FP6-NEST-PATH project no: 29085
Report Version: 1
Report Preparation Date: May 25, 2007
Classication: Pub.
Deliverable no. 2.1

Closing the Loop of Sound Evaluation and Design (CLOSED)

Deliverable 2.1

Algorithms for Ecologically-Founded Sound Synthesis: Library and Documentation

Stefano Delle Monache, Delphine Devallez, Carlo Drioli,
Federico Fontana, Stefano Papetti, Pietro Polotti,
Davide Rocchesso (UNIVERONA)

Physically-based

Sound Design Tools

User's Guide

version 0.4

VIPS group - University of Verona
project CLOSED [FP6-2004-NEST-PATH]
http://closed.ircam.fr

Contents

Introduction

The Physically-based **Sound Design Tools** (**SDT** from now on) aims at providing perception-oriented and physically-coherent advanced tools for the next generation of sound designers.

The **SDT** package is the main software product of a project activity which begun in 2001 with the EU project *SOb - the Sounding Object* [3]. In continuous development, the **SDT** package inherits basic theoretical concepts of sound synthesis initiated with the physical analysis of impact sounds, later complemented with more low-level physically-based sound models (such as friction) and further provided with higher level synthesis paradigms, integrating low-level models into more elaborate synthesis contexts (such as rolling and crumpling).

Such theoretical concepts found early realizations in the form of `pure data` patches and *externals* for Linux and Windows which had to be *ad hoc* compiled. In **SDT** all externals have been thoroughly revised and rewritten to comply with the multi-OS programming layer *flext* and to run in the Max/MSP environment. Moreover, the adoption of Max/MSP has allowed to restyle the early `pure data` patches in order to create more accessible, usable sound synthesis controls and better online documentations.

The **SDT** package is maintained and continuously developed by the UNIVERONA (VIPS group) unit of the EU project *CLOSED - Closing the Loop Of Sound Evaluation and Design*.

The basic (low-level) algorithms, the corresponding *externals* and some example patches are described in Chapter 1.

Higher level algorithms, based on the basic ones, are described in Chapter 2. In some cases the higher level algorithms are just implemented as Max/MSP patches which exploit the basic *externals*; in others they rely on both low-level and additional *externals*.

Externals descriptions are easily identifiable by boxed titles.

Throughout this manual we follow the Mac OS X (or UNIX) file-system conventions: e.g. paths are expressed with slashes "/". Of course Windows users just need to consider back-slashes "\" instead of slashes.

How to use the SDT package

The **SDT** package is split into three main sections (corresponding to subdirectories): source code (under `/SDT_sources`), Max/MSP patches (under `/SDT_patches`) and, for the faint hearted, precompiled Max/MSP *externals* for both Mac OS X[1] and Windows (under `/SDT_binaries`).

All main source files are coded in C++ (while some other auxiliary files are coded in C) using *flext*[2], a layer for cross-platform development of Max/MSP and `pure data` *externals*. This means that, with minimum effort, one can build the **SDT** package on Mac OS X, Windows and Linux platforms, this way getting *externals* for Max/MSP or `pure data`. In other words, although this manual is oriented towards Max/MSP, still sound designers keen on `pure data` can build the very same *externals* for their application of choice. Unfortunately though, the **SDT** package only include patches for Max/MSP.

Building flext

Despite the provided Max/MSP *externals* being ready for use, dauntless users would surely like to build their own ones. Other users may want to modify some of the provided *externals*, therefore

[1] Mac OS X *externals* are in universal binary format, that is they are compatible with both PPC and Intel Macs
[2] `http://grrrr.org/ext/flext/`

they would need to alter the corresponding source code, and then build their updated *externals* from scratch.

For all these advanced users the first thing to do is to download and install the **Max/MSP Software Development Kit** (**SDK**) from `http://www.cycling74.com/downloads/maxmsp`.

Secondly, it is necessary to download and install *flext*. At the time of writing, the version of *flext* required to correctly build the **SDT** package is only available as **CVS** check-out (starting with the current v0.5.1, any bleeding edge version should fit).

If your OS is Windows and you are not using **cygwin**, make sure that the option "`use UNIX line endings`" (or the like) of your **CVS** client is checked. If you don't know how to do that, and your client does not do it by default, after having checked-out *flext* you'd most likely need to open[3] all text-type files lying inside the directory `/flext` and its subdirectories, change all line endings to UNIX style, and save them.

Here are all the informations needed in order to check-out the **CVS** version of *flext* with any **CVS** client:

Protocol: `:pserver:`

User name: `anonymous`

Server: `pure-data.cvs.sourceforge.net`

Repository directory: `/cvsroot/pure-data`

Module: `externals/grill/flext`

Once the check-out is finished, it is necessary to configure, build and install *flext* on your machine. Please refer to the files `readme.txt` and `build.txt` inside the directory `/flext` of your *flext* distribution for detailed instructions on how to do that.

Building the SDT package

All source files lie inside the directory `/SDT_sources` (under `/Solids` and `/Liquids`).

To each external corresponds a `.txt` file containing instruction for the compiler and being used by *flext* own build system. In *flext*-based distributions, the default name for this kind of file is `package.txt` so that, when invoking the build batch-file, you don't have to specify its name. However, since SDT is made up of many externals which often share the same source files, we opted for putting all shared sources together, providing a `EXTERNAL_NAME.txt` file for each external. Hence, when invoking *flext* build batch-file it is necessary to add the macro `‘‘PKGINFO=EXTERNAL_NAME.txt’’` (including quotation marks) at the end of your sequence of instructions.

Hint 1: In order to refer to them more quickly, you can rename each package-type `.txt` file as you wish, e.g. with progressive numbers: `1.txt`, `2.txt`, etc..

Hint 2: The easiest way to compile the externals is to copy the whole directory `/SDT_sources` under the directory `/flext` of your *flext* distribution. Then it will be straightforward to invoke *flext* build batch-file as:

```
sh ../build.sh
```

from bash shell, or

```
..\build
```

[3]With a suitable text editor like the free Notepad++ (`http://notepad-plus.sourceforge.net/`)

from DOS command prompt.

Example 1: To build the external impact_2modalb∼ with **gcc** under Mac OS X, just open the terminal, go inside `/SDT_sources` and write:

```
sh ../build.sh max gcc "PKGINFO=impact_2modalb~.txt"
```

or change `../build.sh` accordingly to the specific path of your *flext* distribution. Then you'll find the compiled external within the directory `/max-darwin` under `/SDT_sources`.

Example 2: To build the external onebubble∼ with **Microsoft Visual Studio** under Windows, open the **Visual Studio Command Prompt**, go inside `\SDT_sources` and write:

```
..\build max msvc "PKGINFO=onebubble~.txt''
```

or change `..\build` accordingly to the specific path of your *flext* distribution. Then you'll find the compiled external within the directory `\max-msvc` under `\SDT_sources`

Installing the SDT package

To install the **SDT** package:

1. Copy all the *externals* for your OS into Max/MSP `/externals` directory or, provided that you update accordingly Max/MSP preferences, into your directory of choice.
2. Copy all `.help` files found under `/SDT_patches` and subdirectories into Max/MSP `/max-help` directory.
3. Start using the **SDT** by double-clicking on any `.mxb` example patch under `/SDT_patches` and have fun!

1 Low-level models

Basic models for solids-contact and liquids sound events are presented. After some theory, *externals* which implement the models and some example patches are described.

We will see in Chapter 2 how these basic models serve as a basis for more complex sound events, textures and processes.

1.1 Solids contact

The models considered here apply to basic contact events between two solid objects. As the most relevant contact sound events in everyday life come down to impacts and frictions, the provided *externals* model these two kinds of interactions.

The algorithms implemented here share a common structure: two **solid object** models interact through (what we call) an **interactor** (see Fig. 1.1).

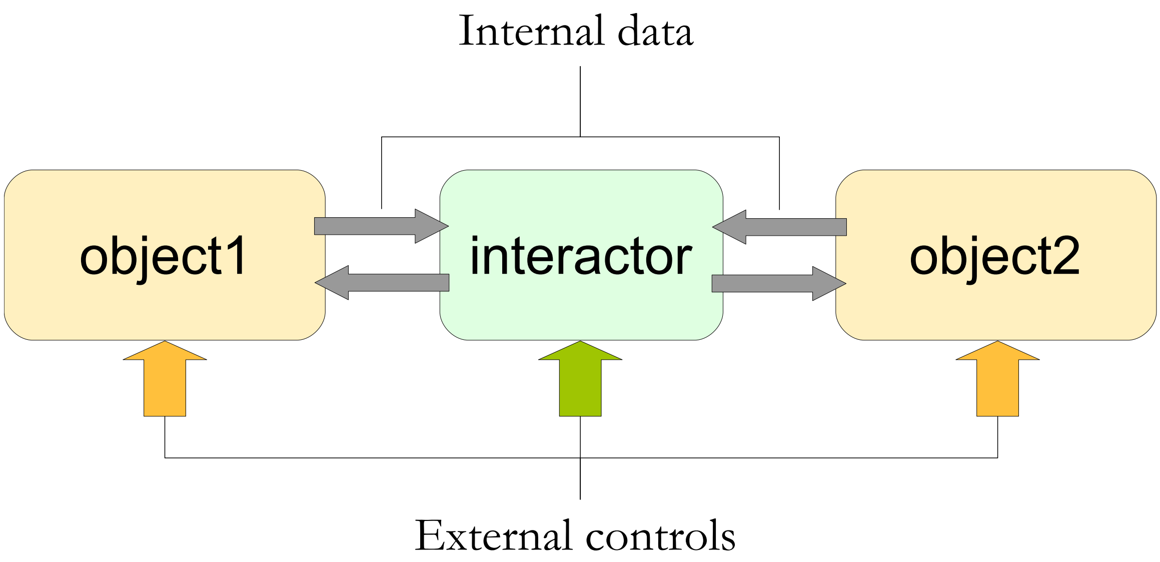

Figure 1.1: The common structure underlying solid contact algorithms.

Contact models

An interactor represents a contact model or, so to say, the "thing" between the two interacting objects. As for the impact model, it can be seen as the "felt" between the striking object and the struck object, while in the friction model it simulates friction as if the surfaces of the two rubbing objects would be covered with "micro-bristles".

Two impact models (one of which is a simplified version) and a friction model are provided:

impact interactor - implements a non-linear impact force. It receives the total compression (the difference of displacements of the two interacting objects at interaction point) and returns the computed impact force.

The latter is made of the sum of an *elastic* component and a *dissipative* one. The elastic component is parameterized by the *force stiffness* (or *elasticity*) and a *non-linear exponent* which depends on the local geometry around the contact area. The dissipative component is parameterized by the *force dissipation* (or *damping weight*). For further details refer to [3].

simplified impact interactor - implements a linear impact force. As it is a linearized version of the impact interactor described above, the non-linear exponent is not present (or, equivalently, is equal to 1), while the *force stiffness* and the *force dissipation* still remain.

friction interactor - implements a non-linear friction force. It receives the relative velocity of the two rubbing objects and returns the computed friction force.
The latter is made of the sum of four components, each of them corresponding to one coefficient. These are: an *elasticity* coefficient, an *internal dissipation* coefficient, a *viscosity* coefficient, and finally the gain of a pseudo-random function (noise related to *surface roughness*). The model is parametrized by several others quantities: the *dynamic friction* and the *static friction* coefficients, a *break-away coefficient* and the *Stribeck velocity* (both of them relate to the transient). For further details refer to [3].

