played back through the speaker outputs `loudspeakerports` and the response is recorded through the microphone inputs `micports`. In the filter phase, the output signals are filtered with the phase-inverted corresponding responses, and the signal is added to the microphone signal. An overview of the signal flow is given in the figure 12.

Please note that no feedback jack connections are possible for the echo cancellation to work, because feedback connections cause an additional delay which results in a mismatch of the cancellation signal. This also means that a graph from the microphone to the loudspeaker (e.g., self monitoring) is not possible for the echo cancellation to work. Future versions may compensate for this extra delay.

The filter is implemented in frequency domain as overlap-save algorithm.

Attributes of element **echoc**

name	description (type, unit)	def.
`autoreconnect`	Automatically re-connect ports after jack port change (bool)	false
`bypass`	Bypass filter stage (bool)	false
`filterlen`	Minimal length of filters (uint32, samples)	65
`level`	Playback level (float, dB SPL)	70
`loudspeakerports`	Loudspeaker ports (string array)	system:playback_1 system:playback_2
`maxdist`	Maximum distance between microphone and loudspeaker (float, m)	2

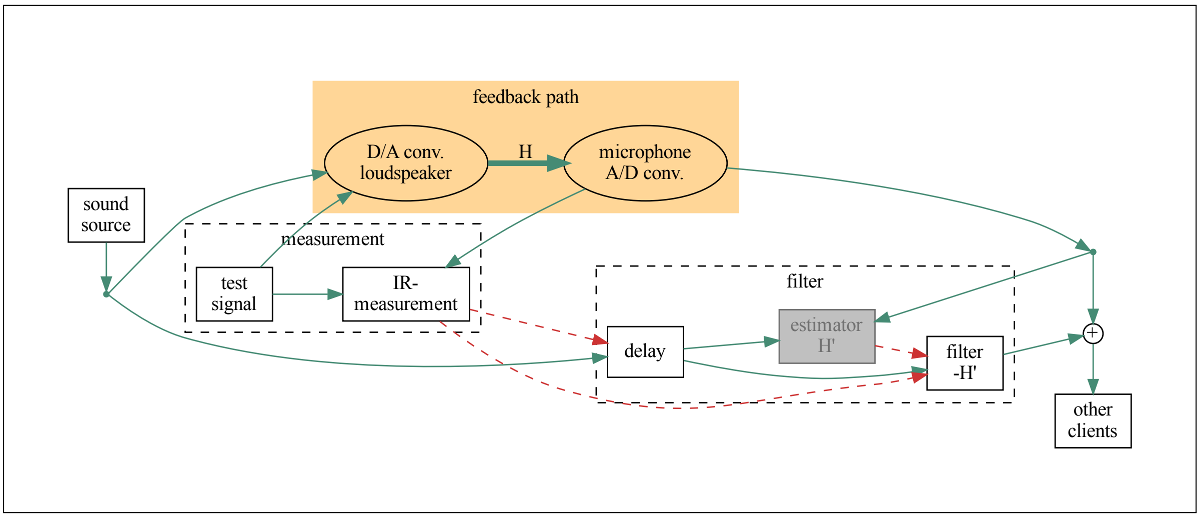


Figure 12: Signal flow of the echoc plugin. The sound source (left side) is played back through the loudspeakers. In the signal sent to the other clients (on the right side), the sound source is cancelled. The adaptive filter estimator (gray box) is not yet implemented.

`measureatstart`	Perform a measurement when the plugin is loaded (bool)	false
`micports`	Microphone ports (string array)	system:capture_1
`name`	Client name, used for jack and IR file name (string)	echoc
`nrep`	Number of measurement repetitions (uint32)	16
`premax`	Time before to maximum to add to filter (uint32, samples)	8

6.4 glabsensors

The module *glabsensors* provides an interface to error reporting in sensor drivers, and it can load drivers for various sensors.

Attributes of element **glabsensors**

name	description (type, unit)	def.
`ontop`	Keep window on top of other windows (bool)	true
`url_critical`	OSC URL to send critical messages to (string)	
`url_warning`	OSC URL to send warning messages to (string)	
`x`	Screen x position (uint32, px)	0
`y`	Screen y position (uint32, px)	0
`w`	Window width (uint32, px)	320
`h`	Window height (uint32, px)	1080

The **trackir** sensor (optical marker tracking) supports these attributes:

Attributes:	
`name`	Sensor name (default: “trackir”)
`linethreshold`	Maximal deviation from line (default: 1)
`maxdist`	Maximal distance error of marker 2D projection (default: 0.05)
`margin`	Warning margin in pixels (default: 100)
`use_calib`	Use camera calibration (default: true)
`flipx`	Flip x coordinate, required for some TrackIR models (default: false)
`flipy`	Flip y coordinate, required for some TrackIR models (default: false)
`f`	Focal length of camera (default: 640)
`maxframedist`	Maximal distance between consecutive frames for warnings (default: 0.05)
`camcalibfile`	Name of camera calibration file (default: “${HOME}/tascartrackircamcalib.txt”)
`crownfile`	Name of camera calibration file (default: “${HOME}/tascartrackircrown.txt”)
`camview`	Draw camera view (default: true)

The module provides three LSL streams: `trackir` contains 6 channels (translation xyz, rotation zyx) as provided by the underlying openCV camera solving algorithm. `trackirpresolve` contains also 6 channels with translation and rotation, but based on the 2-dimensional projection. The rotation around x and y will always be zero. This estimation might be more robust than the camera solving algorithm based estimation in some conditions. `trackirmarker` contains 30 channels, with the camera position (x,y) and pixel size of up to 10 markers. Untracked markers contain a pixel size of 0.

Camera calibration can be provided in a simple text file. White space will be ignored, comments are allowed after the '#' comment character. The filed must contain six numbers: Camera position (x,y,z) and camera Euler orientation (z,y,x). If camera calibration is provided in an external file and locally in the XML configuration, then the data from the external file is used.

The **eog** sensor is a bluetooth serial stream based device, with these attributes:

Attributes:	
`device`	Serial device (default: “/dev/rfcomm1”)
`baudrate`	Baud rate (default: 38400)
`charsize`	Character size (default: 8)
`offset`	Data offset (default: 512)
`scale`	Data scale (default: 0.0032227)
`range`	Data range (default: “0 1023”)
`unit`	Data unit (default: mV)

The LSL output stream contains one channel.

The **midicc** sensor receives MIDI CC messages:

Attributes:	
`connect`	Connect input to this MIDI port
`range`	Value range mapping, input values of 0 are mapped to the first element, input values of 127 are mapped to the second, with linear interpolation ("0 1")
`controllers`	Channel/Parameter pairs of controllers to receive
`data`	Start values sent to device upon initialization

The **serial** sensor reads data from the serial device, with these attributes:

Attributes:	
`device`	Serial device (default: "/dev/ttyS0")
`baudrate`	Baud rate (default: 38400)
`charsize`	Character size (default: 8)
`offset`	Data offset (default: 0)
`scale`	Data scale (default: 1)
`channels`	Number of channels (default: 1)

The LSL output stream contains one channel.

The **emergency** sensor reacts on continuous OSC messages on path `/noemergency`, and executes a command when no OSC message arrives within a given timeout.

Attributes of element **emergency**

name	description (type, unit)	def.
`alivetimeout`	Timeout after which the sensor is seen as not alive (double, s)	1
`name`	Module name (string)	emergency
`on_alive`	Command to be executed when sensor is alive again (string)	
`on_timeout`	Command to be executed on timeout (string)	
`path`	OSC path on which messages are arriving (string)	/noemergency
`startlock`	Lock detecting at start for this amount of time (double, s)	5
`timeout`	Timeout after which an emergency is detected (double, s)	1

OSC variables:

path	fmt.	range	r.	description
`/noemergency`	f		no	

For the custom-made ESP-based combined head tracking and EOG amplifier of the Gesture Lab, the module **espheadtracker** was developed. This module requires a session port number of 9800 to work, since the port number is hard-coded into the firmware of the sensor.

Attributes:	
timeout	Time out for re-connection/re-initialization of LSL stream in seconds

The **jackstatus** sensor analyses the JACK backend performance (xruns and CPU load). Warnings are issued if xruns occur or if the CPU load is above the given threshold. Critical errors are issued when the average xrun frequency is above the given threshold, or if the CPU load is above the threshold.

Attributes:	
warnload	CPU load threshold for warnings, in percent (default: 70)
criticalload	CPU load threshold for critical errors, in percent (default: 95)
maxxrunfreq	Critical average xrun frequency threshold in Hz (default: 0.1)
oncritical	Shell command to be executed when critical state is reached

The **qualisys** is an interface between the OSC interface of the commercial QTM software by Qualisys and TASCAR. It creates an LSL stream and optional OSC output for each rigid object tracked by QTM. The attributes are:

name	description (type, unit)	def.
dataprefix	OSC path prefix, will be followed by slash + rigid names (string)	
dataurl	OSC URL where data is sent to (or empty for no OSC sending) (string)	
qtmurl	Qualisys Track Manager URL of USC interface (string)	osc.udp://localhost:22225/
timeout	Timeout (double, s)	1
uselsl	Create LSL output stream (bool)	true

The **smiley** sensor is used for testing and learning only. It has no configurable attributes.

6.5 granularsynth

Granular synthesis

Attributes of element **granularsynth**

name	description (type, unit)	def.
active	active (bool)	true
bpm	Tempo (double, bpm)	120
bypass	bypass (bool)	false
durations	Durations (double array, beats)	
f0	frequency of pitch 0 (double, Hz)	415
gain	Gain (double, dB)	0
id	ID used in jack name and OSC path (string)	granularsynth
loop	Time when to loop (double, beats)	64
numgrains	Number of grains to keep (uint32)	100

`pitches`	Pitch numbers (double array, semitones)	
`ponset`	Onset playback probabbility (double)	1
`prefix`	prefix used in OSC path (string)	/c/
`psustain`	Sustained sound probability (double)	0
`t0`	Melody start time (double, s)	0
`wet`	Mixing gain (float)	1
`wlen`	window length (uint32, samples)	8192

These parameters can be controlled interactively:

OSC variables:

path	fmt.	range	r.	description
`/.../active`	i	bool	yes	
`/.../bypass`	i	bool	yes	
`/.../gain`	f	[-40,10]	yes	
`/.../oscactive`	i	bool	yes	
`/.../ponset`	f		yes	
`/.../psustain`	f		yes	
`/.../reset`			no	
`/.../t0`	f		yes	
`/.../wetapply`	f		no	
`/.../wet`	f		yes	

6.6 hoafdnrot

A higher-order-ambisonics feedback delay network (FDN) with rotation line-filters, and circulant feedback matrix design after Rocchesso and Smith (1997).

Attributes of element **hoafdnrot**

name	description (type, unit)	def.
`id`	Jack / OSC id (string)	fdn
`amborder`	Ambisonics order (uint32)	3
`fdnorder`	FDN order (uint32)	5
`w`	Rotation velocity in rounds per second (double, rps)	1
`dw`	Angular spread (double, rps)	0.1
`t`	Average delay line length (double, s)	0.01
`dt`	Delay line spread (double, s)	0.002
`decay`	Decay time (double, s)	1
`damping`	Damping coefficient (double)	0.3
`dry`	Dry signal ratio (double)	0
`wet`	Wet signal ratio (double)	1
`prefilt`	Use pre-filters (bool)	false
`logdelays`	Use logarithmic delay distribution between dt and t (bool)	false

Real-time parameters can be remote-controlled with the OSC variables `/id/par`, accepting six floats (w, dw, t, dt, decay, damping), and the variable `/id/dry` with one float, to control the dry signal ratio.

6.7 hossustain

Cluster generator by moving spectral averaging with random phase.

Attributes of element **hossustain**

name	description (type, unit)	def.
`bass`	Linear gain of subsonic component (float)	0
`bassratio`	Frequency ratio of subsonic component (float)	2
`delayenvelope`	Delay envelope to match processed signal (bool)	false
`fcut`	Low-cut edge frequency (float, Hz)	40
`gain`	Gain (double, dB)	0
`id`	ID used for jack and OSC (string)	sustain
`tau_envelope`	Envelope tracking time constant (float, s)	1
`tau_sustain`	Clustering time constant (float, s)	20
`wet`	Wet-dry ratio (float)	1
`wlen`	Window length (uint32, samples)	8192

6.8 hrirconv

The module **hrirconv** is intended for convolution of multi-channel loudspeaker signals with head related impulse responses (HRIR), to generate signals for binaural listening or hearing aid processing. To activate the module, add

```
<hrirconv>
  ...
</hrirconv>
```

to your session configuration.

Attributes:

`id`	Name used for jack
`fftlen`	FFT length (need to be longer than jack fragment size)
`inchannels`	Number of input channels
`outchannels`	Number of output channels
`autoconnect`	Auto-connect input to all receivers with matching channel count (true|false)
`connect`	Input port connections (port name globbing possible)
`hrirfile`	file name of HRIR file, channel order i1o1,i1o2,i1o3,...,i2o1,...

The convolution matrix can be defined with the `<entry/>` element. Each entry defines a convolution with an impulse response; typically a convolution for each combination of input channel and output channel is defined. The recognized attributes of `<entry/>` are:

Attributes:	
`in`	Input channel number (zero-based)
`out`	Output channel number (zero-based)
`file`	File name of impulse response
`channel`	File channel of impulse response (zero-based)

A typical configuration for binaural listening can look like this:

```
<?xml version="1.0"?>
<session name="hrir" license="CC BY-SA 3.0" attribution="Giso Grimm">
  <scene name="test">
    <receiver name="out" type="vbap" layout="8ch.spk"/>
    <!-- ... -->
  </scene>
  <modules>
    <hrirconv inchannels="8" outchannels="2" autoconnect="true">
      <entry in="0" out="0" file="hrir_000.wav" channel="0"/>
      <entry in="0" out="1" file="hrir_000.wav" channel="1"/>
      <entry in="1" out="0" file="hrir_045.wav" channel="0"/>
      <entry in="1" out="1" file="hrir_045.wav" channel="1"/>
      <entry in="2" out="0" file="hrir_090.wav" channel="0"/>
      <entry in="2" out="1" file="hrir_090.wav" channel="1"/>
      <entry in="3" out="0" file="hrir_135.wav" channel="0"/>
      <entry in="3" out="1" file="hrir_135.wav" channel="1"/>
      <entry in="4" out="0" file="hrir_180.wav" channel="0"/>
      <entry in="4" out="1" file="hrir_180.wav" channel="1"/>
      <entry in="5" out="0" file="hrir_225.wav" channel="0"/>
      <entry in="5" out="1" file="hrir_225.wav" channel="1"/>
      <entry in="6" out="0" file="hrir_270.wav" channel="0"/>
      <entry in="6" out="1" file="hrir_270.wav" channel="1"/>
      <entry in="7" out="0" file="hrir_315.wav" channel="0"/>
      <entry in="7" out="1" file="hrir_315.wav" channel="1"/>
    </hrirconv>
  </modules>
  <connect src="hrirconv:out_0" dest="system:playback_1"/>
  <connect src="hrirconv:out_1" dest="system:playback_2"/>
</session>
```

Example 13: examples/example_hrirconv.tsc

A configuration file for binaural convolution together with a binaural head model HRIR set (Duda, 1993) can be created with the Matlab/GNU Octave script “tascar_hrir_duda.m” (in /usr/share/tascar/matlab). See documentation of the script for details.

6.9 jackrec

OSC controlled audio recorder

List of configuration variables:

Attributes of element **jackrec**

name	description (type, unit)	def.
buflen	audio buffer length (double, s)	10
fileformat	File format (string, WAV AIFF AU RAW PAF SVX NIST VOC IRCAM W64 MAT4 MAT5 PVF XI HTK SDS AVR WAVEX SD2 FLAC CAF WVE OGG MPC2K RF64)	WAV
name	Name used for OSC prefix and jack (string)	jackrec
path	File path where to store and search for files (string)	
pattern	search pattern (string)	rec*.wav
ports	List of ports to record (string array)	
prefix	file prefix (string)	rec
sampleformat	Audio sample format (string, PCM_S8 PCM_16 PCM_24 PCM_32 PCM_U8 FLOAT DOUBLE ULAW ALAW IMA_ADPCM MS_ADPCM GSM610 VOX_ADPCM G721_32 G723_24 G723_40 DWVW_12 DWVW_16 DWVW_24 DWVW_N DPCM_8 DPCM_16 VORBIS)	PCM_16
url	URL of OSC controller interface (string)	
usetransport	Record only when transport is rolling (bool)	false

Here is an example of the communication protocol:

TASCAR response	control client request
/jackrec/ready	
	/jackrec/listports
/jackrec/portlist /jackrec/port system:capture_1 /jackrec/port system:capture_2 /jackrec/port render.scene:out_l ...	
	/jackrec/addport system:capture_1 /jackrec/addport system:capture_2 ...
	/jackrec/start
/jackrec/rectime 0.18575963377952576 /jackrec/rectime 0.38603174686431885 /jackrec/rectime 0.5863038301467896 /jackrec/rectime 0.786575973033905 /jackrec/rectime 0.9868480563163757 ...	
	/jackrec/stop
	/jackrec/listfiles
/jackrec/file rec20201122_101133.wav	
	/jackrec/clear /jackrec/start
/jackrec/error Failure: No sources selected.	
	/jackrec/addport system:capture_1 /jackrec/tag _id1234_ /jackrec/start
/jackrec/rectime 0.18575963377952576 /jackrec/rectime 0.38603174686431885 /jackrec/rectime 0.5863038301467896 /jackrec/rectime 0.786575973033905 /jackrec/rectime 0.9868480563163757 ...	
	/jackrec/stop
	/jackrec/listfiles
/jackrec/file rec20201122_101133.wav /jackrec/file rec_id1234_20201122_101633.wav	

File formats:

```
WAV AIFF AU RAW PAF SVX NIST VOC IRCAM W64 MAT4 MAT5
PVF XI HTK SDS AVR WAVEX SD2 FLAC CAF WVE OGG MPC2K RF64
```

Sample formats:

```
PCM_S8 PCM_16 PCM_24 PCM_32 PCM_U8
FLOAT DOUBLE
```

```
ULAW ALAW
IMA_ADPCM MS_ADPCM
GSM610 VOX_ADPCM G721_32 G723_24 G723_40
DWVW_12 DWVW_16 DWVW_24 DWVW_N DPCM_8 DPCM_16
VORBIS
```

OSC variables:

path	fmt.	range	r.	description
`/.../addport`	s		no	Add the given port to the list of recorder input ports
`/.../clear`			no	Clear list of ports
`/.../listfiles`			no	Send list of sound files (matching pattern provided in XML)
`/.../listports`			no	List all available jack ports
`/.../name`	s	string	yes	Output file name, leave empty for automatic file names
`/.../rmfile`	s		no	Remove a file on disk
`/.../start`			no	Start recording (or recording standby when usetransport is set)
`/.../stop`			no	Stop recording and close output file
`/.../tag`	s	string	yes	Set tag of output file
`/.../usetransport`	i	bool	yes	Control wether to use jack transport during recording when started next

6.10 levels2osc

Attributes:	
`pattern`	Source port names
`noisepattern`	Source port names for noise signals, to calculate SNR
`url`	Target OSC URL
`ttl`	Time to live of OSC multicast messages

This module reads the level meters of the specified ports and sends their values as OSC data and LSL streams. If `pattern` and `noisepattern` match the same number of ports and each port has the same number of audio channels, then the SNR is calculated and transmitted instead of levels. A potential application is data logging of levels or SNRs.

6.11 lightcolorpicker

Provide a simple color picking dialog to control HSV values via OSC.

