SpatDIF specification V 0.3

Draft Version: July 12, 2012

This document is the specification for the SpatDIF terminology, structure and the namespace. It does not define specific implementation details.

Notes for this Draft Version

This document is an unapproved draft of a proposed SpatDIF format. As such, this document is subject to change. USE AT YOUR OWN RISK! Because this is an unapproved draft, this document must not be utilized for any conformance/compliance purposes. Permission is hereby granted for participants to reproduce this document for purposes of standardization consideration.

Publication Plan

This final version of this document will be published on the SpatDIF web site.
This document should be cited as:

SpatDIF specification Version 0.3, draft version - rev. 271

Participants

This specification is primarily developed and written by:

Nils Peters	Jan Schacher	Trond Lossius
ICSI, CNMAT	ICST	BEK
UC Berkeley	Zurich University of the Arts	Bergen Center for Electronic Arts
Berkeley, USA	Zürich, Switzerland	Bergen, Norway

with additional contributions by:

Jean Bresson	Marlon Schuhmacher	Gary Kendall	Scott Wilson
Giorgio Zoia	Frank Melchior	Matthias Geier	

Contents

1 What is SpatDIF

SpatDIF, the Spatial Sound Description Interchange Format, is a collaborative effort[1] that aims to create a format (semantic and syntactic) as well as best-practice implementations for storing and transmitting spatial audio-scene descriptions.

The goal of SpatDIF is to simplify and enhance the methods of working with spatial audio content in the context of authoring, storage of pieces and their distribution, as well as performance and study of spatial music. SpatDIF strives to be human-readable i.e., easily understood and unambiguous, platform- and implementation-independent, extendable, and free of license restrictions. Typical users include composers, sound installation artists, sound engineers, acousticians, virtual reality researchers, musicologists and many more[2].

One of the guiding principles for SpatDIF is that authoring and rendering of spatial audio might occur at separate times and places, and be executed or rendered with tools whose capabilities cannot be known in advance. The goal was to formulate a concise semantic structure that is capable of carrying the necessary information, without being tied to a specific implementation, thought-model or technical method. SpatDIF is a syntax rather than a programming interface or file-format. SpatDIF may be represented in any of the structured mark-up languages or message systems that are in use now or in the future. Examples of streaming (OSC) and storing SpatDIF data (XML, YAML, SDIF) accompany this document.

SpatDIF describes only the aspects required for the storage and transmission of *spatial information*. A complete work typically contains additional dimensions outside the scope of SpatDIF. These are only addressed to the extent necessary for linking the elements to the descriptions of the spatial dimension.

[1] `http://redmine.spatdif.org/projects/spatdif/wiki/Meetings`

[2] For further explanation see `http://redmine.spatdif.org/projects/spatdif/wiki/User_scenarios`.

2 Terminology and Structure

This section gives a brief overview of the terminology which will be used throughout the SpatDIF specification and the general structure of the SpatDIF format. The subsequent sections will provide further details.

2.1 Terminology

A SpatDIF representation is the combination of a space and the actions that are unfolding within it. A scene consists of a number of SpatDIF **entities**. Entities are all objects that are affecting or interacting with the sound of that scene. Entities can be of different **kinds** e.g., sources or sinks. Each entity instance is assigned a **name**, so that it may be uniquely identified within the scene. The properties of entities are described and transmitted via SpatDIF **descriptors**. A complete SpatDIF statement consists of an **address** unambiguously identifying an **entity**, its **descriptor**, and its associated **value**. The values of descriptors may change over time. All **entities** and **descriptors** are defined within the SpatDIF **namespace**.

OSC messages for example, need to comply with the SpatDIF namespace in order to be valid SpatDIF statements. An OSC message such as `/src/1/pos 1.0 5.0 0.0` is considered invalid, since neither the kind `src` nor the descriptor `pos` are defined in the SpatDIF namespace. Figure 1 shows a valid SpatDIF statement in streaming OSC-style: the entity is of kind `source` and named `romeo`, the `position` descriptor is set by the vector `(1.0 5.0 0.0)`, which is its value.

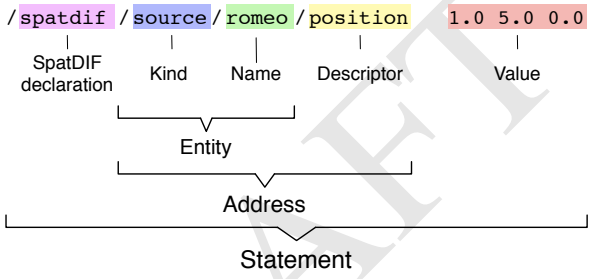


Figure 1: SpatDIF terminology

2.2 Meta Section and Time Section

A SpatDIF representation consists of two sections - a Meta Section and a Time Section (Figure 3). The Meta Section serves to configure and initialize the system, while the Time Section describes the temporal unfolding of a scene.