Object models

Three distinct object models are provided:

modal object - in the modal description, a resonating object is described as a system of a finite number of parallel mass-spring-damper structures. Each mass-spring-damper structure models a mechanical oscillator which represents a normal mode of resonance of the object. The oscillation period, the mass and the damping coefficient of each oscillator correspond respectively to the *resonance frequency*, the *gain* and the *decay time* of each mode.
In our implementation it is possible to choose the wanted number of modes and to separately control their properties. Furthermore, each modal object has a number of pickup points, from which the sound is output. There must be at least one pickup point but they must be less than the number of modes. The first pickup point is also where the contact takes place (interaction point).

inertial object - simulates a simple inertial point mass. Obviously this kind of objects is useful solely as an exciter for other resonators.
The only settable object property is its *mass*.

waveguide object - the digital waveguide technique [4] models the propagation of waves along elastic media. In the one-dimensional case implemented here [2], the waveguide object models an ideal elastic string.
In our implementation it is possible to set *length*, *tension* and *mass* of the string. Further, one can set the position of the *interaction point* along its length. The interaction point coincides with the only available pickup point (i.e. the place from where the sound is output).

Generalities

Having a look at Fig. 1.1, the way two objects interact through an interactor appears evident: at each discrete time instant (sample) both objects send their internal states (displacement and velocity at the interaction point) to the interactor, which in turn sends the newly computed (opposite) forces to the objects. Knowing the new applied forces, the objects are able to compute their new states for the next time instant. In other words, there's a feedback communication between the three models.

The **SDT** framework differs remarkably from the approach to physically-based sound synthesis found in most existing implementations and literature. Being this not the seat for an exhaustive and analytical dissertation and comparison, of interest is the IRCAM software **Modalys** [1], a powerful physical modeling interactive tool for musical applications, based on modal synthesis.

The **Modalys** working space is characterized by a modular set of modal objects, such as tubes, membranes, strings, "two-mass" objects and hybrids; types of linear connections, that is types of interactions between objects; controllers, that specify exactly how the connections will be executed; "accesses", that set the physical location of the connection on a **Modalys** object in order to interact with other objects.

Recently, **Modalys** developments included the possibility to create three dimensional meshes and to compute their modes by finite elements numerical methods.

Available classes of modal objects include the following opcodes: "bi-string", a string or rod whose vibration moves in two transverse directions; "cello-bridge"; "circ-membrane", a circular membrane with zero thickness; "clamped-circ-membrane", a circular plate fixed at its edges; "closed-closed-tube", an acoustic tube sealed at both ends; "closed-open-tube", an acoustic tube sealed at one end and opened at the other; "melt-hybrid" and "mix-hybrid" that create a hybrid of two different objects. The latter can be seen as a box with the two objects inside, with a sound mix of the two objects, whose excitation transmits energy to the sub-objects in proportion to the current position of the hybrid.

Connections-interactions can be modularly set on each object. Among the others, the relative opcodes includes: "adhere", adherence between two accesses; "bi-fingerboard", that simulates the interaction between a finger and a string with a fingerboard underneath; "bow", a two dimensional connection between four accesses; "hole", that makes a hole of variable diameter in an acoustic tube; "pluck", where one access plucks another one; "position", for changing the access position along its axis. At last, controllers include noise generators, break-point envelope functions, filters, oscillators and arithmetical operators.

It can be noticed how types of objects and interactions already form parametrically high level subclasses to be connected in order to achieve more complex virtual musical instruments.

As far as the interaction structures are concerned, a main difference between the **SDT** package and **Modalys** resides in the feed-forward structure of the latter: in **Modalys** a form **exciter** ↔ **resonator** is adopted, where non-linearities are concentrated in the interaction part of a structure (e.g. blow, bow, etc.).

Bearing in mind the musical purposes conception of **Modalys**, elements like key noise for an acoustic tube are sometimes simulated by means of a usual random generator, rather than actually being calculated as a physical result of other stimuli.

On the contrary, the **SDT** package takes advantage of a *cartoonified* approach in sound design and implements a feedback network within the interaction **object1** ↔ **interactor** ↔ **object2**, with non-linear characteristics of the interactor. This allows the accurate modeling of complex interactions (e.g friction) and to output the sound of both the interacting objects. Besides, the continuous feedback approach adopted into the **SDT** is memory consistent, that is the system takes record of each previous state, during the interaction and manipulation. Secondly, the **SDT** package implements low level models that are already capable of all possible issues and interactions, with an open and wide variety of sound possibilities, and a consistent physical behavior.

Externals conventions

According to the described framework, the naming of each *external* conforms to the general form interactor_object1_object2~.

The linear impact interactor is referred as linpact, the non-linear impact interactor as impact, while the friction interactor is referred as friction.

The modal and the inertial objects are respectively referred as modalb and inertialb[1], while the waveguide object is referred as wg. The only naming exception happens when two modal objects interact, in which case the whole _object1_object2 section becomes _2modalb.

[1] The ending b is just a reminder for developers which tells that the *bilinear transformation* has been used in order to discretize the objects' continuous-time equations.

Notice that object1 is always either a modal or an inertial object. This is because those are the only objects which are able to move: as already said, an inertial object behaves as a point mass, while the first mode of a modal object1 is its **inertial mode**, that is the whole object behaves as a mass-spring-damper structure. In both cases it is therefore possible to make object1 move (e.g. imposing an external force or an initial velocity on it) so that it starts interacting with object2.

As modal objects have a variable number of modes and pickup points, when an *external* involves a modal object the arguments are of variable length and composition. Moreover, each pickup point corresponds to a signal outlet, therefore outlets of such *externals* are also in variable number.

In the following *externals* descriptions, elements which are in variable number are underlined. If applicable, each argument type is followed by the name of the element (interactor, object1, object2) it refers to.

Help patch conventions

The best way to understand how the supplied *externals* work is to have a look to their help patches. Just be sure to put them under Max/MSP `/max-help` directory to have access to them when invoking the help.

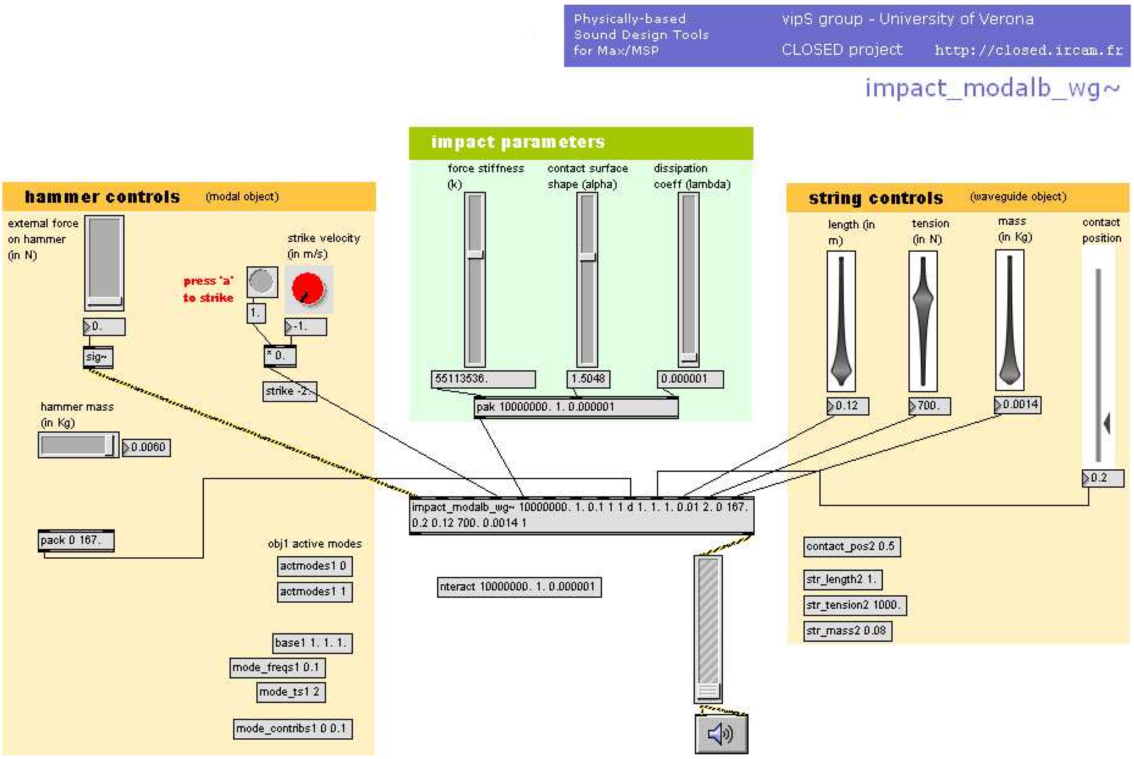

Figure 1.2: Solids contact example (help) patch

All help patches share a common structure (cfr. Fig. 1.2): the control panels on the left and right sides (orange backgrounds) are respectively for object1 and object2, those in the center (green background) are for the interactor.

Control panels for object1 only operate on a subset of its own properties (basically for screen space's sake), yet they allow to trigger the contact with object2. Conversely, the control panels provided for the interactor and object2 operate on the whole set of their available physical-geometric properties.

1.1.1 linpact_inertialb_modalb∼

Linear impact between one inertial and one modal object.

Input

signal	object1	1st inlet. External force applied to object1.
signal	object2	2nd inlet. External force applied to object2.
signal		3rd inlet. Additional displacement offset between the objects.
float	object1	4th inlet. Initial strike velocity.
list: [3×**float**]		5th inlet. The first two arguments are respectively the interactor's force stiffness and damping coefficient. The third is the mass of object1.
list: [3×**float**]	object2	6th inlet. Base (global) factors. These multiply respectively: the frequency, the decay time and the gain of all modes.
list: [**float**]	object2	7th inlet. As many arguments as the number of modes. Frequency of each mode.
list: [**float**]	object2	8th inlet. As many arguments as the number of modes. Decay time of each mode.
list: [**int float**]	object2	9th inlet. The int value indicates the considered pickup point. The subsequent list of float values sets the gain of each mode at the specified pickup point.
strike	object1	1st inlet. Followed by a float value, sets the initial strike velocity.
nteract_strikr		1st inlet. Followed by three float values. The first two set respectively the interactor's force stiffness and damping coefficient. The third sets the mass of object1.
actmodes2	object2	1st inlet. Followed by one int value which sets the number of currently active modes. Obviously the maximum number of active modes must be lower than the total number of available modes.
base2	object2	1st inlet. Followed by three float values correspondent to the base (global) factors. These multiply respectively: the frequency, the decay time and the gain of all modes.
mode_freqs2	object2	1st inlet. Followed by as many float values as the number of modes which set each mode frequency. Notice that for this message to have effect you need to refresh the base message.

mode_ts2	object2	1st inlet. Followed by as many float values as the number of modes which set their decay times. Notice that for this message to have effect you need to refresh the base message.
mode_contribs2	object2	1st inlet. Followed by one int value which specifies a pickup point, and as many float values as the number of modes. These set the gain (0-100 on a logarithmic scale) of each mode at the specified pickup point.