Attributes of element **lightcolorpicker**

name	description (type, unit)	def.
`color`	undocumented (string)	
`path`	undocumented (string)	

6.12 lightctl

The module has these attributes:

name	description (type, unit)	def.
`fps`	Frames per second (double, Hz)	30
`universe`	DMX universe (uint32)	0
`driver`	Driver name (string, “artnetdmx”, “opendmxusb”, or “osc”)	
`hue_warp_rot`	Hue warping rotation (double, deg)	0
`hue_warp_x`	Hue warping x offset (double)	0
`hue_warp_y`	Hue warping y offset (double)	0
`rawsrvchannels`	Number of channels to receive as RAW DMX (uint32)	0
`rawsrvhost`	multicast address for raw DX OSC server (string)	
`rawsrvpath`	Path for raw DMX OSC server, empty for no raw DMX OSC server (string)	
`rawsrvport`	Port of raw DMX OSC server, or empty to use session OSC server (string)	
`rawsrvproto`	Protocol of raw DMX OSC server (string)	UDP

Additional attributes of “artnetdmx” driver:

name	description (type, unit)	def.
`hostname`	Hostname of ArtnetDMX receiver (string)	localhost
`port`	Port number of ArtnetDMX receiver (uint32)	6454

Additional attributes of “opendmxusb” driver

name	description (type, unit)	def.
`device`	Device name	/dev/ttyUSB0

Additional attributes of “osc” driver

name	description (type, unit)	def.
`hostname`	Hostname of OSC destination (string)	localhost
`port`	Port number of OSC destination (uint32)	9000
`path`	Destination path (string)	/dmx
`maxchannels`	Maximum number of channels to transmit (uint32)	512

One or more `<lightscene/>` elements can be defined, with these attributes:

Attributes of element **lightscene**

name	description (type, unit)	def.
`channels`	Number of DMX channels per fixture (uint32)	3
`layout`	name of speaker layout file (string)	
`master`	undocumented (float)	1
`method`	undocumented (string)	

`mixmax`	undocumented (bool)	false
`name`	Scene name (string)	lightscene
`objects`	Pattern of objects to track (string array)	
`objval`	DMX value of objects (float array)	
`objw`	weight of objects (float array)	
`parent`	Name of parent object for relative position measurement (string)	
`sendsquared`	Send squared values for smoother intensity fades (bool)	false
`usecalib`	Use calibrated values instead of raw values (bool)	true

Fixtures are defined using the `fixture` element within the `fixtures` element. Syntax is the same as for speaker layout definitions, with these additional attributes for each element:

Attributes of element **fixture**

name	description (type, unit)	def.
`addr`	start address (uint32)	1
`az`	Azimuth (double, deg)	0
`el`	Elevation (double, deg)	0
`label`	fixture label (string)	
`dmxval`	start DMX value (int32 array)	

For each fixture, sub-elements in the form `<calib channel="0" in="255" out="127"/>` can be provided, to calibrate the input-output function of the lamps. The attributes `in` needs to be larger than zero, the attributes `channel` and `out` need to be larger or equal to zero. Instead of linear DMX values these can be squared ($\mathrm{DMX}_o = 255\,\mathrm{ceil}(\mathrm{DMX}_i/255)^2$), to achieve constant intensity light.

Calibration tools for MATLAB/GNU Octave are available in `tascar_fixtures_calib.m`. See source code repository for more examples.

6.13 lsl2osc

Convert LSL streams into OSC messages. Each variable will contain the LSL time stamp in the first entry, followed by all stream channels. This means that if the LSL stream contains N channels, then the OSC variable will contain $N+1$ double entries. Only LSL streams with 32 or 64 Bit floating point data are supported. Both types will be forwarded as 64 Bit floating points.

Attributes of element **lsl2osc**

name	description (type, unit)	def.
`prefix`	OSC path prefix, "/" + name will be appended. (string)	/lsl2osc
`streams`	List of stream names to transmit (string array)	
`url`	OSC target URL, or empty to dispatch locally. (string)	

6.14 lsljacktime

This module sends the current jack time as an LSL stream of the name “TASCARtime”.

Example:

```
<lsljacktime sendwhilestopped="false"/>
```

6.15 ltcgen

A Linear Time Code (LTC) generator module encodes either session time or wall clock time into LTC code, such as for synchronizing cameras. A jack port is provided through which the signal is transmitted.

Attributes of element **ltcgen**

name	description (type, unit)	def.
`addtime`	Add time, e.g., for time zone compensation (double, s)	0
`connect`	Space-separated list of output port connections (string array)	
`fpsden`	Frames per second, denominator (double)	1
`fpsnum`	Frames per second, numerator (double)	25
`usewallclock`	Use wallclock time instead of session time (bool)	false
`volume`	Signal volume (double, dB re FS)	-18

6.16 matrix

Create a jack based matrix multiplication, e.g., for Ambisonics decoding.

Attributes:	
`id`	Jack identifier
`decoder`	Empty (for explicit matrix), or maxre2d (for mode-matching 2D-HOA max-rE decoding)

Outputs are defined as in speaker based layout files, except that they use the element `<output/>`. In addition, each speaker can contain the attribute `m`, which contains a list of floating point values. Each output channel is the sum of the product of `m` with the corresponding input channel.

Inputs are defined by the sub-elements `<input/>`. Each input can have the attribute `connect`.

An Ambisonics decoder configuration can be created with the MATLAB/GNU Octave script `tacsar_generatedecmatrix.m`.

Example:

```
<matrix id="dec" decoder="maxre2d">
  <input connect="hoa:out.0" label=".0_0"/>
  <input connect="hoa:out.1" label=".1_-1"/>
  <input connect="hoa:out.2" label=".1_1"/>
  <input connect="hoa:out.3" label=".2_-2"/>
  <input connect="hoa:out.4" label=".2_2"/>
  <input connect="hoa:out.5" label=".3_-3"/>
  <input connect="hoa:out.6" label=".3_3"/>
  <input connect="hoa:out.7" label=".4_-4"/>
  <input connect="hoa:out.8" label=".4_4"/>
  <input connect="hoa:out.9" label=".5_-5"/>
  <input connect="hoa:out.10" label=".5_5"/>
  <input connect="hoa:out.11" label=".6_-6"/>
  <input connect="hoa:out.12" label=".6_6"/>
  <output az="12" connect="render.tostereo:in.0"/>
  <output az="36" connect="render.tostereo:in.1"/>
  <output az="60" connect="render.tostereo:in.2"/>
  <output az="84" connect="render.tostereo:in.3"/>
  <output az="108" connect="render.tostereo:in.4"/>
  <output az="132" connect="render.tostereo:in.5"/>
  <output az="156" connect="render.tostereo:in.6"/>
  <output az="180" connect="render.tostereo:in.7"/>
  <output az="204" connect="render.tostereo:in.8"/>
  <output az="228" connect="render.tostereo:in.9"/>
  <output az="252" connect="render.tostereo:in.10"/>
  <output az="276" connect="render.tostereo:in.11"/>
  <output az="300" connect="render.tostereo:in.12"/>
  <output az="324" connect="render.tostereo:in.13"/>
  <output az="348" connect="render.tostereo:in.14"/>
</matrix>
```

An example with explicit matrix element definitions:

```
<matrix id="mix_out">
  <input label="proc1_l" connect="adm1:out_1"/>
  <input label="proc1_r" connect="adm1:out_2"/>
  <input label="proc2_l" connect="adm2:out_1"/>
  <input label="proc2_r" connect="adm2:out_2"/>
  <output label="S_out_l" m="0.5 0 0.5 0"/>
  <output label="S_out_r" m="0 0.5 0 0.5"/>
  <output label="N_out_l" m="-0.5 0 0.5 0"/>
  <output label="N_out_r" m="0 -0.5 0 0.5"/>
</matrix>
```

6.17 midicc2osc

Convert MIDI CC events from ALSA devices into OSC messages.

Attributes of element **midicc2osc**

name	description (type, unit)	def.
connect	name of input ALSA MIDI source (string)	
controllers	List of controllers, in “channel/param” form (e.g., 0/13 0/28) (string array)	
dumpmsg	Dump unprocessed messages to console (bool)	false
max	maximum output value (corresponding to MIDI 127) (double)	1
min	minimum output value (corresponding to MIDI 0) (double)	0
name	name of MIDI client (string)	
path	OSC path (string)	/midicc
url	OSC destination URL (string)	osc.udp://localhost:7777/

6.18 midictl

Control gains with a MIDI controller.

Attributes:	
pattern	Pattern to select gain controllers in TASCAR.
dumpmsg	Dump unprocessed messages to console
name	name of MIDI client
connect	name of MIDI device
controllers	List of controllers, in “channel/param” form (e.g., 0/13 0/28)
min	minimum output value (corresponding to MIDI 0)
max	maximum output value (corresponding to MIDI 127)

Here is an example which selects the gain controller of the sound “in.0”, the receiver gain “out” (both in the scene “scene”) and the gain controller of the “route” module “/test”:

```
<?xml version="1.0" encoding="UTF-8"?>
<session license="CC0">
  <scene name="scene">
    <source name="in">
      <sound name="0"/>
    </source>
    <receiver name="out"/>
  </scene>
  <modules>
    <midictl name="master" min="-30" max="0" controllers="0/0 0/1 0/2 0/8 0/9
    0/10" dumpmsg="true" connect="BCF2000:0" pattern="/scene/in/0 /scene/out
    /test"/>
    <route name="test" channels="2"/>
  </modules>
</session>
```

Example 14: examples/example_midictl.tsc

6.19 mididispatch

This plugins can dispatch OSC messages upon MIDI events (CC or note events). Event handlers can be registered via OSC or in the XML configuration, using the <ccmsg/>

or <notemsg/> elements (see below). Parameters to the message can be added using the <f v="1.234"/>, <i v="1"/> or <s v="string"/> sub-elements. Multiple event handlers for the same event can be registered. In that case all event handlers will be called. Event handler can be removed via OSC. The communication is bi-directional; MIDI events can be emitted by sending an OSC message to /mididispatch/send/cc or /mididispatch/send/note.

Attributes of element **mididispatch**

name	description (type, unit)	def.
connect	ALSA device name to connect to (string)	
copyccpath	OSC path for copied CC events (string)	/cc
copynotepath	OSC path for copied note events (string)	/note
copyurl	OSC URL to copy outgoing MIDI messages to. (string)	
dumpmsg	Dump all unrecognized messages to console (bool)	true
name	ALSA MIDI name (string)	mididispatch
oscinput	Create additional OSC inputs (bool)	false

Attributes of element **ccmsg**

name	description (type, unit)	def.
channel	MIDI channel (uint32)	0
param	MIDI CC parameter (uint32)	0
mode	message mode, float|trigger (string)	trigger
path	OSC path (string)	
min	lower bound (float)	0
max	upper bound (float)	127

Attributes of element **notemsg**

name	description (type, unit)	def.
channel	MIDI channel (uint32)	0
note	MIDI note (uint32)	0
mode	message mode, float|trigger (string)	trigger
path	OSC path (string)	
min	lower bound (float)	0
max	upper bound (float)	127

An example configuration can look like this:

```
<modules>
  <mididispatch connect="US-2x2:US-2x2 MIDI 1" dumpmsg="true">
    <notemsg channel="0" note="0" path="/runscript"><s v="note0"/></notemsg>
  </mididispatch>
</modules>
```

OSC variables:

path	fmt.	range	r.	description
/.../add/cc/float	iisff		no	
/.../add/cc/float	iisffs		no	
/.../add/cc/trigger	iisii		no	
/.../add/cc/trigger	iisiis		no	
/.../add/note/float	iisff		no	
/.../add/note/float	iisffs		no	
/.../add/note/trigger	iisii		no	
/.../add/note/trigger	iisiis		no	
/.../clear/launchpadaction			no	
/.../del/cc/all			no	
/.../del/cc	ii		no	
/.../del/launchpadaction	i		no	
/.../del/note/all			no	
/.../del/note	ii		no	
/.../select/launchpadaction	s		no	
/.../send/cc	iii		no	
/.../send/note	iii		no	

6.20 osc2lsl

Convert OSC messages into an LSL stream.

Attributes of element **osc2lsl**

name	description (type, unit)	def.
first_row_is_timestamp	Use data of first row as LSL time stamp (bool)	false
lslname	LSL name (string)	osc2lsl
lsltype	LSL type (string)	osc2lsl
path	OSC path name (string)	/osc2lsl
retval	OSC return value: 0 = handle messages also locally, non-0 = mark message as handled, do not handle locally (int32)	1
size	Dimension of variable (uint32)	1
source_id	LSL source ID (string)	osc2lsl29

6.21 osceog

OSC based EOG sensor driver

Attributes of element **osceog**

name	description (type, unit)	def.
connectwlan	connect to sensor to external WLAN (bool)	false
eogpath	OSC target path for EOG data, or empty for no EOG (string)	/eog
name	Prefix in OSC control variables (string)	osceog
srate	Sensor sampling rate (8, 16, 32, 64, 128, 250, 475, 860) (uint32, Hz)	128

`targetip`	target IP address when using external WLAN (string)
`wlanpass`	passphrase of external WLAN (string)
`wlanssid`	SSID of external WLAN (string)

6.22 oscevents

Emit OSC events at given time instances.

Note: The interface will change in near future, thus it remains undocumented.

6.23 oscjacktime

This module sends the current jack time as OSC messages.

Attributes of element **oscjacktime**

name	description (type, unit)	def.
`path`	Destination OSC path (string)	/time
`skip`	Skip this number of blocks between sending (uint32, blocks)	0
`ttl`	Time-to-live of UDP messages (uint32)	1
`url`	Destination URL (string)	osc.udp://localhost:9999/

Example:

```
<oscjacktime url="osc.udp://localhost:7000/" path="/time"/>
```

6.24 oscrelay

Relay OSC messages, e.g., for distribution of motion sensors.

Attributes of element **oscrelay**

name	description (type, unit)	def.
`newpath`	Replace incoming path with this path, or empty for no replacement (string)	
`path`	Path filter, or empty to match any path (string)	
`retval`	Return value: 0 = handle messages also locally, non-0 = do not handle locally (int32)	1
`startswith`	Forward only messags which start with this path (string)	
`trimstart`	Trim startswith part of the path before forwarding (bool)	false
`url`	Target OSC URL (string)	osc.udp://localhost:9000/

6.25 oscserver

Optional additional OSC server, e.g., for simultaneous access via TCP and UDP.

Attributes:	
`srv_addr`	OSC server address for multicasting (or empty for unicast)
`srv_port`	OSC server port number (default: 9877)
`srv_proto`	OSC transport protocol, “UDP” or “TCP” (default: “TCP”)

6.26 route

Create a jack bus with OSC controllable gain.

Attributes of element **route**

name	description (type, unit)	def.
`caliblevel`	calibration level (float, dB SPL)	93.9794
`caliblevel_in`	Input calibration levels (float array, dB SPL)	
`channels`	Number of channels (uint32)	1
`connect`	Regular expressions of input port names (string array)	
`connect_out`	Regular expressions of output port names (string array)	
`gain`	Route gain (float, dB)	0
`id`	Unique route id, empty to autogenerate (string)	
`inv`	phase invert (bool)	false
`levelmeter_tc`	Leq level metering time constant (double, s)	2
`levelmeter_weight`	level meter weighting (f-weight)	Z
`lingain`	linear gain (float)	1
`mute`	Mute flag of route (bool)	false
`name`	Jack and OSC identifier (string)	
`solo`	Solo flag of route (bool)	false

The session OSC server is used for control.

6.27 sampler

Play audio samples via jack, triggered by OSC messages.

Attributes of element **sampler**

name	description (type, unit)	def.
`multicast`	Multicast address (string)	
`port`	OSC port number (string)	9999

Sound files can be loaded with the sub-element `<sound/>`:

Attributes of element **sound**

name	description (type, unit)	def.
`gain`	Gain to be applied (double, dB)	0
`name`	File name of sound file (string)	

6.28 savegains

This module can save the gains of all input and output ports into a plain text file, containing OSC paths and dB values of the current gain. The file name is “savedgains” (with an optional path prefix, see below). The save action can be triggered via an empty OSC message to the OSC path `/savegains/save`. The same file can be restored by sending an empty OSC message to the path `/savegains/restore`. To switch between different gain settings, the file name can be changed remotely via OSC.

Attributes:	
`pattern`	Pattern of routes to be saved (default: “*”)
`filename`	File name (default: “savedgains”)
`path`	Path prefix of output file name

6.29 sleep

Block loading of additional modules for a given amount of time.

Attributes:	
`sleep`	Sleep time in seconds (default: 1)

6.30 system

Start system processes, e.g., to load helper programs, external decoders or video render tools.

Attributes of element **system**

name	description (type, unit)	def.
`allowoscmod`	allow modifications of timed commands via OSC (bool)	false
`command`	command to be executed (string)	
`id`	undocumented (string)	system
`noshell`	do not use shell to spawn subprocess (bool)	true
`onunload`	command to be executed when unloading session (string)	
`relaunch`	relaunch process if ended before session unload (bool)	false
`relaunchwait`	Time to wait before relaunching subprocess (double, s)	0
`sleep`	wait after starting the command before continuing to load session (double, s)	0

`timedcmdpipe`	start timed commands using a pipe (true) or fork (false) (bool)	true
`timedprefix`	Prefix for timed commands added via OSC (string)	
`triggered`	command to be executed upon trigger signal (string)	

If using a shell, on Unix systems the commands are started into the background using this shell command line:

```
sh -c "cd sessionpath;command >/dev/null & echo \$!"
```

6.31 systime

Dispatch system time as OSC message to a local variable with 6 entries (year, month, day, hour, minute, second).

Attributes of element **systime**

name	description (type, unit)	def.
`path`	OSC path where time stamps (calendar) are dispatched (string)	/systime
`secpath`	OSC path where time stamps (seconds since midnight) are dispatched (string)	/seconds
`sendsessiontime`	Send session time in first data field (bool)	true

6.32 timedisplay

Create a window with a time display.

Attributes of element **timedisplay**

name	description (type, unit)	def.
`colbg`	background color (string, html color)	#ffffff
`colneg`	font color for negative times (string, html color)	#cc1a1a
`colpos`	font color for positive times (string, html color)	#000000
`digits`	Number of decimals (uint32)	1
`fontscale`	font scale (double)	1
`fps`	Display update rate (not granted) (double, Hz)	10
`prefix`	OSC variable prefix (string)	/timedisplay
`remaining`	show remaining time (bool)	false
`showtc`	Show time code (bool)	false
`threshold`	Change color to red if displayed time is below this value (double, s)	0
`times`	List of time thresholds (double array, s)	
`w`	window width (int32, px)	148
`h`	window height (int32, px)	17
`x`	window x position (int32, px)	26
`y`	window y position (int32, px)	23

It is possible to set a timer using OSC the variable:

OSC variables:

path	fmt.	range	r.	description
/.../time	d		no	

6.33 touchosc

Interface to TouchOSC control surface.

6.34 transportgui

Show transport controls and a time line.

Attributes:	
times	List of marker times, or empty to use session time
x, y, w, h	Position and size of window

6.35 waitforjackport

Block loading of additional modules until specified jack ports exist.

Attributes of element **waitforjackport**

name	description (type, unit)	def.
name	Name used in jack (string)	waitforjackport
ports	List of port names to wait for (string array)	
timeout	Timeout (double, s)	30

Ports can also be specified with `<port/>` sub-elements. This way it is possible to include whitespace in port names, e.g.:

```
<modules>
  <waitforjackport ports="obs:in_1 obs:in_2">
    <port>ardour:Giso/audio_in 1</port>
    <port>ardour:stereo/audio_in 1</port>
    <port>ardour:stereo/audio_in 2</port>
  </waitforjackport>
</modules>
```

6.36 waitforlslstream

Block loading of additional modules until specified LSL streams exist.

