# OSC-Namespace and OSC-State: Schemata for Describing the Namespace and State of OSC-Enabled Systems

Ilias Bergstrom
Fondation Agalma
Rue Adrien-Lachenal 18, CH-1207 Genève
ilias.bergstrom@agalma.ch

EventLAB,
Universitat de Barcelona, Campus de Mundet - Edifici Teatre, Passeig de la Vall d'Hebron 171, 08035 Barcelona, Spain

Joan Llobera
Fondation Agalma
Rue Adrien-Lachenal 18, CH-1207 Genève
joan.llobera@agalma.ch

Immersive Interaction Group
École Polytechnique Fédérale de Lausanne
EPFL-IC-IIG
Station 14
CH-1015 Lausanne

## ABSTRACT

We introduce two complementary OSC schemata for two contexts of use. The first is for the complete description of an OSC namespace: detailing the full set of messages each OSC-enabled system can receive or send, alongside choice metadata we deem necessary to make full use of each system's description. The second context of use is a snapshot (partial or full) of the system's state. We also relate our proposed schemata to the current state of the art, and how using these resolves issues that were left pending with previous research.

## Keywords

OSC, Open Sound Control, Mapping, Schema, Namespace

## 1. INTRODUCTION

The development of new instruments for musical performance has always been at the forefront of technology, from the mechanical and electromechanical instruments of past times to the electronic and digital instruments of today.

The advent of synthesizers decoupled musical instruments into the gestural controller and the sound generator. The gestural controller forms the part musicians physically interact with, emitting data for the sound generator to interpret and ultimately produce the corresponding sound. This separation between control and output devices can today be witnessed in most forms of artistic performance, e.g. VJ-ing, new media art, light-shows, etc., usually each with its own standards and protocols for communication.

Today, modern performance systems converge towards using what has emerged as a potential future universal standard for real-time digital control data: Open Sound Control (OSC) [1]. The great advantage of OSC is that while there is a per-message schema, there is no overall fixed schema to define or restrict the set of possible messages, as is the case with legacy protocols (e.g. MIDI, DMX). A second advantage is that older protocols can be translated to OSC data with relative ease.

To describe OSC we paraphrase its creators [2]: the basic unit of OSC is a message, consisting of an Address Pattern (AP), a Type Tag String (TTS), and arguments. The AP is a string specifying the entity or entities within the OSC server to which the message is directed. The TTS gives the data type of each argument. Finally the arguments are the data contained in the message. So in the message /voices/3/freq, 'f' 261.62558, the AP is followed by the TTS and finally the corresponding argument. All points of control of an OSC server are organized into a tree-structured hierarchy called the server's address space. An OSC AP is the full path from the root of the address space tree to a particular node. In the above example the AP points to a node named "freq" that is a child of a node named "3", itself a child of a node named "voices". The full set of possible combinations of APs and TTSs that an OSC server responds to, we here refer to as that server's namespace.

OSC provides for several advantages compared to the previous de facto standards of their respective fields, MIDI, DMX, etc. Using OSC, interoperability between an arbitrary number of disparate sources and destinations is straightforward. No longer are digital musical instruments forced to adhere to the strained façade that they can behave as keyboard instruments, as was the case with MIDI, when in fact they are nothing of the sort (see for example drum, wind and guitar controllers).

However, OSC also introduces new obstacles. First, since there is no fixed set of messages, each participating server needs to know what messages it can send to the servers it intends to communicate with. Currently the OSC standard does not provide for a means of programmatically discovering all messages a server responds to. We will refer to this as the *namespace discovery problem*. Second, each server's events need to be mapped to the messages that the recipient servers expect to receive, which we will here refer to as the *mapping problem*. In this article we propose a solution to the namespace discovery problem, and relate it to previous research addressing the mapping problem.

## 2. BACKGROUND

Both obstacles mentioned are active areas of research. A solution that has been proposed towards both is reintroducing namespaces, using one namespace per context of use instead of a single fixed namespace for all, as was the case with legacy protocols. SynOSCopy [3] is one such standard namespace designed for describing synthesizers. The only other such initiative known to us is the now defunct OpenMediaControl repository for OSC namespaces [4]. In our view however, this idea is too restrictive and to some extent contradicting the main benefit of OSC: its openness and flexibility. No matter the extent of standardization, cases will always appear that need functionality beyond that described in the standardized namespaces. Be it conflicting versions of the same namespace, or two entirely different namespaces, mapping is still needed in both cases.