2.2.1 Meta Section

The Meta Section contains meta descriptions, and is located at the beginning of a SpatDIF representation. It contains information that is not executed at runtime; timed events are therefore excluded from this section. The Meta descriptions contain extension setup information (see Section 5), general annotation and documentation information, information about the organization of the subsequent time section, higher-level process and compositional information and technical setup information referring to the original authoring situation. The Meta Section can also be used to describe a static scene or the initial state of a dynamic scene. The Meta Section is mandatory for a SpatDIF representation.

2.2.2 Time Section

The Time Section holds information about entities and their descriptors as they unfold over time. Each statement is located at a specific point in time. If the scene to be described is static, no temporal data will be required. For this reason the time section is optional.

2.3 Core and Extensions

SpatDIF contains of a lightweight set of core descriptors and various extensions.

2.3.1 The SpatDIF Core

The fundamental core descriptors by themselves offer a compact set of information necessary to describe simple works in a light-weight format. All SpatDIF-compliant renderers are required to be able to interpret these core descriptors. Core descriptors are specified in Section 4.

2.3.2 Extensions

The core descriptions can be augmented via *extensions*. Extensions (Section 5) enable a more detailed description of the scene, its authoring and rendering by introducing more descriptors to existing entities as well as additional entities if necessary. The use of extensions is optional, but their use must be declared in the meta section. A rendering engine is not mandated to respect all extensions. In the case that a renderer does not know how to deal with an extension, those related descriptors are either simply ignored, or interpreted by the renderer to "gracefully fail".

SpatDIF extensions are arranged according to a multi-layer structure (Figure 2), suggested in [4]. This structure mediates essential components in sound spatialization and strives to facilitate artistic work with spatialization systems regarding structure, flexibility, and interoperability. Figure 4 on page 8 illustrates various extensions that are currently being considered, organized by the layer they belong to.

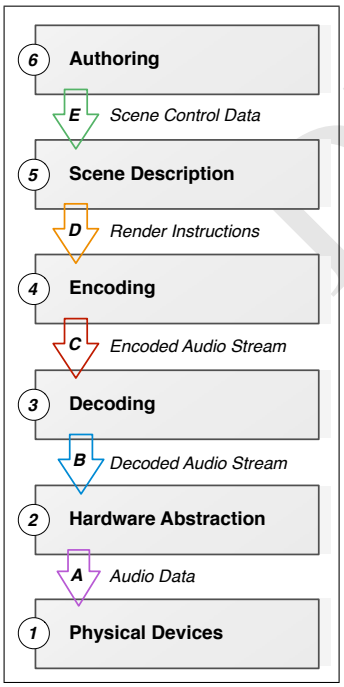


Figure 2: Layers and streams in sound spatialization [4].

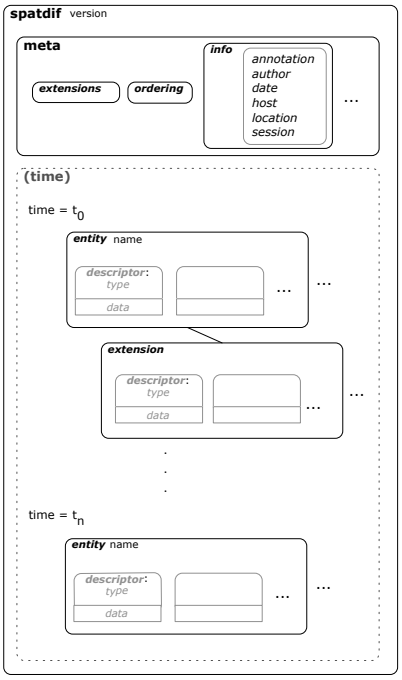


Figure 3: General structure of a SpatDIF representation - Meta Section and Time Section

3 Sections

3.1 Meta Section

The Meta section is mandatory for a SpatDIF representation. Meta descriptions (see Figure 3) are stored at the beginning of a SpatDIF representation and serve as information storage. These informations are not executed at runtime. The meta descriptions contain general annotation and documentation information, information about the organization of the following time section(s), higher-level process and compositional information, technical setup information referring to the original authoring situation and extension setup information. If extensions are used in the time section, they must be declared in the meta-section. Some extensions may require a setup in the meta section.

Descriptor	Description	data type	possible values	default value	possible units	default unit
`extensions`	A list of all extensions used within the SpatDIF representation, must be declared if extensions are used (see Section 5).	1 string	-	-	-	-

`ordering`	how is the time section is organized
	b) `time`: all entities are sorted on a time axis.
	c) `tracks`: all instances of each entity are sorted in tracks
		1 string	
			`time`
			`tracks`
				`time`	-	-

`info`	session information block					
`author`	Creator of the scene	1 string	-	-	-	-
`host`	Authoring tool used for creating the scene	1 string	-	-	-	-
`date`	Storing date in ISO 8601 format	1 string	-	-		
`session`	Session number	1 string	-	-	-	-
`location`	Studio or venue location	1 string	-	-	-	-
`annotation`	General comments about this scene	1 string	-	-	-	-

Table 1: Meta Section

3.2 Time Section

While the meta section serves to provide general information, the time section contains the audio scene rendering instructions in form of SpatDIF addresses and their values. These values can change over time.