Output

All outlets come from pickup points of object2, in progressive order from left to right.

signal	object2	As many signal outlets as the number of pickup points. Depending on the chosen output mask (see the Arguments section below), they output either the object's velocity or displacement.

Arguments

All arguments are mandatory. They initialize the two objects and the interactor with their physical-geometric properties. In progressive order they are:

list: [3×**float**]		The first two arguments are respectively the interactor's force stiffness and damping coefficient. The third is the mass of object1.
int	object2	Number of modes.
int	object2	Number of pickup points.
symbol	object2	As many arguments as the number of pickup points. Mask for the output of each pickup point: '1' sets the pickup point to output the object's velocity, while anything else (e.g. 'd') sets it to output the object's displacement.
list: [3×**float**]	object2	Base (global) factors. These multiply respectively: the frequency, the decay time and the gain of all modes.
list: [**float**]	object2	As many arguments as the number of modes. Frequency of each mode.
list: [**float**]	object2	As many arguments as the number of modes. Decay time of each mode.
list: [**int float**]	object2	As many lists as the number of pickup points. The length of each list is equal to the number of modes +1. The int value indicates the considered pickup point. The subsequent list of float values sets the gain of each mode at the specified pickup point.

1.1.2 linpact_2modalb~

Linear impact between two modal objects.

Input

signal	object1	1st inlet. External force applied to object1.
signal	object2	2nd inlet. External force applied to object2.
signal		3rd inlet. Additional displacement offset between the objects.
float	object1	4th inlet. Initial strike velocity.
list: [2×**float**]	interactor	5th inlet. The two arguments are respectively the force stiffness and the damping coefficient.
list: [3×**float**]	object1	6th inlet. Base (global) factors. These multiply respectively: the frequency, the decay time and the gain of all modes.
list: [**float**]	object1	7th inlet. As many arguments as the number of modes. Frequency of each mode.
list: [**float**]	object1	8th inlet. As many arguments as the number of modes. Decay time of each mode.
list: [**int float**]	object1	9th inlet. The int value indicates the considered pickup point. The subsequent list of float values sets the gain of each mode at the specified pickup point.
list: [3×**float**]	object2	10th inlet. Same as for the 6th inlet.
list: [**float**]	object2	11th inlet. Same as for the 7th inlet.
list: [**float**]	object2	12th inlet. Same as for the 8th inlet.
list: [**int float**]	object2	13th inlet. Same as for the 9th inlet.
strike	object1	1st inlet. Followed by a float value, sets the initial strike velocity.
nteract	interactor	1st inlet. Followed by two float values which set respectively the interactor's force stiffness and damping coefficient.
actmodes1	object1	1st inlet. Followed by one int value which sets the number of currently active modes. Obviously the maximum number of active modes must be lower than the total number of available modes.
base1	object1	1st inlet. Followed by three float values correspondent to the base (global) factors. These multiply respectively: the frequency, the decay time and the gain of all modes.

mode_freqs1	object1	1st inlet. Followed by as many float values as the number of modes which set each mode frequency. Notice that for this message to have effect you need to refresh the base message.
mode_ts1	object1	1st inlet. Followed by as many float values as the number of modes which set their decay times. Notice that for this message to have effect you need to refresh the base message.
mode_contribs1	object1	1st inlet. Followed by one int value which specifies a pickup point, and as many float values as the number of modes. These set the gain (0-100 on a logarithmic scale) of each mode at the specified pickup point.
actmodes2	object2	1st inlet. Same as actmodes1.
base2	object2	1st inlet. Same as base1.
mode_freqs2	object2	1st inlet. Same as mode_freqs1.
mode_ts2	object2	1st inlet. Same as mode_ts1.
mode_contribs2	object2	1st inlet. Same as mode_contribs1.

Output

Outlets come from pickup points of both object1 and object2 (starting from left).

For example, considering that object1 has one pickup and object2 has three, the first outlet would come from pickup0 of object1 and the subsequent ones respectively from pickup0, pickup1 and pickup2 of object2.

signal	object1	As many signal outlets as the number of pickup points of object1. Depending on the chosen output mask (see the Arguments section below), they output either the object's velocity or displacement.
signal	object2	Same as for object1.

Arguments

All arguments are mandatory. They initialize the two objects and the interactor with their physical-geometric properties. In progressive order they are:

list: [2×**float**]	interactor	The two arguments are respectively the force stiffness and the damping coefficient.
int	object1	Number of modes.
int	object1	Number of pickup points.

symbol	object1	As many arguments as the number of pickup points. Mask for the output of each pickup point: '1' sets the pickup point to output the object's velocity, while anything else (e.g. 'd') sets it to output the object's displacement.
int	object2	Number of modes.
int	object2	Number of pickup points.
symbol	object2	As many arguments as the number of pickup points. Mask for the output of each pickup point: '1' sets the pickup point to output the object's velocity, while anything else (e.g. 'd') sets it to output the object's displacement.
list: [3×**float**]	object1	Base (global) factors. These multiply respectively: the frequency, the decay time and the gain of all modes.
list: [**float**]	object1	As many arguments as the number of modes. Frequency of each mode.
list: [**float**]	object1	As many arguments as the number of modes. Decay time of each mode.
list: [**int** **float**]	object1	As many lists as the number of pickup points. The length of each list is equal to the number of modes +1. The int values indicates the considered pickup point. The subsequent list of float values sets the gain of each mode at the specified pickup point.
list: [3×**float**]	object2	Base (global) factors. These multiply respectively: the frequency, the decay time and the gain of all modes.
list: [**float**]	object2	As many arguments as the number of modes. Frequency of each mode.
list: [**float**]	object2	As many arguments as the number of modes. Decay time of each mode.
list: [**int** **float**]	object2	As many lists as the number of pickup points. The length of each list is equal to the number of modes +1. The int value indicates the considered pickup point. The subsequent list of float values sets the gain of each mode at the specified pickup point.

1.1.3 impact_inertialb_modalb~

Non-linear impact between one inertial and one modal object.

Input

signal	object1	1st inlet. External force applied to object1.
signal	object2	2nd inlet. External force applied to object2.
signal		3rd inlet. Additional displacement offset between the objects.
float	object1	4th inlet. Initial strike velocity.
list: [4×**float**]		5th inlet. The first three arguments are for the interactor. They are respectively the force stiffness, a parameter which depends on the contact-surface's shape, and the dissipation coefficient. The fourth is the mass of object1.
list: [3×**float**]	object2	6th inlet. Base (global) factors. These multiply respectively: the frequency, the decay time and the gain of all modes.
list: [**float**]	object2	7th inlet. As many arguments as the number of modes. Frequency of each mode.
list: [**float**]	object2	8th inlet. As many arguments as the number of modes. Decay time of each mode.
list: [**int float**]	object2	9th inlet. The int value indicates the considered pickup point. The subsequent list of float values sets the gain of each mode at the specified pickup point.
strike	object1	1st inlet. Followed by a float value, sets the initial strike velocity.
nteract_strikr		1st inlet. Followed by four float values. The first three set respectively the interactor's force stiffness, a parameter which depends on the contact-surface's shape, and the dissipation coefficient. The fourth sets the mass of object1.
actmodes2	object2	1st inlet. Followed by one int value which sets the number of currently active modes. Obviously the maximum number of active modes must be lower than the total number of available modes.
base2	object2	1st inlet. Followed by three float values correspondent to the base (global) factors. These multiply respectively: the frequency, the decay time and the gain of all modes.
mode_freqs2	object2	1st inlet. Followed by as many float values as the number of modes which set each mode frequency. Notice that for this message to have effect you need to refresh the base message.

mode_ts2	object2	1st inlet. Followed by as many float values as the number of modes which set their decay times. Notice that for this message to have effect you need to refresh the base message.
mode_contribs2	object2	1st inlet. Followed by one int value which specifies a pickup point, and as many float values as the number of modes. These set the gain (0-100 on a logarithmic scale) of each mode at the specified pickup point.

Output

All outlets come from pickup points of object2, in progressive order from left to right.

signal	object2	As many signal outlets as the number of pickup points. Depending on the chosen output mask (see the Arguments section below), they output either the object's velocity or displacement.

Arguments

All arguments are mandatory. They initialize the two objects and the interactor with their physical-geometric properties. In progressive order they are:

list: [4×**float**]		The first three arguments are for the interactor. They are respectively the force stiffness, a parameter which depends on the contact surface's shape, and the dissipation coefficient. The fourth is the mass of object1.
int	object2	Number of modes.
int	object2	Number of pickup points.
symbol	object2	As many arguments as the number of pickup points. Mask for the output of each pickup point: '1' sets the pickup point to output the object's velocity, while anything else (e.g. 'd') sets it to output the object's displacement.
list: [3×**float**]	object2	Base (global) factors. These multiply respectively: the frequency, the decay time and the gain of all modes.
list: [**float**]	object2	As many arguments as the number of modes. Frequency of each mode.
list: [**float**]	object2	As many arguments as the number of modes. Decay time of each mode.
list: [**int float**]	object2	As many lists as the number of pickup points. The length of each list is equal to the number of modes +1. The int value indicates the considered pickup point. The subsequent list of float values sets the gain of each mode at the specified pickup point.

1.1.4 impact_2modalb∼

Non-linear impact between two modal objects.

Input

signal	object1	1st inlet. External force applied to object1.
signal	object2	2nd inlet. External force applied to object2.
signal		3rd inlet. Additional displacement offset between the objects.
float	object1	4th inlet. Initial strike velocity.
list: [3×**float**]	interactor	5th inlet. The three arguments are respectively the force stiffness, a parameter which depends on the contact-surface's shape, and the dissipation coefficient.
list: [3×**float**]	object1	6th inlet. Base (global) factors. These multiply respectively: the frequency, the decay time and the gain of all modes.
list: [**float**]	object1	7th inlet. As many arguments as the number of modes. Frequency of each mode.
list: [**float**]	object1	8th inlet. As many arguments as the number of modes. Decay time of each mode.
list: [**int float**]	object1	9th inlet. The int value indicates the considered pickup point. The subsequent list of float values sets the gain of each mode at the specified pickup point.
list: [3×**float**]	object2	10th inlet. Same as for the 6th inlet.
list: [**float**]	object2	11th inlet. Same as for the 7th inlet.
list: [**float**]	object2	12th inlet. Same as for the 8th inlet.
list: [**int float**]	object2	13th inlet. Same as for the 9th inlet.
strike	object1	1st inlet. Followed by a float value, sets the initial strike velocity.
nteract	interactor	1st inlet. Followed by three float values which set respectively the force stiffness, a parameter which depends on the contact-surface's shape, and the dissipation coefficient.
actmodes1	object1	1st inlet. Followed by one int value which sets the number of currently active modes. Obviously the maximum number of active modes must be lower than the total number of available modes.
base1	object1	1st inlet. Followed by three float values correspondent to the base (global) factors. These multiply respectively: the frequency, the decay time and the gain of all modes.

mode_freqs1	object1	1st inlet. Followed by as many float values as the number of modes which set each mode frequency. Notice that for this message to have effect you need to refresh the base message.
mode_ts1	object1	1st inlet. Followed by as many float values as the number of modes which set their decay times. Notice that for this message to have effect you need to refresh the base message.
mode_contribs1	object1	1st inlet. Followed by one int value which specifies a pickup point, and as many float values as the number of modes. These set the gain (0-100 on a logarithmic scale) of each mode at the specified pickup point.
actmodes2	object2	1st inlet. Same as actmodes1.
base2	object2	1st inlet. Same as base1.
mode_freqs2	object2	1st inlet. Same as mode_freqs1.
mode_ts2	object2	1st inlet. Same as mode_ts1.
mode_contribs2	object2	1st inlet. Same as mode_contribs1.