The mapping problem has many proposed solutions, each with its distinct advantages and disadvantages depending on context of use. The Libmapper software [5] was first to provide a GUI and toolset for the manual defining of mappings. The idea of Mutable Mapping, as implemented in the Mediator software [6], takes the extra step of facilitating that mappings be gradually altered, created and destroyed, even during performance as a form of expression in and of itself. Many more mapping approaches exist [7], and much room is still left for further innovation.

To address the namespace discovery problem, OSC querying solutions have been suggested, the first being by OSC's creators [8]. Through a series of extensions to the OSC protocol, their solution allows for querying an OSC server of its namespace, as well as its current state.

Querying is intended to address two different problems:

- Describing the server's namespace: this includes the set of OSC messages it understands, plus metadata for each message, such as units, default values, ranges, etc.
- Describing the full state of the server; that is, storing current values for each of the parameters that can be controlled with the above set of messages.

The above however, needs applying in two contexts of use:

- Run-time query of the namespace and state, directly extracting this information from the servers.
- Long-term storage and retrieval of the namespace and state information for future recall.

Querying systems allows discovering namespace, state and metadata during run-time, provided a server supports querying.

While query systems provide useful functionality, they face important obstacles. First, they exclude the vast number of existing systems which only support OSC up to v 1.1, since query functionality necessarily requires either that extensions to the OSC protocol's set of messages are introduced, or that a particular address-space structure is adhered to. Second, query solutions are inherently volatile: all namespace discovery necessarily occurs during runtime. Mappings set-up are consequently also volatile, a major obstacle when users have to define complex mappings between a large number of systems, arguably a common use case for OSC.

More query solutions have since appeared [9], [10], [11], but none has seen any significant level of adoption, with the vast majority of OSC capable systems today implementing no support for query functionality at all.

## 3. A DIFFERENT APPROACH FOR DESCRIBING NAMESPACE AND STATE

Without directly replacing any of the abovementioned solutions, we here address the namespace discovery problem with an approach that does not require extending the OSC protocol, and without volatility: we introduce two XML schemata, one for the description of an OSC server's namespace, and one for the description of its state. We therefore also address the long-standing call on the opensoundcontrol.org website, for machine-readable OSC schemata [12].

While our suggestion of using namespaces may seem familiar, an important distinction is that we do not at all intend for these namespaces to ever be standardized. Each simply describes one server, in one context of use. Moreover, for one particular server (e.g. a synthesizer), we expect and even condone the existence of several namespaces, either because the server has evolved (e.g. new capabilities through a firmware update), or because a user simply had no use for some aspects of the servers' functionality, and so did not include them in the namespace specification. For this reason, we also predict for the use of version fields per tag, to aid in differentiating between evolutionary iterations of the same namespace.

Each schema file also begins with a compulsory tag which specifies whether it is an OSC-Namespace or OSC-State file, and a version field to specify which version of the schema the file follows (the schemata in this article are both v1).

```
<OSC-Namespace Version="1">
<Node ID="SubtractiveSynth_1" AP="Synth_1" V="3" Continuity="Continuous" Direction="Bi">
  <Node ID="Oscillator_1" AP="Osc_1" V="3" Continuity="Continuous" Direction="Bi">
    <Node ID="Freq_OSC1" AP="Frequency" V="3" Continuity="Continuous" Direction="Bi">
      <TTS ID="TTS_OSC1_Freq" V="1">
        <TT ID="TT_F_OSC1" V="1" Tag="f" Min="0" Max="20000" Default="440" Unit=”Hertz”>
      </TTS>
    </Node>
  </Node>
  <Node ID="Filter_F1" AP="Filter_1" V="3" Continuity="Continuous" Direction="Bi">
    <TTS ID="TTS_OSC1_Filter" V="1">
      <TT ID="TT_OSC1_F1_Cutoff" V="1" Tag="f" Default="1.0">
      <TT ID="TT_OSC1_F1_Resonance" V="1" Tag="f" Default="0.0">
    </TTS>
  </Node>
  <Node ID="Apply_Preset" AP="Apply_Preset" V="3" Continuity="Discreet" Direction="In">
    <TTS ID="TTS_AP" V="1" Description="Apply 's' immediately">
        <TT ID="TT_P " V="1" Tag="s" Defalut="Preset_1">
    </TTS>
    <TTS ID="TTS_AP_Interp" V="2" Description="Interpolate to 's' over 'f' seconds">
        <TT ID="TT_P_Interp" V="1" Tag="s" Default="Preset_1" Trigger=”1”>
        <TT ID="TT_P_InterpTime" V="2" Tag="f" Min="0" Max="1" Default=".5" Trigger=”0”>
    </TTS>
  </Node>
</Node>
</OSC-Namespace>
```