SpatDIF offers two concepts for ordering the statements within the time section: Ordering by "time" is equivalent to an orchestral score and provides a complete overview. Ordering by "tracks" groups the statements into individual parts or tracks. In the context of real-time streaming of scenes, ordering by time is necessary, while in storage-type scenarios the "tracks" ordering principle may be more adequate.

- Time is considered continuous and linear as defined in the base quantities of the Intl. System of Units, see also Appendix A.
- All descriptions outside the meta section are considered to be audio scene rendering instructions and considered to be positioned along a time axis.
- Time declarations are relative to the beginning of the scene which starts at time 0.0 seconds by default, see also Table 2.
- A time declaration places all subsequent declarations as taking place at this specific point in time, until a new declaration is observed.

Descriptor	Data type	Default unit	Default value	Alternative units
`time`	1 double	s	0.0	h, min, ms
	1 string		0:00:00.000	hms

Table 2: Time descriptor

4 The Core

4.1 Descriptor conventions

4.2 Entities

Entities are all objects that are affecting or interacting with the sound of that scene. In the core, there is currently only one kind of entity: source. Other entities, such as sinks or reference-points, can be added if necessary via extensions.

Each entity is identified through a unique name.

Source A source injects sound into the scene. The origin of that sound can be defined through the media resources (see Section 4.4). A source is further defined through its `type`. In the core, there is only one type:

- `point`, omnidirectional point source (default)

4.3 Descriptors

It is not mandatory for SpatDIF representations to use all of the core descriptors. Undefined descriptors are assumed to have the default value. The core descriptors are listed in Table 3.

Entity	Descriptor	Data type	Default unit	Default value	Alternative units
`source`	`type`	1 string	-	`point`	
	`present`	1 boolean	-	1	-
	`position`	3 double	`xyz`	0. 0. 0.	`aed, openGL`
	`orientation`	3 or 4 double	`euler`	0. 0. 0.	`quaternion, angle-axis`

Table 3: Core descriptors

- For the definition of coordinate systems and their conversion please refer to Appendix C.
- For the definition of orientation systems and their conversion please refer to Appendix D.

A source definition implicitly activates the present flag. When removing a source using the 'present false' message, the source's internal state is deleted. The next time the source is activated it reverts to the default state. Setting any source descriptor after it has been deactivated, this will reactivate it implicitly, i.e., set the present flag to true.

4.4 Media resources

The media resources are used to assign media content to sources within a scene. The media resources may be defined within the Meta Section and be referenced within the scene through their `id`.
For the definition of gain units and their conversion please refer to Appendix E.

4.5 Loop

This features provides support for looping behaviors at different levels of hierarchy. This loop features can be used in conjunction with the media resources as well as to time-based behaviors of other extensions.

The loop descriptors are listed in Table 4.5.
The `type` descriptor can have an additional parameter which specifies the number of loops. For instance, a `repeat` will cause an infinite number of repetitions but a `repeat 3` will stop after the third repetition.

Descriptor	Description	data type	possible values	default value	possible units	default unit
`id`	Unique identifier	1 string	—	unique identifier	N/A	N/A
`type`	Where the content comes from	1 string	`stream`, `file`, `live`, `none`	`none`	N/A	N/A
`location`	Location of the file or stream	1 string	—	NULL	N/A	N/A
`channel`	If `type` has more channels, define the channel that is taken as input	1 int	> 0	1	N/A	N/A
`time-offset`	Starting position within media file	1 double	≥ 0.0	0.0	ms, s, min, h, hms	sec
`gain`	gain value of the media	1 double		1.0	db, linear	linear

Table 4: Media descriptors

Descriptor	Description	data type	possible values	default value	possible units	default unit
`type`	Looping method	1 string (+1 int)	`none`, `repeat`, `palindrome`	`none`	-	-
`points`	Defining the region of the data which is looped	2 double	start-point end-point	0.0 eof	ms, s, min, h, hms	s
`wait-time`	Time before a loop is repeated after coming to an end	1 double	≥ 0.0	0.0	ms, s, min, h, hms	s

eof: end of file

Table 5: Loop descriptors

4.6 Interpolation

This functionality defines how temporally sparse information are up-sampled. Interpolation is computed with respect to the unit in which the target is expressed e.g., if a position is defined in cartesian coordinates, the interpolation will be also performed in cartesian coordinates. Please note that for the interpolation of orientation statements, a *gimbal lock* can occur depending on the chosen system.

Descriptor	Description	data type	possible values	default value	possible units	default unit
`type`	Interpolation method	1 int	0, 1	0	-	-

`type` value	Description
0	disabled - no interpolation
1	enabled - linear interpolation

Table 6: Interpolation descriptors

See Appendix B for interpolation equations.