Output

Outlets come from pickup points of both object1 and object2 (starting from left).

For example, considering that object1 has one pickup and object2 has three, the first outlet would come from pickup0 of object1 and the subsequent ones respectively from pickup0, pickup1 and pickup2 of object2.

signal	object1	As many signal outlets as the number of pickup points of object1. Depending on the chosen output mask (see the Arguments section below), they output either the object's velocity or displacement.
signal	object2	Same as for object1.

Arguments

All arguments are mandatory. They initialize the two objects and the interactor with their physical-geometric properties. In progressive order they are:

list: [3×**float**]	interactor	The three arguments are respectively the force stiffness, a parameter which depends on the contact-surface's shape, and the dissipation coefficient.
int	object1	Number of modes.
int	object1	Number of pickup points.

symbol	object1	As many arguments as the number of pickup points. Mask for the output of each pickup point: '1' sets the pickup point to output the object's velocity, while anything else (e.g. 'd') sets it to output the object's displacement.
int	object2	Number of modes.
int	object2	Number of pickup points.
symbol	object2	As many arguments as the number of pickup points. Mask for the output of each pickup point: '1' sets the pickup point to output the object's velocity, while anything else (e.g. 'd') sets it to output the object's displacement.
list: [3×**float**]	object1	Base (global) factors. These multiply respectively: the frequency, the decay time and the gain of all modes.
list: [**float**]	object1	As many arguments as the number of modes. Frequency of each mode.
list: [**float**]	object1	As many arguments as the number of modes. Decay time of each mode.
list: [**int float**]	object1	As many lists as the number of pickup points. The length of each list is equal to the number of modes +1. The int value indicates the considered pickup point. The subsequent list of float values sets the gain of each mode at the specified pickup point.
list: [3×**float**]	object2	Base (global) factors. These multiply respectively: the frequency, the decay time and the gain of all modes.
list: [**float**]	object2	As many arguments as the number of modes. Frequency of each mode.
list: [**float**]	object2	As many arguments as the number of modes. Decay time of each mode.
list: [**int float**]	object2	As many lists as the number of pickup points. The length of each list is equal to the number of modes +1. The int value indicates the considered pickup point. The subsequent list of float values sets the gain of each mode at the specified pickup point.

1.1.5 impact_inertialb_wg~

Non-linear impact between one inertial and one waveguide object.

Input

signal	object1	1st inlet. External force applied to object1.
signal	object2	2nd inlet. External force applied to object2.
signal		3rd inlet. Additional displacement offset between the objects.
float	object1	4th inlet. Initial strike velocity.
list: [4×**float**]		5th inlet. The first three arguments are for the interactor. They are respectively the force stiffness, a parameter which depends on the contact-surface's shape, and the dissipation coefficient. The fourth is the mass of object1.
float	object2	6th inlet. Normalized (0-1) contact position along string's length.
float	object2	7th inlet. String's length in meters.
float	object2	8th inlet. String's tension in Newtons.
float	object2	9th inlet. String's mass in kilograms.
strike	object1	1st inlet. Followed by a float value, sets the initial strike velocity.
nteract_strikr		1st inlet. Followed by four float values. The first three set respectively the interactor's force stiffness, a parameter which depends on the contact-surface's shape, and the dissipation coefficient. The fourth sets the mass of object1.
contact_pos2	object2	1st inlet. Followed by a float value, sets the normalized (0-1) contact position along the string.
str_length2	object2	1st inlet. Followed by a float value, sets the string's length in meters.
str_tension2	object2	1st inlet. Followed by a float value, sets the string's tension in Newtons.
str_mass2	object2	1st inlet. Followed by a float value, sets the string's mass in kilograms.

Output

The outlet comes from the only pickup point of object2.

signal	object2	Depending on the chosen output mask (see the Arguments section below), the outlet outputs either the object's velocity or displacement at the contact position.

Arguments

All arguments are mandatory. They initialize the two objects and the interactor with their physical-geometric properties. In progressive order they are:

list: [4×**float**]		The first three arguments are for the interactor. They are respectively the force stiffness, a parameter which depends on the contact surface's shape, and the dissipation coefficient. The fourth is the mass of object1.
float	object2	Normalized (0-1) contact position along string's length.
float	object2	String's length in meters.
float	object2	String's tension in Newtons.
float	object2	String's mass in kilograms.
symbol	object2	Mask for the output of pickup point: '1' sets the pickup point to output the object's velocity, while anything else (e.g. 'd') sets it to output the object's displacement.

1.1.6 impact_modalb_wg∼

Non-linear impact between one modal and one waveguide object.

Input

signal	object1	1st inlet. External force applied to object1.
signal	object2	2nd inlet. External force applied to object2.
signal		3rd inlet. Additional displacement offset between the objects.
float	object1	4th inlet. Initial strike velocity.
list: [3×**float**]	interactor	5th inlet. The three arguments are respectively the force stiffness, a parameter which depends on the contact-surface's shape, and the dissipation coefficient.
list: [3×**float**]	object1	6th inlet. Base (global) factors. These multiply respectively: the frequency, the decay time and the gain of all modes.
list: [**float**]	object1	7th inlet. As many arguments as the number of modes. Frequency of each mode.
list: [**float**]	object1	8th inlet. As many arguments as the number of modes. Decay time of each mode.
list: [**int** **float**]	object1	9th inlet. The int value indicates the considered pickup point. The subsequent list of float values sets the gain of each mode at the specified pickup point.
float	object2	10th inlet. Normalized (0-1) contact position along string's length.
float	object2	11th inlet. String's length in meters.
float	object2	12th inlet. String's tension in Newtons.
float	object2	13th inlet. String's mass in kilograms.
strike	object1	1st inlet. Followed by a float value, sets the initial strike velocity.
nteract	interactor	1st inlet. Followed by three float values which set respectively the force stiffness, a parameter which depends on the contact-surface's shape, and the dissipation coefficient.
actmodes1	object1	1st inlet. Followed by one int value which sets the number of currently active modes. Obviously the maximum number of active modes must be lower than the total number of available modes.
base1	object1	1st inlet. Followed by three float values correspondent to the base (global) factors. These multiply respectively: the frequency, the decay time and the gain of all modes.

mode_freqs1	object1	1st inlet. Followed by as many float values as the number of modes which set each mode frequency. Notice that for this message to have effect you need to refresh the base message.
mode_ts1	object1	1st inlet. Followed by as many float values as the number of modes which set their decay times. Notice that for this message to have effect you need to refresh the base message.
mode_contribs1	object1	1st inlet. Followed by one int value which specifies a pickup point, and as many float values as the number of modes. These set the gain (0-100 on a logarithmic scale) of each mode at the specified pickup point.
contact_pos2	object2	1st inlet. Followed by a float value, sets the normalized (0-1) contact position along the string.
str_length2	object2	1st inlet. Followed by a float value, sets the string's length in meters.
str_tension2	object2	1st inlet. Followed by a float value, sets the string's tension in Newtons.
str_mass2	object2	1st inlet. Followed by a float value, sets the string's mass in kilograms.

Output

Outlets come from pickup points of both object1 and object2 (starting from left). The rightmost one comes from the only pickup point of object2.

signal	object1	As many signal outlets as the number of pickup points of object1. Depending on the chosen output mask (see the Arguments section below), they output either the object's velocity or displacement.
signal	object2	Depending on the chosen output mask (see the Arguments section below), the rightmost outlet outputs either the object's velocity or displacement at the contact position.

Arguments

All arguments are mandatory. They initialize the two objects and the interactor with their physical-geometric properties. In progressive order they are:

list: [3×**float**]	interactor	The three arguments are respectively the force stiffness, a parameter which depends on the contact-surface's shape, and the dissipation coefficient.
int	object1	Number of modes.

int	object1	Number of pickup points.
symbol	object1	As many arguments as the number of pickup points. Mask for the output of each pickup point: '1' sets the pickup point to output the object's velocity, while anything else (e.g. 'd') sets it to output the object's displacement.
list: [3×**float**]	object1	Base (global) factors. These multiply respectively: the frequency, the decay time and the gain of all modes.
list: [**float**]	object1	As many arguments as the number of modes. Frequency of each mode.
list: [**float**]	object1	As many arguments as the number of modes. Decay time of each mode.
list: [**int float**]	object1	As many lists as the number of pickup points. The length of each list is equal to the number of modes +1. The int value indicates the considered pickup point. The subsequent list of float values sets the gain of each mode at the specified pickup point.
float	object2	Normalized (0-1) contact position along string's length.
float	object2	String's length in meters.
float	object2	String's tension in Newtons.
float	object2	String's mass in kilograms.
symbol	object2	Mask for the output of pickup point: '1' sets the pickup point to output the object's velocity, while anything else (e.g. 'd') sets it to output the object's displacement.

1.1.7 friction_2modalb~

Non-linear friction between two modal objects.

Input

signal	object1	1st inlet. External force applied to object1.
signal	object2	2nd inlet. External force applied to object2.
signal		3rd inlet. Additional displacement offset between the objects.
float	object1	4th inlet. Initial rubbing velocity.
list: [9×**float**]		5th inlet. The nine arguments are respectively: the mean bristles-stiffness, the mean bristles-dissipation, a viscosity coefficient, the dynamic-friction coefficient, the static-friction coefficient, the break-away coefficient, the Stribeck velocity, the perpendicular pressure which object1 applies on object2, and finally the noise intensity.
list: [3×**float**]	object1	6th inlet. Base (global) factors. These multiply respectively: the frequency, the decay time and the gain of all modes.
list: [**float**]	object1	7th inlet. As many arguments as the number of modes. Frequency of each mode.
list: [**float**]	object1	8th inlet. As many arguments as the number of modes. Decay time of each mode.
list: [**int float**]	object1	9th inlet. The int value indicates the considered pickup point. The subsequent list of float values sets the gain of each mode at the specified pickup point.
list: [3×**float**]	object2	10th inlet. Same as for the 6th inlet.
list: [**float**]	object2	11th inlet. Same as for the 7th inlet.
list: [**float**]	object2	12th inlet. Same as for the 8th inlet.
list: [**int float**]	object2	13th inlet. Same as for the 9th inlet.
start_rubbing	object1	1st inlet. Followed by a float value, sets the initial rubbing velocity.
nteract_pressr		1st inlet. Followed by nine float values which set respectively: the mean bristles-stiffness, the mean bristles-dissipation, a viscosity coefficient, the dynamic-friction coefficient, the static-friction coefficient, the break-away coefficient, the Stribeck velocity, the perpendicular pressure which object1 applies on object2, and finally the noise intensity.

actmodes1	object1	1st inlet. Followed by one int value which sets the number of currently active modes. Obviously the maximum number of active modes must be lower than the total number of available modes.
base1	object1	1st inlet. Followed by three float values correspondent to the base (global) factors. These multiply respectively: the frequency, the decay time and the gain of all modes.
mode_freqs1	object1	1st inlet. Followed by as many float values as the number of modes which set each mode frequency. Notice that for this message to have effect you need to refresh the base message.
mode_ts1	object1	1st inlet. Followed by as many float values as the number of modes which set their decay times. Notice that for this message to have effect you need to refresh the base message.
mode_contribs1	object1	1st inlet. Followed by one int value which specifies a pickup point, and as many float values as the number of modes. These set the gain (0-100 on a logarithmic scale) of each mode at the specified pickup point.
actmodes2	object2	1st inlet. Same as actmodes1.
base2	object2	1st inlet. Same as base1.
mode_freqs2	object2	1st inlet. Same as mode_freqs1.
mode_ts2	object2	1st inlet. Same as mode_ts1.
mode_contribs2	object2	1st inlet. Same as mode_contribs1.