**Figure 1: Minimal example of a file following the OSC-Namespace XML schema**

## 3.1 OSC-NAMESPACE SCHEMA

The OSC-Namespace schema consists of only three XML tags (see Figure 1 for an example namespace): Node, TT and TTS.

**Node** tags represent one node in the OSC address tree. So, for example, the address "/Synth_1/Oscillator_1/Frequency" therefore needs three nested Nodes to be represented. Node tags only have one compulsory attribute, "AP", which is the OSC address-part of the Node. They may then have any number of optional attributes. We deem the most useful to be:

- "Continuity" (Discreet, Continuous): states if the message to be interpreted as an event or not. For example, a button press message is a new event, even if it has the same value as the previous button press message, and is thus labelled Discreet. A message communicating the output sound volume level however, only makes sense as an event if the value to be sent is different from the previous one, thus labelled Continuous.
- "Direction" (in, out, bi): Is the message one the server responds to, transmits to, or both? A musical controller might transmit notes, but not respond to received ones (in). A synth might respond but not transmit (out). A toggle-button might transmit when pushed, but also change state when a value is received (bi).

**TT** tags represent one node's individual type-tag. They have the compulsory attribute of "Tag", which is a single character representing the OSC type-tag, e.g. 'i', 'f', 's', etc. Our recommended optional attributes are:

- "Default" holds a value with which to initialize a new instance.
- "Min" & "Max" define the range of numerical values expected. If either or both are omitted, plus and/or minus infinity should be assumed.
- "Trigger" (1, 0): should a new value set to the type-tag, result in an entire new message being triggered? Usually only one of the values should be set to trigger, leftmost or rightmost, depending on whether the list of parameters should be treated left to right, or right to left (Pure Data style or Max/MSP style).
- "Unit" is a string description of the expected measurement unit.
- "Clip": if a numerical value reaches the minimum or maximum, should it clip or be allowed to go beyond these?

**TTS** stands for Type Tag String. TTS tags are used to group several type-tags together. Their use is compulsory. They have no compulsory or recommended attributes.

The OSC-Namespace schema's syntax is such that each child node can be copied out of the address space where it appears, and still stand as a valid namespace specification. So, in Figure 1, node "Oscillator_1", were it to be copied out along with its sub-nodes, forms a valid namespace specification also outside of the tree-structure within which it appears in the figure.

Also, all tags (Node, TT & TTS) can have the following optional attributes:

- "ID" is meant to hold an ID String, unique within the tree structure level, to represent the object to the host. It is strongly recommended to use ID fields throughout, at the very least for Node tags, as this facilitates absolute associations between saved states and nodes.
- "V" is a version number to be incremented every time the element is changed, useful when choosing between conflicting versions of two files. If you increase a version number, the version numbers of all its parent nodes preceding it in the tree hierarchy should most usefully also be increased. Note this refers to the version of the field as decided by the maker of the file. The schema version, deciding how the file is parsed, is the field "Version".
- "Description" is intended for a verbose, free-text description of the node.

So, a TTS tag which uses all three optional attributes would look as follows:
<TTS ID="TTS_AP" V="1" Description="Apply 's' immediately">.

## 3.2 OSC-STATE SCHEMA (A PRESET)

The OSC-State schema too consists of three XML tags (see Figure 2 for an example):

**Node_State** delineates a node's state. It has no compulsory attributes, but using an "ID" is recommended, as is specifying a Node ID Path, "NodeIDP", to associate the state with an OSC namespace hierarchy's root node.

**Tuple** describes an OSC destination for the state values. Each Node_State can contain any number of Tuples. These are not hierarchically organized to reflect the tree structure of the OSC-Namespace whose state they describe. Instead the tree structure is exploded sequentially, thus enabling that also partial sets of state can be stored. Per Tuple, the compulsory attribute is "NodeIDP" or "AP", holding the full address to the node for which the values held in the Tuple are intended. If it is set to "NodeIDP", it is expected that "ID" tags have been set in the OSC-Namespace hierarchy referred to, and the "NodeIDP" path directs to the node following the path of ID's. If it is set to "AP", instead it refers to the OSC AP of the target node.