5 Extensions

The use of extensions permits the use of descriptors for defining further rendering instructions and/or structural information of a sound scene. Extensions enrich the core descriptors and the descriptors of other extensions. As depicted in Figure 4, the extensions in this specification are organized in a multi-layer structure.

Future extensions will initially be developed and validated as a collaborative effort within the SpatDIF community, drawing on experts within the relevant fields. As the definition of an extension reaches maturity, it will be added to the SpatDIF specification.

5.1 Supporting Extensions

The use of any SpatDIF extension must be declared in the Meta Section using its `extension` descriptor. Thus, it becomes immediately apparent what rendering capabilities are necessary to interpret all descriptors, see accompanied scene examples.

In contrast to core descriptors, the execution of extended rendering instructions depends on the abilities of the renderer. In the case that a renderer is not equipped with a specific extension, it is up to the renderer how to deal with those data: In the standard case, descriptors of unsupported extensions are ignored. A smarter renderer however might be able to interpret those descriptors according to its given feature set to "fail gracefully". If a known descriptor is addressed with unknown or wrongly formatted data, the default value is to be used. In this case it might be useful to notify the user.

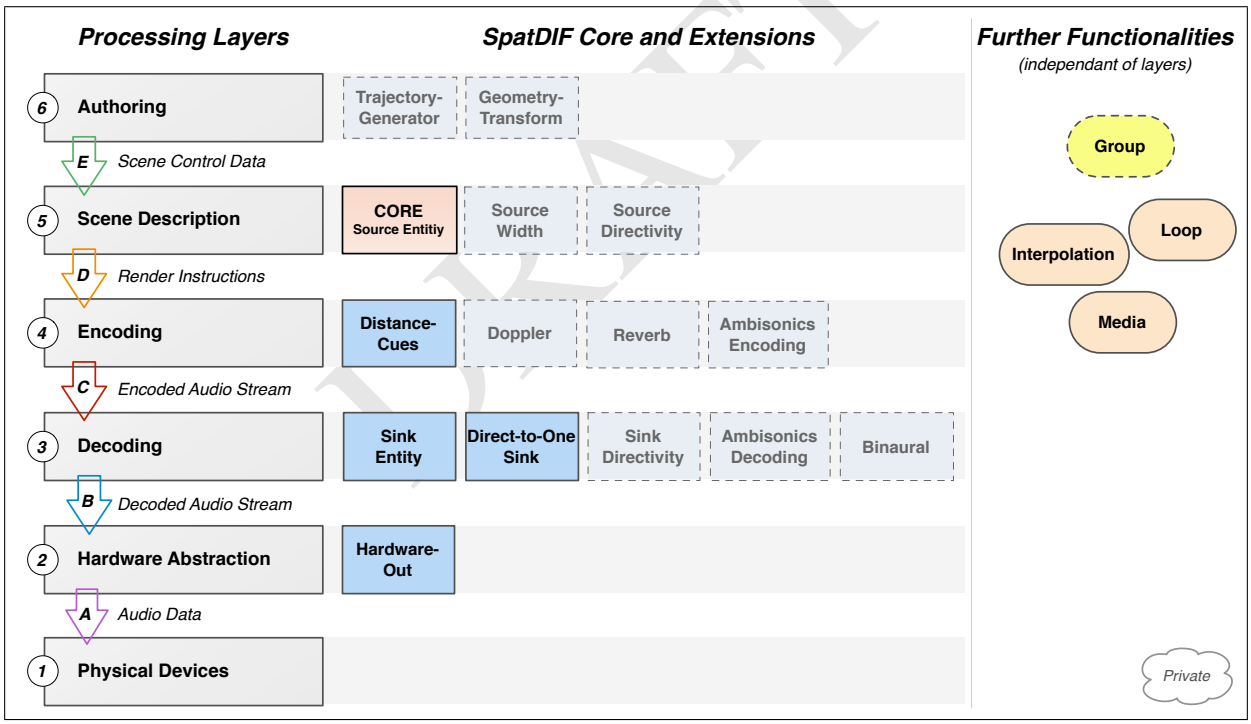


Figure 4: The layer model of the SpatDIF namespace. Extensions with a dashed frame are work-in-progress.

5.2 Extensions for Spatial Authoring Layer

There are currently no extensions defined relating to this layer. Some considered extensions are related to trajectory generators, trajectory transforms, and group hierarchies.

5.3 Extensions for Scene Description Layer

This is the layer where all core descriptors are located. There are currently no extensions defined relating to this layer. Some considered extensions are related to apparent source width, and source directivity.

5.4 Extensions for Spatial Encoding Layer

5.4.1 Distance-cues

This extension provides descriptors related to simulating distance of sound sources through gain adjustment and spectral filtering. There are two distance attenuation function provided. All default values are set to compute the classic inverse-square law until an attenuation of ≈ 96 dB. For more details see Appendix F.

