



AUDIENCE and OpenAUDIENCE for Pd MANUAL

Manual version 1.1.4 (Aug.2010)

AUDIENCE version 2 .0.1 (Aug.2010)

OpenAUDIENCE version 1.0.1 (Aug.2010)



INDEX

Organization of this manual	1
INTRODUCTION	2
What is AUDIENCE and OpenAUDIENCE?	2
AUDIENCE – The architecture	4
How the system architecture is organized?	4
The functional layers	4
Further information on the system architecture	6
AUDIENCE – The software	7
How does the AUDIENCE software work?	7
Functional blocks and interfaces	9
Messages	15
Help files	17
INSTALLATION	18
Installing in all OS	18
USAGE	22
What can be done?	22
How to start using the software	22
Tutorials	23
Troubleshooting	23
DEVELOPING (how to create blocks in AUDIENCE for Pd)	25
Building an AUDIENCE block in Pd	25
Naming convention	25
Design and implementation steps	25
Creating an external Pd block	26
How to create a project for a new Pd object	26



How to make a Pd external	27
Compiling and building-up OA / AUDIENCE externals.....	30
Testing built AUDIENCE and Pd blocks	30
RELEASE NOTES	31
Licenses	31
ANNEXES	33
A. Instalando o ambiente de desenvolvimento Eclipse/CDT/MinGW	33
1. Instalando o Eclipse.....	33
2. Instalando o MinGW	33
3. Baixando e instalando o CDT (plugin C++)	33
4. Testando o ambiente.	34



This guide is in constant update. Consult the latest version and date of the newest release.

Document history:

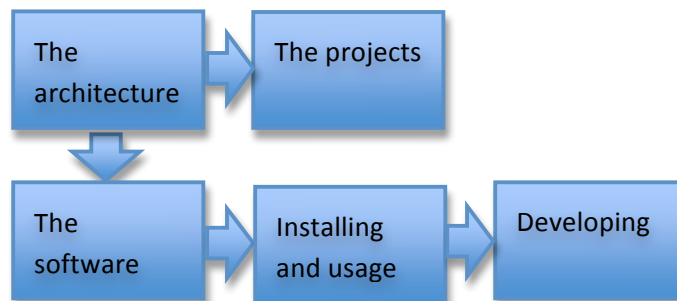
22/08/2010 – Release v.1.1.4 of the manual, updating text revised
13/08/2010 – Release v.1.1.3 of the manual, updating installation, development notes and new text review
29/07/2010 – Release v.1.1.2 of the manual, updating info about Audience architecture and images
20/07/2010 – Release v.1.1 of the manual, featuring tables, text and figures updates and English revisions
27/04/2010 – Release v.1.0 of the manual for integrated Audience and OpenAudience for Pd (Regis Faria)
04/04/2008 – Updated block diagram figure (Regis Faria)
04/09/2007 – Updated structure, added tests sections and text about the software (Team)
15/05/2007 – Updated external creation guidelines (Leandro F. Thomaz)
09/03/2007 – First “How-to File” creation (Team)

See **RELEASE NOTES** for software versions.

Copyright © 2010 – AUDIENCE Team

Organization of this manual

This manual is organized in sections with information flowing in this way:



A reader interested in **immediate installing** the software may go directly to the respective section without loss. **To use the software** however the reader **must understand** the concept of **patching blocks** and **should** understand the underlying **architecture of the system**, as this is the safest way to rapidly learning how to find the right tools to implement desired functions and to build applications.

The reader interested in **being a contributor** in the developing projects shall read the introduction describing the projects and their references.

The reader that wants to contribute **developing new processing objects** and functions in the system must read the corresponding section.

This manual version is **preliminary**. Some information is pending to document and there may be few errors and outdated information on it. If you need further information or want to send feedbacks, including submit problems and faults in the manual, please write to the email audience@lsi.usp.br.

Good reading and usage of the AUDIENCE system!

The team, August 2010.

INTRODUCTION

What is AUDIENCE and OpenAUDIENCE?

AUDIENCE is at the same time the name of a *system architecture* proposed in 2005 for *spatial audio* and the name of a *software library* implementing key functionalities for the chain of spatial/multichannel audio production, distribution and reproduction based on this architecture.

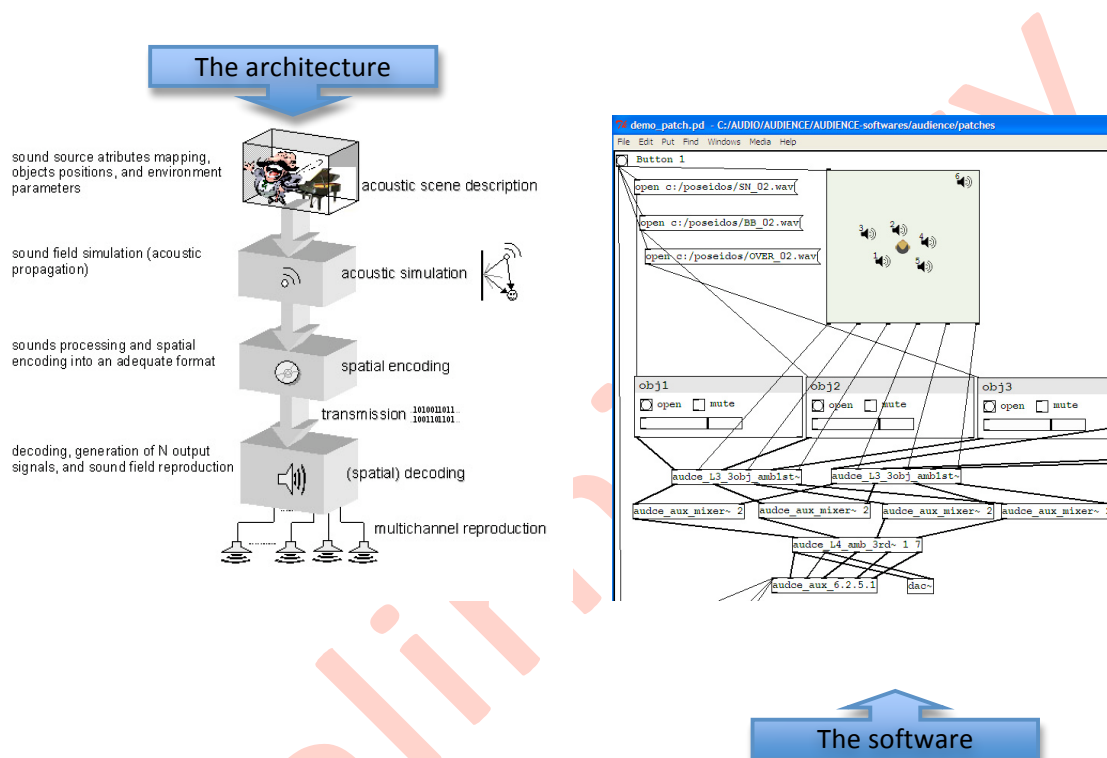


Figure 1 – The architecture and the software.

The *AUDIENCE for Pd* (*Audience4Pd*) is a software library made using Pure Data (Pd¹) objects, patches and *externals*². The software development started in 2005 and, together with some customized hardware for multichannel audio cabling and multichannel amplifier designs, constitute the main results of the namesake project “*AUDIENCE – AUDio Immersion ExperieNce by Computer Emulation*” developed from 2003 to 2007 in the University of Sao Paulo. The main objectives were the development of a software adherent to the architecture and released as a library for Pd. Many contributors have been involved in software development, testing concepts, designs and applications since then.

¹ Pure Data is a graphical programming environment for audio and video. The software platform is written in C, follows a BSD-like license, and is the substrate platform on top of which *AUDIENCE4Pd* is built. Plenty information about it can be found at its main website at puredata.info.

² An external in Pd’s terminology is a specific purpose object built from a C/C++ code.

The AUDIENCE project is documented online at www.lsi.usp.br/interativos/neac/audience/. Published papers about both the system architecture and the software are referenced in the website.