Output

Outlets come from pickup points of both object1 and object2 (starting from left).
For example, considering that object1 has one pickup and object2 has three, the first outlet would come from pickup0 of object1 and the subsequent ones respectively from pickup0, pickup1 and pickup2 of object2.

signal	object1	As many signal outlets as the number of pickup points of object1. Depending on the chosen output mask (see the Arguments section below), they output either the object's velocity or displacement.
signal	object2	Same as for object1.

Arguments

All arguments are mandatory. They initialize the two objects and the interactor with their physical-geometric properties. In progressive order they are:

list: [9×**float**]		The nine arguments are respectively: the mean bristles-stiffness, the mean bristles-dissipation, a viscosity coefficient, the dynamic-friction coefficient, the static-friction coefficient, the break-away coefficient, the Stribeck velocity, the perpendicular pressure which object1 applies on object2, and finally the noise intensity.
int	object1	Number of modes.
int	object1	Number of pickup points.
symbol	object1	As many arguments as the number of pickup points. Mask for the output of each pickup point: '1' sets the pickup point to output the object's velocity, while anything else (e.g. 'd') sets it to output the object's displacement.
int	object2	Number of modes.
int	object2	Number of pickup points.
symbol	object2	As many arguments as the number of pickup points. Mask for the output of each pickup point: '1' sets the pickup point to output the object's velocity, while anything else (e.g. 'd') sets it to output the object's displacement.
list: [3×**float**]	object1	Base (global) factors. These multiply respectively: the frequency, the decay time and the gain of all modes.
list: [**float**]	object1	As many arguments as the number of modes. Frequency of each mode.
list: [**float**]	object1	As many arguments as the number of modes. Decay time of each mode.
list: [**int float**]	object1	As many lists as the number of pickup points. The length of each list is equal to the number of modes +1. The int value indicates the considered pickup point. The subsequent list of float values sets the gain of each mode at the specified pickup point.
list: [3×**float**]	object2	Base (global) factors. These multiply respectively: the frequency, the decay time and the gain of all modes.
list: [**float**]	object2	As many arguments as the number of modes. Frequency of each mode.
list: [**float**]	object2	As many arguments as the number of modes. Decay time of each mode.

list: [**int** **float**]	object2	As many lists as the number of pickup points. The length of each list is equal to the number of modes +1. The int value indicates the considered pickup point. The subsequent list of float values sets the gain of each mode at the specified pickup point.

1.1.8 friction_modalb_wg~

Non-linear friction between one modal and one waveguide object.

Input

signal	object1	1st inlet. External force applied to object1.
signal	object2	2nd inlet. External force applied to object2.
signal		3rd inlet. Additional displacement offset between the objects.
float	object1	4th inlet. Initial bowing velocity.
list: [9×**float**]		5th inlet. The nine arguments are respectively: the mean bristles-stiffness, the mean bristles-dissipation, a viscosity coefficient, the dynamic-friction coefficient, the static-friction coefficient, the break-away coefficient, the Stribeck velocity, the perpendicular pressure which object1 applies on object2, and finally the noise intensity.
list: [3×**float**]	object1	6th inlet. Base (global) factors. These multiply respectively: the frequency, the decay time and the gain of all modes.
list: [**float**]	object1	7th inlet. As many arguments as the number of modes. Frequency of each mode.
list: [**float**]	object1	8th inlet. As many arguments as the number of modes. Decay time of each mode.
list: [**int float**]	object1	9th inlet. The int value indicates the considered pickup point. The subsequent list of float values sets the gain of each mode at the specified pickup point.
float	object2	10th inlet. Normalized (0-1) contact position along string's length.
float	object2	11th inlet. String's length in meters.
float	object2	12th inlet. String's tension in Newtons.
float	object2	13th inlet. String's mass in kilograms.
start_bowing	object1	1st inlet. Followed by a float value, sets the initial bowing velocity.
nteract_pressr		1st inlet. Followed by nine float values which set respectively: the mean bristles-stiffness, the mean bristles-dissipation, a viscosity coefficient, the dynamic-friction coefficient, the static-friction coefficient, the break-away coefficient, the Stribeck velocity, the perpendicular pressure which object1 applies on object2, and finally the noise intensity.

actmodes1 object1 1st inlet. Followed by one int value which sets the number of currently active modes. Obviously the maximum number of active modes must be lower than the total number of available modes.

base1 object1 1st inlet. Followed by three float values correspondent to the base (global) factors. These multiply respectively: the frequency, the decay time and the gain of all modes.

mode_freqs1 object1 1st inlet. Followed by as many float values as the number of modes which set each mode frequency.
Notice that for this message to have effect you need to refresh the base message.

mode_ts1 object1 1st inlet. Followed by as many float values as the number of modes which set their decay times.
Notice that for this message to have effect you need to refresh the base message.

mode_contribs1 object1 1st inlet. Followed by one int value which specifies a pickup point, and as many float values as the number of modes. These set the gain (0-100 on a logarithmic scale) of each mode at the specified pickup point.

contact_pos2 object2 1st inlet. Followed by a float value, sets the normalized (0-1) contact position along the string.

str_length2 object2 1st inlet. Followed by a float value, sets the string's length in meters.

str_tension2 object2 1st inlet. Followed by a float value, sets the string's tension in Newtons.

str_mass2 object2 1st inlet. Followed by a float value, sets the string's mass in kilograms.

Output

Outlets come from pickup points of both object1 and object2 (starting from left). The rightmost one comes from the only pickup point of object2.

signal object1 As many signal outlets as the number of pickup points of object1. Depending on the chosen output mask (see the Arguments section below), they output either the object's velocity or displacement.

signal object2 Depending on the chosen output mask (see the Arguments section below), the rightmost outlet outputs either the object's velocity or displacement at the contact position.

Arguments

All arguments are mandatory. They initialize the two objects and the interactor with their physical-geometric properties. In progressive order they are:

list: [9×**float**]		The nine arguments are respectively: the mean bristles-stiffness, the mean bristles-dissipation, a viscosity coefficient, the dynamic-friction coefficient, the static-friction coefficient, the break-away coefficient, the Stribeck velocity, the perpendicular pressure which object1 applies on object2, and finally the noise intensity.
int	object1	Number of modes.
int	object1	Number of pickup points.
<u>**symbol**</u>	object1	As many arguments as the number of pickup points. Mask for the output of each pickup point: '1' sets the pickup point to output the object's velocity, while anything else (e.g. 'd') sets it to output the object's displacement.
list: [3×**float**]	object1	Base (global) factors. These multiply respectively: the frequency, the decay time and the gain of all modes.
list: [<u>**float**</u>]	object1	As many arguments as the number of modes. Frequency of each mode.
list: [<u>**float**</u>]	object1	As many arguments as the number of modes. Decay time of each mode.
<u>**list**</u>: <u>[**int** <u>**float**</u>]</u>	object1	As many lists as the number of pickup points. The length of each list is equal to the number of modes +1. The int value indicates the considered pickup point. The subsequent list of float values sets the gain of each mode at the specified pickup point.
float	object2	Normalized (0-1) contact position along string's length.
float	object2	String's length in meters.
float	object2	String's tension in Newtons.
float	object2	String's mass in kilograms.
symbol	object2	Mask for the output of pickup point: '1' sets the pickup point to output the object's velocity, while anything else (e.g. 'd') sets it to output the object's displacement.

1.2 Liquids

Low level models of basic events responsible for acoustic emission in liquids are considered. In particular, we consider the formation of single resonating cavities (bubbles) observed in dripping, boiling, or pouring.

Bubble model

The formation of radially oscillating bubbles under the surface of a liquid volume is modeled assuming that the bubble cavity acts as a simple Helmoltz resonator, its impulse response being a damped sinusoid with time-varying frequency [5]. One of the most common events involving the formation of radially oscillating bubbles under the surface of a liquid volume is dripping, the falling of a drop or an object into a quiescent liquid. In the simplest case, when the initial impact sound is neglected and when there is no crown formation following the initial impact, dripping can be represented by a single bubble event.

However, the more the initial impact sound is perceivable and the cavity of the bubble becomes larger (e.g., for large impacting mass), the less the simple single bubble sound model becomes adequate to represent the dripping event. This is even more evident when large objects or drops falling into a resting liquid generate many secondary bubbles and droplets events due to the mass displaced by the principal impact event (splashing).

The *external* onebubble∼ provides controls for the bubble radius, and for the slope of the frequency rise due to radius change in time.

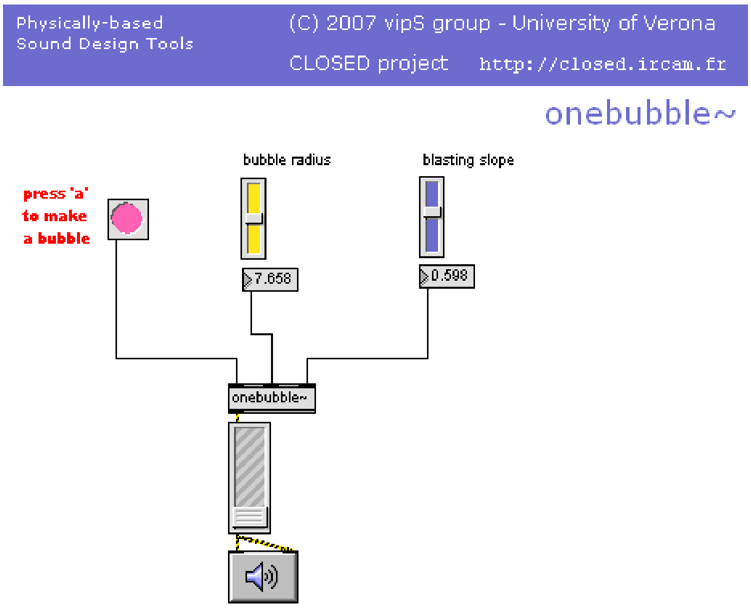

Figure 1.3: Single bubble help patch

1.2.1 onebubble~

Single radially oscillating bubble.

Input

bang	1st inlet. Bubble event trigger.
float	2nd inlet. Bubble initial radius.
float	2nd inlet. Frequency slope.
radius	1st inlet. Followed by a float value, sets the initial bubble radius.
slope	1st inlet. Followed by a float value, sets the frequency slope.

Output

Radiated pressure.

signal	Signal outlet.

Arguments

None.

2 Higher-level models

This chapter covers algorithms which exploit the low-level models seen in Chapter 1. Notice that the expression "higher-level" indicates more complex and structured algorithms, corresponding to somewhat large-scale events, processes or textures. In a way, that matches the meaning of the expression "high-level" in Computer Science, where it often denotes languages similar to those of human beings. Of course, in order to achieve that, high-level languages are indeed more complex and structured than low-level ones.