Attributes of element **waitforlslstream**

name	description (type, unit)	def.
`streams`	List of stream names to wait for (string array)	
`timeout`	Timeout (double, s)	30

7 Actor modules

Actor modules can be used in the same way as general purpose modules, however, their purpose is to change or query the position one or more objects by using an actor name definition:

```
<simplecontroller actor="/scene/obj" .../>
```

Name matching with `*` is possible. For example, we can choose all the objects from the scene, whose names start with `N`:

`actor="/scene/N*"`

Or if we have more than one scenes, we can choose all the objects called `out` from all scenes:

`actor="/*/out"`

List of actor modules:

- **accmovement**
- **accrotator**
- **epicycles**
- **geopresets**
- **joystick**
- **linearmovement**
- **locationmodulator**
- **locationvelocity**
- **lslactor**
- **motionpath**
- **nearsensor**
- **orientationmodulator**
- **oscactor**

- oscheadtracker
- ovheadtracker
- pendulum
- pos2lsl
- pos2osc
- qualisystracker
- rotator
- serialheadtracker
- simplecontroller
- skyfall
- snapangle
- tracegui

7.1 accmovement

Accelerated movement, with the position vector $\boldsymbol{r}$ as

$$\boldsymbol{r} = \boldsymbol{p}_{acc\_onset} + t\boldsymbol{v} + \begin{cases} \frac{1}{2}t^2\boldsymbol{a} & t > 0 \\ 0 & t \leq 0 \end{cases} \quad (9)$$

with t being the time since $t_{\mathrm{acc\_onset}}$.

Attributes of element **accmovement**

name	description (type, unit)	def.
`a`	acceleration vector (pos, m/s^2)	0 0 0
`actor`	pattern to match actor objects (string array)	
`p_acc_onset`	start position at time t_acc_onset (pos, m)	0 0 0
`t_acc_onset`	onset of acceleration time t_acc_onset (double, s)	0
`v`	velocity vector (pos, m/s)	1 1 0

OSC variables:

path	fmt.	range	r.	description
`/.../a/x`	f		yes	acceleration in x-direction in m/s^2
`/.../a/y`	f		yes	acceleration in y-direction in m/s^2
`/.../a/z`	f		yes	acceleration in z-direction in m/s^2
`/.../avpt`	dddddddddd		no	
`/.../p_acc_onset/x`	f		yes	start x-position at time t_{acc_onset} in m
`/.../p_acc_onset/y`	f		yes	start y-position at time t_{acc_onset} in m

/.../p_acc_onset/z	f		yes	start z-position at time t_{acc_onset} in m
/.../t_acc_onset	f		yes	reference session time in s
/.../v/x	f		yes	velocity in x-direction in m/s
/.../v/y	f		yes	velocity in y-direction in m/s
/.../v/z	f		yes	velocity in z-direction in m/s

7.2 accrotator

Accelerated rotation, with Euler angle around z axis Ω_z as

$$\Omega_z = \theta_{acc\_onset} + t\omega + \begin{cases} \frac{1}{2}t^2 a & t > 0 \\ 0 & t \leq 0 \end{cases} \tag{10}$$

with t being the time since $t_{\mathrm{acc\_onset}}$.

Attributes of element **accrotator**

name	description (type, unit)	def.
acc	angular acceleration at origin (double, rad/s^2)	0
actor	pattern to match actor objects (string array)	
omega	angular velocity vector (double, rad/s)	1
t_acc_onset	onset of angular acceleration time t_acc_onset (double, s)	0
theta_acc_onset	start angular rotation at time t_acc_onset (double, rad)	0

OSC variables:

path	fmt.	range	r.	description
/.../acc	f		yes	angular acceleration rad/s^2
/.../awzt	dddd		no	
/.../omega	f		yes	angular velocity in rad/s
/.../t_acc_onset	f		yes	time of acceleration onset
/.../theta_acc_onset	f		yes	angular rotation at time t_{acc_onset} in rad

7.3 epicycles

Parametric cycle/epicycle generator, to be controlled via OSC.

The algorithm was originally presented in Grimm and Herzke (2012).

Attributes of element **epicycles**

name	description (type, unit)	def.
actor	pattern to match actor objects (string array)	
home	Home direction of sound source (double, deg)	0
path	Path prefix of plugin (string)	
targetaddr	Target url where the current position is sent to on trigger (string)	
use_transport	Update traces only while transport is running (bool)	true

OSC variables are:

name	format	meaning
`phi0`	f	Starting direction in degrees
`random`	f	Amount of randomness
`f`	f	Rounds per second of main rotation
`r`	f	Normalized radius of main rotation
`theta`	f	Alignment of Keppler ellipse
`e`	f	Excentricity of movement
`f_epi`	f	Rounds per second of epicycles
`r_epi`	f	Radius of epicycles
`phi0_epi`		Starting direction of epicycle, in degrees
`sendphi`	s	OSC path to send current position
`locate`	f	Trigger movement to starting directions, paramter defines time to reach in seconds
`apply`	f	Apply non-angular parameters, parameter defines time to reach in seconds
`stopat`	f	Stop movement when given direction is next reached, in degrees
`applyat`	ff	Apply parameters when position is reached, in seconds
`az`	f	Move to this direction immediately
`gohome`		Trigger movement to home position
`home`	f	Overwrite configured home direction

OSC variables:

path	fmt.	range	r.	description
`/applyat`	ff		no	
`/apply`	f		no	
`/az`	f		no	
`/e`	f		yes	
`/f_epi`	f		yes	
`/f`	f		yes	
`/gohome`			no	
`/home`	f	[0,360]	yes	
`/incbpm`	f		yes	
`/incbpmphi`	f	[0,360]	yes	
`/incphi0`	f	[0,360]	yes	
`/locate`	f		no	
`/phi0_epi`	f	[0,360]	yes	
`/phi0`	f	[0,360]	yes	
`/r_epi`	f		yes	
`/random`	f		yes	
`/r`	f		yes	
`/sendphi`	s		no	
`/stopat`	f		no	
`/tcnt`	i		yes	
`/theta`	f	[0,360]	yes	

7.4 geopresets

The module **geopresets** allows to define preset positions and orientations of objects. Objects are moved to the defined preset delta-transformation following a von-Hann ramp from the current delta-transformation to the new delta-transformation.

```
<scene>
  <receiver name="out"/>
  <source name="in">
    <sound/>
  </source>
</scene>
<modules>
  <geopresets actor="/*/in" showgui="true">
    <preset name="loc" position="2 1 0"/>
    <preset name="locrot" position="1 -1 0" orientation="70 0 0"/>
    <preset name="rot" orientation="0 0 0"/>
  </geopresets>
</modules>
```

Example 15: examples/example_geopresets.tsc

In this example, the presets “pos”, “posrot” and “rot” can be reached with OSC commands, e.g.,

```
/geopresets pos
```

The enable state and the duration can be controlled via OSC.

```
/geopresets/enable 1
/geopresets/duration 3
```

To use **geopresets** in combination with the simplecontroller (Section 7.22) or joystick (Section 7.5) actor plugin, configure this module to appear before the others in the session file, and set `unlock="true"`.

Attributes:	
`duration`	Duration of ramp in seconds (default: 2)
`enable`	Enable (true, default) or disable (false) the module.
`id`	ID used as OSC prefix (default: geopresets)
`startpreset`	Starting preset (or empty for no starting preset)
`unlock`	Unlock delta transformation after motion
`showgui`	Show GUI (default: false)
`width`	Window width in pixels (default: 200)
`buttonheight`	Button height in pixels

Presets can be defined with one or more `<preset/>` elements, which support these attributes:

Attributes:	
`name`	Preset name
`position`	Position (optional).
`orientation`	Orientation (optional).

Within a preset, a number of `<osc/>` elements can be defined. These attributes are supported:

Attributes:	
`path`	OSC varaibale path, either "/pos" or "/zyxeuler" are appended
`pos`	Position x y z Cartesian coordinates in meter (optional).
`rot`	Orientation z y x Euler angles in degree (optional).

OSC messages are dispatched in the current session. No position fades are applied here.

7.5 joystick

Very simple joystick motion controller module. The recognized attributes are:

Attributes:	
`maxnorm`	Maximum distance of object from origin, or zero for no limit.
`x_ax`	Axis number for control
`x_scale`	Maximum velocity
`x_min`	minimum value
`x_max`	maximum value
`x_threshold`	Threshold for noise suppression
`preset`	Preset selection, currently "xbox360" or "logitechX3d"
`device`	Device name, or empty (default) for auto-detection

`x_` can be replaced by `x_` (movement forward/backward), `y_` (lateral movement), `r_` (rotation) or `tilt_` (tilt). If a preset is selected and parameters set explicitly, then the preset defaults will be overridden.

7.6 linearmovement

The `<locationvelocity/>` module can create linear motion of objects. The velocity `v` and starting position `p0` can be given in cartesian coordinates, e.g.,

```
<locationvelocity actor="/scene/obj" v="1 2 3" p0="0 0 1" t0="2"/>
```

Attributes of element **linearmovement**

name	description (type, unit)	def.
`actor`	pattern to match actor objects (string array)	
`p0`	start position at time t0 (pos, m)	0 0 0
`t0`	start time t0 (double, s)	0
`v`	velocity vector (pos, m/s)	1 1 0

All variables can be controlled via OSC; the `actor` attribute is used as path prefix. In the example above this would result in these OSC variables:

```
/scene/obj/v/x (d)
/scene/obj/v/y (d)
/scene/obj/v/z (d)
/scene/obj/p0/x (d)
/scene/obj/p0/y (d)
/scene/obj/p0/z (d)
/scene/obj/t0 (d)
/scene/obj/vpt (ddddddd)
```

Note that only setting the last OSC variable `/scene/obj/vpt` ensures an atomic operation of setting the variables. If you set it variable by variable, you may get undefined (and possibly extreme) intermediate values.

OSC variables:

path	fmt.	range	r.	description
`/.../p0/x`	f	start x-position at time t0 in m	yes	
`/.../p0/y`	f	start y-position at time t0 in m	yes	
`/.../p0/z`	f	start z-position at time t0 in m	yes	
`/.../t0`	f	reference session time in s	yes	
`/.../v/x`	f	velocity in x-direction in m/s	yes	
`/.../v/y`	f	velocity in y-direction in m/s	yes	
`/.../v/z`	f	velocity in z-direction in m/s	yes	
`/.../vpt`	ddddddd		no	

7.7 locationmodulator

Modify location periodically.

Attributes:	
`m`	Modulation depth in meter along “x y z” axis
`f`	Modulation frequency in Hz
`p0`	Start phase in degrees

7.8 locationvelocity

The `<locationvelocity/>` module was renamed to `<linearmovement/>`.

7.9 lslactor

Control position from an LSL stream (e.g., via EEG).

The translation is assumed to be in meters, the rotation is ZYX-Euler angles in radians.

Attributes:	
predicate	LSL stream predicate
channels	LSL channels, for the six translation channels (x,y,z,rotz,roty,rotx), -1 for unused
influence	Weights of channels
local	Use local (true) or global translation
incremental	Use incremental changes

7.10 motionpath

Allow motion along a predefined trajectory independently from the session time line, or optionally based on the TASCAR time line. This module needs an active session-OSC server. These OSC methods are added:

```
/motionpath/go from to
/motionpath/start
/motionpath/stop
/motionpath/locate time
/motionpath/stoptime time
```

`go` moves along the path from the time `from` until the time `to`. `start` starts the motion at the current time, `stop` stops the motion. `locate` sets the path time to `time` without changing the motion state. `stoptime` sets the time when the motion will be stopped. If the current time is after the stop time, then the current time is set to the stop time.

Attributes:	
active	Play trajectory (true), or ignore trajectory (false); default: true
tascartime	Use OSC time control (false) or tascar time line (true); default: false
id	Use id in OSC path; default: “motionpath”
sampledorientation	Sample orientation along trajectory with this distance (default: 0)

7.11 nearsensor

Attributes:	
`url`	target OSC url
`ttl`	time-to-live of UDP packets
`pattern`	pattern ob objects to detect
`parent`	name of parent object (= sensor position)
`radius`	sensor radius in meter
`mode`	operation mode: 0 = detect object origin, 1 = detect sound vertex
`path`	OSC message target path

Emit an OSC message when an object or sound vertex is near the parent object. The OSC message can be composed from sub-elements of the types `<f/>` (e.g., `<f v="1.0"/>`) (float), `<i/>` (`<i v="123"/>`) (integer) or `<s/>` (`<s v="abc"/>`) (string). Multiple sub-elements are possible.

Any number of sub-elements `<msgapp/>` (messages to be sent on approaching a target) and `<msgdep/>` (messages to be sent on departing from a target) are possible. Each message has the attribute

Attributes:	
`path`	OSC message target path

and the same sub-elements `<f/>`, `<i/>` and `<s/>` as described before.

7.12 orientationmodulator

Modify orientation around z axis periodically.

Attributes:	
`m`	Modulation depth in degrees
`f`	Modulation frequency in Hz
`p0`	Start phase in degrees

7.13 oscactor

Control position from an OSC stream

The translation is assumed to be in meters, the rotation is ZYX-Euler angles in radians.

Attributes of element **oscactor**

name	description (type, unit)	def.

actor	pattern to match actor objects (string array)	
channels	Which channels to use (int32 array)	
incremental	Add transformation to current delta transformation, e.g., when used together with other motion controllers (bool)	false
influence	Influence of OSC values on the selected movement channels (float array)	
inputchannels	Number of OSC channels (uint32)	6
local	Use transformations in local coordinates (bool)	false
path	OSC path (string)	

The influence can be controlled during run-time:

OSC variables:

path	fmt.	range	r.	description
/path/influence	ffffff		no	Influence of OSC values on the selected movement channels
/path	ffffff		no	OSC data variable

7.14 oscheadtracker

Headtracking module for MPU6050 with WiFi module, using OSC communication. To use this headtracker, connect to the WiFi provided by the headtracker.

Attributes of element **oscheadtracker**

name	description (type, unit)	def.
actor	pattern to match actor objects (string array)	
apply_loc	Apply translation based on accelerometer (not implemented) (bool)	false
apply_rot	Apply rotation based on gyroscope and accelerometer (bool)	true
autoref	Filter coefficient for estimating reference orientation from average direction, or zero for no auto-referencing (double)	1e-05
autoref_zonly	Compensate z-rotation only, requires sensor alignment (bool)	true
combinegyr	Combine quaternions with gyroscope based second estimate for increased resolution of pose estimation. (bool)	true
connectwlan	connect to sensor to external WLAN (bool)	false
eogpath	OSC target path for EOG data, or empty for no EOG (string)	
rawpath	OSC target path for raw data, or empty for no raw data (string)	
name	Prefix in OSC control variables (string)	oscheadtracker
rotpath	OSC target path for rotation data (string)	
roturl	OSC target URL for rotation data (string)	
smooth	Filter coefficient for smoothing of quaternions (double)	0.1
targetip	target IP address when using external WLAN (string)	
ttl	Time-to-live of OSC multicast data (uint32)	1
url	Target URL for OSC data logging, or empty for no datalogging (string)	
wlanpass	passphrase of external WLAN (string)	
wlanssid	SSID of external WLAN (string)	

7.15 ovheadtracker

headtracking module for MPU6050 from repository https://github.com/gisogrimm/ov-client.

Attributes of element **ovheadtracker**

name	description (type, unit)	def.
`accscale`	Scaling factor of accelerometer, default value scales to m/s^2 (double)	1670.13
`actor`	pattern to match actor objects (string array)	
`apply_loc`	Apply translation based on accelerometer (not implemented) (bool)	false
`apply_rot`	Apply rotation based on gyroscope and accelerometer (bool)	true
`autoref`	Filter coefficient for estimating reference orientation from average direction, or zero for no auto-referencing (double)	0
`autoref_zonly`	Compensate z-rotation only, requires sensor alignment (bool)	false
`axes`	Order of axes, or -1 to not use axis (int32 array)	0 1 2
`calib0path`	OSC-Path to which a trigger is sent on start of calibration path (string)	/calib0
`calib1path`	OSC-Path to which a trigger is sent on end of calibration path (string)	/calib1
`combinegyr`	Combine quaternions with gyroscope based second estimate for increased resolution of pose estimation. (bool)	true
`devices`	List of serial port device candidates (string array)	/dev/ttyUSB0 /dev/ttyUSB1 /dev/ttyUSB2
`gyrscale`	Scaling factor of gyroscope, default value scales to deg/s (double)	16.4
`levelpattern`	TASCAR internal path of level meter to read level data (string array)	
`name`	Prefix in OSC control variables (string)	ovheadtracker
`rotpath`	OSC target path for rotation data (string)	
`roturl`	OSC target URL for rotation data (string)	
`send_only_quaternion`	Send only quaternion data instead of raw sensor data (bool)	false
`smooth`	Filter coefficient for smoothing of quaternions (double)	0
`tiltmap`	tilt mapping, [in1 out1 in2 out2] (float array)	0 0 180 180
`tiltpath`	OSC path for tilt (string)	/tilt
`tilturl`	OSC target URL for tilt (string)	
`ttl`	Time-to-live of OSC multicast data (uint32)	1
`url`	Target URL for OSC data logging, or empty for no datalogging (string)	

7.16 pendulum

Generate pendular movements

Attributes:	
`amplitude`	Starting amplitude in degrees
`frequency`	Swinging frequency in Hz
`decaytime`	50% decay time of pendulum movement
`starttime`	Time when movement starts
`distance`	Length of pendulum

7.17 pos2lsl

The module **pos2lsl** sends position and orientation of TASCAR objects as OSC message. This can be used to control objects in computer graphics tools. Example:

```
<pos2lsl pattern="/*/out" transport="false"/>
```

The `pattern` attribute specifies the object (or objects) whose geometry information will be sent.

Attributes:	
`pattern`	Pattern of TASCAR object names (default: /*/*). See actor modules for details.
`transport`	Send data only while transport is rolling (default: true)

7.18 pos2osc

The module **pos2osc** sends position and orientation of TASCAR objects as OSC message. This can be used to control objects in computer graphics tools. Example:

```
<pos2osc url="osc.udp://localhost:9999/" pattern="/*/cg_*" mode="2"/>
```

The `pattern` attribute specifies the object (or objects) whose geometry information will be sent. In the example above all objects, whose name starts with `cg_` will send geometry data. The Euler-angles are sent in degrees, Cartesian coordinates in meter.

Attributes of element **pos2osc**

name	description (type, unit)	def.
`addparentname`	When sending sound vertex positions, add parent name to vertex name (bool)	false
`avatar`	Name of object to be controlled (for control of game engines) (string)	
`ignoreorientation`	Ignore delta-orientation of source, send zeros instead (bool)	false
`lookatlen`	Duration of look-at animation (for control of game engines) (double, s)	1

`mode`	Message format mode (uint32)	0
`name`	Default name used in OSC variables (string)	pos2osc
`orientationname`	Name for orientation variables (string)	/headGaze
`oscale`	Scaling factor for orientations (float)	1
`pattern`	Pattern of TASCAR object names; see actor module documentation for details. (string array)	/*/*
`sendsounds`	Send also position of sound vertices (modes 2 and 3 only) (bool)	false
`skip`	Skip frames to reduce network traffic (uint32)	0
`taumin`	Minimum period time between two transmissions. (float, s)	0
`threaded`	Use additional thread for sending data to avoid blocking of real-time audio thread (bool)	true
`transport`	Send only while transport is rolling (bool)	true
`triggered`	Send data only when triggered via OSC (bool)	false
`ttl`	Time to live of OSC multicast messages (uint32)	1
`url`	Target URL (string)	osc.udp://localhost:9999/

The operation modes are:

mode	send to...
0	/scene/name/pos (x,y,z) and /scene/name/rot (Euler-Z,Euler-Y,Euler-X)
1	/scene/name/pos (x,y,z,Euler-Z,Euler-Y,Euler-X)
2	/tascarpos (/scene/name,x,y,z,Euler-Z,Euler-Y,Euler-X)
3	/tascarpos (name,x,y,z,Euler-Z,Euler-Y,Euler-X)
4	/avatar /lookAt x,y,z,lookatlen
5	/avatar Euler-Z
6	/avatar <orientationname> Euler-Y, Euler-Z, Euler-X (delta orientation only)
7	/avatar <orientationname> Euler-Y, Euler-Z, Euler-X
8	/avatar Euler-Y, Euler-Z, Euler-X (delta orientation only, degree)
9	/avatar <orientationname> Euler-X, Euler-Y, Euler-Z (delta orientation only)
11	/avatar/<objname> x, y, z, Euler-Z, Euler-Y, Euler-X

7.19 qualisystracker

Interface for Qualisys tracking software

Attributes:

`qtmurl`	URL of qualisys track manager
`timeout`	Response timeout in seconds
`rigid`	Name of rigid to be tracked
`influence`	Weights of channels
`local`	Use local (true) or global translation
`incremental`	Use incremental changes

7.20 rotator

The `<rotator/>` module can create parametric rotation of objects around the z-axis. Four modes are supported, *linear* (`mode="0"`, default), *sigmoid* (`mode="1"`), *cosine* (`mode="2"`), and *free* (`mode="3"`).