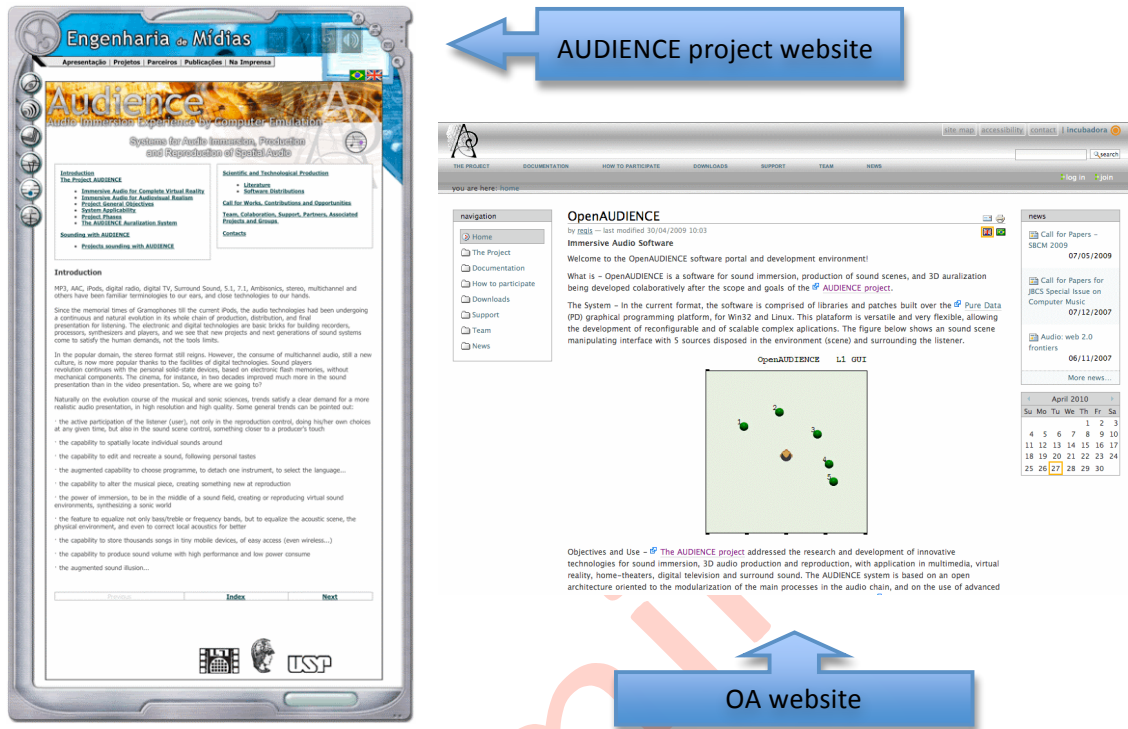


Figure 2 – Websites pics.

The OpenAUDIENCE (OA) is a free/open distribution of this software library, based on the full version. The main difference between them is the existence of *non-free objects* and *proprietary objects and libraries* included in the full Audience4Pd version. The OA can be downloaded freely from the website <http://openaudience.incubadora.fapesp.br>. The project in which this version is developed started in late 2006 and is open to any contributors from both technical and artistic/musical backgrounds willing to develop applications and new features.

The full-version AUDIENCE is available to the team of developers and researchers involved with the AUDIENCE development projects. It is also available for contractors and companies willing to use it as spatial audio engines in specialized audio systems and product designs. Interested parties must get in touch with the team in charge of the project for any prospective agreements, sending email to audience@lsi.usp.br.

AUDIENCE – The architecture

How the system architecture is organized?

The AUDIENCE system architecture is based on the concept of *decoupled functional layers*. Simplicity, layers and interfaces are keywords to understand this architecture.

The first proposal of the AUDIENCE architecture was published in 2005, and since then it has been detailed and further consolidated in later published papers. Many practical applications were valuable for validating the concepts and distinguishing features.

In the next section a practical overview of the architecture is given, explaining its *core*, introducing *features* and key *aspects of the interfaces* between layers. The section includes a discussion on the hypothesis supporting the architecture and the novelties it introduces when compared to existing approaches. The reader should have a basic knowledge on engineering and software terminologies to fully understand all the concepts.

The functional layers

The core of the architecture is on the *functional layers*, which can be viewed as *clusters of related functions*. There are *four main audio processing layers* in the AUDIENCE architecture, which are mandatory for any spatial production-reproduction chain. They are:

- scene composing and description layer, resolving the auditory scene (the layer 1 or just “L1”)
- scene acoustic model rendering or simulation, using some acoustics method (the layer L2)
- spatial audio coding, packaging audio data (with metadata or not) into a specific format for distribution or storage (the layer L3)
- spatial audio reproduction, decoding and generating multichannel output for the final audition configuration or surround loudspeaker mode (the layer L4).

These *four layers* constitute *the core* of the architecture.

Besides the functions within the main 4 layers there are also other functions that fit in what can be called the *auxiliary layers* (or *service layers*), implementing functionalities which are not directly connected to the spatial audio processing tasks, but are required to link the core to running applications and devices. Usually they are platform-dependent and do not interfere in the subject dealt by the main audio processing chain. They are:

- *communication interfaces*, implementing communication mechanisms and input/output (I/O) ports for transmission and reception of all sort of data to/from the main layers. It is responsible for providing audio data and also metadata input/output to/from the main core.

- *user interface/control*, implementing the GUI³ blocks that control the main core. It is responsible for admitting control and interactive actions from the user/listener and sending commands to the main 4 layers. Usually if GUI blocks embed L1 functionalities⁴ they will however get a L1 status.

Figure 3 below shows the architecture's hierarchy of layers.

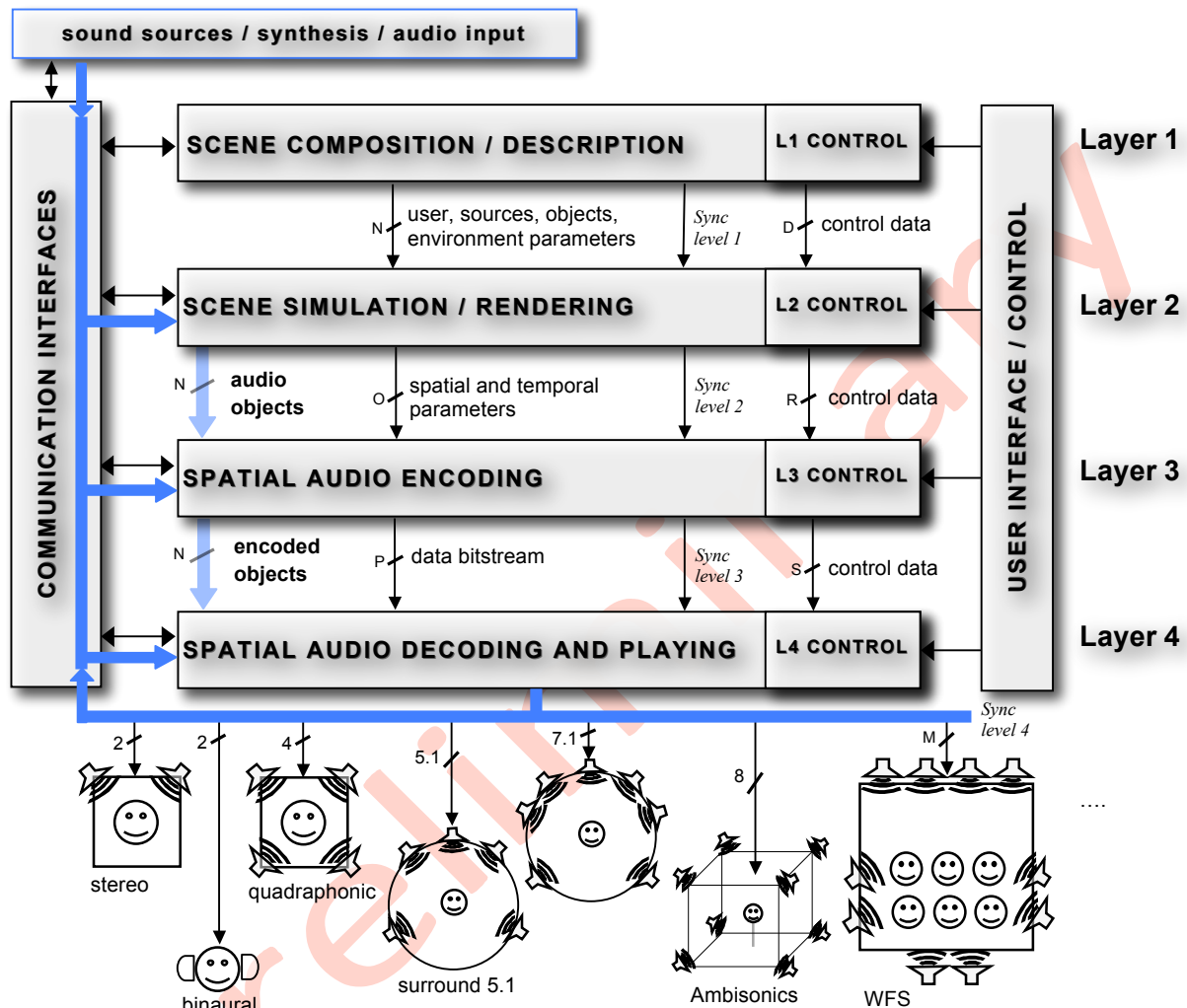


Figure 3 – AUDIENCE's layer architecture (updated aug/2010)

Comments:

Layers hierarchy. Notice that the architecture proposes a hierarchy of layers, where L1 is highest semantic level and L4 is the lowest one. Synchronization and other data rate increase from L1 to L4.