The higher-level algorithms here discussed implement temporal patterns or other physically consistent controls (e.g. external forces) superimposed to low-level models. These algorithms are made available either by dedicated *externals*, or by using Max/MSP visual programming, that is as patches. In the former case, the *external* can be self-contained - that is it implements both the control and the low-level synthesis layers - or just meant to drive other *externals* (in which case it is labeled with a final tilde ∼) - that is it implements only the control layer (in which case it is labeled without a final tilde ∼).

2.1 Solids

2.1.1 Bouncing

Bouncing is a sound process resulting from repeated macro-impact events.

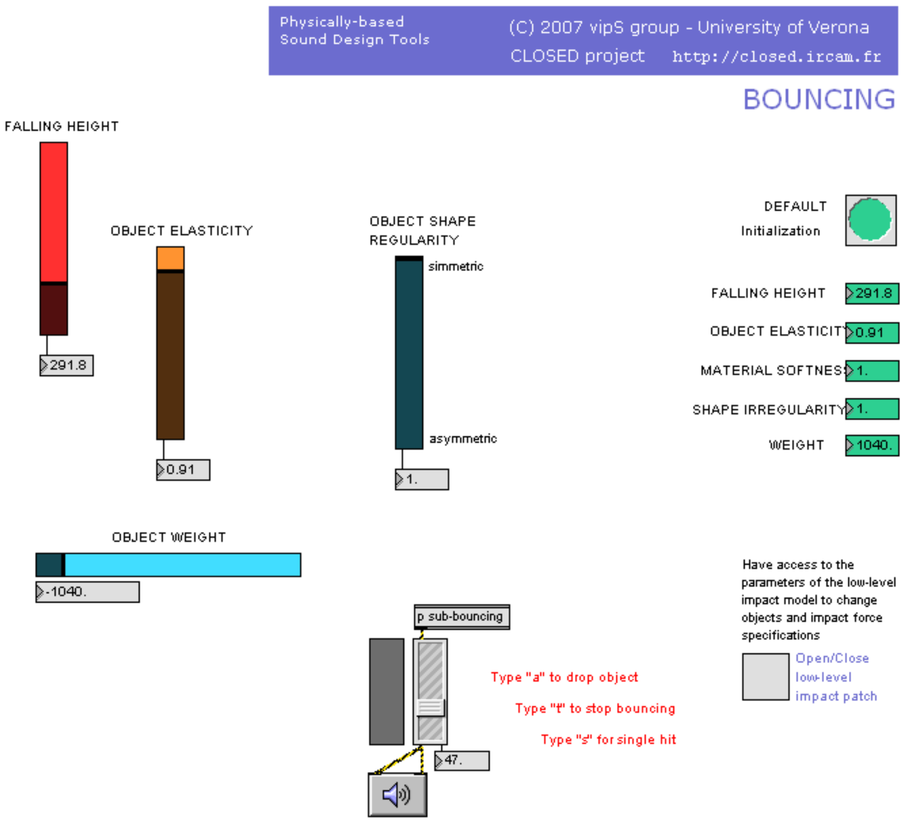

Figure 2.1: Bouncing (falling object) patch

The Max/MSP patch (`BOUNCING.mxb`) (Fig. 2.1) simulates the case of a falling object - a typical situation in which bouncing happens. There, an instance of the low-level *external* impact_inertialb_modalb∼ is driven by a control layer implemented into the sub-patch dropper (which lies inside the sub-patch sub-bouncing).

The available high-level user's controls are:

falling height: sets the time interval between the first two bounces.

object elasticity: sets the acceleration and deceleration rate of bounces.

object shape regularity: sets the level of randomness to be applied to bounces. This corresponds to the symmetry properties of the falling object.

object weight: sets the maximum hitting velocity.

2.1.2 control_crump

Crumpling is a widely general-purpose sonification paradigm based on the control of *atomic* sonic events (for instance impacts). The crumpling process governs the timbre and dynamics of every atomic event as well as the temporal gap in between two successive events.

- Concerning dynamics and time, their control is exerted by statically setting macroscopic parameters of specific stochastic laws which, in their turn, produce sequences of micro events whose overall intensity and temporal density follow directly from the parameter values. However, every sequence individually exhibits statistical variability.
- Concerning timbre, its control follows by informing the model with microscopic features (*viz.* the nature of the atomic sounds) along with changing (usually coarse) structural evolutionary features of the physical model dynamically along time.

Detailed explanations about the physical nature of this sonification paradigm, in particular the correspondence existing between an atomic crumpling event and its representation as an impact between a hammering and a resonating object, can be found in [3].

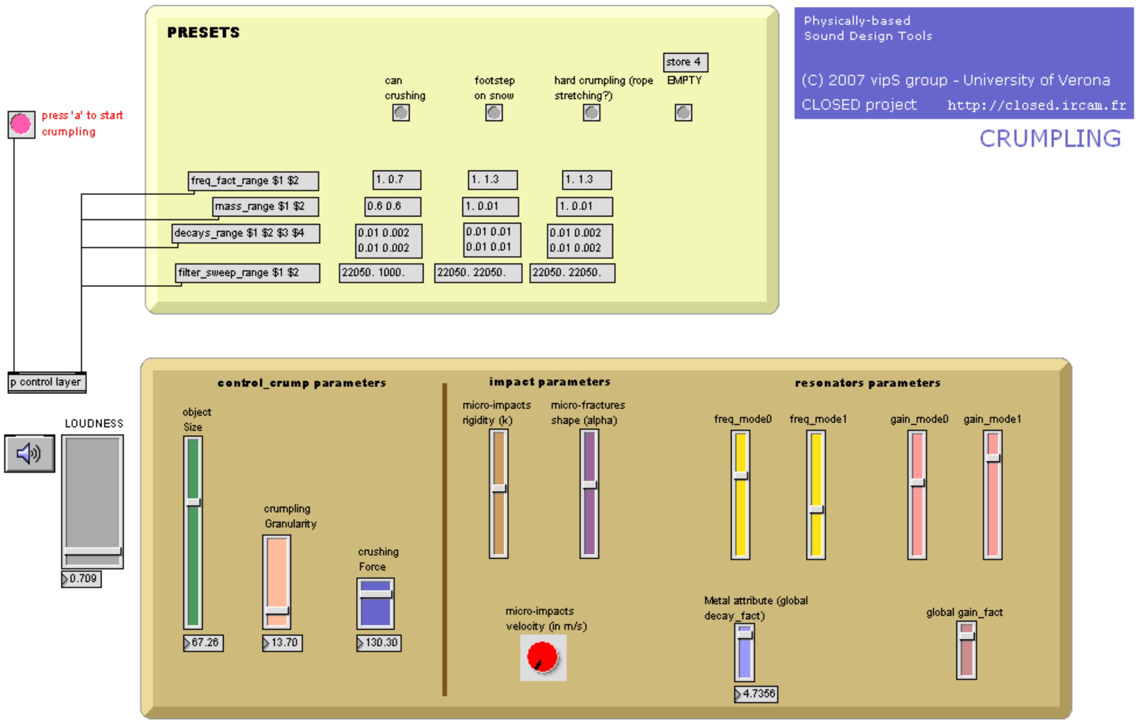

Figure 2.2: Crumpling example patch

The *external* control_crump implements such controls and is expressly meant to drive impact events modeled by two instances of impact_inertialb_modalb∼ *externals* in which both object2s (modal objects) have two resonating modes. Hence it controls two couples of interacting objects (left: object1 and object2, right object1 and object2). In Fig. 2.2 is represented the provided example patch (`CRUMPLING.mxb`) in which control_crump drives the low-level sound synthesis models.

Input

bang	1st inlet. A new atomic crumpling event is triggered.
reset	1st inlet. The crumpling process is reset by restoring initial energy and settings.

float	2nd inlet. Size of the crumpling. Initializes the energy.
float	3rd inlet. Force of the crumpling.
float	4th inlet. Smoothness of the crumpling.
list: [2×**float**]	5th inlet. Sets the range in which the frequency-factors (multiplying coefficients for resonances) of both object2s vary along the duration of crumpling process. The two float arguments are respectively the initial and the final values of the coefficients.
list: [4×**float**]	6th inlet. Sets the range in which the decay times of the two modes of both object2s vary along the duration of crumpling process. The four float arguments are, in order: the initial and the final values of the decay for the first mode of both object2s; the initial and the final values of the decay for the second mode of both object2s.
list: [2×**float**]	7th inlet. Sets the range in which the masses of both object1s vary along the duration of crumpling process. The two float arguments are the initial and the final values of the masses.

Output

Outlets provide controls to two instances of impact_inertialb_modalb∼ *externals*, where both object2s have two resonating modes.

float	1st outlet. Delay value after which to back-bang the control_crump *external* itself by means of a loopback containing a delay object.
float	2nd outlet. Mass of the left object1.
float	3rd outlet. Frequency-factor of the left object2.
float	4th outlet. Decay value of the left object2's first mode.
float	5th outlet. Decay value of the left object2's second mode.
float	6th outlet. Energy of the signal outcoming from the left object2.
float	7th outlet. Mass of the right object1.
float	8th outlet. Frequency-factor of the right object2.
float	9th outlet. Decay value of the right object2's first mode.
float	10th outlet. Decay value of the right object2's second mode.
float	11th outlet. Energy of the signal outcoming from the right object2.

Arguments

All arguments are mandatory.

float	Crumpling size.
float	Crumpling force.
float	Crumpling smoothness.
list: [2×**float**]	Default range (initial and final values) of the frequency-factors of both object2s.
list: [4×**float**]	Default range of the decay times. The four float arguments are, in order: the initial and the final values of the decay for the first mode of both object2s; the initial and the final values of the decay for the second mode of both object2s.
list: [2×**float**]	Default range (initial and final values) of the mass of the both object1s.
list: [2×**float**]	Cutoff frequency sweep range (initial and final values). It is meant to be connected to a lowpass filter with controllable cutoff frequency.

2.2 Liquids

High level models of two classes of complex liquid events are considered: 1. the formation of high quantities of bubbles observed in boiling, frying, streaming or pouring, and 2. the temporal pattern involving initial impacts, principal bubble formation, and secondary droplets, occurring in splash-like events.

Boiling, frying, streaming, pouring

The single resonating bubble model is well suited to serve as the elementary brick to represent populations of bubbles, whose parameters and triggering instants are designed according to given statistical distributions. Typical phenomena in which bubble distributions are observed are boiling or frying liquids, water streaming, pouring of a liquid into a container or into another quiescent liquid, breaking waves.

The *external* bubblestream~ provides controls for bubble triggering frequency, for the mean and variance of radius and radius slope values, for the variance of bubbles amplitude, and for the smoothness of bubbles onset. A sample patch is provided which contains presets for boiling, frying, and running water sounds.