Attributes of element **rotator**

name	description (type, unit)	def.
`actor`	pattern to match actor objects (string array)	
`mode`	Operation mode (uint32, 0|1|2|3)	0
`phi0`	Start angle (sigmoid/cosine movement) (double, deg)	-90
`phi1`	End angle (sigmoid/cosine movement) (double, deg)	90
`t0`	Starting time (double, s)	0
`t1`	End time (sigmoid/cosine movement) (double, s)	1
`w`	Angular velocity (double, deg/s)	10

OSC variables:

path	fmt.	range	r.	description
`/.../mode`	i		yes	Operation mode
`/.../phi0`	f		yes	
`/.../phi1`	f		yes	
`/.../t0`	f		yes	
`/.../t1`	f		yes	
`/.../w`	f		yes	Angular velocity in deg/s

Examples:

Linear rotation

```
<rotator mode="0" t0="2" w="10" actor="/*/out"/>
```

$$O_z = w(t - t_0) \tag{11}$$

Sigmoid rotation

```
<rotator mode="1" t0="2" t1="5" phi0="-120" phi1="10" actor="/*/out"/>
```

$$O_z = \varphi_0 + \frac{\varphi_1 - \varphi_0}{1 + e^{-2\pi(t-0.5(t_0+t_1))/(t_1-t_0)}} \tag{12}$$

Cosine rotation

```
<rotator mode="2" t0="2" t1="5" phi0="-120" phi1="10" actor="/*/out"/>
```

$$O_z = \begin{cases} \varphi_0 & t < t_0 \\ \varphi_0 + \frac{1}{2}(\varphi_1 - \varphi_0)(1 - \cos(\pi\frac{t-t_0}{t_1-t_0})) & t_0 \leq t \leq t_1 \\ \varphi_1 & t_1 < t \end{cases} \tag{13}$$

Free rotation

Same as linear, but the rotation phase is continuously incremented independent of the transport time.

7.21 serialheadtracker

Serial port headtracking module for MPU6050, equivalent to oscheadtracker, but communication via (USB) serial port.

Attributes of element **serialheadtracker**

name	description (type, unit)	def.
`actor`	pattern to match actor objects (string array)	
`apply_loc`	Apply translation based on accelerometer (not implemented) (bool)	false
`apply_rot`	Apply rotation based on gyroscope and accelerometer (bool)	true
`autoref`	Filter coefficient for estimating reference orientation from average direction, or zero for no auto-referencing (float)	1e-05
`autoref_zonly`	Compensate z-rotation only, requires sensor alignment (bool)	true
`combinegyr`	Combine quaternions with gyroscope based second estimate for increased resolution of pose estimation. (bool)	true
`name`	Prefix in OSC control variables (string)	oscheadtracker
`rotpath`	OSC target path for rotation data (string)	
`roturl`	OSC target URL for rotation data (string)	
`smooth`	Filter coefficient for smoothing of quaternions (float)	0.1
`ttl`	Time-to-live of OSC multicast data (uint32)	1
`url`	Target URL for OSC data logging, or empty for no datalogging (string)	

7.22 simplecontroller

This module creates a minimal graphical user interface for mouse and keyboard motion control of objects within a TASCAR scene. The recognized attributes are:

Attributes of element **simplecontroller**

name	description (type, unit)	def.
`actor`	pattern to match actor objects (string array)	
`maxnorm`	Maximum distance of object from origin, or zero for no limit. (double, m)	0
`vr`	Angular velocity (double, deg/s)	90
`vx`	Velocity in x direction (double, m/s)	1
`vy`	Velocity in y direction (double, m/s)	1
`vz`	Velocity in y direction (double, m/s)	1

Example:

```
<simplecontroller actor="/*/out" maxnorm="0"/>
```

7.23 skyfall

Simple physical simulation of sky dive

Attributes of element **skyfall**

name	description (type, unit)	def.
`actor`	pattern to match actor objects (string array)	
`bypass`	Bypass plugin (bool)	true
`deceleration`	Deceleration during sprung phase (double, m/s^2)	40
`friction_fall`	friction during falling phase (double)	1
`friction_jump`	friction during jumping phase (double)	0.3
`gravitation`	Gravitation constant (double, m/s^2)	-9.81
`prefix`	OSC prefix (string)	/skyfall
`vmax`	maximum velocity (double, m/s)	40
`wx`	deg/s (double, angular velocity around x axis)	0
`wy`	deg/s (double, angular velocity around y axis)	11
`wz`	deg/s (double, angular velocity around z axis)	45
`z0`	starting point (double, m)	2

OSC variables:

path	fmt.	range	r.	description
`/.../bypass`	i	bool	yes	
`/.../deceleration`	f		yes	
`/.../friction_fall`	f		yes	
`/.../friction_jump`	f		yes	
`/.../gravitation`	f		yes	
`/.../vmax`	f		yes	
`/.../wx`	f	[0,360]	yes	
`/.../wy`	f	[0,360]	yes	
`/.../wz`	f	[0,360]	yes	
`/.../z0`	f		yes	

7.24 snapangle

This plugin adjusts the orientation of some objects to the most appropriate orientation between a controller and a list of candidates.

Attributes of element **snapangle**

name	description (type, unit)	def.
`actor`	pattern to match actor objects (string array)	
`bypass`	Bypass algorithm (bool)	false
`candidates`	Path of target candidates (string)	
`name`	Default name used in OSC variables (string)	snapangle
`srcobj`	Path of source object (string)	

7.25 tracegui

A GUI module to show traces of a subset of objects, controlled by the `actor` attribute.

Attributes:	
`tracelen`	Length of trace in seconds (default: 4)
`fps`	Display frame rate, frames per second (default: 10)
`guiscale`	Zoom factor of GUI (default: 10)
`unitcircle`	Show unit circle (default: true)
`origin`	Show cross in origin (default: true)
`x`,`y`,`w`,`h`	Window position and size

An example which shows traces of all objects not starting with an “o”:

```
<modules>
  <tracegui actor="/*/[!o]*" fps="20" guiscale="2.2" tracelen="1.6"/>
</modules>
```

8 Audio plugins

Each sound vertex <sound/>, each diffuse sound field <diffuse/>, and each receiver <receiver/> can contain a list of audio plugins for processing and analysis, such as tone generators or speech analysis for lip synchronization modeling. These audio plugins are specified within the <plugins/> section within a <sound/> or <receiver/> element, e.g.:

```
    <sound name="wheel" z="-0.5">
      <plugins>
        <sndfile name="sounds/redcar_loop1.wav" levelmode="rms" level="85"/>
        <sine f="1000" a="70"/>
      </plugins>
    </sound>
```

Example 16: examples/example_audioplugins.tsc

Audio plugins may share their variables via OSC. See the list of OSC variables to check which variables can be accessed.

Audio plugins are processed in the order they appear in the configuration within the <plugins/> section. For sound vertices, they are processed before the sound is handed to the acoustic model. For receivers, audio plugins are processed after the post processing function of the render format.

To profile the plugin performance, it is possible to set the attribute `profilingpath` to an OSC path that can be recorded using the datalogging plugin. The `size` attribute of the OSC variable in the datalogging must match the number of plugins, see Example 3 in the <session/> section. The data contains the time spent in each processing cycle in seconds, for each plugin. Please note that the clock granularity is one microsecond on Linux machines.

List of audio plugins:

- **allpass**
- **bandlevel2osc**
- **bandpass**
- **const**
- **delay**
- **feedbackdelay**
- **fence**
- **filter**
- **flanger**
- **gain**
- **gainramp**

- gate
- hannenv
- identity
- level2hsv
- level2osc
- lipsync
- lipsync_paper
- lookatme
- loopmachine
- metronome
- noise
- onsetdetector
- pink
- pulse
- reclevelanalyzer
- sessiontime
- simplesynth
- sine
- sndfile
- sndfileasync
- speechactivity
- spkcalib
- spksim
- transportramp
- tubesim

8.1 allpass

Allpass filter plugin with filter design in the z-plane.

Attributes of element **allpass**

name	description (type, unit)	def.
`bypass`	Bypass plugin (bool)	false
`f`	Phase jump frequency (double, Hz)	1000
`nstages`	Number of biquad-stages (uint32)	3
`r`	Allpass pole radius (double)	0.9

OSC variables:

path	fmt.	range	r.	description
`/.../bypass`	i	bool	yes	

8.2 bandlevel2osc

Send band levels via OSC.

Attributes of element **bandlevel2osc**

name	description (type, unit)	def.
`bandwidth`	band width (float, octaves)	1
`f`	Center frequencies (float array, Hz)	250 500 1000 2000
`mode`	Level mode [dbspl|rms|max] (string)	dbspl
`path`	Target path (string)	/level
`sendwhilestopped`	Send also when transport is stopped (bool)	false
`skip`	Skip frames (uint32)	0
`threaded`	Use additional thread for sending data (bool)	true
`url`	Target URL (string)	osc.udp://localhost:9999/

If N is the number of channels and B the number of frequency bands, the OSC message will contain $N * B + 1$ floating point values. The first value contains the object time in seconds, the other floats contain the RMS level within the current audio block in dB SPL.

8.3 bandpass

4th order (two biquads) bandpass filter. Gain is normalized to zero at the geometric average of the frequencies.

Attributes of element **bandpass**

name	description (type, unit)	def.
`bypass`	bypass plugin (bool)	false
`fmax`	Maximum frequency (float, Hz)	20000
`fmin`	Minimum frequency (float, Hz)	100

OSC variables:

path	fmt.	range	r.	description
`/.../bypass`	i	bool	yes	
`/.../fmax`	f	]0,20000]	yes	Upper cutoff frequency in Hz
`/.../fmax`	ff		no	Fade the upper cutoff frequency, first parameter is new frequency in Hz, second parameter is fade duration in s
`/.../fmin`	f	]0,20000]	yes	Lower cutoff frequency in Hz
`/.../fmin`	ff		no	Fade the lower cutoff frequency, first parameter is new frequency in Hz, second parameter is fade duration in s

8.4 const

Generate constant numbers as audio signal.

Attributes of element **const**

name	description (type, unit)	def.
`a`	amplitude, one entry per channel (float array, Pa)	1

OSC variables:

path	fmt.	range	r.	description
`/.../a`	f	[0,120]	no	

8.5 delay

Delay the vertex audio signal. One entry for each audio channel is possible. If fewer values than channels are provided, the delay values starting from index zero are repeated.

Attributes of element **delay**

name	description (type, unit)	def.
`delay`	Delays in seconds (double array, s)	1

8.6 feedbackdelay

Feedback delay line.

Attributes of element **feedbackdelay**

name	description (type, unit)	def.
`dry`	Linear gain of direct input (float)	1
`f`	Resonance frequency (float, Hz)	1000

`feedback`	Linear feedback gain (float)	0.5
`maxdelay`	Maximum delay line length (uint64, samples)	44100
`wet`	Linear gain of input to delayline (float)	1

OSC variables:

path	fmt.	range	r.	description
`/.../dry`	f	[0,1]	yes	Linear gain of direct input
`/.../feedback`	f	]-1,1[	yes	Linear feedback gain
`/.../f`	f	]0,8000]	yes	Resonance frequency
`/.../wet`	f	[0,1]	yes	Linear gain of input to delayline

8.7 fence

Create an acoustic fence by increasing the gain when the object is outside a given distance from an origin. See `example_fence.tsc` for an example.

Attributes of element **fence**

name	description (type, unit)	def.
`alpha`	alpha (float)	1
`origin`	origin (pos, m)	0 0 0
`r`	r (float, m)	1
`range`	range (float, m)	0.1

OSC variables:

path	fmt.	range	r.	description
`/.../alpha`	f		yes	
`/.../range`	f		yes	
`/.../r`	f		yes	

8.8 filter

Biquad filter stage. Low-pass and high-pass use Butterworth filter design.

Attributes of element **filter**

name	description (type, unit)	def.
`Q`	quality factor (float)	1
`fc`	Cut-off frequncy (float, Hz)	1000
`gain`	equalizer gain (float, dB)	0
`highpass`	Highpass filter (true) or lowpass filter (false) (bool)	false
`mode`	filter mode: lohi, lowpass, highpass, equalizer, highshelf, lowshelf (string)	lohi

OSC variables:

path	fmt.	range	r.	description
`/.../fc`	f	]0,20000]	yes	Cutoff frequency in Hz

8.9 flanger

Flanger plugin.

Attributes of element **flanger**

name	description (type, unit)	def.
`dmax`	Upper bound of delay (float, s)	0.01
`dmin`	Lower bound of delay (float, s)	0
`feedback`	Feedback, must be between 0 and 0.999 (float)	0
`maxdelay`	Maximum delay line length (uint64, samples)	44100
`modf`	Modulation frequency (float, Hz)	1
`wet`	Linear gain of input to delayline (float)	1

OSC variables:

path	fmt.	range	r.	description
`/.../dmax`	f	[0,1]	yes	Upper bound of delay, in s
`/.../dmin`	f	[0,1]	yes	Lower bound of delay, in s
`/.../feedback`	f	[0,0.999]	yes	Feedback
`/.../modf`	f	[0,100]	yes	Modulation frequency
`/.../wet`	f	[0,1]	yes	Linear gain of input to delayline

8.10 gain

Modify gain.

Attributes of element **gain**

name	description (type, unit)	def.
`gain`	gain (float, dB)	0
`lingain`	lingain (float)	1

OSC variables:

path	fmt.	range	r.	description
`/.../fade`	ff		no	
`/.../gain`	f	[-40,10]	yes	
`/.../lingain`	f		yes	

8.11 gainramp

Modify gain.

Attributes of element **gainramp**

name	description (type, unit)	def.
`gain`	Set current gain (double, dB)	0
`maxgain`	Set maximal gain (double, dB)	0
`slope`	Set gain slope in dB/s (double, dB)	-inf

OSC variables:

path	fmt.	range	r.	description
`/.../gain`	f	[-40,10]	yes	
`/.../maxgain`	f	[-40,10]	yes	
`/.../slope`	f	[-40,10]	yes	

8.12 gate

Gate the vertex audio signal.

Attributes of element **gate**

name	description (type, unit)	def.
`bypass`	Start in bypass mode (bool)	true
`fadeinlen`	Duration of von-Hann fade in (double, s)	0.01
`fadeoutlen`	Duration of von-Hann fade out (double, s)	0.125
`holdlen`	Time to keep output after level decay below threshold (double, s)	0.125
`taurms`	RMS level estimation time constant (double, s)	0.005
`tautrack`	Min/max tracking time constant (double, s)	30
`threshold`	Threshold value between 0 and 1 (double)	0.125

OSC variables:

path	fmt.	range	r.	description
`/.../bypass`	i	bool	yes	
`/.../taurms`	f		yes	
`/.../tautrack`	f		yes	
`/.../threshold`	f		yes	

8.13 hannenv

Apply periodic von-Hann ramps to the signal.

Attributes of element **hannenv**

name	description (type, unit)	def.
`period`	Period time (double, s)	2
`ramp1`	First ramp length (double, s)	0.25
`ramp2`	Second ramp length (double, s)	0.25
`steady`	Duration of steady state (double, s)	0.5
`t0`	Start time (double, s)	0

8.14 identity

As the name suggests, this plugin returns the unmodified input signal.

8.15 level2hsv

Convert sound pressure level to light intensity (value component of hsv variable) of a OSC lamp path.

When more than one channel is available, only the first channel is used.

Attributes of element **level2hsv**

name	description (type, unit)	def.
`active`	start activated (bool)	true
`decay`	decay filter coefficient (double)	0
`frange`	Frequency range in bandpass mode (float array, Hz)	62.5 4000
`hue`	Hue component (0-360) (float, degree)	0
`lrange`	Level range (float array, dB)	40 90
`mode`	Level mode [dbspl|rms|max] (string)	dbspl
`path`	Target path (string array)	/hsv
`saturation`	Saturation component (0-1) (float)	1
`skip`	Skip frames (uint32)	0
`tau`	Leq duration, or 0 to use block size (float, s)	0
`url`	Target URL (string)	osc.udp://localhost:9999/
`weight`	Level meter weight (f-weight)	Z

OSC control:

OSC variables:

path	fmt.	range	r.	description
`/.../active`	i	bool	yes	
`/.../decay`	f	[0,1[	yes	decay coeficient
`/.../hue`	f	[0,360]	yes	Hue component (0-360 degree)
`/.../lrange`	ff		no	Level range in dB
`/.../saturation`	f	[0,1]	yes	Saturation component (0-1)

8.16 level2osc

Send levels via OSC.

Attributes of element **level2osc**

name	description (type, unit)	def.
firstpar	First parameter, or -1 to use current session time. (double)	-1
frange	Frequency range in bandpass mode (float array, Hz)	62.5 4000
mode	Level mode [dbspl|rms|max] (string)	dbspl
path	Target path (string)	/level
sendwhilestopped	Send also when transport is stopped (bool)	false
skip	Skip frames (uint32)	0
tau	Leq duration, or 0 to use block size (float, s)	0
threaded	Use additional thread for sending data (bool)	true
url	Target URL (string)	osc.udp://localhost:9999/
weights	Level meter weights (f-weight array)	Z

The number of channels, denoted by N, and the number of frequency weights, represented by W, determine the number of floating-point values contained in the OSC message. The first of these values represents the object time in seconds, while the remaining values indicate the RMS level within the current audio block in dB SPL.

OSC variables:

path	fmt.	range	r.	description
/.../firstpar	f		yes	

8.17 lipsync

Lip synchronization module, similar to lipsync_paper .

Attributes of element **lipsync**

name	description (type, unit)	def.
dynamicrange	Mapped dynamic range (float, dB)	165
energypath	OSC destination for sending format energies, or empty for no energy messages (string)	
maxspeechlevel	Level normalization (float, dB)	48
onchangecount	Maximum number of repetitions of equal messages in “onchange” mode (uint32)	3
path	OSC destination of blendshape messages (empty: use parent name) (string)	
scale	Scaling factor of blend shapes; 3 values: kiss, jaw, lipsclosed (pos)	1 1 1
sendmode	Sending mode, one of “always”, “transport”, or “onchange” (string)	always
smoothing	Smoothing time constant (float, s)	0.02
strmsg	Message string to be added to OSC messages before blend shapes (string)	/lipsync

`threaded`	Use additional thread for sending data (bool)	true
`threshold`	Noise threshold, range 0-1 (float)	0.5
`url`	Target OSC URL (string)	osc.udp://localhost:9999/
`vocalTract`	Vocal tract scaling factor (float)	1

OSC variables:

path	fmt.	range	r.	description
`/.../active`	i	bool	yes	
`/.../dynamicrange`	f		yes	
`/.../maxspeechlevel`	f		yes	
`/.../smoothing`	f		yes	
`/.../threshold`	f		yes	
`/.../vocalTract`	f		yes	

8.18 lipsync_paper

Module to control lip synchronization as used in Llorach et al. (2016).