³ Graphical User Interface.

⁴ such as permitting the localization of sound objects in a scene and exporting this information.

Signals in the interfaces (metadata, audio data, sync signals, control data). Observe that there are different classes of data in the interface between layers, as indicated by the figure 3.

Top-down data flow arrangement. Figure 3 clearly suggests a main direction for the data flow (top-down). Inside a basic chain the data flows from layer i to layer $i+1$. In several circumstances one chain must feed input to another chain and then it happens that a layer i will feed a layer j , where $i, j \in \{1, 2, 3, 4\}$. The sequence encompasses functions done in different stages and these data naturally include audio payload, metadata, control parameters and sync signals.

The concept of layer control. As layers must be independent from each other, every layer must have a self-contained and sufficient set of working rules and algorithms to control its behavior, e.g. algorithms to act in case of audio clipping, invalid data range, etc.

Inputs coming from GUI provide control to all layers. GUI's can deliver specific commands or instructions to control functions and behaviors in all layers.

Layers exchanging data through the communication layer. All layers functions can deliver messages, data and instructions to other layers. This is done using some sort of communication interface, which is shown in a generalized form in the figure 3.

Audio data bus showing possible routes. Audio routes are shown in blue color in the figure 3. It is detached that audio data can flow directly from L2 to L3 and from L3 to L4. Audio data flowing back from L4 to other layers is indicated by the feedback of audio from L4 reinserted into the audio interfaces.

Further information on the system architecture

More information on the following items about the architecture will be complemented in a series of paper to published soon:

- basic processes and data flows
- applications build-up
- advanced features of the architecture (such as semantic levels and rate of information across layers)
- hypothesis supporting the 4-layer approach and innovative aspects of the architecture

AUDIENCE – The software

The *AUDIENCE for Pd* (*Audience4Pd*) is a *collection of functional components* in the form of Pd objects and patches⁵. In other words, the software is a library tool for Pd. A functional component in this library is simply a *processing block* built as a self-contained Pd *object* or a *patch*.

The blocks are sorted in 4 functional layers plus an auxiliary one, running on top of Pd. The figure 4 shows the software stack.

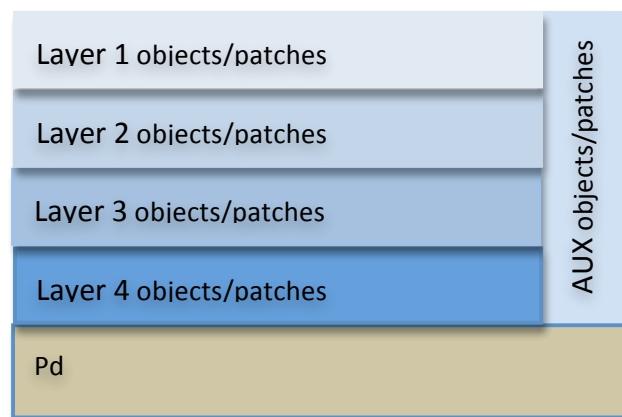


Figure 4 – Software stack. All objects are instantiated as Pd blocks.

The software library is a direct result of the development projects. In the first versions it worked with 2, 4, 6 and 8 loudspeakers setups running Ambisonics and incorporated a simple GUI for L1 scene creation and control. The perceptual spatial impressions were quite good for open fields virtual worlds and got improved as layer-2 processes were implemented⁶, such as an image-source L2-algorithm. Progressively new methods and techniques were incorporated and presently the full version works with Ambisonics (up to 3rd order), multichannel PCM, MPEG-4 AAC, MPEG Surround, and further blocks are on the way.

This sections describes the contents of the library and how it works.

How does the AUDIENCE software work?

A practical spatial audio application is built *creating data flows* by *instantiating* and *connecting blocks*, as shown in figure 1 below. Blocks can be created inside a Pd patch and connected to other objects, so to execute desired tasks, such as creating and manipulating sound scenes with sound objects inside, simulating acoustics, encoding the results to a certain distribution format, or simply playing a rendered spatial sound scene in a local loudspeaker array.

⁵ And so it is OpenAUDIENCE (OA) then.

⁶ For subjective testes please refer to the published papers.

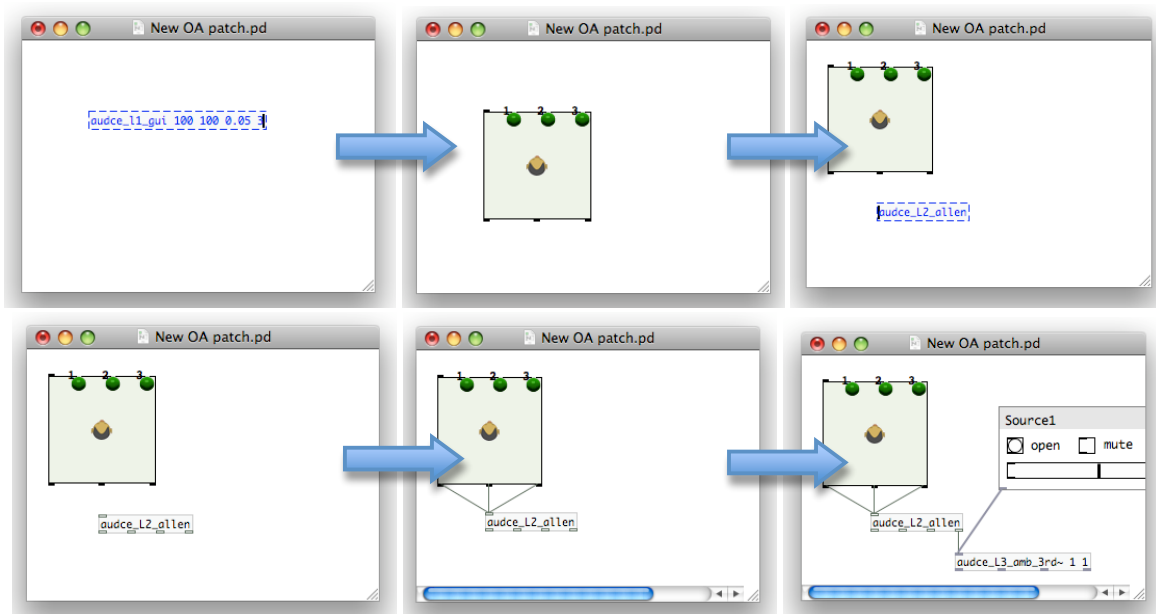


Figure 5 – Patching in Pd. Objects are instantiated and connected establishing a data flow

Many pipelines of processes can be created in parallel and real-time operation may be established. As long as a data path exists, live processes can work around the clock, continuously executing, exactly as spatial processes occur in the nature. For example, as long as a reverberant chamber exists it will always process the incoming sounds. Similarly, processes can be roughly terminated by simply dismounting the chain, disconnecting the objects and destroying them.

This intuitive process of creating and destroying applications is a powerful and typical feature of the Pd and other similar object-oriented graphical programming platforms, and is not a novel approach in computing.

In the OA/AUDIENCE software each functional component – or simply a “block” – is designed and built following the design rules of the AUDIENCE system architecture. This mandates that *one block must belong to one and only one functional layer* of the architecture. A real application may have several blocks from several functional layers, executing a bunch of both parallel and serial processes.

In the next section we will see what is a *functional layer* and how the system architecture is devised.