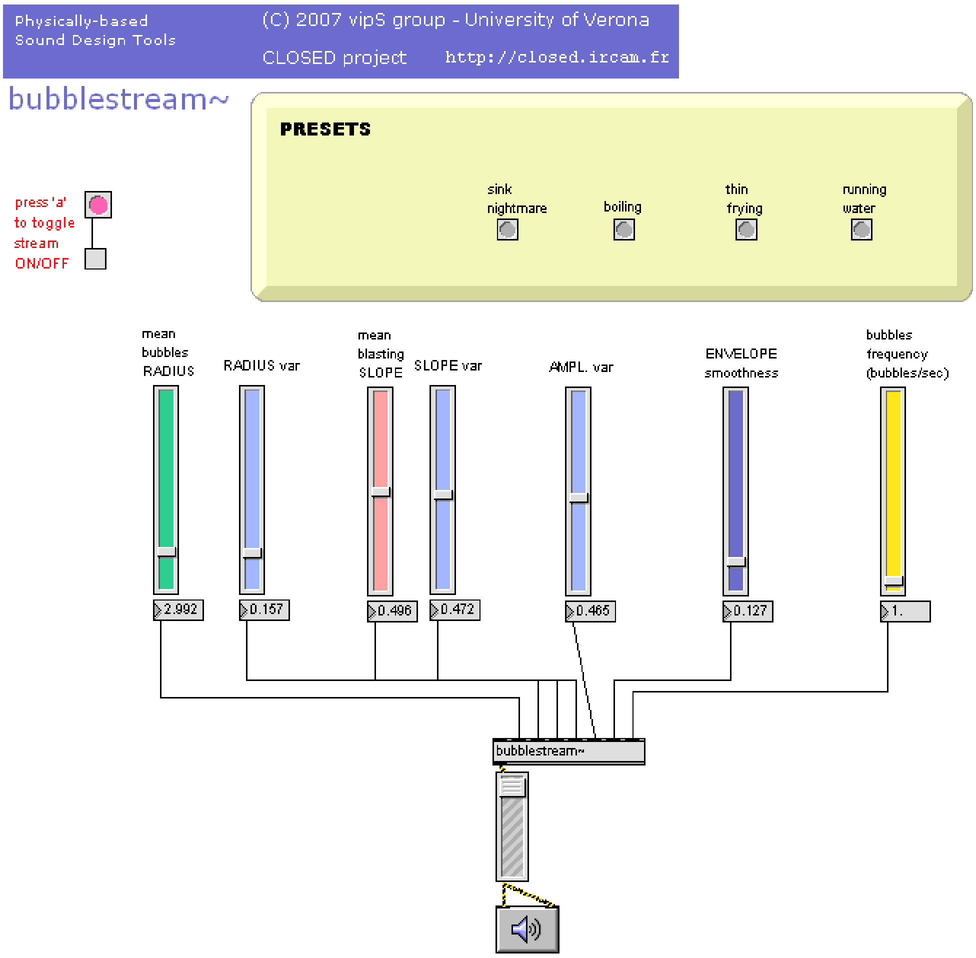

Figure 2.3: Bubblestream patch

2.2.1 `bubblestream~`

Population of bubbles.

Input

bang	1st inlet. Toggles the bubbles flow.
float	2nd inlet. Mean bubbles radius.
float	3rd inlet. Variance of bubbles radius.
float	4th inlet. Mean radius slope.
float	5th inlet. Variance of radius slope.
float	6th inlet. Mean amplitude.
float	7th inlet. Bubbles onset smoothness.
float	8th inlet. Bubbles frequency.

Output

Radiated sound pressure.

Arguments

None.

Dripping, splashing

As an attempt to reproduce the temporal structure of a splashing sound, we choose to design it as a sequence of three distinct low-level events: *1.* a short initial impact sound, *2.* a bubble sound modeled as detailed in the low-level models section, and *3.* a secondary droplets event texture. In the present implementation, sampled waveforms are used to reproduce the initial impact sound and the final sound due to droplets formation, whereas the principal bubble formation is based on the single bubble model.

The *external* splash∼ provides controls for gain balance of the single components, and for the parameters of the principal bubble sound. A sample patch is provided which contains presets for single bubble and complete splashing sounds.

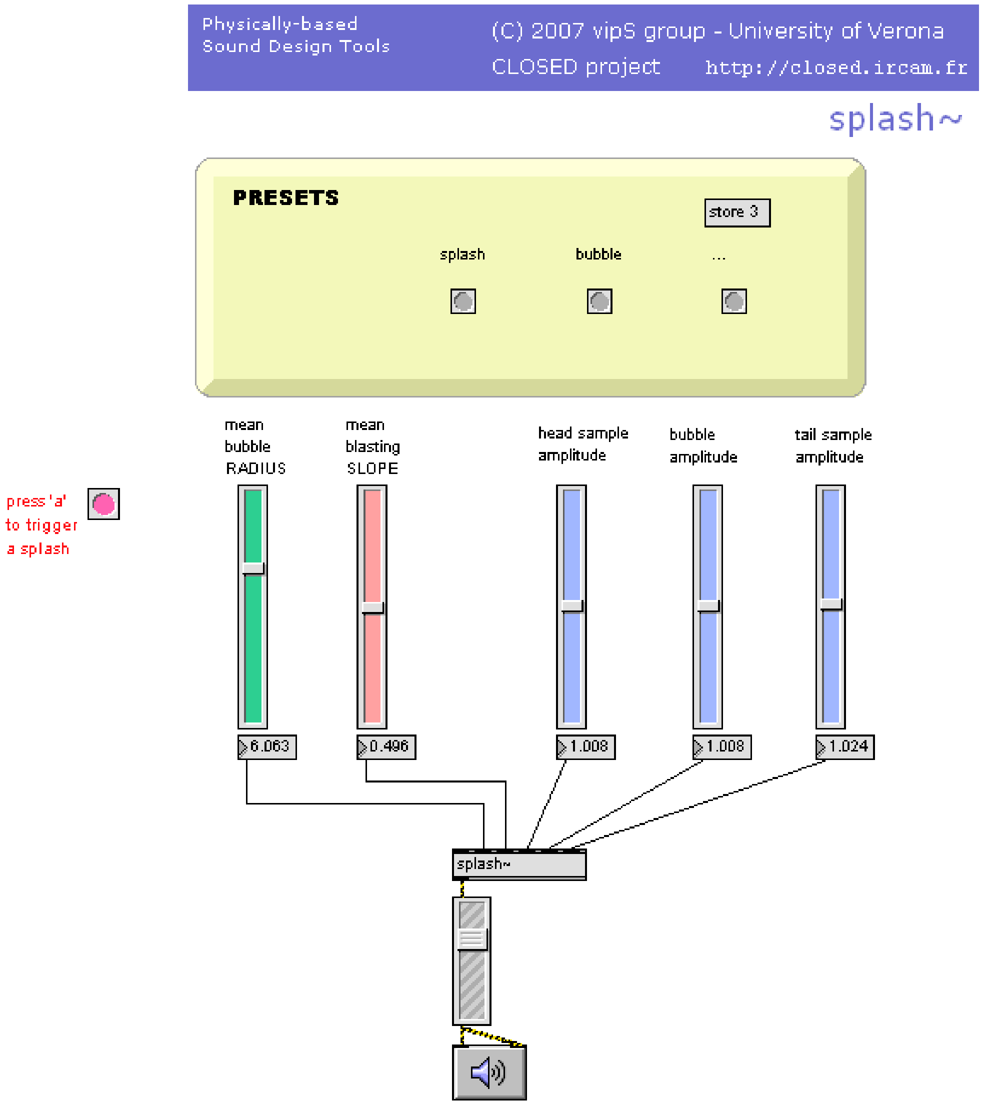

Figure 2.4: Splash patch

2.2.2 `splash~`

Splash or dripping event.

Input

bang	1st inlet. Splash event trigger.
float	2nd inlet. Principal bubble initial radius.
float	2nd inlet. Radius slope.
float	3d inlet. Initial impact event gain.
float	4th inlet. Principal bubble gain.
float	5th inlet. Droplet event gain.

Output

Radiated pressure.

Arguments

None.

GNU Free Documentation License

Version 1.2, November 2002

0. PREAMBLE

The purpose of this License is to make a manual, textbook, or other functional and useful document "free" in the sense of freedom: to assure everyone the effective freedom to copy and redistribute it, with or without modifying it, either commercially or noncommercially. Secondarily, this License preserves for the author and publisher a way to get credit for their work, while not being considered responsible for modifications made by others.

This License is a kind of "copyleft", which means that derivative works of the document must themselves be free in the same sense. It complements the GNU General Public License, which is a copyleft license designed for free software.

We have designed this License in order to use it for manuals for free software, because free software needs free documentation: a free program should come with manuals providing the same freedoms that the software does. But this License is not limited to software manuals; it can be used for any textual work, regardless of subject matter or whether it is published as a printed book. We recommend this License principally for works whose purpose is instruction or reference.

1. APPLICABILITY AND DEFINITIONS

This License applies to any manual or other work, in any medium, that contains a notice placed by the copyright holder saying it can be distributed under the terms of this License. Such a notice grants a world-wide, royalty-free license, unlimited in duration, to use that work under the conditions stated herein. The "Document", below, refers to any such manual or work. Any member of the public is a licensee, and is addressed as "you". You accept the license if you copy, modify or distribute the work in a way requiring permission under copyright law.

A "Modified Version" of the Document means any work containing the Document or a portion of it, either copied verbatim, or with modifications and/or translated into another language.

A "Secondary Section" is a named appendix or a front-matter section of the Document that deals exclusively with the relationship of the publishers or authors of the Document to the Document's overall subject (or to related matters) and contains nothing that could fall directly within that overall subject. (Thus, if the Document is in part a textbook of mathematics, a Secondary Section may not explain any mathematics.) The relationship could be a matter of historical connection with the subject or with related matters, or of legal, commercial, philosophical, ethical or political position regarding them.

The "Invariant Sections" are certain Secondary Sections whose titles are designated, as being those of Invariant Sections, in the notice that says that the Document is released under this License. If a section does not fit the above definition of Secondary then it is not allowed to be designated as Invariant. The Document may contain zero Invariant Sections. If the Document does not identify any Invariant Sections then there are none.

The "Cover Texts" are certain short passages of text that are listed, as Front-Cover Texts or Back-Cover Texts, in the notice that says that the Document is released under this License. A Front-Cover Text may be at most 5 words, and a Back-Cover Text may be at most 25 words.

A "Transparent" copy of the Document means a machine-readable copy, represented in a format whose specification is available to the general public, that is suitable for revising the document straightforwardly with generic text editors or (for images composed of pixels) generic paint programs or (for drawings) some widely available drawing editor, and that is suitable for input to text formatters or for automatic translation to a variety of formats suitable for input to text formatters. A copy made in an otherwise Transparent file format whose markup, or absence of markup, has been arranged to thwart or discourage subsequent modification by readers is not Transparent. An image format is not Transparent if used for any substantial amount of text. A copy that is not "Transparent" is called "Opaque".

Examples of suitable formats for Transparent copies include plain ASCII without markup, Texinfo input format, LaTeX input format, SGML or XML using a publicly available DTD, and standard-conforming simple HTML, PostScript or PDF designed for human modification. Examples of transparent image formats include PNG, XCF and JPG. Opaque formats include proprietary formats that can be read and edited only by proprietary word processors, SGML or XML for which the DTD and/or processing tools are not generally available, and the machine-generated HTML, PostScript or PDF produced by some word processors for output purposes only.

The "Title Page" means, for a printed book, the title page itself, plus such following pages as are needed to hold, legibly, the material this License requires to appear in the title page. For works in formats which do not have any title page as such, "Title Page" means the text near the most prominent appearance of the work's title, preceding the beginning of the body of the text.

