ToscA: An OSC Communication Plugin for Object-Oriented Spatialization Authoring
Thibaut Carpentier

➢ To cite this version:
Thibaut Carpentier. ToscA: An OSC Communication Plugin for Object-Oriented Spatialization Authoring. 41st International Computer Music Conference (ICMC), Sep 2015, Denton, TX, United States. 41st International Computer Music Conference (ICMC), pp.368 – 371, 2015. <hal-01247588>

HAL Id: hal-01247588
https://hal.archives-ouvertes.fr/hal-01247588
Submitted on 23 Dec 2015

HAL is a multi-disciplinary open access archive for the deposit and dissemination of scientific research documents, whether they are published or not. The documents may come from teaching and research institutions in France or abroad, or from public or private research centers.

L’archive ouverte pluridisciplinaire HAL, est destinée au dépôt et à la diffusion de documents scientifiques de niveau recherche, publiés ou non, émanant des établissements d’enseignement et de recherche français ou étrangers, des laboratoires publics ou privés.
**ToscA: an OSC communication plugin for object-oriented spatialization authoring**

Thibaut Carpentier  
UMR 9912 STMS IRCAM-CNRS-UPMC  
1, place Igor Stravinsky, 75004 Paris  
thibaut.carpentier@ircam.fr

**ABSTRACT**

The paper presents ToscA, a plugin that uses the OSC protocol to transmit automation parameters between a digital audio workstation and a remote application. A typical use case is the production of an object-oriented spatialized mix independently of the limitations of the host applications.

1. **INTRODUCTION**

There is today a growing interest in sound spatialization. The movie industry seems to be shifting towards “3D” sound systems; mobile devices have radically changed our listening habits and multimedia broadcast companies are evolving accordingly, showing substantial interest in binaural techniques and interactive content; massively multichannel equipments and transmission protocols are available and they facilitate the installation of ambitious speaker setups in concert halls [1] or exhibition venues.

In all these contexts, object-based production appears to be the favored approach. In the paradigm of object-based spatialization, an audio source is linked with side informations and both streams are stored or transmitted to a rendering engine which reproduces the spatial sound scene. Many object-oriented spatialization processors are used (for instance: Sound Element Spatializer [2], SoundScape Renderer [3], ICST Ambisonics Tools [4], Spatium [5], VBAP [6], HoaLibrary [7], Zirkonium [8], Wave 1 [9], Spatialisateur [10], etc.), and from a signal processing viewpoint they are efficient and they offer great possibilities of rendering. However the authoring of spatial sound scenes remains a major challenge. There are several reasons for this. 1) Most of the previously mentioned processors are integrated into realtime environments — such as Max/MSP or PureData — which are inadequate to mixing or spatial composing as they are ill-suited to manipulation of temporal structures. 2) On the other hand, digital audio workstations (DAW) provide an efficient timeline and transport bar, but they lack flexibility for multichannel streams: most of the DAWs only support “limited” multichannel tracks/buses (stereo, 5.1 or 7.1) and inserting spatialization plugins is difficult and/or tedious. 3) Finally the absence of standardization or consensus on the representation of spatialization metadata (in spite of several initiatives [11, 12]) complicates the interchange between applications.

This paper introduces ToscA, a plugin that allows to communicate automation parameters between a digital audio workstation and a remote application. The main use case is the production of massively multichannel object-oriented spatialized mixes.

2. **WORKFLOW**

We propose a workflow where the spatialization rendering engine lies outside of the workstation (Figure 1). This architecture requires (bi-directional) communication between the DAW and the remote renderer, and both the audio streams and the automation data shall be conveyed. After being processed by the remote renderer, the spatialized signals could either feed the reproduction setup (loudspeakers or headphones), be bounced to disk, or be sent back to the DAW.

In the remainder of this paper, the remote spatialization renderer (or equivalently the environment that hosts such renderer) will be denoted by “auxiliary application”.

![Figure 1. Principle of object-oriented spatialization rendering with a remote application.](image-url)

Typically, the audio tracks to be spatialized are mono or stereophonic, and the automation parameters refers to spatial data (coordinates, and optionally orientation and/or directivity data). Localization data are most frequently expressed with 3D Cartesian or spherical coordinates and are consequently stored on three automation tracks.
2.1 Audio transmission

The “remote rendering” workflow requires to transmit the audio data from the DAW to the auxiliary application. There are several ways to accomplish such inter-application communication, and two different use cases shall be distinguished:

- the two applications (DAW and auxiliary) run on two different computers; in this case one just needs to physically inter-connect the audio interfaces of each computer,
- both applications lie on the same computer. If possible, one can physically connect the output channels of the DAW to input channels of the auxiliary application. When hardware connection is not possible, the routing can be made software, e.g. using Jack\(^1\), SoundFlower\(^2\) or similar virtual drivers. Some audio interface drivers also allow (software) internal feedback without latency.

2.2 Automation transmission

For the transmission of automation data, a solution previously used in various productions at Ircam (or elsewhere) was based on the MIDI protocol. Indeed MIDI is widely used and perfectly supported in all DAWs and in auxiliary applications. However this approach suffered from many limitations:

- setting up the DAW session is extremely tedious, inconvenient, and prone to errors,
- due to its low resolution (8 bits), the MIDI protocol is ill-suited for encoding some spatialization parameters (e.g. azimuth or distance of the sources),
- the number of objects that can be controlled is typically limited by the number of available MIDI channels (i.e. 16),
- as MIDI messages are not typed, transmitted data are not easily readable, which makes the session difficult to maintain.