Attributes of element **lipsync_paper**

name	description (type, unit)	def.
`dynamicrange`	Mapped dynamic range (double, dB)	165
`energypath`	OSC destination for sending format energies, or empty for no energy messages (string)	
`maxspeechlevel`	Level normalization (double, dB)	48
`onchangecount`	Maximum number of repetitions of equal messages in “onchange” mode (uint32)	3
`path`	OSC destination of blendshape messages (empty: use parent name) (string)	
`scale`	Scaling factor of blend shapes; 3 values: kiss, jaw, lipsclosed (pos)	1 1 1
`sendmode`	Sending mode, one of “always”, “transport”, or “onchange” (string)	always
`smoothing`	Smoothing time constant (double, s)	0.04
`strmsg`	Message string to be added to OSC messages before blend shapes (string)	/lipsync
`threaded`	Use additional thread for sending data (bool)	true
`threshold`	Noise threshold, range 0-1 (double)	0.5
`url`	Target OSC URL (string)	osc.udp://localhost:9999/
`vocalTract`	Vocal tract scaling factor (double)	1

OSC variables:

path	fmt.	range	r.	description
`/.../active`	i	bool	yes	
`/.../dynamicrange`	f		yes	
`/.../maxspeechlevel`	f		yes	
`/.../smoothing`	f		yes	
`/.../threshold`	f		yes	
`/.../vocalTract`	f		yes	

8.19 lookatme

Onset-detector for avatar head orientation control.

Attributes of element **lookatme**

name	description (type, unit)	def.
`animation`	Animation name (or empty for no animation) (string)	
`fadelen`	Motion duration after threshold (double, s)	1
`levelpath`	Destination path of level logging (or empty) (string)	
`paths`	Space-separated list of target paths (string array)	
`pos_offset`	Position to look at on offset (or empty for no change of look direction) (pos, m)	0 0 0
`pos_onset`	Position to look at on onset (or empty to look at vertex position) (pos, m)	0 0 0
`tau`	Time constant of level estimation (double, s)	1
`threshold`	Level threshold (double, dB SPL)	53.9794
`thresholdpath`	Destination path of threshold criterion (or empty) (string)	
`url`	Target OSC URL (string)	osc.udp://localhost:9999/

OSC variables:

path	fmt.	range	r.	description
`/.../active`	i	bool	yes	
`/.../discordantLS`	i	bool	yes	
`/.../threshold`	f	[0,120]	yes	

8.20 loopmachine

Simple loop machine with OSC control.

Attributes of element **loopmachine**

name	description (type, unit)	def.
`bpm`	Beats per minute (double)	120
`bypass`	Start in bypass mode (bool)	false
`delaycomp`	Delay compensation (double, s)	0
`durationbeats`	Record duration (double, beats)	4
`gain`	Playback gain (float, dB)	0
`muteinput`	Mute input while not recording (bool)	false
`ramplen`	Ramp length (double, s)	0.01

OSC variables:

path	fmt.	range	r.	description

/.../bypass	i	bool	yes	bypass, 0 means loop is added to output
/.../clear			no	clear current recording
/.../gaindb	f		yes	dB gain applied to loop
/.../gain	f		yes	linear gain applied to loop
/.../muteinput	i	bool	yes	mute the input (play only loop)
/.../record			no	start recording

8.21 metronome

Attributes of element **metronome**

name	description (type, unit)	def.
a1	Amplitude of first beat (double, dB SPL)	40
ao	Amplitude of other beats (double, dB SPL)	33.9794
bpb	Beats per bar (int32 array)	4
bpm	Beats per minute (double)	120
bypass	Load in bypass mode (bool)	false
changeonone	Apply OSC parameter changes on next bar (bool)	false
fres1	Resonance frequency of first beat (double, Hz)	1000
freso	Resonance frequency of other beats (double, Hz)	600
q1	Filter resonance of first beat (double)	0.997
qo	Filter resonance of other beats (double)	0.997
sync	Use object time synchronization (bool)	false

OSC messages can be dispatched on beat one using the “/dispatchin” OSC variables.

OSC variables:

path	fmt.	range	r.	description
/.../a1	f	[0,120]	yes	
/.../ao	f	[0,120]	yes	
/.../bpb	i		no	
/.../bpm	f		yes	
/.../bypass	i	bool	yes	
/.../changeonone	i	bool	yes	
/.../dispatchin	i		yes	
/.../dispatchmsg	(any)		no	
/.../dispatchpath	s	string	yes	
/.../filter/f1	f		yes	
/.../filter/fo	f		yes	
/.../filter/q1	f		yes	
/.../filter/qo	f		yes	
/.../sync	i	bool	yes	

Each sub-message can be defined using a `<msg/>` element.

Attributes of element **msg**

name	description (type, unit)	def.
`path`	OSC path name (string)	

8.22 noise

White noise generator.

Attributes of element **noise**

name	description (type, unit)	def.
`a`	Noise level (double, dB SPL)	33.9794

OSC variables:

path	fmt.	range	r.	description
`/.../a`	f	[0,120]	yes	

8.23 onsetdetector

Onset detector for automated animations.

Attributes of element **onsetdetector**

name	description (type, unit)	def.
`path`	Destination OSC path (string)	
`side`	(string)	
`tau`	Level estimator time constant (double, s)	1
`taumin`	Trigger blocking time (double, s)	0.05
`threshold`	Detection threshold (double, dB SPL)	53.9794
`url`	Destination OSC URL (string)	osc.udp://localhost:9999/

8.24 pink

Add a (band limited) frequency-dependent noise to the input signal. The power spectral density P is

$$P(f) \propto \frac{1}{f^{\alpha}} \tag{14}$$

in the interval $f_{min} \leq f \leq f_{max}$, and zero otherwise.

Attributes of element **pink**

name	description (type, unit)	def.
`alpha`	Frequency exponent alpha, 1 = pink (double)	2

`fmax`	Maximum frequency (double, Hz)	4000
`fmin`	Minimum frequency (double, Hz)	62.5
`level`	RMS level (double, dB SPL)	33.9794
`mute`	load muted (bool)	false
`period`	Period time of frozen noise (double, s)	4
`use_transport`	Play only if transport is running (bool)	false

If `use_transport` is activated, the object time is used for the frozen noise position.

OSC variables:

path	fmt.	range	r.	description
`/.../level`	f	[0,120]	yes	
`/.../mute`	i	bool	yes	
`/.../use_transport`	i	bool	yes	

8.25 pulse

Add a pulse train to the input signal.

Attributes of element **pulse**

name	description (type, unit)	def.
`a`	Pulse amplitude (double, Pa)	0.001
`f`	Pulse frequency (double, Hz)	1000

OSC variables:

path	fmt.	range	r.	description
`/.../a`	f	[0,120]	yes	
`/.../f`	f		yes	

8.26 reclevelanalyzer

This plugin can analyse input levels and send percentile peak levels and A-weighted RMS levels as an internal OSC message. The audio signal is divided into segments of `tau_segment` duration. The peak level and the A-weighted RMS level are calculated in each segment. All short-term levels within `tau_analysis` are collected and on each analysis call the levels are sorted to derive percentile levels.

Attributes of element **reclevelanalyzer**

name	description (type, unit)	def.
`p`	Percentile values (float array)	0 0.5 0.65 0.95 1

`path`	OSC path. It will contain 2*sizeof(p)+1 values, the first is channel, then sizeof(p) peak values, then sizeof(p) RMS values. (string)	/reclevelanalyzer
`tau_analysis`	Length of analysis window (float, s)	30
`tau_segment`	Period time of one level segment (float, s)	0.125
`triggered`	Update analysis only when triggered via OSC message. (bool)	false
`update_interval`	Update interval of analysis (each update will analyse full window) (float, s)	5

OSC variables:

path	fmt.	range	r.	description
`/.../trigger`			no	

8.27 sessiontime

This audio plugin returns the session time in seconds as output. It has no configurable parameters.

8.28 simplesynth

Simple MIDI synthesizer.

Attributes of element **simplesynth**

name	description (type, unit)	def.
`autoconnect`	Autoconnect to input ports (bool)	false
`connect`	ALSA device name to connect to (string)	
`decay`	Tone decay time (float, s)	4
`decaydamping`	Damping tone decay time (float, s)	8
`decaynoise`	Noise decay time (float, s)	0.5
`decayoffset`	Tone offset decay time (float, s)	0.5
`detune`	Detuning frequency in Hz (float, Hz)	1
`f0`	Tuning frequency (float, Hz)	440
`gamma`	Velocity gamma value (float)	1
`level`	Sound level (float, dB SPL)	69.5424
`maxvoices`	Maximum number of polyphonic voices (uint32)	8
`midichannel`	MIDI channel (int32)	0
`noisemin`	Minimum noise amplitude during sustain (float)	0
`noiseq`	Noise resonance filter Q factor (float)	0.5
`noiseweight`	Noise to tone ratio (float)	0
`onset`	Onset time (float, s)	0.02
`partialweights`	Linear amplitudes of tone components (float array)	1 0.562 0.316 0.355 0.282 0.355 0.2 0.0891 0.0398 0.0398 0.0398
`tuning`	Tuning (string, equal|werkmeister3|meantone4|meantone6|valotti)	equal

OSC variables:

path	fmt.	range	r.	description
`/.../decaydamping`	f	[0,10]	yes	Damping decay in s
`/.../decay`	f	]0,20]	yes	Decay time in s
`/.../decaynoise`	f	[0,4]	yes	Noise decay time in s
`/.../decayoffset`	f	]0,20]	yes	Offset decay time in s
`/.../detune`	f	[-10,10]	yes	Detuning in Hz
`/.../f0`	f	[100,1000]	yes	Tuning frequency in Hz
`/.../level`	f	[0,100]	yes	Sound level in dB SPL
`/.../noiseq`	f	]0,1[	yes	Noise resonance filter Q factor
`/.../noiseweight`	f	[0,1]	yes	Noise to tone ratio
`/.../onset`	f	[0,0.2]	yes	Onset duration in s

8.29 sine

Add a sine wave to the input signal.

Attributes of element **sine**

name	description (type, unit)	def.
`a`	Amplitude (double, dB SPL)	33.9794
`f`	Frequency (double, Hz)	1000

OSC variables:

path	fmt.	range	r.	description
`/.../a`	f	[0,100]	yes	Amplitude in dB SPL
`/.../f`	f	]0,20000]	yes	Frequency in Hz

8.30 sndfile

The 'sndfile' plugin reads sound files and adds their content to the audio signal. Playback can be controlled by the session timeline, triggered by OSC messages, or independent of both. The libsndfile library (http://www.mega-nerd.com/libsndfile/) is used internally, so all file and sample formats supported by this library are also supported by this plugin.

Attributes of element **sndfile**

name	description (type, unit)	def.
`attribution`	attribution of license, if applicable (string)	
`channel`	First sound file channel to be used, zero-base (uint32)	0
`channelorder`	Channel order in case of First Order Ambisonics files, "FuMa", "ACN" or "none" (string, FuMa|ACN|none)	
`length`	length of sound sample, or 0 to use whole file length (double, s)	0
`level`	level, meaning depends on `levelmode` (double, dB)	-inf

`levelmode`	level mode, “rms”, “peak” or “calib” (string)	rms
`license`	license type (string)	
`loop`	loop count or 0 for infinite looping (uint32)	1
`loopcrossexp`	exponent of von-Hann crossfade for seamless loop (float)	1
`loopcrosslen`	duration of crossfade for seamless loop (float, s)	0
`mute`	Load muted (bool)	false
`name`	Sound file name (string)	
`normalization`	Normalization in case of First Order Ambisonics files. (string, FuMa|SN3D)	FuMa
`position`	Start position within the scene (double, s)	0
`rampend`	von-Hann ramp duration at end of sound (float, s)	0
`rampstart`	von-Hann ramp duration at start of sound (float, s)	0
`resample`	Allow resampling to current session sample rate (bool)	false
`start`	Start position within the file (double, s)	0
`transport`	Use session time base (bool)	true
`triggered`	Use OSC variable ‘/loop’ to trigger playback (ignores attributes ‘position’ and ‘loop’) (bool)	false
`weighting`	level weighting for RMS mode (f-weight)	Z

Multi-channel sound files

If the plugin receives multiple channels (e.g., when used in a receiver, a diffuse sound field or a multichannel route), all channels starting with the channel number `channel` are returned. If the file does not contain a sufficient number of channels, silence is returned for all channels not available in the sound file.

If the number of plugin channels (not sound file channels) is four, and the attribute `channelorder` is not “none”, a First Order Ambisonics sound file with SN3D normalization is assumed. In that case, the `channelorder` should be set to the correct channel order.

Calibration of levels

In the level mode “rms”, the RMS value of the first used channel will be used for adjusting the level, i.e., all channels will be scaled with the same value such that the first channel has the RMS level `level`.

- Level mode “rms” scales the signal so the RMS of the first channel corresponds to `level`.
- Level mode “peak” scales the signal so the peak over all channels corresponds to `level`.
- Level mode “calib” scales the signal by `level` minus 93.979 dB.

Internally, the signal is measured in Pascal. Therefore, a signal with an RMS value of 1 corresponds to a sound pressure level of 93.979 dB.

Please note that currently the calibration level and the gain of input ports also affects the calibration of the plugins.

The level calibration is applied before calculating any ramps.

Temporal alignment

All times are defined relative to the object time of the sound file plugin's parent object. In most cases this is equivalent to the session time, however, it can be changed with the `start` attribute of the objects in scenes. If the parent object is not within a scene (e.g., a 'route' module), the session time is used.

See also Figure 13 for more details on the time and position conventions.

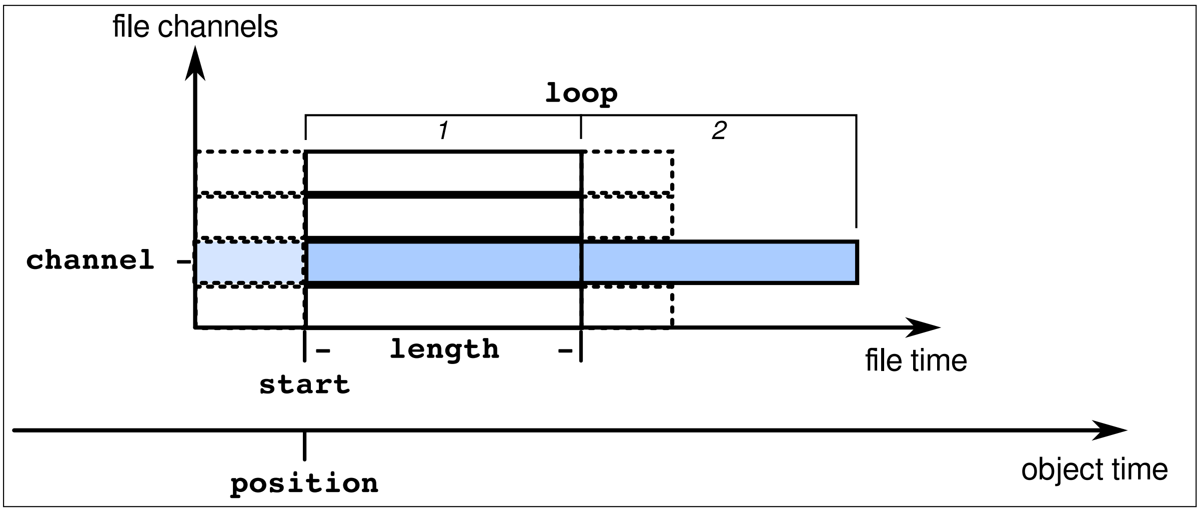


Figure 13: Temporal alignment of sounds added with the `sndfile` audio plugin.

OSC control

To load a file via OSC, send to the path `/loadfile` (for full path check the list of OSC variables in TASCAR) with two strings and a float parameter. The first string is the file name, either as absolute path or relative to the current session file. The second string is the level mode, see above for details. The third parameter is the level in dB, again see above for details.

```
/scene/in/0/ap0/sndfile/loadfile sound.wav rms 50
```

If an invalid file name or level mode is provided, a warning is printed to the console running TASCAR (to see such warnings start TASCAR from a terminal).

List of OSC variables:

name	format	description
`/loop`	i	In normal mode: Loop count, 0 for infitie loop. In "triggered" mode: Trigger of playback, number defines number of repetitions, 0 will stop playback.
`/position`	f	Position in scene in seconds
`/start`	f	Position in sound file in seconds
`/loadfile`	s	Load file with pre-configured level mode and level
`/loadfile`	ssf	Load file with level mode and level (see above)

`/mute`	i	Mute state

OSC variables:

path	fmt.	range	r.	description
`/.../loadfile`	s		no	
`/.../loadfile`	ssf		no	
`/.../loop`	i		yes	
`/.../mute`	i	bool	yes	
`/.../position`	f		yes	temporal position relative to object time, in seconds
`/.../rampend`	f	[0,10]	yes	Ramp duration in s at end of sound
`/.../rampstart`	f	[0,10]	yes	Ramp duration in s at start of sound
`/.../start`	f		yes	number of seconds to cut at the beginning of the sound file

`/position` and `/loop` will affect the file which is loaded next. It will not affect the current file.

8.31 sndfileasync

Add a sound file and play back at a given time, with asynchronous file access. This plugin provides an alternative to the sndfile audio plugin (see 8.30). It does not require to load the full file during session load, which can be advantageous for huge files. As a drawback, it is not possible to configure absolute RMS value, and dropouts may occur if the file system is slower than required. If the plugin receives multiple channels (e.g., when used in a receiver or a diffuse sound field), all channels starting at channel number `channel` will be returned. If the file does not contain a sufficient number of channels, silence will be returned for all channels not available in the sound file.

Attributes of element **sndfileasync**

name	description (type, unit)	def.
`attribution`	attribution of license, if applicable (string)	
`caliblevel`	Calibration level (double, dB SPL)	93.9794
`channel`	First sound file channel to be used, zero-base (uint32)	0
`license`	license type (string)	
`loop`	loop count or 0 for infinite looping (uint32)	1
`mute`	Load muted (bool)	false
`name`	Sound file name (string)	
`position`	Start position within the scene (double, s)	0
`transport`	Use session time base (bool)	true

OSC variables:

path	fmt.	range	r.	description
`/.../mute`	i	bool	yes	

93.979 dB corresponds internally to a full-scale signal.

8.32 speechactivity

Speech activity and onset detector. This plugin creates an LSL outlet and sends the states via OSC.

Attributes of element **speechactivity**

name	description (type, unit)	def.
`path`	OSC destination path (string)	/in.0
`tauenv`	Envelope tracking time constant (double, s)	1
`tauonset`	Onset detection time constant (double, s)	1
`threshold`	Envelope threshold (double, dB SPL)	48.9794
`transitionsonly`	Send only when a transition occurs (bool)	false
`url`	OSC destination URL (string)	osc.udp://localhost:9999/

8.33 spkcalib

This plugin allows to use a loudspeaker definition file for calibration processing. Typical application is in a standalone route or as post processing of virtual stereo microphones. Please note that diffuse sound field properties are not applicable. Also port connections defined in the loudspeaker layout are not applied.

The number of channels must match the total number of output channels (main speaker, subwoofer, and convolution channels).

See also section 10.1 for a description of the loudspeaker calibration method.

Attributes of element **spkcalib**

name	description (type, unit)	def.
`layout`	name of speaker layout file (string)	

8.34 spksim

This plugin implements a loudspeaker simulation, which creates distortion.

First, the input $x(t)$ is filtered with the 2nd order resonance filter. The filtered signal $x_r(t)$ is then distorted sample-wise,

$$x_d(t) = \frac{s}{s + |x_r(t)|} x_r(t), \tag{15}$$

with the distortion factor s. Larger values of s lead to a smaller distortion. The coupling of the speaker membrane to the air is simulated using a derivative high-pass filter:

$$y(t) = g \frac{\mathrm{d}}{\mathrm{d}t} x_d(t) \tag{16}$$

Attributes of element **spksim**

name	description (type, unit)	def.
`bypass`	Bypass plugin (bool)	false
`fres`	Resonance frequency (double, Hz)	1200
`gain`	Post-gain g (double, dB)	0
`q`	q-factor of the resonance filter (double)	0.8
`scale`	Distortion factor s (double)	0.5
`wet`	Wet (1) - dry (0) mixture gain (float)	1

OSC variables:

path	fmt.	range	r.	description
`/.../bypass`	i	bool	yes	
`/.../fres`	f	[1,10000]	yes	Resonance frequency in Hz
`/.../gain`	f	[-40,40]	yes	Post-gain in dB
`/.../q`	f	]0,1[	yes	q-factor of the resonance filter
`/.../scale`	f		yes	
`/.../wet`	f	[0,1]	yes	

8.35 transportramp

Apply a raised cosine-ramp after changes of the transport state. The duration of the ramp can be controlled separately for transitions from stopped to rolling (`startduration`) and from rolling to stopped (`endduration`).