A section "Entitled XYZ" means a named subunit of the Document whose title either is precisely XYZ or contains XYZ in parentheses following text that translates XYZ in another language. (Here XYZ stands for a specific section name mentioned below, such as "Acknowledgements", "Dedications", "Endorsements", or "History".) To "Preserve the Title" of such a section when you modify the Document means that it remains a section "Entitled XYZ" according to this definition.

The Document may include Warranty Disclaimers next to the notice which states that this License applies to the Document. These Warranty Disclaimers are considered to be included by reference in this License, but only as regards disclaiming warranties: any other implication that these Warranty Disclaimers may have is void and has no effect on the meaning of this License.

2. VERBATIM COPYING

You may copy and distribute the Document in any medium, either commercially or noncommercially, provided that this License, the copyright notices, and the license notice saying this License applies to the Document are reproduced in all copies, and that you add no other conditions whatsoever to those of this License. You may not use technical measures to obstruct or control the reading or further copying of the copies you make or distribute. However, you may accept compensation in exchange for copies. If you distribute a large enough number of copies you must also follow the conditions in section 3.

You may also lend copies, under the same conditions stated above, and you may publicly display copies.

3. COPYING IN QUANTITY

If you publish printed copies (or copies in media that commonly have printed covers) of the Document, numbering more than 100, and the Document's license notice requires Cover Texts, you must enclose the copies in covers that carry, clearly and legibly, all these Cover Texts: Front-Cover Texts on the front cover, and Back-Cover Texts on the back cover. Both covers must also clearly and legibly identify you as the publisher of these copies. The front cover must present the full title with all words of the title equally prominent and visible. You may add other material on the covers in addition. Copying with changes limited to the covers, as long as they preserve the title of the Document and satisfy these conditions, can be treated as verbatim copying in other respects.

If the required texts for either cover are too voluminous to fit legibly, you should put the first ones listed (as many as fit reasonably) on the actual cover, and continue the rest onto adjacent pages.

If you publish or distribute Opaque copies of the Document numbering more than 100, you must either include a machine-readable Transparent copy along with each Opaque copy, or state in or with each Opaque copy a computer-network location from which the general network-using public has access to download using public-standard network protocols a complete Transparent copy of the Document, free of added material. If you use the latter option, you must take reasonably prudent steps, when you begin distribution of Opaque copies in quantity, to ensure that this Transparent copy will remain thus accessible at the stated location until at least one year after the last time you distribute an Opaque copy (directly or through your agents or retailers) of that edition to the public.

It is requested, but not required, that you contact the authors of the Document well before redistributing any large number of copies, to give them a chance to provide you with an updated version of the Document.

4. MODIFICATIONS

You may copy and distribute a Modified Version of the Document under the conditions of sections 2 and 3 above, provided that you release the Modified Version under precisely this License, with the Modified Version filling the role of the Document, thus licensing distribution and modification of the Modified Version to whoever possesses a copy of it. In addition, you must do these things in the Modified Version:

A. Use in the Title Page (and on the covers, if any) a title distinct from that of the Document, and from those of previous versions (which should, if there were any, be listed in the History section of the Document). You may use the same title as a previous version if the original publisher of that version gives permission.

B. List on the Title Page, as authors, one or more persons or entities responsible for authorship of the modifications in the Modified Version, together with at least five of the principal authors of the Document (all of its principal authors, if it has fewer than five), unless they release you from this requirement.

C. State on the Title page the name of the publisher of the Modified Version, as the publisher.

D. Preserve all the copyright notices of the Document.

E. Add an appropriate copyright notice for your modifications adjacent to the other copyright notices.

F. Include, immediately after the copyright notices, a license notice giving the public permission to use the Modified Version under the terms of this License, in the form shown in the Addendum below.

G. Preserve in that license notice the full lists of Invariant Sections and required Cover Texts given in the Document's license notice.

H. Include an unaltered copy of this License.

I. Preserve the section Entitled "History", Preserve its Title, and add to it an item stating at least the title, year, new authors, and publisher of the Modified Version as given on the Title

Page. If there is no section Entitled "History" in the Document, create one stating the title, year, authors, and publisher of the Document as given on its Title Page, then add an item describing the Modified Version as stated in the previous sentence.

J. Preserve the network location, if any, given in the Document for public access to a Transparent copy of the Document, and likewise the network locations given in the Document for previous versions it was based on. These may be placed in the "History" section. You may omit a network location for a work that was published at least four years before the Document itself, or if the original publisher of the version it refers to gives permission.

K. For any section Entitled "Acknowledgements" or "Dedications", Preserve the Title of the section, and preserve in the section all the substance and tone of each of the contributor acknowledgements and/or dedications given therein.

L. Preserve all the Invariant Sections of the Document, unaltered in their text and in their titles. Section numbers or the equivalent are not considered part of the section titles.

M. Delete any section Entitled "Endorsements". Such a section may not be included in the Modified Version.

N. Do not retitle any existing section to be Entitled "Endorsements" or to conflict in title with any Invariant Section.

O. Preserve any Warranty Disclaimers.

If the Modified Version includes new front-matter sections or appendices that qualify as Secondary Sections and contain no material copied from the Document, you may at your option designate some or all of these sections as invariant. To do this, add their titles to the list of Invariant Sections in the Modified Version's license notice. These titles must be distinct from any other section titles.

You may add a section Entitled "Endorsements", provided it contains nothing but endorsements of your Modified Version by various parties–for example, statements of peer review or that the text has been approved by an organization as the authoritative definition of a standard.

You may add a passage of up to five words as a Front-Cover Text, and a passage of up to 25 words as a Back-Cover Text, to the end of the list of Cover Texts in the Modified Version. Only one passage of Front-Cover Text and one of Back-Cover Text may be added by (or through arrangements made by) any one entity. If the Document already includes a cover text for the same cover, previously added by you or by arrangement made by the same entity you are acting on behalf of, you may not add another; but you may replace the old one, on explicit permission from the previous publisher that added the old one.

The author(s) and publisher(s) of the Document do not by this License give permission to use their names for publicity for or to assert or imply endorsement of any Modified Version.

5. COMBINING DOCUMENTS

You may combine the Document with other documents released under this License, under the terms defined in section 4 above for modified versions, provided that you include in the combination all of the Invariant Sections of all of the original documents, unmodified, and list them all as Invariant Sections of your combined work in its license notice, and that you preserve all their Warranty Disclaimers.

The combined work need only contain one copy of this License, and multiple identical Invariant Sections may be replaced with a single copy. If there are multiple Invariant Sections with the same name but different contents, make the title of each such section unique by adding at the end of it, in parentheses, the name of the original author or publisher of that section if known, or else a unique number. Make the same adjustment to the section titles in the list of Invariant Sections in the license notice of the combined work.

In the combination, you must combine any sections Entitled "History" in the various original documents, forming one section Entitled "History"; likewise combine any sections Entitled "Acknowledgements", and any sections Entitled "Dedications". You must delete all sections Entitled "Endorsements".

6. COLLECTIONS OF DOCUMENTS

You may make a collection consisting of the Document and other documents released under this License, and replace the individual copies of this License in the various documents with a single copy that is included in the collection, provided that you follow the rules of this License for verbatim copying of each of the documents in all other respects.

You may extract a single document from such a collection, and distribute it individually under this License, provided you insert a copy of this License into the extracted document, and follow this License in all other respects regarding verbatim copying of that document.

7. AGGREGATION WITH INDEPENDENT WORKS

A compilation of the Document or its derivatives with other separate and independent documents or works, in or on a volume of a storage or distribution medium, is called an "aggregate" if the copyright resulting from the compilation is not used to limit the legal rights of the compilation's users beyond what the individual works permit. When the Document is included in an aggregate, this License does not apply to the other works in the aggregate which are not themselves derivative works of the Document.

If the Cover Text requirement of section 3 is applicable to these copies of the Document, then if the Document is less than one half of the entire aggregate, the Document's Cover Texts may be placed on covers that bracket the Document within the aggregate, or the electronic equivalent of covers if the Document is in electronic form. Otherwise they must appear on printed covers that bracket the whole aggregate.

8. TRANSLATION

Translation is considered a kind of modification, so you may distribute translations of the Document under the terms of section 4. Replacing Invariant Sections with translations requires special permission from their copyright holders, but you may include translations of some or all Invariant Sections in addition to the original versions of these Invariant Sections. You may include a translation of this License, and all the license notices in the Document, and any Warranty Disclaimers, provided that you also include the original English version of this License and the original versions of those notices and disclaimers. In case of a disagreement between the translation and the original version of this License or a notice or disclaimer, the original version will prevail.

If a section in the Document is Entitled "Acknowledgements", "Dedications", or "History", the requirement (section 4) to Preserve its Title (section 1) will typically require changing the actual title.

9. TERMINATION

You may not copy, modify, sublicense, or distribute the Document except as expressly provided for under this License. Any other attempt to copy, modify, sublicense or distribute the Document 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.

10. FUTURE REVISIONS OF THIS LICENSE

The Free Software Foundation may publish new, revised versions of the GNU Free Documentation 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. See http://www.gnu.org/copyleft/.

Each version of the License is given a distinguishing version number. If the Document specifies that a particular numbered version of this License "or any later version" applies to it, you have the option of following the terms and conditions either of that specified version or of any later version that has been published (not as a draft) by the Free Software Foundation. If the Document does not specify a version number of this License, you may choose any version ever published (not as a draft) by the Free Software Foundation.

ADDENDUM: How to use this License for your documents

To use this License in a document you have written, include a copy of the License in the document and put the following copyright and license notices just after the title page:

> Copyright (c) YEAR YOUR NAME. Permission is granted to copy, distribute and/or modify this document under the terms of the GNU Free Documentation License, Version 1.2 or any later version published by the Free Software Foundation; with no Invariant Sections, no Front-Cover Texts, and no Back-Cover Texts. A copy of the license is included in the section entitled "GNU Free Documentation License".

If you have Invariant Sections, Front-Cover Texts and Back-Cover Texts, replace the "with...Texts." line with this:

> with the Invariant Sections being LIST THEIR TITLES, with the Front-Cover Texts being LIST, and with the Back-Cover Texts being LIST.

If you have Invariant Sections without Cover Texts, or some other combination of the three, merge those two alternatives to suit the situation.

If your document contains nontrivial examples of program code, we recommend releasing these examples in parallel under your choice of free software license, such as the GNU General Public License, to permit their use in free software.

References

[1] N. Misdariis, R. Caussé, L. Dandrel, J. Bensoam, and C. Vergez. Modalys, un outil pour le design sonore. In *1ères Journées du Design Sonore*, Paris, 2002.

[2] S. Papetti. Sintesi audio in tempo-reale mediante modelli waveguide di risonatori e modelli non-lineari di contatto. Master's Thesis in Computer Engineering, University of Padua, A.A. 2004/2005.

[3] D. Rocchesso and F. Fontana, editors. *The Sounding Object*. Mondo Estremo, 2003. Freely available from `http://www.soundobject.org/`.

[4] J. O. Smith. Principles of digital waveguide models of musical instruments. In M. Kahrs and K. Brandeburg, editors, *Applications of DSP to Audio and Acoustic*, pages 417–466. Kluwer Academic Publishers, 2002.

[5] K. van den Doel. Physically-based models for liquid sounds. In *Proc. ICAD 04*, Sydney, July 2004.