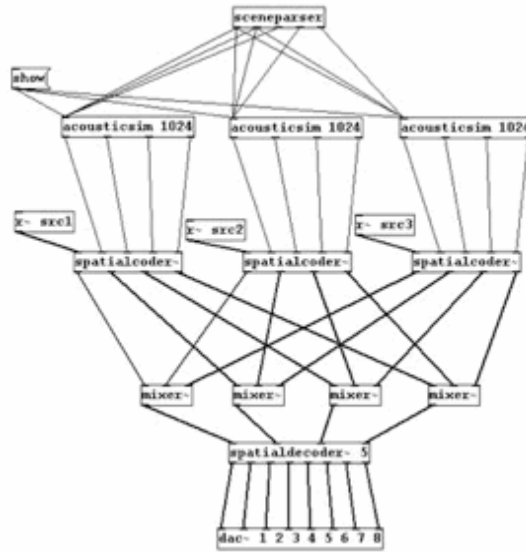


Figure 6 – An example of a complete top-down data flow. A L1 block (top) feeds three L2-L3 parallel chains, one per sound object in scene. A following auxiliary layer combines the encoded sounds and feeds a L4 decoder (bottom).

Functional blocks and interfaces

Functional blocks of the software executing the core roles will be implemented into and tagged with a corresponding *main layer* (i.e. layer 1, 2, 3 or 4). All other functions in the software are collectively tagged and grouped into the *auxiliary layer*. This includes many glue functions and supportive blocks, such as data formatting, arithmetic and DSP operations (taken for granted in the Pd substrate), mixing operations, buttons and sliders, timing and scheduling, synchronization, start/stop/turn on and off signals, among others. These are important to build patches and to set and to control parameters. All available Pd libraries are considered auxiliary functions in this sense.

The table 1 below lists the functional blocks implemented in the current OA/AUDIENCE for Pd. It is the most complete source information about the blocks in this manual. For each block it is presented:

- Layer group (e.g. L1, L2, L3, L4 or Auxiliary)
- Name of the block (object name in Pd)
- Arguments (the creation arguments to create the pd object)
- Data it holds (e.g. sound object position in space, order of Ambisonics, etc.)
- Input and Output (the interface signals, i.e. the valid signals that are accepted in inlets and signals available in the outlets)
- Status (the development status, e.g. stable, in test, frozen, etc.)
- Usage and short description

Table 1 – Available functional blocks ⁷

Layer	Block (object in Pd)	Arguments	Data it holds	Input	Output	Status	Usage / Short description
L1	audce_L1_gui	grid width, grid height, pixel ration, no. of sources			receiver & sources positions	stable	general user interface for L1 functions
L2	audce_L2_allen	no. points of impulse response		receiver & source position, room dimensions, walls absorp. coef.	Ambisonic set of impulse responses	stable	Allen's algorithm adapted to simulate a box room acoustic and generate an Ambisonic set of impulse responses
L3	audce_L3_amb_3rd~					stable	Ambisonics encoder (up to 3 rd order)
L4	audce_L4_amb_3rd~					stable	Ambisonics decoder (up to 3 rd order)
Aux	audce_aux_objplay~					stable	sound object player (file+transport)

⁷ Functional blocks updated from version 19/01/07.

	audce_aux_objplay1~					stable	sound object player (file+ transport+ time)
	audce_aux_objplay2~					stable	sound object player (file + transport+ time+ volume)
	audce_aux_mixer					stable	audio signals mixer
	audce_aux_move					stable	moves dynamically an object following a given path/trajectory

preliminary

Table 2 lists blocks that belong to one of these categories: obsolete function, discontinued block, frozen development, abandoned development, development not-started, expected development, and incomplete development.

Table 2 – Obsolete, Discontinued, Frozen, Abandoned, Not started, Under design, Expected, and Incomplete blocks

Layer	Block (object in Pd)	Arguments	Data it holds	Input	Output	Status	Usage and short description
L1	audce_L1_control	t.b.d	t.b.d	init state	t.b.d ⁸	frozen	layer 1 controller
L1	audce_L1_data_transmit	-	-	-	-	abandoned	to parse, store, transmit scene data to other blocks. Reason: specific needs require specific objects for this.
L1	audce_L1_test	t.b.d	t.b.d.	usr request	t.b.d.	expected	test scripts for L1
L1	audce_L1_error_mng	t.b.d	t.b.d	scene data	t.b.d	expected	error management for L1
L1	audce_L1_room	-	-	-	-	abandoned	box storing environment data; reason: L1 scene block must hold it
L1	audce_L1_usr	-	-	-	-	abandoned	reason: box storing listener data; reason: L1 scene block must hold it
L1	audce_L1_src	-	-	-	-	abandoned	box storing sound source data; reason: scene block must hold it
L1	audce_L1_obstruction	-	-	-	-	abandoned	scene block must control obstructions

⁸ To Be Defined

L2	audce_L2_control	t.b.d	t.b.d	init state	t.b.d	frozen	layer 2 controller
L2	audce_L2_acousticsim					obsolete	acoustic simulator (based on Allen's algorithm)
L2	audce_L2_obstruction					frozen	acoustic obstruction solver
L2	audce_L2_test					expected	test scripts for L2
L2	audce_L2_error_mng					expected	error management for L2
L3	audce_L3_context_control					frozen	layer 3 controller
L3	audce_L3_ambcoder					obsolete	Ambisonics encoder
L3	audce_L3_test					expected	test scripts for L3
L3	audce_L3_error_mng					expected	error management for L3
L4	audce_L4_context_control					frozen	layer 4 controller
L4	audce_L4_ambdecoder					obsolete	Ambisonics decoder
L4	audce_L4_dereverb					abandoned	de-reverberating
L4	audce_L4_test					expected	test scripts for L4
L4	audce_L4_error_mng					expected	error management for L4
aux	audce_aux_error_mng					expected	error management for aux
aux	audce_aux_calibrate					expected	loudspeaker configuration calibration

Aux	audce_aux_timer					abandoned	timer for process control in general
Aux	audce_aux_delay					abandoned	time delayer for audio process in general
Aux	audce_aux_raydelay					abandoned	calculates the delay of the direct ray of a IR vector
Aux	audce_aux_terminate					abandoned	block that causes a patch to terminate
Aux	audce_aux_msg_rcv					abandoned	receive messages from external world via TCP/UDP for all layers
Aux	audce_aux_control_transmit					frozen	parses, stores and transmits commands and process control parameters to all layers
Aux	audce_aux_data_transmit					frozen	template block to parse, store and transmit data
Aux	audce_aux_test					frozen	general template block for tests
--	audce_gui_srcfile	name of snd obj				discontinued	defined audio input file
--	audce_gui_volume					obsolete	gui to control volume (fader), solo and mute
--	audce_gui_transport					obsolete	gui to start, stop, control and terminate patch

Messages

The following table lists the set of messages implemented in the current version of AUDIENCE for Pd.

Table 3 – AUDIENCE messages

Layer	Message	Arguments	Status	Usage and short description
L1	src_pos n x y z	n = number of sound object; x, y, z = cartesian coordinates	stable	to move sound object n to position x,y,z
L1	rcv_pos x y z	x, y, z = cartesian coordinates	stable	to move listener to position x,y,z

Table 4 – AUDIENCE blocks: fast map view

audce_L1_gui					L1
audce_L2_allen		audce_L2_crr	audce_L2_RMgen		L2
audce_L3_amb_3rd~	audce_L3_3obj_amb1st~	audce_L3_3obj_amb2nd~	audce_L3_3obj_amb3rd~	audce_L3_5.1render	L3
audce_L3_aacenc	audce_L3_5obj_amb1st~	audce_L3_5obj_amb3rd~	audce_L3_crr2amb1~	audce_L3_ambirconv~	
audce_L3_MPSEnc					
audce_L4_amb_3rd~	audce_L4_MPSEnc	audce_L4_objplay1~	audce_L4_sceneplay		L4
audce_L4_aacdec	audce_L4_objplay~	audce_L4_objplay2~	audce_L4_amb_3rd~		
audce_aux_mixer~	audce_aux_convolver~	audce_aux_dbvol~	audce_aux_mute~	audce_aux_objvol~	AUX
audce_aux_2.2.2.0	audce_aux_4.2.4.0	audce_aux_4.2.5.1	audce_aux_6.2.5.1	audce_aux_objvol1~	
audce_aux_file	audce_aux_move	audce_aux_n.2.mc	audce_aux_nwaves2mc	audce_aux_volume~	
audce_aux_objtime	audce_aux_objtransp	audce_aux_samples2time	audce_aux_time2samples	audce_aux_vol~	
audce_aux_pf~	audce_aux_playmcfile~	audce_aux_playmonofile~	audce_aux_recordmc~	audce_aux_writemc	

Help files

Besides the source of information in table 1, most blocks will also have a “help” file associated, which can be accessed by right-clicking the mouse over the block, and opening it.