If the ramps are not pre-calculated (`precalc="false"`), the duration can be changed via OSC.

Attributes of element **transportramp**

name	description (type, unit)	def.
`endduration`	Duration of ramp when transport is switched from “rolling” to “stopped” (float, s)	0.025
`precalc`	Operation mode, to switch between precalculated and online-generated ramps (bool)	true
`startduration`	Duration of ramp when transport is switched from “stopped” to “rolling” (float, s)	0.025

8.36 tubesim

This plugin implements a vacuum tube simulation that generates distortions.

This simulation applies a qualitative model of vacuum tubes. It consists of two stages: First, the output characteristics of a triode vacuum tube are simulated:

$$I(x) = \max\{x + x_0, 0\}^{\frac{3}{2}} \tag{17}$$

Here x corresponds to the grid voltage, x_0 to the grid bias voltage and I to the anode current. In this qualitative model, the anode voltage is not explicitly considered. The second stage simulates the overdrive:

$$\hat{I}(x) = \frac{I(x)}{I(x) + s} \tag{18}$$

with the saturation parameter s, and the limited anode current $\hat{I}$. The offset of the simulation output signal is then corrected, and a pre-gain p_i and a post-gain p_o is applied:

$$y(x) = g_o \cdot \left(\hat{I}\left(g_i \cdot x\right) - \hat{I}\left(x_0\right)\right) \tag{19}$$

The resulting input-output characteristics, sine waveform and distortion spectrum is shown in Figure 14.

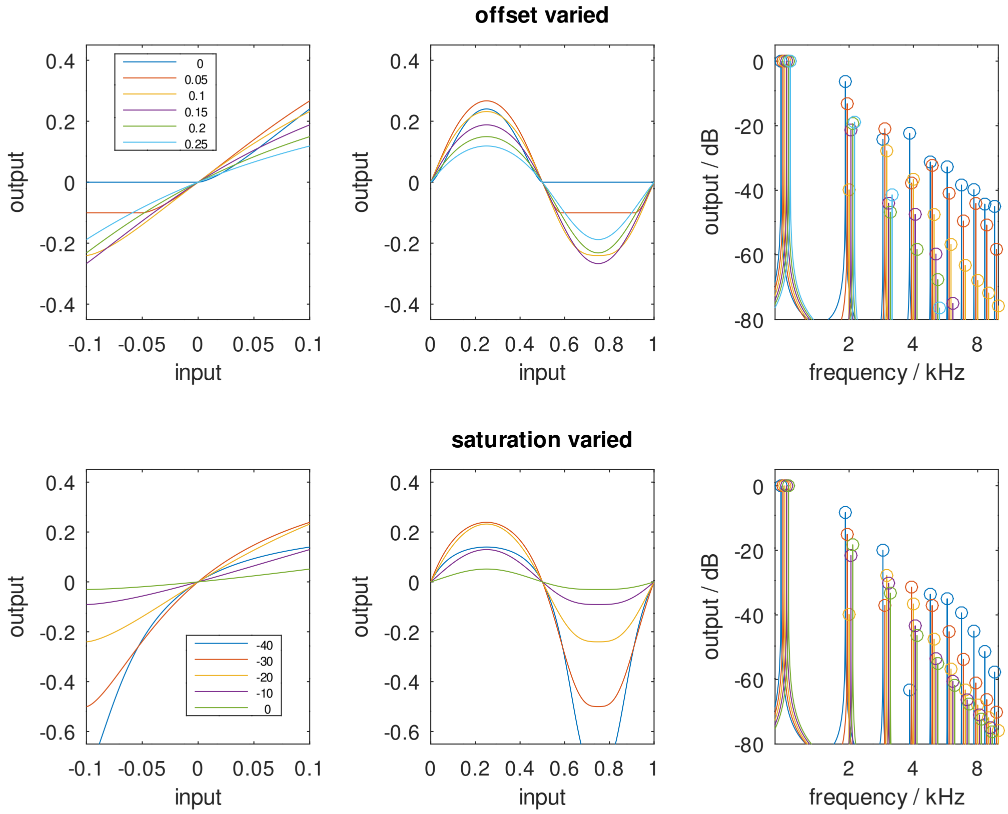


Figure 14: Input-output characteristics (left panel), sine waveform (middle panel) and distortion spectrum (right panel) of the tube simulation for various offset parameters x_0 (upper row) and saturation values $20 \log 10(s)$ (lower row).

Attributes of element **tubesim**

name	description (type, unit)	def.
`bypass`	Bypass plugin (bool)	false
`offset`	Input offset x_0 (float)	0.5
`postgain`	Post-gain g_o (float, dB)	0

`pregain`	Pre-gain g_i (float, dB)	0
`saturation`	Saturation parameter s (float, dB)	-6.0206
`wet`	Wet (1) - dry (0) mixture gain (float)	1

OSC variables:

path	fmt.	range	r.	description
`/.../bypass`	i	bool	yes	
`/.../offset`	f	[0,1]	yes	Input offset
`/.../postgain`	f	[-50,10]	yes	Output gain in dB
`/.../pregain`	f	[-10,50]	yes	Input gain in dB
`/.../saturation`	f	[-40,0]	yes	Saturation threshold in dB
`/.../wet`	f	[0,1]	yes	
`/.../wet`	ff		no	

9 Spatial mask plugins

Spatial masks can be used to control a direction dependent gain in receivers. This gain is applied independent of the receiver type, i.e., the same spatial gain map can be created for any type of receiver, from omni-directional, via binaural up to multi-channel loudspeaker receiver types.

To add a spatial mask to a receiver, in any receiver type add a `<maskplugin/>` element within the receiver section, e.g.:

```
<receiver type="omni">
  <maskplugin type="multibeam" numbeams="2" az="30 -90"/>
</receiver>
```

List of mask plugins:

- **fig8**
- **multibeam**

9.1 fig8

Attributes of maskplugin element **fig8**

name	description (type, unit)	def.
`drawradius`	Draw mask plugin with this radius in TASCAR GUI, 0 for no drawing. (float, m)	0
`type`	mask plugin type (string)	

9.2 multibeam

Add multiple steerable beams. The directional gain g as a function of the incident direction $\mathbf{p}$ is defined as

$$g(\mathbf{p}) = g_{\min} + (1 - g_{\min}) \sum_{k=1}^{N} g_k \frac{(1 + \cos(\min\{\pi, s_k \arccos(\mathbf{p} \cdot \mathbf{p}_k)\}))}{2} \tag{20}$$

with the minimum gain $g_{\min}$, the number of beams N, the on-axis gain g_k, the selectivity s_k and the steering vector $\mathbf{p}_k$.

Attributes of maskplugin element **multibeam**

name	description (type, unit)	def.
`az`	Azimuth of steering vectors (float array, deg)	0
`drawradius`	Draw mask plugin with this radius in TASCAR GUI, 0 for no drawing. (float, m)	0

`el`	Elevation of steering vectors (float array, deg)	0
`gain`	On-axis gain (float array, dB)	0
`maxgain`	Maximum gain (float, dB)	0
`mingain`	Minimum gain (float, dB)	-inf
`numbeams`	Number of beams (uint32)	1
`selectivity`	Selectivity, 0 = omni, 1 = cardioid (6 dB threshold) (float array, 1/pi)	1
`type`	mask plugin type (string)	

10 Calibration and level metering

TASCAR offers a level meter for each primary or diffuse sound field and receiver. In the level meters, root-mean-square (RMS) values in dB SPL, averaged over the past two seconds, are shown. In TASCAR, internal values are measured in Pa. This means that a sinusoid with an amplitude of one corresponds to a level of 91 dB SPL. The level of sound sources corresponds to the anechoic free field level in a distance of 1 m.

Each input port (`<sound/>` element) and output port (`<receiver/>` element) of TASCAR can be calibrated with the calibration level attribute `caliblevel`.

At the input, a full-scale sine wave corresponds to `caliblevel`-3 dB (because the RMS of a sine wave is -3 dB). This means that in case of the sine wave, the level of that sound source is 91 dB SPL, in a 1 m distance and anechoic conditions. The last bit is important: In virtual acoustics we cannot easily calibrate the level of sound sources at the listening position. In anechoic conditions this can be calculated with the $\frac{1}{r}$ amplitude law, but in case of reflections this $\frac{1}{r}$ law is not valid anymore.

For the sine wave the CREST-factor (difference between peak and RMS level) is 3 dB, but for speech this is roughly 20-24 dB. Thus typically for speech one will need a much higher `caliblevel` than 93 dB, because otherwise a full-scale speech signal would result in only 70 dB SPL. Typically, any speech test software will have some output calibration value. In case of the Oldenburg Measurement Applications (OMA) this is the same as the `caliblevel` of TASCAR. Most likely the value of it will be in the order of 120 dB SPL (similar to the `caliblevel` of the TASCAR receiver). If the `caliblevel`-value of the speech test software is known, exactly the same value should be used for the TASCAR input `caliblevel`. In that case, the input level meters of TASCAR should show the same values as the output level meters of the speech test.

For a calibration of loudspeaker layouts, it is recommended to use the tool “tascar_spkcalib” (see section 10.1).

To measure the sound pressure level in a virtual acoustic environment, one can place an omni-directional microphone at the position of the main output receiver. This omni-directional level meter should show the same numbers as a real physical sound level meter in the center of the physical reproduction system. The sound level meters need to be configured to “unweighted”, “Z-weighted” or “C-weighted” settings. Please be aware of the fact that in “unweighted” mode the background noise levels can be in the order of 40-60 dB, due to ventilation of the room, door slamming in the building, steps, nearby trains and plains etc., which contain extremely low frequencies.

The TASCAR level meters support three different frequency weightings: “Z” or unweighted mode, “C” weighting (62.5 Hz to 4 kHz) and a “bandpass” weighting (500 Hz to 4 kHz).

10.1 Calibrating loudspeaker layouts with `tascar_spkcalib`

All loudspeaker-based rendering methods (e.g., those depending on a loudspeaker layout file) should result in identical levels at the listening positions for virtual sound sources from the directions of the loudspeakers (the levels of interpolated virtual sources may differ due to differences in the rendering method).

The calibration of loudspeaker arrays consists of three steps: a) calibration of the differences between the loudspeakers (with optional spectral correction), b) calibration of the reference level for the reproduction of point sources and c) calibration of the gain correction for the reproduction of diffuse sound fields. The calibration assistant `tascar_spkcalib` guides you step by step through this calibration process.

If the wizard is started without specifying a layout file, a page for selecting a layout opens first. In the next step the calibration parameters can be revised (see Figure 15). “fmin” and “fmax” determine the frequency range of the calibration stimulus. Within the given frequency range a $1/f$ characteristic is used. “duration” defines the duration of the level measurement, “prewait” the waiting time between switching on the test stimulus and starting the level measurement. The target level is specified in the “reference level” field. The other fields refer to the frequency response correction: “bands per octave” defines the frequency resolution of the analysis filter bank. “overlap in bands” designates a spectral smoothing over adjacent frequency bands, e.g. to minimize the influence of notches. “max number of filter stages” is the maximum number of equalizer stages, where each stage is realized by a biquad filter.

If the box “initial calibration” is selected, then first the operating point is determined interactively before the calibration process is started (see Figure 16).

At least one measurement microphone is required for the calibration (if several measurement microphones are used, then the intensities are averaged over all microphones). The inputs to which the measurement microphones are connected as well as their calibration levels can be specified in the lower area of the window.

When the calibration is complete, the layout-specific parameters are saved in the layout file. However, all values can also be saved as default values, these are then stored in the file `.tascardefaults.xml` in the home directory.

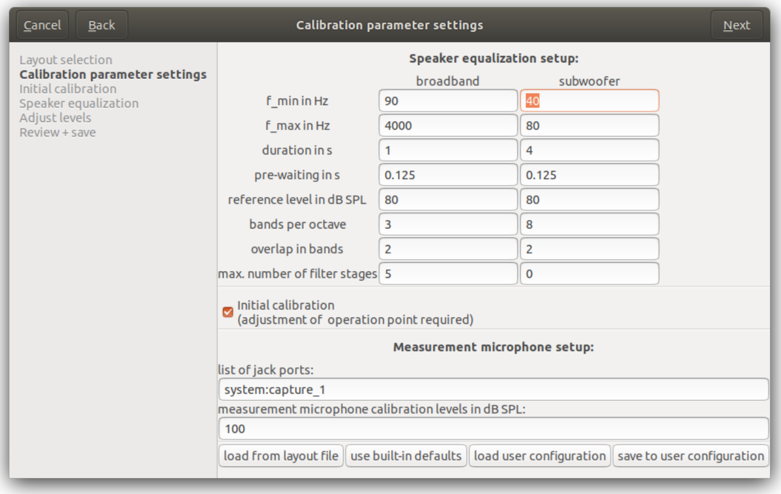


Figure 15: Revision of calibration parameters in the calibration assistant.

In the next step, the differences between the loudspeakers can be equalized (see Figure

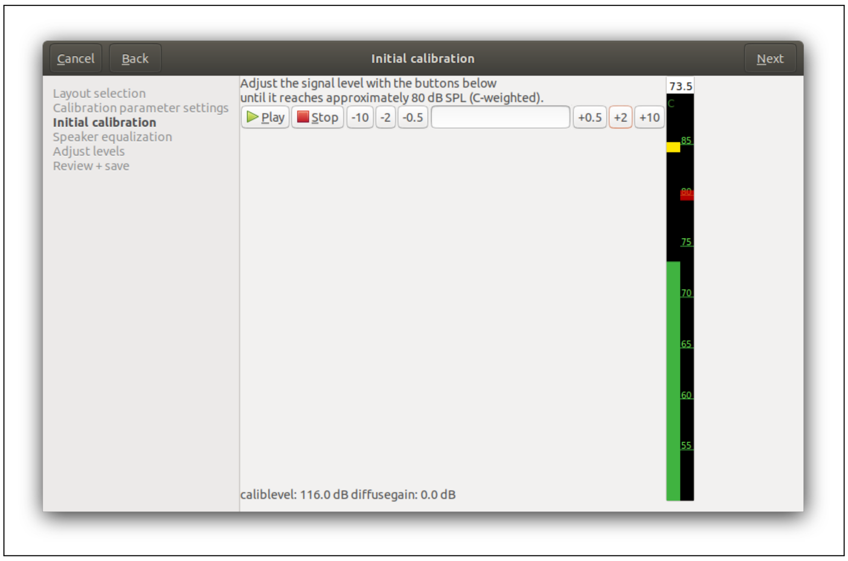


Figure 16: Interactive adjustment of the operation point.

17). If spectral equalization is activated, in the first step the frequency response is measured using an analysis filter bank. Then, the broad band level at the measurement microphone is measured for each loudspaker. Differences between loudspeakers will be equalized.

In the display, the resulting loudspeaker gain is shown (e.g., `g = 0.0 dB`). Furthermore, the recording level `Lmic` and the recording coherence between the test signal and the recorded signal `c` are shown, for each microphone. Recording levels below -50 dB FS can indicate problems with the microphone, e.g., missing phantom power or wrong input channel. Coherence values below 0.75 can be an indication for poor signal-to-noise ratio. If these values are critical for only a single loudspeaker, it is likely that one loudspeaker channel is not connected or distorted.

In the actual calibration step (Figure 18) the playback level of a stimulus can be adjusted until the desired reference level is reached. For level metering either a level meter or the connected measurement microphone (if calibrated) can be used. For the point source calibration, the stimulus is played via the first loudspeaker. The diffuse sound field calibration activates all loudspeakers.

In the final step, the calibration can be revised and saved to the layout file.

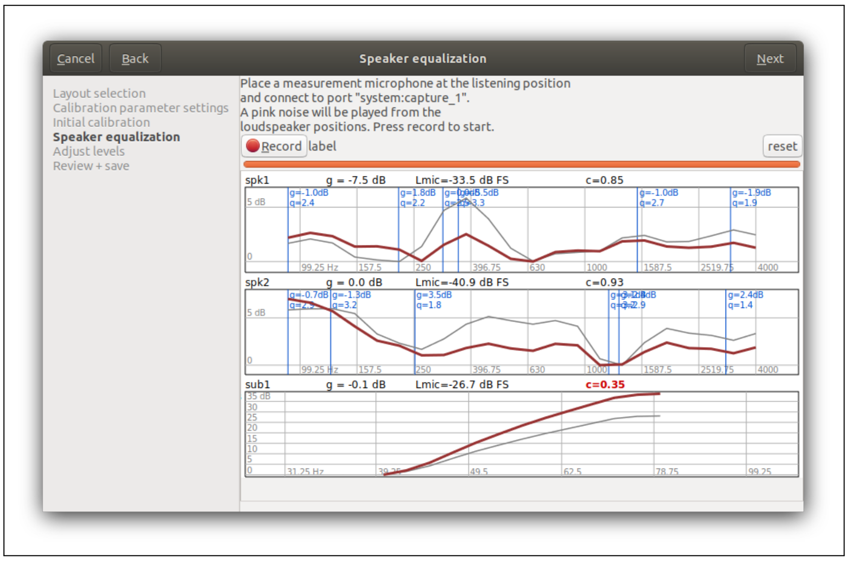


Figure 17: Equalization of loudspeakers, with spectral correction. The center frequencies of the equalizer stages are indicated with blue markers. The thin gray line indicates the frequency response without spectral correction, the thick red line with spectral correction.

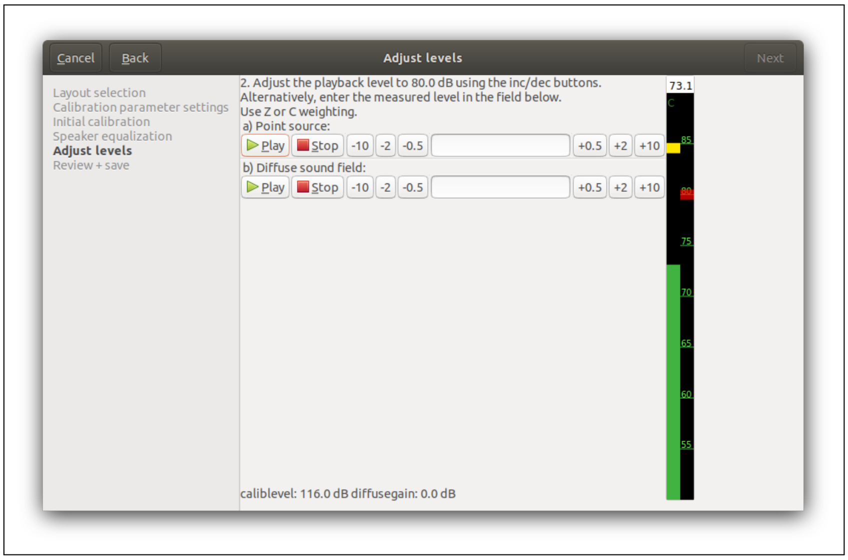


Figure 18: Adjustment of the point source level and the diffuse sound field reproduction gain.

11 Interfacing from MATLAB and GNU/Octave

For the interface between TASCAR and MATLAB or GNU/Octave a set of scripts are provided in `/usr/share/tascar/matlab`. There are the following scripts available:

11.1 `tascar_ctl`

This function can be used for some basic actions. For details see function help in MATLAB/GNU Octave.

- Loading a scene, which already exists:
  `h = tascar_ctl('load','filename.tsc')`
- Controlling the transport:
  `tascar_ctl( 'transport', h, 'play' )`
  `tascar_ctl( 'transport', h, 'locate', 15 )`
- Closing a scene:
  `tascar_ctl('kill',h )`
- Creating a basic scene:
  `tascar_ctl('createscene','filename','my_scene.tsc')`
- Create an audio player:
  `h = tascar_ctl('audioplayercreate',{sig1,sig2});`
  `tascar_ctl('audioplayerselect',1);`
  `tascar_ctl('audioplayerselect',2);`
  `tascar_ctl('kill',h )`

An example of the usage of this (and other MATLAB/GNU Octave functions) can be found in `example_controlTASCAR.m`.