Descriptor	Description	data type	possible values	default value	possible units	default unit
`distance-cues`	Extension name	1 string	-	-	-	-
`reference-distance`	The distance beyond which distance-attenuation is applied, and below which no attenuation happens	1 double	> 0.0	1.0	m	m
`maximum-distance`	The distance below which distance-attenuation is applied, and beyond the `maximum-attenuation` happens	1 double	> 0.0	62500	m	m
`maximum-attenuation`	Amplitude at `maximum-distance`	1 double		0.000016	dB, linear	linear
`attenuation-model`	Function to compute the distance-attenuation, see Appendix F	1 int	0, 1, 2	2	-	-
`absorption-model`	Spectral filtering due to the absorptive character of the transmission medium (e.g., air) defined through the cutoff-frequency of a lowpass filter	1 int	0, 1	1	-	-

Future Extensions

Other extensions considered for this layer are related to Doppler effect, Ambisonics encoding, and reverb effects.

5.5 Extensions for Spatial Decoding Layer

5.5.1 Sink Entity

This extension adds the entity sink to a SpatDIF scene representation. Sinks represent the acoustical outputs of the scene. These sink descriptors are similarly defined to the source descriptors in the SpatDIF core (see Section 4.3). A sink is further described through its `type`: by declaring its type, other descriptors might be interpreted accordingly. For instance a sink of 'type listener' enables the interaction with a `binaural` extension.

- `loudspeaker` (default)
- `listener`
- `microphone`
- `undefined`

Entity	Descriptor	Data type	Default unit	Default value	Alternative units
`sink`	`type`	1 string	-	`point`	
	`present`	1 boolean	-	1	-
	`position`	3 double	`xyz`	0. 0. 0.	`aed, openGL`
	`orientation`	3 or 4 double	`euler`	0. 0. 0.	`quaternion, angle-axis`

Table 7: Sink descriptors

5.5.2 Direct-to-One Sink

Sources which are not intended to be processed according to their spatial position, but rather directly fed in to the nearst sink. It is the renderer's job to assign the signal to the correct output. Because the scene doesn't need to contain an explicit sink description, this is an implicit routing instruction. For an explicit routing to a specific hardware channel use the Hardware-out extension from the Hardware Abstractions Layer (Section 5.6.1).

Descriptor	Description	data type	possible values	default value	possible units	default unit
`direct-to-one`	snap source to nearest sink	1 boolean	1, 0	0	-	-

Table 8: Direct-to-one Sink descriptor

Future Extensions

Other extensions considered for this layer are related to various rendering concepts such as ViMiC, Ambisonics, Wave Field Synthesis, or binaural.

5.6 Extensions for Hardware Abstractions Layer

5.6.1 Hardware-out

This extensions provides support for setting up the hardware layer. It is also used to link a source directly to a particular loudspeaker instead of being virtually rendered (see scene example `Stereo Playback`). This is an explicit routing instruction, for an implicit routing to an output channel use the Direct-to-one extension (Section 5.5.2).

Descriptor	Description	data type	possible values	default value	possible units	default unit
`hardware-out`	Extension name	1 string	-	-	-	-
`physical-channel`	Physical output channel of the hardware	1 int	> 0	-	-	-
`gain`	Gain value of the output	1 double	-	1.0	linear, dB	linear

Table 9: Hardware-out descriptors

5.7 Extensions for Physical Layer

There are currently no extensions defined or considered for this layer.

5.8 Private

The `private` extension serves as a generic container to store statements which are renderer specific and not covered within the SpatDIF specification. A private extension has to be declared with `private` and an individual descriptor (e.g., the name of the renderer). The use of a private extension must be declared with the `extensions` descriptors of the meta data.

Note: One should strive to limit the use of the private extension because private extensions constrain the interchangeability of SpatDIF representations.

6 General Conventions

- SpatDIF descriptors are to be kept as concise as possible, written all lowercase except for acronyms.
- Compound words are joined by a dash, e.g., `wait-time`.
- Descriptors have default values and units (where applicable).
- Alternative units or coordinate systems can be used (see conversion tables in Appendix).
- All entities have internal states i.e., persistence of parameters.
- All entities have default states at the beginning of the scene description, which get overwritten by explicit rendering information.
- When new rendering instructions are received, un-touched parameters remain the same.
- Extensions are required to adhere to this convention.

References

[1] Nils Peters. Proposing SpatDIF - The Spatial Sound Description Interchange Format. In *Proc. of the International Computer Music Conference*, Belfast, UK, 2008.

[2] Nils Peters, Sean Ferguson, and Stephen McAdams. Towards a Spatial Sound Description Interchange Format (SpatDIF). *Canadian Acoustics*, 35(3):64 – 65, 2007.

[3] Nils Peters, Trond Lossius, and Jan C. Schacher. SpatDIF: Principles, specification, and examples. In *Proc. of the 9th Sound and Music Computing Conference*, Copenhagen, DK, 2012.

[4] Nils Peters, Trond Lossius, Jan C. Schacher, Pascal Baltazar, Charles Bascou, and Timothy Place. A stratified approach for sound spatialization. In *Proc. of the 6th Sound and Music Computing Conference*, pages 219–224, Porto, PT, 2009.