Help files gather the following sort of information:

A help file

Object name

Comment *(one sentence info)*

Description

Arguments (of creation)

Example *(this is very useful to see how the block is used in a practical example)*

Additional information for that specific block*

Limitations and known bugs

Version history

See also *(indicates other related and complementary objects/blocks that may help the user to understand more about the related functions)*

(*) optional information

AUDIENCE

Object name: `audce_L3_amb_3rd-`

Comment: Ambisonics encoder up to the 3rd order

Description: The `audce_L3_amb_3rd-` object receives the sound source and receiver positions and encodes a mono signal using the Ambisonics encoding equations.

This kind of encoding is based on a free-field because there are no reflections, just the primary ray.

Arguments: (1) Ambisonics order (1, 2 or 3)
(2) Ambisonics sphere radius (in meters)

Usage example:

Ambisonics coding issues:

- a) encoding sphere radius is orthogonal to decoding sphere radius (used in the L4 decoder block).
- b) sphere radius is the reference distance where the sound is neither amplified or attenuated.
- c) sounds beyond the sphere are attenuated; sounds inside the sphere are amplified until the head radius limit of 0.01m.

INSTALLATION

OpenAUDIENCE and AUDIENCE are supported in three Operating Systems: Windows, MAC OS X and Linux. Installation in all of them is done in *two phases*:

- 1 – install the Pure Data (Pd)
- 2 – install the OA or full AUDIENCE library.

Shortly, installing the Pd is easy: regardless of the OS, a compressed archive will be expanded, creating working directories and a binary (executable) file. Installing the OA/AUDIENCE requires decompressing its distribution zipped file into a working directory and then updating the search *path* of Pd in order to also look for files in the OA/AUDIENCE directories. Notice that this installation is required for both the normal use and development purposes.

Also, please notice that some functions in AUDIENCE require the installation of third parties' software packages/libraries such as: fftw3, Coding Technologies/Dolby libsomedia and Tcl/Tk.

This section presents installation instructions valid for all the three OS. We should anticipate however the note in "troubleshooting" about Windows not permitting any "aux" directory under its file system. Beware of that.

Installing in all OS

1. Download and install the Pure Data on your audio-capable machine. Pure Data is free (open source) and can be downloaded from the following websites:

<http://crca.ucsd.edu/~msp/software.html>

Miller Puckette's "official" version, also known as pd-vanilla. There you will find links for Pd downloads, readme.txt, documentation, source code and stable releases.



<http://puredata.info/downloads>

Pd Community Site. There you find the most recent version of pd-extended, the most complete assembly of all available libraries, extensions, and documentation from the Pd CVS repository.

For **Windows** users: Pd is distributed as a "zip" file. Unzip this creating a directory such as \pd (ex: C:\pd).

For **Linux** users: Download Pd, which will be a ".tar.gz" file. To unpack it, type "zcat [name].tar.gz | tar xf -" to a shell. This creates a directory with a name like "pd-0.35". There are also RPMs available.

For **MAC** users: The web browser will automatically unpack the distributions into a folder such as "pd-0.35" on your desktop. Move it to the Applications directory if the automatic installation does not do it for you. To make easy access to the binary (executable) icon, drag it to your bar.

2. Download the OpenAUDIENCE library from the OA portal or get an authorized full AUDIENCE version distribution.

OA portal: <http://openaudience.iv.org.br/portal/down>



Full AUDIENCE version is available to members of the development team or contracted users. As a researcher or developer you are welcome to apply to become a member of the development team. If you want to license the full AUDIENCE version please send an email to audience@lsi.usp.br stating your interest and contact information.

OA and AUDIENCE are distributed as a "zip" file.

3. Unzip the "zip" to a working directory of your choice, such as `C:\audience` (in Windows) or `~/audience` (Linux and MAC).

Automatically the library directory structure will be expanded into this directory. If this is not your audio-capable working computer, then copy the directory to the target machine.

4. Open the Pd and update the *Path*. The directories where Pd will look for OA/AUDIENCE patches and binaries must be included in the *Pd search path*. Look for the option *Path...* and open the window where you edit the search path.

Include these directories into the path:

<code>/audience/L1</code>	contains L1 functional blocks (objects & patches)
<code>/audience/L2</code>	contains L2 functional blocks (objects & patches)
<code>/audience/L3</code>	contains L3 functional blocks (objects & patches)
<code>/audience/L4</code>	contains L4 functional blocks (objects & patches)
<code>/audience/aux</code>	contains auxiliary functional blocks (objects & patches)
<code>/audience/bin</code>	contains 3rd parties' binaries and <i>dlls</i>
<code>/audience/lib</code>	contains libraries for normal use and development

You may also need to include these directories into the path:

<code>/audience/obs</code>	contains previous and now obsolete patches and objects
<code>/audience/tst</code>	contains test patches and demos

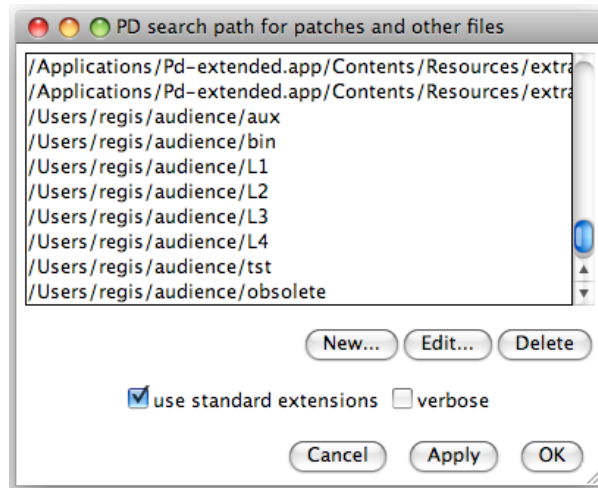


Figure 7 – Pd Search path configuration window (MAC OS)

NOTE 1: All patches and objects to be instantiated must be in one directory listed in the Pd search path. Directories not listed there will not be searched.

NOTE 2: The full AUDIENCE version includes the “app” directory containing several sub-directories for *applications*. The OA version may contain only free/open-source applications.

NOTE 3: If patches, objects or externals from a given application is to be used you must include the application sub-dir into the path:

/audience/app/X contains patches/objects from the application X

NOTE 4: If you develop applications, create a new sub-directory to hold your patches (e.g. */audience/app/my_application*). Then you must also include this directory in the search path as well. For development purposes the “inc” (or “include”) directory will hold include files and the “src” (or “source”) directory will include the source files (in C) and the *makefile* to compile and build-up all externals.

NOTES FOR PREVIOUS VERSIONS:

a) The directory *audience/patches* used to hold applications’ sub-directories. The “app” directory currently replaced it.

b) In previous versions the following directories must also be included in the path:

<i>/audience/patches/audience</i>	that contains the AUDIENCE functional patches
<i>/audience/patches/tests</i>	that contains the AUDIENCE test patches

5. Setup the **audio input** and **audio output** ports and **sample rate** in the Pd “Media” menu. If you use a multichannel card there will be options there to setup their inputs and output ports so that you can address them using the *adc~* and *dac~* objects of Pd.

NOTE 1: Many problems of patches not playing and lack of audio are due to mismatch between the ports used in the patch and those available and setup in the “Media” menu.

6. Test the installation by opening **test patches** in the */audience/tst* directory.

Start the Pd and test the AUDIENCE installation with the patch *test_audience.pd*.

Match the number of channels of the *dac~* object to the available number of channels in your system. For more information, see help documentation of the block *spatialdecoder~*.

preliminary

USAGE

What can be done?

You can build the following applications:

- Recorders, players

You can do the following functions:

- create squared rooms with as many sound objects inside as you wish
- encode a scene to Ambisonics in 1st, 2nd or 3rd order
- decode Ambisonics sound files to several output loudspeaker setups and modes
- record and play multichannel files
- mix audio files
- convolve audio files with arrays (e.g. impulse responses)

To receive/send audio streams from/to other applications you must have something like *Jack* (installed in your computer and have it declared and properly setup in the menu Media/Audio of the Pd).