11.2 `generate_scene`

In TASCAR, virtual acoustic environments are defined in an xml file format. User can write such a file on his own or create it using MATLAB/GNU Octave function `generate_scene`. This function can generate a simple scene with one loudspeaker-based receiver, N sources distributed on a circle around the receiver and K virtual loudspeakers, also equally distributed on a circle around the receiver. There are a couple of parameters which can be specified by a user - the file name, the number of sources and loudspeakers, radius of the circle, receiver type, as well as the length of a delay line.

11.3 `tascar_jackio`

When we already have an XML file with a virtual environment, we may want to start performing some measurements using MATLAB/GNU Octave. The function `tascar_jackio` is used to play and record sound via jack (jack audio connection kit). It means that we are playing the sound using ports responsible for the sources in the virtual scene and recording the sound using ports responsible for the receiver in the scene. Usually, it will be a test signal (for example white noise) played back and recorded (for example to compute the impulse response of the virtual environment).

At the input of the function, we have to specify input and output jack ports which will be used for playing and recording the signal:

```
[y,fs,bufsize] = tascar_jackio( x, ...
                                'output', csOutputPorts, ...
                                'input', csInputPorts );
```

Here `output` is a list of port connections to which the sound vector `x` will be sent (typically corresponding to the virtual sound sources in a TASCAR scene, e.g., `{'render.scene:src.0'}`). The number of channels in `x` must match the number of output port connections. Accordingly, `input` is a list of input port connections from which the content of `y` will be read (e.g., the receiver outputs of a scene, or audio hardware inputs). If you specify writable clients as `input`, e.g., the sound card outputs, then `tascar_jackio` will connect to all readable input ports which are connected to the specified writable port. This can be used to record the signal sent to the loudspeakers, even if it is a mixture from several scenes. The number of channels in `y` will be the number of elements in the input variable.

For more information, type `help tascar_jackio` (usage information), or `tascar_jackio help` (full list of parameters).

11.4 tascar_ir_measure

Measure an impulse response using a sine sweep method after Farina (2000).

11.5 send_osc

The properties of objects placed in the scene or for example the transport state can be manipulated from outside TASCAR using OSC-messages. The parameters (properties) of an object which can be changed by sending an OSC message to TASCAR are called "OSC variables". For example, in MATLAB/GNU Octave it can be done by using the provided function `send_osc` (for UDP transport) or `send_osc_tcp` (for TCP transport).

The functions `send_osc` and `send_osc_tcp` are functions by which we can control a TASCAR session and objects in a TASCAR scene. The default OSC port is 9877, listening on all network devices. For more control, we can use the attributes `srv_addr` and `srv_port` to the element **`<session/>`**, e.g.,

```
<session srv_addr="" srv_port="9999">
```

To check the list of variables and the OSC server port in a TASCAR session, select the sub-menu “OSC variables” in the menu “view” from the main menu bar. Each OSC variable has its path and type. On the right side of each variable path, you can see also its type in brackets. `(f)` means a floating point number, `(fff)` means a vector with 3 floating point numbers (it can for example correspond to 3 coordinates for the position), `(i)` means integer etc.

Function `send_osc` requires specifying the destination host (e.g., ’localhost’) and port number, the path, and the variable values, e.g.,

```
send_osc( 'localhost', 9877, '/scene_name/object_name/pos', ...
          pos_x, pos_y, pos_z, euler_z, euler_y, euler_x );
```

where `pos_x`, `pos_y`, `pos_z` are the cartesian coordinates in meters and `euler_z`, `euler_y`, `euler_x` are the rotations around x, y, and z-axis in degrees. They are always relative to the position and orientation specified in the scene definition file.

Position can also be specified using only 3 numbers:

```
send_osc('localhost',9877,'/scene_name/object_name/pos',...
         pos_x,pos_y,pos_z)!
```

Orientation of the object can be also changed using:

```
send_osc('localhost',9877,'/scene_name/object_name/zyxeuler',...
         euler_z euler_y,euler_x)
```

To mute or solo one object, we use:

```
send_osc('localhost',9877,'/scene_name/object_name/solo',1)
send_osc('localhost',9877,'/scene_name/object_name/mute',1)
```

Sending OSC messages can also be used for starting, stoping or placing a scene at the arbitrary point in time:

Stop/start a scene:

```
send_osc('localhost',9877,'/transport/stop')
send_osc('loclhost',9877,'/transport/start')
```

Go back to beginning / 4th second:

```
send_osc('localhost',9877,'/transport/locate',0)
send_osc('localhost',9877,'/transport/locate',4)
```

12 Command line interfaces

All command line applications of TASCAR start with the prefix `tascar_`. To get a list of valid command line options, use the flag `-h` or `--help`.

12.1 tascar_cli

```
Usage:

tascar_cli [options] configfile

Options:

  -h
  --help

  -j #
  --jackname=#

  -o #
  --output=#
Output sound file name.

  -r #
  --range=#

  -l
  --licenses
Show licenses

  -v
  --validate

  -a
  --variables
Show variables


(version: 0.234.2.0-58f0188)
```

12.2 tascar_genrandlsl

```
Usage:

tascar_genrandlsl [options]

Options:

  -h
  --help
```

```
  -c #
  --channels=#

  -n #
  --name=#

  -r #
  --rate=#


(version: 0.234.2.0-58f0188)
```

12.3 tascar_getcalibfor

```
Usage:

tascar_getcalibfor sessionfile [options]

Get "calibfor" values of speaker-based receivers in the session file.

Options:

  -h
  --help
```

12.4 tascar_gpx2csv

```
Usage:

tascar_gpx2csv [options] gpxfile

Options:

  -h
  --help

  -o #
  --lon=#

  -a #
  --lat=#

  -n
  --znull

  -r #
  --resample=#

  -s #
  --smooth=#
```

```
  -v
  --velocity


(version: 0.234.2.0-58f0188)
```

12.5 tascar_hdspmixer

Simple interface to RME HDSP9652 audio interface.

```
Usage:

tascar_hdspmixer [options]

Simple control of HDSP 9652 matrix mixer.



Options:

  -h
  --help

  -i
  --input

  -a
  --alsa

  -s
  --stereo

  -d #
  --device=#

  -c #
  --channels=#


(version: 0.234.2.0-58f0188)
```

12.6 tascar_jackio

Play and record wav files via jack.

```
Usage:

tascar_jackio [options] input.wav [ ports [...]]
```

```
Options:

  -f
  --freewheeling

  -o #
  --output-file=#

  -n #
  --jack-name=#

  -c
  --autoconnect

  -u
  --unlink

  -h
  --help

  -s #
  --start=#

  -w
  --wait

  -d #
  --duration=#

  -t #
  --statistics=#

  -v
  --verbose


(version: 0.234.2.0-58f0188)
```

12.7 tascar_levelmeter

```
Usage:

tascar_levelmeter [options]

Options:

  -h
  --help

  -j #
  --jackname=#

  -o #
  --osctarget=#
```

```
(version: 0.234.2.0-58f0188)
```

12.8 tascar_listsrc

```
Usage:

tascar_listsrc sessionfile [options]

List external source files (sound files, trajectories, reflectors etc).

Options:

  -h
  --help

  -m
  --missing
```

12.9 tascar_lsjackp

```
Usage:

tascar_lsjackp [options]

Options:

  -h
  --help

  -j #
  --jackname=#

  -o
  --output

  -i
  --input

  -p
  --physical

  -s
  --soft


(version: 0.234.2.0-58f0188)
```

12.10 tascar_lslsl

This command line tool outputs a list of available Lab Streaming Layer (LSL) streams.

```
Usage:

tascar_lslsl [options]

List LSL streams.

Options:

  -h
  --help


(version: 0.234.2.0-58f0188)
```

12.11 tascar_osc2file

```
Usage:

osc2file [options]

To add streams, specify it as '<path>:<format>', e.g., '/path:ff'.
<format> can be 'i' (integer), 'f' (32 bit float) or 's' (string).

Options:

  -h
  --help

  -a #
  --add=#

  -o #
  --output=#

  -p #
  --port=#


(version: 0.234.2.0-58f0188)
```

12.12 tascar_osc2lsl

```
Usage:

osc2lsl [options]
```

```
To add streams manually, specify it as '<path>:<format>', e.g., '/path:ff'.
<format> can be 'i' (integer), 'f' (32 bit float) or 's' (string).

Options:

  -h
  --help

  -a #
  --add=#

  -n
  --noauto

  -p #
  --port=#

  -t
  --timestamp


(version: 0.234.2.0-58f0188)
```

12.13 tascar_osc_jack_transport

```
Usage:

tascar_osc_jack_transport [options]

Options:

  -h
  --help

  -j #
  --jackname=#

  -a #
  --srvaddr=#

  -p #
  --srvport=#

  -l #
  --looptime=#
```

12.14 tascar_pdf

```
Usage:
```

```
tascar_pdf -c sessionfile [options]

Options:

  -o #
  --output=#

  -h
  --help

  -t #
  --time=#

  -a
  --acousticmodel

  -0 #
  --ismmin=#

  -1 #
  --ismmax=#
```

12.15 tascar_renderfile

This command line tool can be used for rendering the image source model of a single scene in a TASCAR session with audio input from a sound file and saving the rendered signal to a sound file. Common usage example:

```
  tascar_renderfile -i input_file.wav -o output_file.wav tascar_scene.tsc
```

The size of the input file `input_file.wav` (number of audio channels) has to correspond with the number of sources in the scene. The size of the file `output_file.wav`, which will be created after calling this tool, will correspond to the number of output channels of the receiver used in the scene. In case of multi-channel output (e.g., speaker based receiver types), the order follows the order of the channel definition in the TASCAR files. This may differ from the order of jack ports, because some jack front ends sort ports alphabetically.

```
Usage:

tascar_renderfile [options] sessionfile

Render a TASCAR session into a sound file.



Options:

  -h
  --help

  -i #
```

```
  --inputfile=#

  -o #
  --outputfile=#

  -s #
  --scene=#
Scene name (or empty to use first scene in session file).

  -m #
  --channelmap=#
List of output channels (zero-base), or empty to use all.
Example: -m 0-5,8,12

  -t #
  --starttime=#

  -r #
  --srate=#
Sample rate in Hz. If input file is provided, then its sample rate is used
    instead

  -u #
  --durartion=#

  -f #
  --fragsize=#

  -c
  --static

  -1 #
  --ismmin=#
Minimum order of image source model.

  -2 #
  --ismmax=#
Maximum order of image source model, or -1 to use value from scene definition.

  -v
  --verbose
Increase verbosity.


(version: 0.234.2.0-58f0188)
```

12.16 tascar_renderir

This command line tool is used to render the impulse response of a TASCAR scene. A typical usage example might be

```
  tascar_renderir -o output_file.wav -f 44100 -1 2 tascar_scene.tsc
```

Here the impulse response is saved in `output_file.wav` with a sampling rate of 44100 Hz and up to 2nd order image source model.

```
Usage:

tascar_renderir [options] sessionfile

Render an impulse response of a TASCAR session.

Options:

  -h
  --help

  -s #
  --scene=#
Scene name, or empty to select first scene.

  -o #
  --outputfile=#
Output sound file.

  -t #
  --starttime=#
Start time in session corresponding to first output sample.

  -l #
  --irlength=#

  -f #
  --srate=#
Sampling rate in Hz. If input file is provided, the sampling rate of the input
    file is used.

  -0 #
  --ismmin=#
Minimum order of image source model.

  -1 #
  --ismmax=#
Maximum order of image source model, or -1 to use value from scene definition.

  -i #
  --inchannel=#
Input channel number. This defines from which sound vertex the IR is measured.
    Sound vertices are numbered in the order of their appearance in the session
    file, starting with zero.

  -v
  --verbose


(version: 0.234.2.0-58f0188)
```

12.17 tascar_sampler

```
Usage:

tascar_sampler [options] soundfont [ jackname ]

Options:

  -a #
  --srvaddr=#

  -p #
  --srvport=#

  -h
  --help


A soundfont is a list of sound file names, one file per line.
```

12.18 tascar_sceneskeleton

```
Usage:

tascar_sceneskeleton [options]

Show a generic TASCAR scene skeleton.

Options:

  -h
  --help


(version: 0.234.2.0-58f0188)
```

12.19 tascar_showlicenses

```
Usage:

tascar_showlicenses -c sessionfile [options]

Options:

  -h
  --help
```

12.20 tascar_spk2obj

```
Usage:

tascar_spk2obj [options] <layout file>

Options:

  -o #
  --output=#

  -h
  --help


(version: 0.234.2.0-58f0188)
```

12.21 tascar_validatetsc

```
Usage:

tascar_validatetsc -c sessionfile [options]

Options:

  -h
  --help

  -g
  --gendoc

  -l
  --latex

  -v
  --verbose
```

12.22 tascar_version

```
Usage:

tascar_version [options]

Show version information.

Options:

  -h
  --help
```

```
(version: 0.234.2.0-58f0188)
```

List of symbols and definitions

symbol	dimension	variable
t	scalar	sampled time
N	scalar	number of receiver output channels
K	scalar	number of point sources in a scene
L	scalar	number of diffuse sound fields in a scene
$\mathbf{p}_{src}$	1×3	source position
$\mathbf{p}_{rec}$	1×3	receiver position
$\mathbf{p}_{spk}$	1×3	loudspeaker position in receiver coordinate system
$(\varrho, \varphi, \theta)$	1×3	Spherical coordinates, distance ϱ, azimuth φ, elevation θ
$\mathbf{D}$, d	$N \times 4$	first order Ambisonics decoder matrix
$\mathbf{w}$, w_n	$1 \times N$	driving weights for point source at relative position $\mathbf{p}_{rel}$
$\mathbf{z}(t)$, $z_n(t)$	$1 \times N$	receiver output signal
$y_k(t)$	scalar	acoustic model output signal for k-th point source
$\mathbf{f}_l(t)$	1×4	first order Ambisonics signal for l-th diffuse sound field
$\mathbf{O}_{rec}$	3×3	receiver orientation matrix
$\mathbf{p}_{rel}$	1×3	relative source direction $\mathbf{p}_{rel} = \mathbf{O}_{rec}^{-1}(\mathbf{p}_{src} - \mathbf{p}_{rec})^T$
$r = \|\|\mathbf{p}_{rel}\|\|$	scalar	distance between source and receiver

The receiver orientation is defined by

$$\mathbf{O}_{rec} = \mathbf{O}_x \left(\mathbf{O}_y \mathbf{O}_z\right) \tag{21}$$

$$\mathbf{O}_x = \begin{pmatrix} 1 & 0 & 0 \\ 0 & \cos(\Omega_x) & -\sin(\Omega_x) \\ 0 & \sin(\Omega_x) & \cos(\Omega_x) \end{pmatrix} \tag{22}$$

$$\mathbf{O}_y = \begin{pmatrix} \cos(\Omega_y) & 0 & -\sin(\Omega_y) \\ 0 & 1 & 0 \\ \sin(\Omega_y) & 0 & \cos(\Omega_y) \end{pmatrix} \tag{23}$$

$$\mathbf{O}_z = \begin{pmatrix} \cos(\Omega_y) & -\sin(\Omega_y & 0 \\ \sin(\Omega_y) & \cos(\Omega_y) & 0 \\ 0 & 0 & 1 \end{pmatrix} \tag{24}$$

$$\hat{\mathbf{O}}_{rec} = \begin{pmatrix} 1 & 0 \\ 0 & \mathbf{O}_{rec}^{-1} \end{pmatrix} \tag{25}$$

13 Appendix

References

C Phillip Brown and Richard O Duda. A structural model for binaural sound synthesis. *IEEE Transition on Speech and Audio Processing*, 6(5):476–488, 1998. doi: 10.1109/89.709673. URL https://doi.org/10.1109/89.709673.

Olliver Buttler. Optimierung und erweiterung einer effizienten methode zur synthese binauraler raumimpulsantworten. MSc thesis, 2018.

Jérôme Daniel. *Représentation de champs acoustiques, application à la transmission et à la reproduction de scènes sonores complexes dans un contexte multimédia*. PhD thesis, Université Pierre et Marie Curie (Paris VI), Paris, 2001.

Florian Denk and Birger Kollmeier. The hearpiece database of individual transfer functions of an openly available in-the-ear earpiece for hearing device research. https://zenodo.org/record/3900114, 2020.

Richard O Duda. Modeling head related transfer functions. In *Signals, Systems and Computers, 1993. Conference Record of The Twenty-Seventh Asilomar Conference on*, pages 996–1000. IEEE, 1993.

Stephan D. Ewert. RAZR. http://www.razrengine.com, 2018.

Angelo Farina. Simultaneous measurement of impulse response and distortion with a swept-sine technique. In *Audio Engineering Society Convention 108*, 2 2000.

Jan Goyvaerts. Regular expressions. https://www.regular-expressions.info/, 2019.

Giso Grimm and Tobias Herzke. A framework for dynamic spatial acoustic scene generation with Ambisonics in low delay realtime. In Frank Neumann, editor, *Proceedings of the Linux Audio Conference*, Stanford, CA, USA, 2012. Center for Computer Research in Music and Acoustics, Stanford University.

Giso Grimm, Joanna Luberadzka, Tobias Herzke, and Volker Hohmann. Toolbox for acoustic scene creation and rendering (tascar): Render methods and research applications. In Frank Neumann, editor, *Proceedings of the Linux Audio Conference*, Mainz, Germany, 2015. Johannes-Gutenberg Universität Mainz.

Giso Grimm, Joanna Luberadzka, and Volker Hohmann. Virtual acoustic environments for comprehensive evaluation of model based hearing devices. *International Journal of Audiology*, 2016.

Giso Grimm, Joanna Luberadzka, and Volker Hohmann. A toolbox for rendering virtual acoustic environments in the context of audiology. *Acta Acustica united with Acustica*, 105 (3):566–578, 2019. doi: 10.3813/AAA.919337. URL https://doi.org/10.3813/AAA.919337.

Aaron Heller and Em Benjamin. The Ambisonic Decoder Toolbox: Extensions for Partial-Coverage Loudspeaker Arrays. *Linux Audio Conference*, pages 1–9, 2014. URL http://lac.linuxaudio.org/2014/papers/17.pdf.

Aaron J Heller, Eric M Benjamin, and Richard Lee. A Toolkit for the Design of Ambisonic Decoders. *Linux Audio Conference*, page 12, 2012. URL http://www.academia.edu/download/30883409/18.pdf.

Dieter Leckschat, Christian Epe, Malte Kob, Bernhard Seeber, Sascha Spors, Stefan Weinzierl, and Franz Zotter. *DEGA-Memorandum VA 1201 Guidelines for implementing and documenting audio productions for scientific applications in acoustics*, January 2020. URL https://doi.org/10.5281/zenodo.3597238.

Gerard Llorach, Alun Evans, Josep Blat, Giso Grimm, and Volker Hohmann. Web-based live speech-driven lip-sync. In *VS-Games*, Barcelona, Spain, 2016.

Trond Lossius, Pascal Baltazar, and Théo de la Hogue. Dbap–distance-based amplitude panning. In *ICMC*, 2009.

Ville Pulkki. Virtual sound source positioning using vector base amplitude panning. *J. Audio Eng. Soc*, 45(6):456–466, 1997.

Davide Rocchesso and Julius O Smith. Circulant and elliptic feedback delay networks for artificial reverberation. *IEEE Transactions on Speech and Audio Processing*, 5(1):51–63, 1997.

Manfred R Schroeder. Natural Sounding Artificial Reverberation. *Journal of the Audio Engineering Society Audio Eng. Soc*, 10(3):219–223, 1962.

Fenja Schwark. Data-driven optimization of parameterized head related transfer functions for the implementation in a real-time virtual acoustic rendering framework. BSc thesis, 2020.