Appendices

A Time Unit Conversion

Name	Unit	Convert to default	Convert from default
Millisecond	ms	$y = x \cdot 0.001$	$y = x \cdot 1000$
Second (default)	s	$y = x$	$y = x$
Minute	min	$y = x \cdot 60$	$y = x/60$
Hour	h	$y = x \cdot 3600$	$y = x/3600$

Table 10: Conversions of time units to and from the default unit *second*

A.1 Examples for hms conversion

Table 11 describes the conversation of a hms string (hour-minute-second) into the ISO time units. Milliseconds are optionally expressed as fractions of a second.

hms time	converted time unit
1 : 00 : 00.000 hms	1 h
1 : 00 : 00 hms	1 h
1 : 00.000 hms	1 min
1.000 hms	1 s
0.100 hms	100 ms

Table 11: Conversions of hms unit to other time units

B Interpolation Equations

No interpolation

$$y_k = f(x_{floor(k)}) \tag{1}$$

assuming the index $k = 0.5$:

$$y_{0.5} = f(x_{floor(0.5)})$$
$$y_{0.5} = f(x_0)$$

Linear interpolation

$$y_k = f(x_{floor(k)}) + (k - floor(k)) \cdot (f(x_{ceil(k)}) - f(x_{floor(k)})) \tag{2}$$

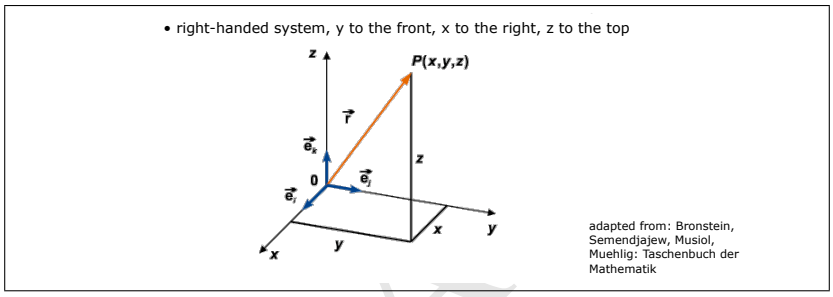
assuming the index $k = 0.5$:

$$y_{0.5} = f(x_0) + (0.5 - 0) \cdot (f(x_1) - f(x_0))$$
$$y_{0.5} = f(x_0) + 0.5 \cdot (f(x_1) - f(x_0))$$

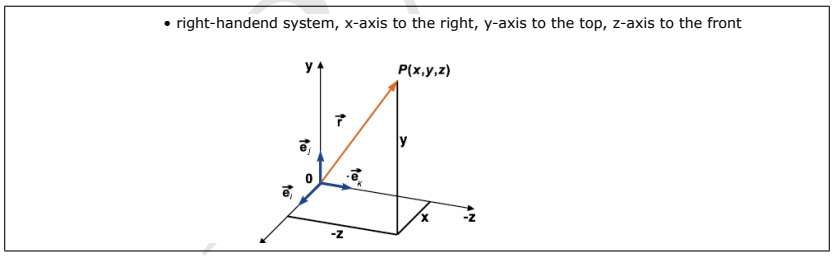
C Coordinate Systems Conversion

Name	Unit	Convert to default	Convert from default
Cartesian Navigational System (default)	xyz	$y_1 = x_1$ $y_2 = x_2$ $y_3 = x_3$	$y_1 = x_1$ $y_2 = x_2$ $y_3 = x_3$
OpenGL	openGL	$y_1 = x_1$ $y_2 = -x_3$ $y_3 = x_2$	$y_1 = x_1$ $y_2 = x_3$ $y_3 = -x_2$
Spherical Navigational System	aed	$y_1 = \sin(\frac{\Pi \cdot x_1}{180}) \cdot \cos(\frac{\Pi \cdot x_2}{180}) \cdot x_3$ $y_2 = \cos(\frac{\Pi \cdot x_1}{180}) \cdot \cos(\frac{\Pi \cdot x_2}{180}) \cdot x_3$ $y_3 = \sin(\frac{\Pi \cdot x_2}{180}) \cdot x_3$	$y_1 = atan2(x_1, x_2) \cdot \frac{180}{\Pi}$ $y_2 = atan2(x_3, \sqrt{x_1^2 + x_2^2}) \cdot \frac{180}{\Pi}$ $y_3 = \sqrt{x_1^2 + x_2^2 + x_3^2}$