How to start using the software

First of all, make sure you have the *Pd search path updated* and the *audio setup completed* in the Pd as explained in the installation section, otherwise you won't be able to use OA/AUDIENCE for Pd.

Second, open the Pd. Create a new patch, name it and start creating objects inside it.

- ☞ To create OA/AUDIENCE objects select "Put" and then write the name of the object in the dashed box that appears. A keyboard short cut for "Put" is CTRL (or Command) + 1.
- ☞ Don't forget to use the correct expected arguments to create your object, otherwise it will not be created and the box will be dashed. You can check the creation arguments in this manual. If you open the help-file of the block you will also learn about the creation arguments. A final option is to try any argument and then see the error messages on the Pd console: some error messages will instruct you on how to put the right arguments.

If you have already patches created, double click on them. This will open the Pd and the patch automatically. From this pre-created patch you can modify it and save-as with other name.

If you have problems, read troubleshooting and the help files of the specific blocks that are "causing problems".

See the TUTORIALS section ahead for more information on how to use OA/AUDIENCE. If you still have problems please contact the AUDIENCE team by email.

Tutorials

You can learn how to use AUDIENCE from TUTORIALS that come in the “app” directory. They were prepared in the format of *Pd patches* and they are a very practical way to see how AUDIENCE works.

There are 7 tutorial patches there. You just open them as any “.pd” file by double-clicking on it. They can teach you the following tasks:

- Tutorial 1: Creating a sound scene
- Tutorial 2: Connecting sound sources
- Tutorial 3: Playing a sound scene
- Tutorial 4: Simulating acoustics
- Tutorial 5: Calibrating parameters
- Tutorial 6: Useful objects
- Tutorial 7: Next steps

More tutorials are to come in the next versions/releases, covering several aspects of sound-scene oriented production, playing, animating, etc.

Troubleshooting

1. Some objects that execute convolution are not instantiating.

OA / AUDIENCE uses some third parties’ libraries, such as TCL/TK (installed with Pd), the “libsomedia” library for AAC coding/decoding, and the FFTW package⁹.

Check if FFTW is installed in your system. This library is required by many objects that perform convolutions. The files *fftw3.lib*, *libfftw3.a* and *libfftw3.a* should be in the *audience/lib* directory. For Windows users also check if the file *ffwt3.dll* is installed in *audience/bin*. If the problem continues, copy it also to the directory *bin* of the Pd installation.

If FFTW is not there, you must download the package from their website and install it in your system. Follow their instructions and use “makeinstall” to install it in the right places. In Windows the files must be either in the Pd directory or in */system32* directory to be found. In MAC OS, they are located in */usr/local/lib*.

If this not solves the problem, then it is not the cause.

⁹ FFTW is a free collection of fast C routine for computing DCT. It was written by M. Frigo and S. G. Johnson and can be downloaded from www.fftw.org. Current version used is 3.2.2.

NOTE: In the MAC OS the directories of the Pd installation are hidden. You should expose them in the finder by opening the Applications directory, finding the Pd, and right-click the mouse to get access to the package contents and browse it in the finder.

2. Some objects are not instantiating (i.e. they can not be created, the block only gets dashed)

Check the Pd search path to see if the blocks you want to create are in directories included in the list. If not, edit the list and include any missing directory. Don't forget to save it for later use.

If it does not solve the problem, see the error message printed in the Pd console, and track down your options to solve it from the messages found there.

Some patches could have been created using obsolete external names. In this case, open the patch file (*patch.pd*) with a TEXT editor, and update the name of the object. Then save and re-open the patch.

If all your "audce_aux..." objects are not instantiating so most likely you are in Windows and the patch is trying to open these auxiliary objects from the "aux" directory. In Windows, however, the auxiliary objects are inside a "auxiliar" directory (because Windows does not permit users to create any directory named "aux" in the file system). In this case, update the path to the objects (by editing the *patch.pd* file with a text editor).

3. Compilation errors due to unistd.h

unistd.h is the reference to UNIX standard library, and will be important for UNIX (including MAC OS X) and Linux. This is found in Cygwin and MinGW distributions, which have their own translations to the functions it implement, mostly system calls.

This is not yet completely clarified its need in AUDIENCE, but if you are having compilation problems with this header file in Windows you may have to find an equivalent for it in your compilation environment. Commenting the lines in the source codes will however prevent compilation errors in Windows.

4. It is not possible to create or use a directory "/aux" under Windows

It has been noticed that Windows does not permit the creation/usage of any directory "aux" in its file system. This is a Windows condition only. In this case, you should change the name of the "aux" directory to "auxiliar" and all references to it. Beware that some patches will try to open auxiliary objects from an "aux" directory.

DEVELOPING (how to create blocks in AUDIENCE for Pd)

Building an AUDIENCE block in Pd

Each block in AUDIENCE implements one functionality in ONE of the main or auxiliary layers.

Naming convention

All blocks are named following a defined syntax to permit immediate identification of the layer it belongs to. Blocks names are built as follows:

audce + _ + layer + _ + functionName

Examples: audce_l2_raytracer; audce_aux_mixer~; audce_l1_x3dparser; audce_L4_mcplayer~

Design and implementation steps

To build a new functional block in the OA or AUDIENCE in Pd, one shall follow the steps below:

1. **Target layer:** This is the conception stage. Define to what layer your block will belong among the main 1, 2, 3 or 4 layer, or a service layer.
2. **Interface and core design:** Define inputs, outputs and processing core. The functions your block will execute will use input data/audio and produce output data/audio. Data can be supplied in several ways but mainly as argument creation and passed in messages to the block. You have to verify from the possible interface signals set which ones you are most likely to use. Keep in mind that a layer *i* block shall accept layer *i-1* outputs and produce inputs signals to the layer *i+1*.
3. **Compliance to the message set:** Your block may use defined parameters, attributes of objects and sound sources, which are used or addressed by other blocks in the scene patch. For example, layer 1 blocks may change the user position and need to propagate this new position using a defined syntax. AUDIENCE has a mechanism to update the values of these data using *message passing*, which can be propagated using a communication send/receive mechanism, the simplest one being a data connection from one block outlet to a block inlet. Your block belongs to one layer, and this layer must be able to receive some messages and to interpret them.

Usually messages have the name of the parameters itself and are followed by its new values. You have to check what messages from the message list you must interpret (see table in this manual) and to build in your block a mechanism to receive and interpret it. Check the message list of types and values for this compliance. There are no strict rules yet on how to build this (if using direct connections with inlets/outlets, or send/receive Pd messages, or UTP messages, etc.) but we are working towards standardizing this. Suggestions are welcome.

4. **Block implementation:** Define whether your block will be built as a *patch block* using the Pd built-in blocks and functions, or if your block will be built as an *external*, linking a function written in C/C++. In this last case you will need to follow the instructions to write externals in the next section of this manual.

These are the main design rules (based on inter-process requirements) for guaranteeing a minimum operational status for your block and making it compliant to what is specified for a OA / AUDIENCE block.

A universal block template is a future wish to be developed, so that creation of new blocks get easier and the incorporation of an external algorithm or integration to an external plug-in or other software is done faster, more in a plug-and-play fashion.

Creating an external Pd block

To create a OA / AUDIENCE block as an external for Pd you need a C/C++ development environment to write your algorithm or function and to compile it to a valid Pd object. First of all you must create a project for your new Pd object. The sub-sections below instruct on how to create a new project to work with the AUDIENCE directory structure and give you minimum information on how to make a Pd external.

For more complete information on this matter you should read the tutorials on “how to write an external” published in the Pd websites.

How to create a project for a new Pd object

The following example is based on the usage of Eclipse C/C++ development environment, and was proposed in 2007. You may need to adapt these instructions to reflect your actual environment conditions.

1. **Create a new project:** Create a new project through the option *File/New/ Managed Make C Project*. In the field *Project Name* write the name of the Pd block you wish to create. Remember that audio processing blocks must terminate with a ~, and then the name of the project must also terminate with ~. Click in *Next*. Select the *Project Type* as *Shared Library (GNU on Windows)*. Click in *Finish*.
2. **Set properties:** Right click over the project name and select *Properties*. In the option *C/C++ Build/Tool Settings/GCC C++ Compiler/Directories*, add the path to the source and include directories (the ... \src and ... \include directories) so that the compiler finds the file *m_pd.h*. In the option *C/C++ Build/Tool Settings/GCC C++ Linker/Libraries*, add the library *Audience4Pd* to the list *Libraries* and add the path to this library in the list *Library search path* (the ... \bin directory). Later on you might need to move the final binary to another directory destination, e.g. one of the *L_i* directories.