Torben Wendt, Steven van de Par, and Stephan Ewert. A Computationally-Efficient and Perceptually-Plausible Algorithm for Binaural Room Impulse Response Simulation. *Journal of the Audio Engineering Society*, 62(11):748–766, dec 2014. ISSN 15494950. doi: 10.17743/jaes.2014.0042. URL http://www.aes.org/e-lib/browse.cfm?elib=17550.

Franz Zotter, Matthias Frank, Matthias Kronlachner, and Jung-Woo Choi. Efficient phantom source widening and diffuseness in ambisonics. In *Proceedings of the EAA Joint Symposium on Auralization and Ambisonics*, 2014.

Examples

Type	N	panning	diffuse decoding
`omni`	1	$w_n = 1$	$d_{1,1} = 1$
`vmic`	1	$w_n = 1 + a(\tilde{p}_{rel,x} - 1)$	$d_{1,1} = \sqrt{2}(1-a),\ d_{1,2} = a$
`cardioid`	1	$w_n = \frac{1}{2}\left(\cos(\varphi) + 1\right)$	$d_{1,1} = 1$
`ortf`	2	ORTF microphone	$\max r_E$
`amb_3h0v`	7	$w_n = \begin{cases} \sqrt{2} & n = 1 \\ \cos(\frac{n}{2}\varphi) & n\ \text{even} \\ \sin(\frac{n-1}{2}\varphi) & n\ \text{odd} \end{cases}$	$d_{n,n} = 1,\ n = \{1, 2, 3\}$
`amb_3h3v`	16	$w = \begin{pmatrix} w_w \\ w_y \\ w_x \\ w_z \\ w_v \\ w_t \\ w_r \\ w_s \\ w_u \\ w_q \\ w_o \\ w_m \\ w_k \\ w_l \\ w_n \\ w_p \end{pmatrix} = \begin{pmatrix} \sqrt{2} \\ \cos(\theta)\sin(\varphi) \\ \cos(\theta)\cos(\varphi) \\ \sin(\theta) \\ 2w_x w_y \\ 2w_z w_y \\ \frac{1}{2}(3w_z^2 - 1) \\ 2w_z w_x \\ w_x^2 - w_y^2 \\ (3w_x^2 - w_y^2)w_y \\ 2.598076 w_z w_v \\ 0.726184(5w_z^2 - 1)w_y \\ \frac{1}{2}w_z(5w_z^2 - 3) \\ 0.726184(5w_z^2 - 1)w_x \\ 2.598076 w_z w_u \\ (w_x^2 - 3w_y^2)w_x \end{pmatrix}$	$d_{n,n} = 1,\ n = \{1, 2, 3, 4\}$
`neukom_basic`	user def.	$w_n = 1 + 2\sum_{l=1}^{order}\cos(l\varphi_n)$	
`neukom_inphase`	user def.	$w_n = \cos(0.5\varphi_n)^{order}$	
`hoa2d`	user def.	see Daniel (2001) for details.	
`nsp`	user def.	$w_{\arg\min\left\{\left\Vert \frac{\mathbf{p}_{rel}}{\Vert\mathbf{p}_{rel}\Vert} - \frac{\mathbf{p}_{spk}}{\Vert\mathbf{p}_{spk}\Vert}\right\Vert\right\}} = 1$	
`hann`	user def.	$w_n = \left(\frac{1}{2} + \frac{1}{2}\cos\left(\min\left\{\left\vert\frac{N}{2}\varphi_n\right\vert, \pi\right\}\right)\right)^\gamma$	
`vbap`, `vbap3d`	user def.	see Pulkki (1997) for details.	

Table 195: Specification of receiver types. $d_{n,wxyz} = 0$ except for the given entries. $(\varrho, \varphi, \theta)$ is the source position in spherical coodinates in the receiver coordinate system. φ_n is the azimuthal angular distance between loudspeaker n and the sound source.

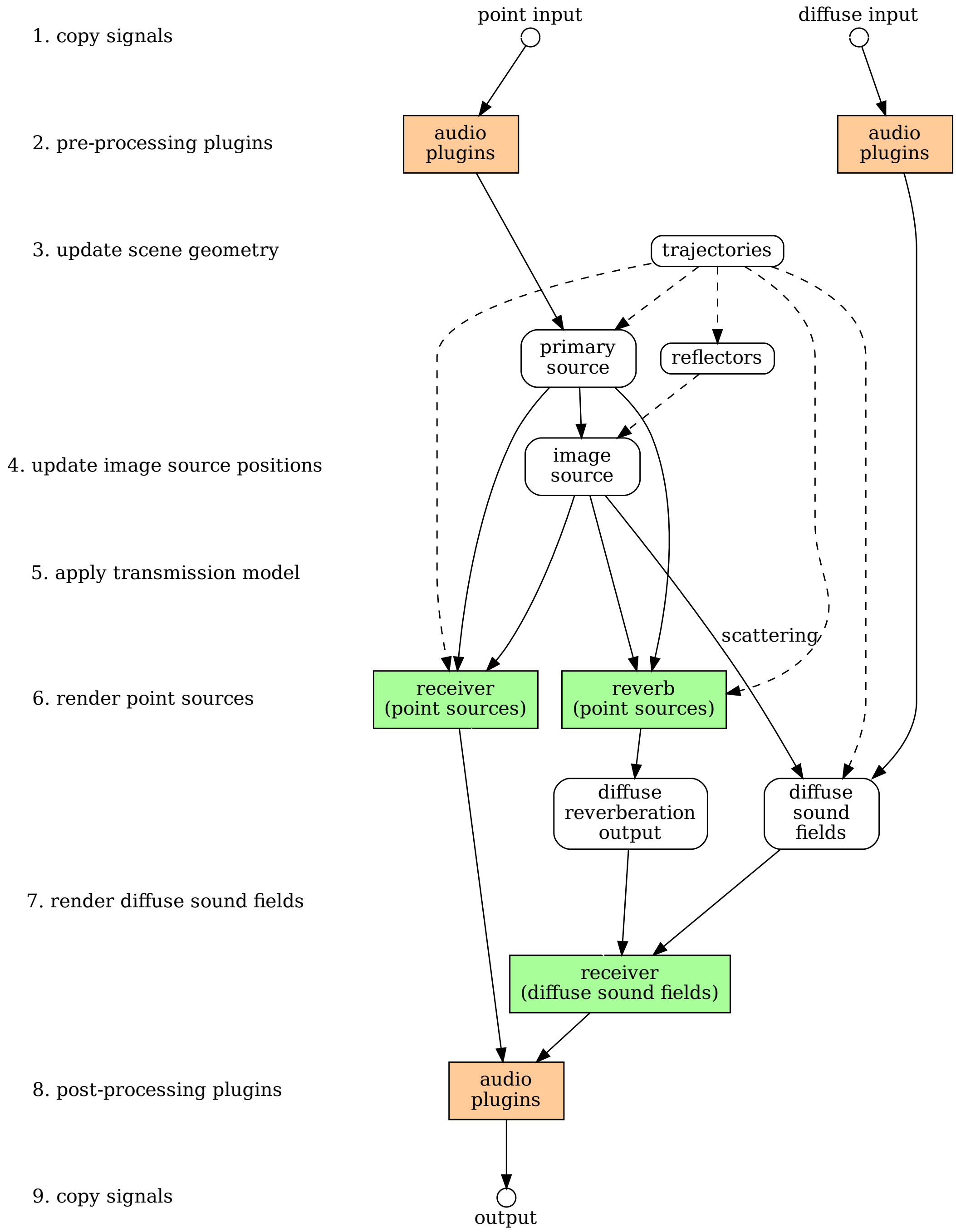


Figure 19: Signal flow in the acoustic model.

Index

Preamble

The licenses for most software are designed to take away your freedom to share and change it. By contrast, the GNU General Public License is intended to guarantee your freedom to share and change free software–to make sure the software is free for all its users. This General Public License applies to most of the Free Software Foundation's software and to any other program whose authors commit to using it. (Some other Free Software Foundation software is covered by the GNU Lesser General Public License instead.) You can apply it to your programs, too.

When we speak of free software, we are referring to freedom, not price. Our General Public Licenses are designed to make sure that you have the freedom to distribute copies of free software (and charge for this service if you wish), that you receive source code or can get it if you want it, that you can change the software or use pieces of it in new free programs; and that you know you can do these things.

To protect your rights, we need to make restrictions that forbid anyone to deny you these rights or to ask you to surrender the rights. These restrictions translate to certain responsibilities for you if you distribute copies of the software, or if you modify it.

For example, if you distribute copies of such a program, whether gratis or for a fee, you must give the recipients all the rights that you have. You must make sure that they, too, receive or can get the source code. And you must show them these terms so they know their rights.

We protect your rights with two steps: (1) copyright the software, and (2) offer you this license which gives you legal permission to copy, distribute and/or modify the software.

Also, for each author's protection and ours, we want to make certain that everyone understands that there is no warranty for this free software. If the software is modified by someone else and passed on, we want its recipients to know that what they have is not the original, so that any problems introduced by others will not reflect on the original authors' reputations.

Finally, any free program is threatened constantly by software patents. We wish to avoid the danger that redistributors of a free program will individually obtain patent licenses, in effect making the program proprietary. To prevent this, we have made it clear that any patent must be licensed for everyone's free use or not licensed at all.

The precise terms and conditions for copying, distribution and modification follow. TERMS AND CONDITIONS FOR COPYING, DISTRIBUTION AND MODIFICATION

0. This License applies to any program or other work which contains a notice placed by the copyright holder saying it may be distributed under the terms of this General Public License. The "Program", below, refers to any such program or work, and a "work based on the Program" means either the Program or any derivative work under copyright law: that is to say,

a work containing the Program or a portion of it, either verbatim or with modifications and/or translated into another language. (Hereinafter, translation is included without limitation in the term "modification".) Each licensee is addressed as "you".

Activities other than copying, distribution and modification are not covered by this License; they are outside its scope. The act of running the Program is not restricted, and the output from the Program is covered only if its contents constitute a work based on the Program (independent of having been made by running the Program). Whether that is true depends on what the Program does.

1. You may copy and distribute verbatim copies of the Program's source code as you receive it, in any medium, provided that you conspicuously and appropriately publish on each copy an appropriate copyright notice and disclaimer of warranty; keep intact all the notices that refer to this License and to the absence of any warranty; and give any other recipients of the Program a copy of this License along with the Program.

You may charge a fee for the physical act of transferring a copy, and you may at your option offer warranty protection in exchange for a fee.

2. You may modify your copy or copies of the Program or any portion of it, thus forming a work based on the Program, and copy and distribute such modifications or work under the terms of Section 1 above, provided that you also meet all of these conditions:

a) You must cause the modified files to carry prominent notices stating that you changed the files and the date of any change. b) You must cause any work that you distribute or publish, that in whole or in part contains or is derived from the Program or any part thereof, to be licensed as a whole at no charge to all third parties under the terms of this License. c) If the modified program normally reads commands interactively when run, you must cause it, when started running for such interactive use in the most ordinary way, to print or display an announcement including an appropriate copyright notice and a notice that there is no warranty (or else, saying that you provide a warranty) and that users may redistribute the program under these conditions, and telling the user how to view a copy of this License. (Exception: if the Program itself is interactive but does not normally print such an announcement, your work based on the Program is not required to print an announcement.)

These requirements apply to the modified work as a whole. If identifiable sections of that work are not derived from the Program, and can be reasonably considered independent and separate works in themselves, then this License, and its terms, do not apply to those sections when you distribute them as separate works. But when you distribute the same sections as part of a whole which is a work based on the Program, the distribution of the whole must be on the terms of this License, whose permissions for other licensees extend to the entire whole, and thus to each and every part regardless of who wrote it.

Thus, it is not the intent of this section to claim rights or contest your rights to work written entirely by you; rather, the intent is to exercise the right to control the distribution of derivative or collective works based on the Program.

In addition, mere aggregation of another work not based on the Program with the Program (or with a work based on the Program) on a volume of a storage or distribution medium does not bring the other work under the scope of this License.

3. You may copy and distribute the Program (or a work based on it, under Section 2) in object

code or executable form under the terms of Sections 1 and 2 above provided that you also do one of the following:

a) Accompany it with the complete corresponding machine-readable source code, which must be distributed under the terms of Sections 1 and 2 above on a medium customarily used for software interchange; or, b) Accompany it with a written offer, valid for at least three years, to give any third party, for a charge no more than your cost of physically performing source distribution, a complete machine-readable copy of the corresponding source code, to be distributed under the terms of Sections 1 and 2 above on a medium customarily used for software interchange; or, c) Accompany it with the information you received as to the offer to distribute corresponding source code. (This alternative is allowed only for noncommercial distribution and only if you received the program in object code or executable form with such an offer, in accord with Subsection b above.)

The source code for a work means the preferred form of the work for making modifications to it. For an executable work, complete source code means all the source code for all modules it contains, plus any associated interface definition files, plus the scripts used to control compilation and installation of the executable. However, as a special exception, the source code distributed need not include anything that is normally distributed (in either source or binary form) with the major components (compiler, kernel, and so on) of the operating system on which the executable runs, unless that component itself accompanies the executable.

If distribution of executable or object code is made by offering access to copy from a designated place, then offering equivalent access to copy the source code from the same place counts as distribution of the source code, even though third parties are not compelled to copy the source along with the object code.

4. You may not copy, modify, sublicense, or distribute the Program except as expressly provided under this License. Any attempt otherwise to copy, modify, sublicense or distribute the Program is void, and will automatically terminate your rights under this License. However, parties who have received copies, or rights, from you under this License will not have their licenses terminated so long as such parties remain in full compliance.

5. You are not required to accept this License, since you have not signed it. However, nothing else grants you permission to modify or distribute the Program or its derivative works. These actions are prohibited by law if you do not accept this License. Therefore, by modifying or distributing the Program (or any work based on the Program), you indicate your acceptance of this License to do so, and all its terms and conditions for copying, distributing or modifying the Program or works based on it.

6. Each time you redistribute the Program (or any work based on the Program), the recipient automatically receives a license from the original licensor to copy, distribute or modify the Program subject to these terms and conditions. You may not impose any further restrictions on the recipients' exercise of the rights granted herein. You are not responsible for enforcing compliance by third parties to this License.

7. If, as a consequence of a court judgment or allegation of patent infringement or for any other reason (not limited to patent issues), conditions are imposed on you (whether by court order, agreement or otherwise) that contradict the conditions of this License, they do not excuse you from the conditions of this License. If you cannot distribute so as to satisfy simultaneously your obligations under this License and any other pertinent obligations, then

as a consequence you may not distribute the Program at all. For example, if a patent license would not permit royalty-free redistribution of the Program by all those who receive copies directly or indirectly through you, then the only way you could satisfy both it and this License would be to refrain entirely from distribution of the Program.

If any portion of this section is held invalid or unenforceable under any particular circumstance, the balance of the section is intended to apply and the section as a whole is intended to apply in other circumstances.

It is not the purpose of this section to induce you to infringe any patents or other property right claims or to contest validity of any such claims; this section has the sole purpose of protecting the integrity of the free software distribution system, which is implemented by public license practices. Many people have made generous contributions to the wide range of software distributed through that system in reliance on consistent application of that system; it is up to the author/donor to decide if he or she is willing to distribute software through any other system and a licensee cannot impose that choice.

This section is intended to make thoroughly clear what is believed to be a consequence of the rest of this License.

8. If the distribution and/or use of the Program is restricted in certain countries either by patents or by copyrighted interfaces, the original copyright holder who places the Program under this License may add an explicit geographical distribution limitation excluding those countries, so that distribution is permitted only in or among countries not thus excluded. In such case, this License incorporates the limitation as if written in the body of this License.

9. The Free Software Foundation may publish revised and/or new versions of the General Public License from time to time. Such new versions will be similar in spirit to the present version, but may differ in detail to address new problems or concerns.

Each version is given a distinguishing version number. If the Program specifies a version number of this License which applies to it and "any later version", you have the option of following the terms and conditions either of that version or of any later version published by the Free Software Foundation. If the Program does not specify a version number of this License, you may choose any version ever published by the Free Software Foundation.

10. If you wish to incorporate parts of the Program into other free programs whose distribution conditions are different, write to the author to ask for permission. For software which is copyrighted by the Free Software Foundation, write to the Free Software Foundation; we sometimes make exceptions for this. Our decision will be guided by the two goals of preserving the free status of all derivatives of our free software and of promoting the sharing and reuse of software generally.

NO WARRANTY

11. BECAUSE THE PROGRAM IS LICENSED FREE OF CHARGE, THERE IS NO WARRANTY FOR THE PROGRAM, TO THE EXTENT PERMITTED BY APPLICABLE LAW. EXCEPT WHEN OTHERWISE STATED IN WRITING THE COPYRIGHT HOLDERS AND/OR OTHER PARTIES PROVIDE THE PROGRAM "AS IS" WITHOUT WARRANTY OF ANY KIND, EITHER EXPRESSED OR IMPLIED, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE. THE ENTIRE RISK AS TO THE QUALITY AND PERFORMANCE OF THE PRO-

GRAM IS WITH YOU. SHOULD THE PROGRAM PROVE DEFECTIVE, YOU ASSUME THE COST OF ALL NECESSARY SERVICING, REPAIR OR CORRECTION.

12. IN NO EVENT UNLESS REQUIRED BY APPLICABLE LAW OR AGREED TO IN WRITING WILL ANY COPYRIGHT HOLDER, OR ANY OTHER PARTY WHO MAY MODIFY AND/OR REDISTRIBUTE THE PROGRAM AS PERMITTED ABOVE, BE LIABLE TO YOU FOR DAMAGES, INCLUDING ANY GENERAL, SPECIAL, INCIDENTAL OR CONSEQUENTIAL DAMAGES ARISING OUT OF THE USE OR INABILITY TO USE THE PROGRAM (INCLUDING BUT NOT LIMITED TO LOSS OF DATA OR DATA BEING RENDERED INACCURATE OR LOSSES SUSTAINED BY YOU OR THIRD PARTIES OR A FAILURE OF THE PROGRAM TO OPERATE WITH ANY OTHER PROGRAMS), EVEN IF SUCH HOLDER OR OTHER PARTY HAS BEEN ADVISED OF THE POSSIBILITY OF SUCH DAMAGES. END OF TERMS AND CONDITIONS How to Apply These Terms to Your New Programs

If you develop a new program, and you want it to be of the greatest possible use to the public, the best way to achieve this is to make it free software which everyone can redistribute and change under these terms.

To do so, attach the following notices to the program. It is safest to attach them to the start of each source file to most effectively convey the exclusion of warranty; and each file should have at least the "copyright" line and a pointer to where the full notice is found.

one line to give the program's name and an idea of what it does. Copyright (C) yyyy name of author

This program 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; either version 2 of the License, or (at your option) any later version.

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.

You should have received a copy of the GNU General Public License along with this program; if not, write to the Free Software Foundation, Inc., 51 Franklin Street, Fifth Floor, Boston, MA 02110-1301, USA.

Also add information on how to contact you by electronic and paper mail.

If the program is interactive, make it output a short notice like this when it starts in an interactive mode:

Gnomovision version 69, Copyright (C) year name of author Gnomovision comes with ABSOLUTELY NO WARRANTY; for details type ‘show w’. This is free software, and you are welcome to redistribute it under certain conditions; type ‘show c’ for details.

The hypothetical commands ‘show w’ and ‘show c’ should show the appropriate parts of the General Public License. Of course, the commands you use may be called something other than ‘show w’ and ‘show c’; they could even be mouse-clicks or menu items–whatever suits your program.

You should also get your employer (if you work as a programmer) or your school, if any, to sign a "copyright disclaimer" for the program, if necessary. Here is a sample; alter the names:

Yoyodyne, Inc., hereby disclaims all copyright interest in the program ‘Gnomovision’ (which makes passes at compilers) written by James Hacker.

signature of Ty Coon, 1 April 1989 Ty Coon, President of Vice

This General Public License does not permit incorporating your program into proprietary programs. If your program is a subroutine library, you may consider it more useful to permit linking proprietary applications with the library. If this is what you want to do, use the GNU Lesser General Public License instead of this License.