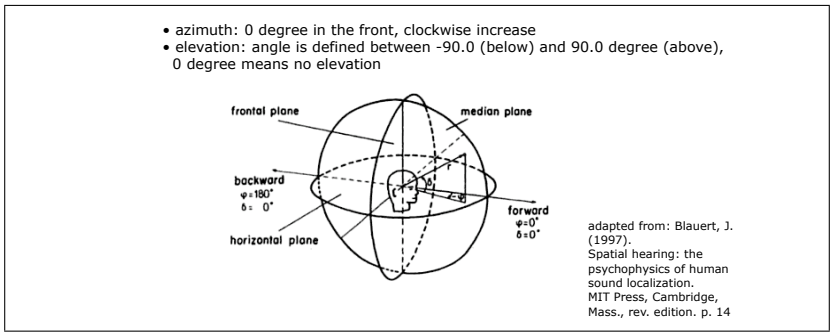
Table 12: Conversions of coordinate systems to and from the default system *xyz*



(a) Cartesian Navigational System - xyz (default)



(b) Computer Graphic System - openGL



(c) Spherical Navigational System - aed

Figure 5: Coordinate Systems conventions

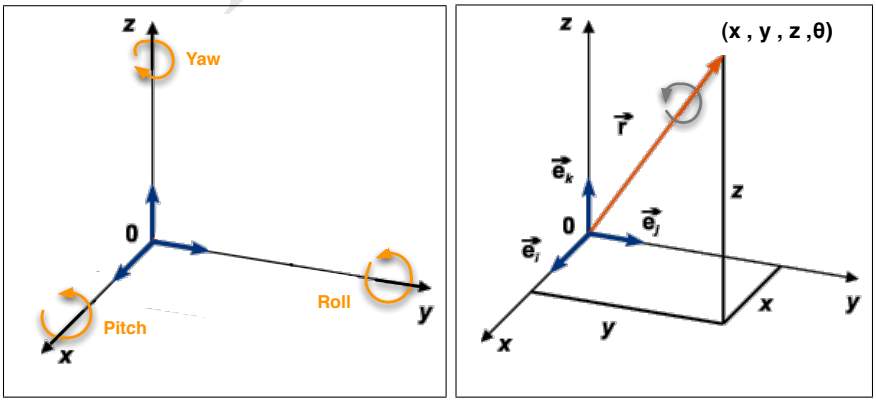
D Orientation System Conversion

The default unit for describing orientation of entities are Euler angles in degree. For converting between different orientation systems, quaternions in the order of `X,Y,Z,W` are used. The following table shows how to convert to and from quaternions.

Name	Unit	Convert to quaternion
Axis-angle	axis	$y_1 = x_1 \cdot \sqrt{x_1^2 + x_2^2 + x_3^2} \cdot \sin(\frac{\Pi \cdot x_4}{360})$ $y_2 = x_2 \cdot \sqrt{x_1^2 + x_2^2 + x_3^2} \cdot \sin(\frac{\Pi \cdot x_4}{360})$ $y_3 = x_3 \cdot \sqrt{x_1^2 + x_2^2 + x_3^2} \cdot \sin(\frac{\Pi \cdot x_4}{360})$ $y_4 = \cos(\frac{\Pi \cdot x_4}{360})$
Euler (yaw/pitch/roll) default	Euler	$y_1 = \cos(-\frac{\Pi \cdot x_1}{360}) \cdot \sin(\frac{\Pi \cdot x_2}{360}) \cdot \cos(\frac{\Pi \cdot x_3}{360}) - \sin(-\frac{\Pi \cdot x_1}{360}) \cdot \cos(\frac{\Pi \cdot x_2}{360}) \cdot \sin(\frac{\Pi \cdot x_3}{360})$ $y_2 = \cos(-\frac{\Pi \cdot x_1}{360}) \cdot \cos(\frac{\Pi \cdot x_2}{360}) \cdot \sin(\frac{\Pi \cdot x_3}{360}) + \sin(-\frac{\Pi \cdot x_1}{360}) \cdot \sin(\frac{\Pi \cdot x_2}{360}) \cdot \cos(\frac{\Pi \cdot x_3}{360})$ $y_3 = \sin(-\frac{\Pi \cdot x_1}{360}) \cdot \cos(\frac{\Pi \cdot x_2}{360}) \cdot \cos(\frac{\Pi \cdot x_3}{360}) + \cos(-\frac{\Pi \cdot x_1}{360}) \cdot \sin(\frac{\Pi \cdot x_2}{360}) \cdot \sin(\frac{\Pi \cdot x_3}{360})$ $y_4 = \cos(-\frac{\Pi \cdot x_1}{360}) \cdot \cos(\frac{\Pi \cdot x_2}{360}) \cdot \cos(\frac{\Pi \cdot x_3}{360}) - \sin(-\frac{\Pi \cdot x_1}{360}) \cdot \sin(\frac{\Pi \cdot x_2}{360}) \cdot \sin(\frac{\Pi \cdot x_3}{360})$
Quaternion	quaternion	*trivial*