3. **Reassure options:** Reassure that inside the option *C/C++ Build/Build Settings* the option *Use default command* is not active, and that in the field below it is written *mingw32-make -f makefile*. This is to use the OA / AUDIENCE makefile.
4. Now create or insert the file with the source code of the new Pd block, with the same name of the project.

Now you need to know how is the basic structure of a Pd external. The next sub-section lists the steps to create a simple Pd external. For a complete reference you should open and inspect yourself existing Pd external source codes and read more complete tutorials on this matter available in the Pd websites.

How to make a Pd external

The steps below show how to make a Pd external.

1. **Define object type:** Define if the block will be of *object class* (messages) or *signal class* (audio and messages).

The basic difference between the two types of blocks is whether it will process audio. In case it does processes audio, one more method must be implemented in the code, *dsp()*, which will process the audio.

2. **Define which data the block must hold.** Define which data the block will store.

Each block instance has its internal variables that must be available for all methods of the block. Instead of a global memory, the Pd external keeps a data structure, which is passed to the method, and in this way all variable can be used.

Basically it is defined a variable named *x_obj*, of type *t_object*, which will be manipulated by Pd. The other variables must be declared and used by the external. Some useful variables that can be created in this structure are *pointers* to the inlets/outlets and configuration parameters of the block.

3. **Define the creation arguments of the block.** Define the creation parameters of the block.

The creation parameters of the block are passed during the instantiation (creation) of the block in the Pd environment (patch), right after the block name. They are used to define parameters that shall not change during the execution of the function in the patch, as, for example, the size of sample block, number of channels of the block, etc.

These parameters are passed by Pd to the method *new()* of the external, and there they can be verified and, if necessary, stored in the block data structure. These parameters can be of type *float* or *symbol*.

The number of parameters (and their types) is specified in the method *setup()*, and it can be fixed or dynamic.

4. **Define the number and type of inlets and outlets.** Define the number of *inlets* (input ports) e *outlets* (output ports), as well as the type of each one (*audio*, *list*, *float* etc.)
5. **Define messages to be sent/received.** Define which messages are to be received and delivered by the block, as well as the data type of each message.
6. **Implement the data structure.** Implement the block data structure, to store all data

```
typedef struct _audce_aux_teste {  
    t_object  x_obj;  
} t_audce_aux_teste;
```

7. **Implement the method *setup()*.**

- a. Add methods for messages.
- b. Add the method *dsp()*, in case this is an audio processing block

```
void helloworld_setup(void)  
{  
    helloworld_class = class_new(gensym("helloworld"),  
        (t_newmethod)helloworld_new,  
        0, sizeof(t_helloworld),  
        CLASS_DEFAULT, 0);  
  
    class_addbang(helloworld_class, helloworld_bang);  
}
```

8. **Implement the method *new()*.**

The method *new()* is called by Pd every time an object is created (instantiated) in a *patch*. It must be of type *void** and must

- a. read and validate input parameters
- b. create *inlets* and *outlets*
- c. allocate memory
- d. initialize the variables

```
void *helloworld_new(void)  
{  
    t_helloworld *x = (t_helloworld *)pd_new(helloworld_class);  
  
    return (void *)x;  
}
```

9. Implement the method *free()*.

- a. release memory

10. Implement the methods of every message.

11. Implement the methods *dsp* and *perform* (in case it is an audio processing object).

The whole process can be summarized as in the flowchart below.

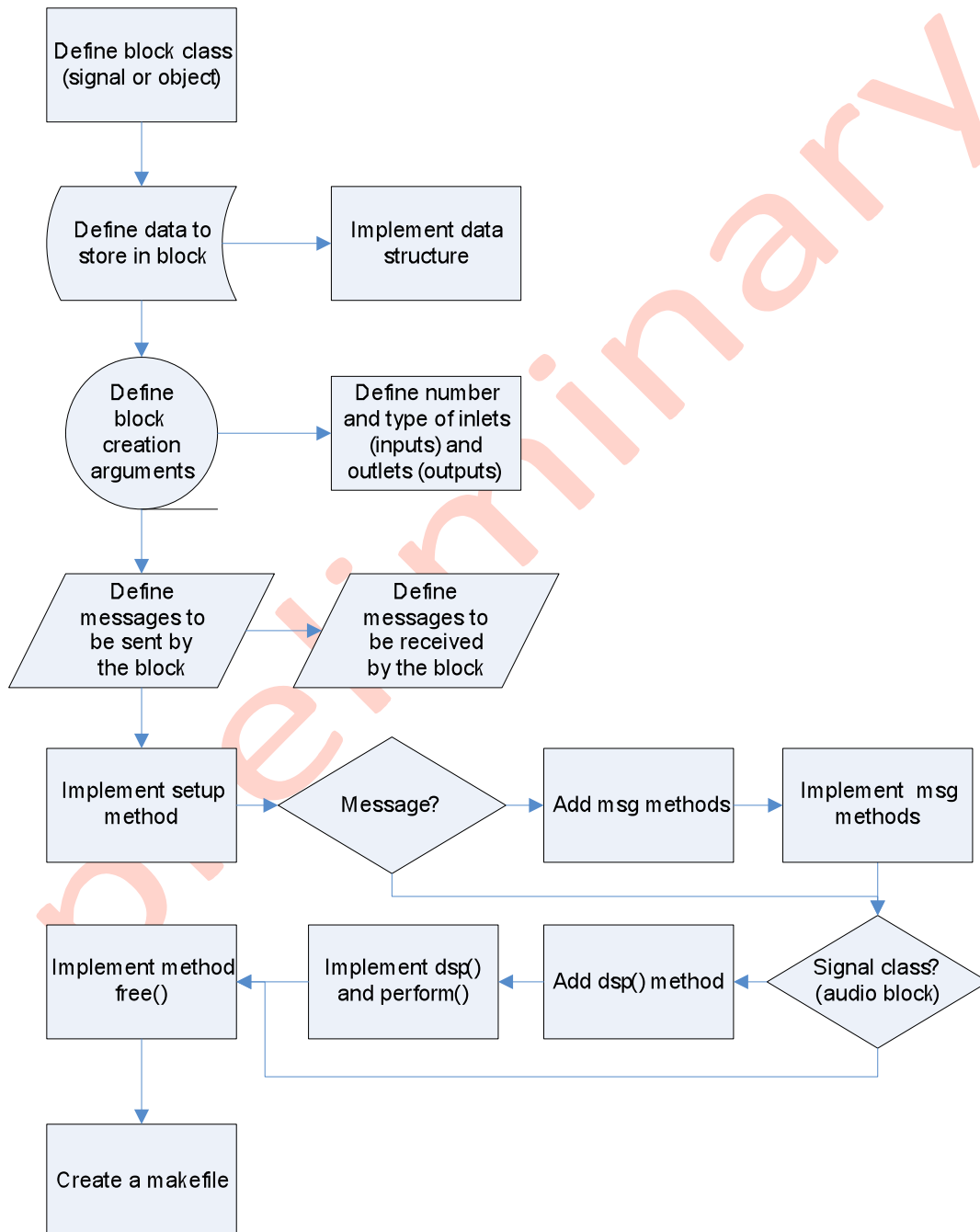


Figure 8 – How to make a Pd block

Compiling and building-up OA / AUDIENCE externals

Compiling the OA / AUDIENCE externals will require access to the source files, headers and the *makefile*. The externals' source files and the *"makefile"* must lie into the *"src"* directory. Header files must be inside the *"inc"* (or *"include"*) directory. The *makefile* script holds compilation and build-up instructions for the compiler and it is meant for use with all supported OS: MAC OS, Windows and Linux.

To compile the distribution you need to check if some parameters in the *makefile* will work for you:

a) if you use Windows and *Visual Studio C/C++*, under the *"NT"* section update the *"VC"* and *"PD"* variables to point respectively to your *Visual Studio* and *Pd* directories. You must substitute *"username"* by your Windows username.

If the compiler claims that *"fftw3"* lib is missing, check if the variable *"PDNTLIB"* points to where it lies. Also check if *"PDNTINCLUDE"* points to all needed *"include"* directories, including the OA/AUDIENCE directory *"include"*.

b) if you use Linux, check if *"LINUXINCLUDE"* variable points to the OA/AUDIENCE directory *"include"*.