As a consequence, it was decided to develop an ad-hoc tool: ToscA aims at overcoming the constraints of the MIDI protocol and it is based on the Open Sound Control protocol (OSC)\(^3\) which was chosen for its wide acceptance in the computer music community and for its ease of integration.

3. TOSCA

ToscA\(^3\) is a plugin that can be inserted on each audio track of the DAW. The audio signals on these tracks are unaffected (bypassed) but several parameters are exposed for automation. In order to ensure compatibility with a wide range of DAW softwares, the maximum number of exposed parameters is restricted to 32.

During playback, active automation tracks are read by ToscA and the corresponding OSC messages are sent over UDP. When the automation tracks are armed for recording, ToscA accepts incoming OSC packets from remote applications and data are written in the sequencer.

\(^{1}\)http://jackaudio.org
\(^{2}\)http://rogueamoeba.com/freebies/soundflower
\(^{3}\)ToscA stands for Thibaut’s OpenSoundControl Automation.

3.1 Syntax of the messages

Each instance of the ToscA plugin has an identifier (ID) which can be adjusted by the user (see Figures 2 and 3). This ID constitutes the root of the OSC address pattern of incoming/outgoing messages. The syntax is as follows:

“/ID/ParameterName ParameterValue”

for instance: “/3/azimuth 135.0”

For the sake of simplicity, all parameters are treated as floating-point numbers in double precision. Other OSC types are currently not supported.

3.2 Mapping file

ToscA is not tied to a specific spatialization renderer and the exposed automation parameters are generic\(^4\). By default the 32 parameters are labelled param1, param2, etc. It is possible to tweak these labels by means of a XML configuration

\(^{4}\)Incidentally, ToscA is not limited to spatialization applications and it could be used for any other purposes.
file (called “mapping file”). Examples of XML syntax are presented in Figures 4 and 5.

In order to keep the setup easy and quick, it was decided to share the names of the parameters between all instances of ToscA. Whenever an XML configuration file is loaded (or edited) in one instance, all other instances are updated accordingly.

```xml
<tosca version="0.4">
  <parameter index="1" name="x" min="-10" max="10" scaling="linear"/>
  <parameter index="2" name="y" min="-10" max="10" scaling="linear"/>
  <parameter index="3" name="z" min="-10" max="10" scaling="linear"/>
</tosca>
```

Figure 4. Example of a mapping file: three automation parameters representing Cartesian coordinates.

```xml
<tosca version="0.4">
  <parameter index="3" name="azim" min="-180" max="180" scaling="linear"/>
  <parameter index="4" name="gain" min="-64" max="64" scaling="linear"/>
  <parameter index="5" name="aperture" min="0" max="100" scaling="linear"/>
  <parameter index="6" name="orientation" min="-180" max="180" scaling="linear"/>
</tosca>
```

Figure 5. Another example of mapping file.

The XML file can further be used to specify a scaling range for each parameter: within the DAW, automation tracks are coded in the [0 – 1] range; ToscA performs a scaling from [0 – 1] to the user-defined [min – max] range. The scaling is applied on outgoing data and an inverse transform is applied on incoming data. The benefits of such scaling transform are twofolds:

- session setup is fast as no parameter mapping is needed in the auxiliary application (unlike the MIDI approach presented in paragraph 2.2),
- it is possible to re-scale an existing mix by editing the [min – max] range of one or several parameters. Indeed, it is common practice in sound spatialization to re-work an existing piece depending on the size or acoustical properties of the venue. For instance, artists often have to emphasize or decrease some spatial parameters (see e.g. [14]) such as the radius of sound trajectories in order to “fit” the playback room.

### 3.3 Touch/latch modes

When authoring spatial sound scenes, it is often necessary to “touch” the trajectories several times using the DAW *touch* or *latch* modes. In this case, the auxiliary application shall inform the DAW of the beginning and end of the touch phase. ToscA uses the following syntax:

```
/ID/ParameterName touch 1": start touching
/ID/ParameterName ParameterValue": update the parameter value
/ID/ParameterName touch 0": stop touching.
```

Figure 6. ToscA inside ProTools. ① plugin inserted on track 1. ② list of exposed parameters. ③ plugin window. ④ automation tracks.

This appears especially useful for editing spatial trajectories with OSC-aware external devices such as Liine’s Lemur⁵ or Hexler’s TouchOSC⁶.

### 3.4 Software development

ToscA uses the [Juce]⁷ framework which provides a handy plugin wrapper. ToscA is thus available in VST, VST3, AU, AAX formats, for MacOS and Windows platforms (32 and 64

---

⁵ http://liine.net/en/products/lemur
⁶ http://hexler.net/software/touchosc
⁷ http://www.juce.com
4. CONCLUSION AND OUTLOOK

This paper presented ToscA, a plugin allowing OSC communication between a DAW and an auxiliary application. The typical use case of object-oriented spatialized mixing has been discussed.

ToscA is completely generic and it can control any type of parameters. In practice, the sound engineers/composers’ needs are essentially focused on spatial trajectories authoring. In order to (even) better meet their expectations, a new plugin is being developed: ToscA-Spat (Figure 8) builds on top of ToscA and further integrates a graphical visualization of the sound scene. It aims at providing a fast and intuitive workflow for geometrical editing.

Figure 8. Preview of ToscA-Spat in Logic Pro. ToscA-Spat is based on Ircam spat.viewer graphical interface.

5. ACKNOWLEDGEMENTS

The author would like to thank Alexis Baskind for valuable feedback, ideas, and beta-testing.

6. REFERENCES