Name	Unit	Convert from quaternion
Axis-angle	axis	$y_1 = \frac{x_1}{\sqrt{1-x_4^2}}$ $y_2 = \frac{x_2}{\sqrt{1-x_4^2}}$ $y_3 = \frac{x_3}{\sqrt{1-x_4^2}}$ $y_4 = \frac{360}{\Pi} \cdot atan2(\sqrt{1-x_4^2}, x_4)$
Euler (yaw/pitch/roll) default	Euler	$y_1 = \frac{180}{\Pi} \cdot atan2(-2 \cdot (x_3 \cdot x_4 - x_1 \cdot x_2), x_4^2 - x_1^2 + x_2^2 - x_3^2)$ $y_2 = \frac{180}{\Pi} \cdot \arcsin(2 \cdot (x_4 \cdot x_1 + x_2 \cdot x_3)$ $y_3 = \frac{180}{\Pi} \cdot atan2(2 \cdot (x_4 \cdot x_2 + x_1 \cdot x_3), x_4^2 - x_1^2 - x_2^2 + x_3^2)$
Quaternion	quaternion	*trivial*

Table 13: Conversions of orientation units to and from the *quaternion*



(a) Euler angle (yaw/pitch/roll) convention (b) axis-angle convention

Figure 6: Orientation convention within the default coordinate system *xyz*

E Gain Unit Conversion

Name	Unit	Convert to default	Convert from default
linear (default)	linear	$y = x$	$y = x$
Decibel	db	$y = 10^{x \cdot 0.05}$	$y = 20 \cdot \log 10(x)$

Table 14: Conversions of gain units to and from the default unit *linear*

F Distance-cue Functions

F.1 Distance-attenuation

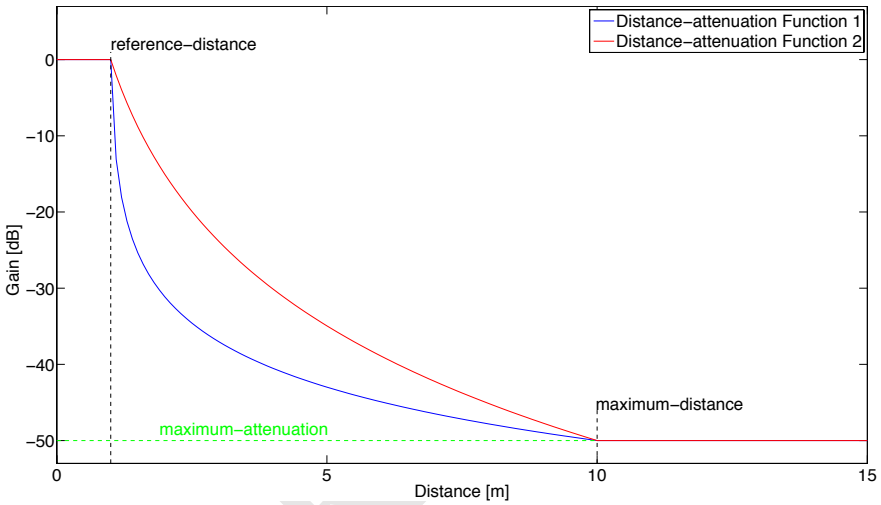


Figure 7: Comparison of the distance-attenuation functions.
`reference-distance`: 1 m, `maximum-distance`: 10 m, `maximum-attenuation`: 50 dB

Distance in SpatDIF assumes Euclidean calculation.
The distance attenuation function 1 used in SpatDIF are known from the literature and also used in OpenAL or DirectSound3D.
Function 2 defines the classic inverse-square law with an additional parameter a to manipulate the slope of the attenuation curve.
The default attenuation function in SpatDIF is function 2, because many spatialization renderer have this type implemented.

Function 0

No attenuation due to distance.

Function 1

$$\texttt{gain} = 20 \cdot \log_{10}\left(\frac{\texttt{reference\_distance}}{\texttt{reference\_distance} + ROF \cdot (distance - \texttt{reference\_distance})}\right) \quad (3)$$

with

$$ROF = \frac{\texttt{reference\_distance} \cdot 10^{-0.05 \cdot \texttt{maximum\_attenuation}} - \texttt{reference\_distance}}{\texttt{maximum\_distance} - \texttt{reference\_distance}}. \tag{4}$$

Function 2 (default)

$$\texttt{gain} = 20 \cdot \log_{10} \left(\frac{\texttt{reference\_distance}}{distance} \right)^{a} \tag{5}$$

with

$$a = \frac{\texttt{maximum\_attenuation}}{20 \cdot \log_{10}(\frac{\texttt{reference\_distance}}{\texttt{maximum\_distance}})}. \tag{6}$$

F.2 Distance-absorption

The high frequency absorption due to the medium can be simulated with a low-pass filtering of the audio signal. Based on the distance information, SpatDIF uses the following formula to compute the cutoff frequency of the low pass filter[3].

Function 0

No absorption due to distance.

Function 1 (default)

$$f_c = 15849 + distance \cdot (-785.71 + distance \cdot (18.919 - 0.1668 \cdot distance)); \tag{7}$$

[3]The ninja-style attenuation function based on temperature, static pressure, relative humidity, sound propagation distance, and frequency of sound can be found at `http://www.engr.uky.edu/~donohue/audio/Arrays/atmAtten.m`