Testing built AUDIENCE and Pd blocks

Testing a Pd block

To test a brand new Pd or OA / AUDIENCE block you have created, you can open a patch (or create a new one) and instantiate the object by putting it on the workspace. Just create the box and write the name of the block, with all necessary parameters for the creation (creation arguments).

After this, just click outside the block in the workspace and if your block is OK it will be get a solid line around, showing that it was identified in the library. In case it does not happen, and the box has dashed lines, you may have entered the wrong text to instantiate it.

Some blocks may require for functioning some other parameters inputs, provided by other boxes (e.g. data boxes and messages, bangs, start and stops messages, etc.). If it is the case, you should create the other boxes with all necessary data to send to your new built block.

Connect your block with other Pd or OA / AUDIENCE blocks to create a valid patch with inputs, processing and outputs. Test then your functionality, and interact with the control blocks and data boxes that change the way your block works. Also, remember to make Audio ON before, as it may be turned off.

After validating the functioning of the block, you can create a test-patch for testing this block and saving it in the ...\\tst directory for future reference/usage.

RELEASE NOTES

AUDIENCE version: v. 2.0.1 (Aug.2010)
OpenAUDIENCE version: v.1.0.1 (Aug.2010)

Individual files releases within the same version are tagged by their creation or updating dates.

Licenses

AUDIENCE and OpenAUDIENCE libraries for Pd are released under a series of different licenses, which apply on a block-usage basis. Individual files must be consulted for individual licensing conditions. AUDIENCE blocks used in applications must be cited in their documentation indicating *"This application uses AUDIENCE for Pd library objects"* with a list of all used blocks. If used blocks are linked to other licenses, all the terms of other licenses must also apply. The following states different conditions for Native blocks and Third-parties' blocks.

Native blocks. OpenAUDIENCE and AUDIENCE Native blocks (developed by AUDIENCE/OA team) are BSD licensed. The following terms inside the box apply for them:

AUDIENCE and OpenAUDIENCE native blocks BSD License Terms. Copyright (c) 2005-2010 Regis Faria and others.

This program is free to use and redistribute. Any modification to it must be documented in any redistributed version indicating the modifications and the authors. Commercial usage of the code is allowed with specific written permission. THIS SOFTWARE IS PROVIDED BY THE AUTHORS "AS IS" AND ANY EXPRESS OR IMPLIED WARRANTIES, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE ARE DISCLAIMED. IN NO EVENT SHALL THE AUTHORS BE LIABLE FOR ANY DIRECT, INDIRECT, INCIDENTAL, SPECIAL, EXEMPLARY, OR CONSEQUENTIAL DAMAGES (INCLUDING, BUT NOT LIMITED TO, PROCUREMENT OF SUBSTITUTE GOODS OR SERVICES; LOSS OF USE, DATA, OR PROFITS; OR BUSINESS INTERRUPTION) HOWEVER CAUSED AND ON ANY THEORY OF LIABILITY, WHETHER IN CONTRACT, STRICT LIABILITY, OR TORT (INCLUDING NEGLIGENCE OR OTHERWISE) ARISING IN ANY WAY OUT OF THE USE OF THIS SOFTWARE, EVEN IF ADVISED OF THE POSSIBILITY OF SUCH DAMAGE.

Third-parties' blocks. AUDIENCE and OpenAUDIENCE libraries blocks (codes) that use third-party libraries, proprietary software or GPL software must comply with the corresponding licensing terms and include it. Below are detailed NOTES on all possible licensing conditions:

Note 1: Pd is licensed under GPL (version 2, June 1991) and parts of the package can be used under "Pd BSD" license (a "Standard Improved BSD License"). Pd is copyrighted by Miller Puckette and others. Users must consult the license.txt in Pd distribution for further details on commercial usage and other purposes. The AUDIENCE team does not respond by the Pd license and terms. All applications using the AUDIENCE and OA for Pd must comply with the Pd license terms and conditions.

Note 2: GPL-licensed codes. Applications using items/code under GPL can not be commercialized. GPL terms must be followed for the files and libraries released under this license.

Note 3: FFTW3 is released under GNU GPL license by Copyright (c) 2003, 2007-8 Matteo Frigo (fftw@fftw.org). Some AUDIENCE objects use this library. All applications developed further must include the license terms in the documentation and comply with its conditions.

Note 4: MPEG-4 AAC for aacPlus© is released under limited license by Coding Technologies and later on by Dolby Laboratories, Inc. Users are restricted to associated research teams and must comply with terms on comprehensive signed license agreement. This is not released under OpenAUDIENCE.

Note 5: MPEG Surround files are ISO copyrighted. Commercialization is not allowed.

Note 6: CRR (Combined Ray-Tracing and Radiosity) code is copyrighted by the University of Sheffield. The following licenses apply by using it: RayTrace Software Package, rel. 3.0, May 3 2006, by Samuel R. Buss (sbuss@ucsd.edu) and SRC (Secret Rabbit Code, GNU GPL) by Erik de Castro Lopo (erikd@mega-nerd.com).

preliminary

ANNEXES

A. Instalando o ambiente de desenvolvimento Eclipse/CDT/MinGW

Annex A is available only in Portuguese. It describes how to install the Eclipse/CDT/MinGW development environment. This text version was released in 04/04/2008.

1. Instalando o Eclipse

Com o Java (JDK) instalado, vá até a página do Eclipse (<http://www.eclipse.org>), clique em *Downloads*, em *All Versions* e selecione o SDK do tipo de build *Stream Stable Build*. A versão baixada para criar este artigo foi a 3.2M5a. Aguarde o download e descompacte em qualquer diretório. Crie um atalho para o *eclipse.exe* em seu desktop e pronto. Já temos um ambiente de desenvolvimento Java instalado. Partiremos, então, para o ambiente C++.

2. Instalando o MinGW

Faça o download do *MinGW-5.0.2.exe* ou superior da página <http://sourceforge.net/projects/mingw>. Execute o aplicativo, clique em *Next* e Selecione um *mirror*. Clique em *Next* novamente e selecione o tipo de instalação *Custom*, marcando *MinGW base tools, g++ compiler e make*. Clique em *next* e selecione o diretório de instalação. Selecione o atalho que você deseja criar e clique em *Install*. Após instalar clique em *Finish*.

Adicione o diretório `<MINGW_HOME>/bin` à variável de ambiente PATH, caso o instalador não tenha feito isso.

3. Baixando e instalando o CDT (plugin C++)

Execute o Eclipse, clique no menu *Help/Software Updates/Find And Install*. Na tela que abrirá, selecione *Search for new features to install* e clique em *Next*. Clique no botão *New Remote Site*, e na tela que abrirá, coloque C++ como nome e esta URL: <http://download.eclipse.org/tools/cdt/releases/eclipse3.1>

Clique em *OK* para fechar a tela e em *Finish*. O Eclipse irá procurar por plugins na URL. Se abrir uma tela para escolher um mirror, selecione o mais próximo de você e clique em *OK*. Ao aparecer a versão do CDT para instalar, *selecione-a* e clique em *Finish*. O Eclipse irá baixar os plugins. Após o download, o Eclipse mostrará a licença do Plugin, *aceite* os termos e clique em *Install All*. Depois de instalar, o Eclipse reiniciará a IDE e seu ambiente C++ está pronto para ser utilizado.

Antes de testarmos, vamos alterar uma configuração. A IDE tem, por default, o ambiente de instalação Cygwin e, portanto, temos que alterá-la para o MinGW. Entre no menu

Window/Preferences. Abra o *C/C++/Make/New Make Project* e desmarque a opção *Use Default*. No build command, coloque: `mingw32-make -f makefile`

Marque a opção *Build on resource save* que fará executar o build a cada alteração de suas classes e clique em *Finish*.

4. Testando o ambiente.

Clique no menu *File/New/Project*. Abra a pasta C++, selecione *Standard Make C++ Project* e clique em *Next*. Coloque o nome do projeto, exemplo *teste*, e *Next*. Clique em *Finish*. Se pedir para trocar de perspectiva clique em *Sim*. Seu projeto C++ está criado.

Botão direito sobre o projeto e entre em propriedades. Abra o *C/C++ Build* na paleta *Build Settings* e se certifique que opção *Use Default* esteja desmarcada e no *Build command*, esteja: `mingw32-make -f makefile`