**Value** holds a single value. Each Tuple can hold any number of Values, each of which has two compulsory attributes: "Tag" holding the OSC type-tag, and "Val" holding the value.

Note that specifying a NodeIDP in a Node_State or in a Tuple does not guarantee correct association, as the node pointed to may not be found in the current state of the system into which the state file is loaded. Also note the distinction between NodeIDP, the AP pointing to a specific node within a

```
<OSC-State Version=”1”>
<Node_State ID="Cool Synth preset 1" NodeIDP="/SubtractiveSynth_1">
  <Tuple NodeIDP="/SubtractiveSynth_1/Oscillator_1/Freq_OSC1">
    <Value Tag="f" Val="440.0"/>
  </Tuple>
  <Tuple AP="/Synth_1/Filter_1" V="3">
    <Value Tag="f" Val="1.0" />
    <Value Tag="f" Val="0.5"/>
  </Tuple>
</Node_State>
</OSC-State>
```

**Figure 2: Minimal example of a file following the OSC-State XML schema**

namespace schema, and the AP stored in the node for which the values are destined. Depending on the application, these could be the same, but there are also cases where it is convenient that they differ, hence providing for using either.

## 4. DISCUSSION/CONCLUSION

Using such schemata, namespace information is no longer volatile, making for reliably reproducible mappings between sessions. Moreover, any OSC capable system can be addressed, old or new, as no extension to the OSC protocol is necessary.

If schemata such as ours gain adoption, all application-agnostic OSC-servers will be able to seamlessly share data regarding the OSC-servers they interact with. We therefore believe that through using our schemata, interconnecting several OSC-capable systems becomes more straightforward.

Wide adoption would imply that a user only needs to download namespace files for his/her OSC-servers. In case no existing definitions are found, since the format is minimal and straightforward, it is relatively simple to create and upload them for others to build on.

Our proposal is fundamentally different from how many OSC-capable applications currently operate (e.g. TouchOSC [13], Lemur [14]): we insist on maintaining exhaustive descriptions of all input and output parameters of each OSC-server type that can be connected to, rather than require users to enter each message's AP, type-tags and metadata manually, one by one, at every instance of their use. Our solution may require more work once up front, but it pays off quickly as it requires no repetition of the effort.

While our design may be criticized for not providing syntax for dynamic address spaces, we argue that this is in fact part of its strength, and a deliberate choice made after careful consideration. Discovery in a dynamic address space is necessarily a task performed in real-time, and so best served by a query system, specifications for which there are already plenty. Furthermore, syntax for dynamic address spaces would introduce additional software complexity and bring back the volatility problem previously outlined, with no added benefit that we are aware of. Our solution is instead purposefully directed to the still very large number of OSC use contexts which do not require dynamic address spaces. Finally, there is nothing to preclude the possibility of using our solution alongside a query system, thus combining the benefits of both.

The use of these or similar schemata provides functionality that has been missing to date. It also places few restrictions on their use, aiming for their adoption in a wide range of use contexts, both predicted and unknown, in the spirit of the original OSC protocol. Two software applications are currently in advanced stages of development, which make use of these schemata. First, an updated version the open-source, modular visuals synthesis program Mother [15], is released to coincide with the publication of this article, including functionality to automatically generate OSC-Namespace files describing its state. Second, a significantly updated version of the OSC mapper/controller application Mediator [6] is to be released in the near future, which can load and save both OSC-Namespace and OSC-State files.

We have created a repository to hold documentation, and future revisions of the schemata (http://code.google.com/p/osc-namespace-and-state-schemata/). Readers of this article are encouraged to refer to that address before implementing use of the schemata, for access to the latest updates and developments. The authors believe a more widespread adoption would be beneficial for the community of OSC users.

## 5. REFERENCES

[1] M. Wright, "OpenSound Control: A New Protocol for Communicating with Sound Synthesizers," *Proc. 1997 Int. Comput. Music Conf. ICMA*, pp. 101–104, 1997.

[2] M. Wright, A. Freed, and A. Momeni, "OpenSound Control: state of the art 2003," in *Proceedings of the 2003 conference on New interfaces for musical expression*, 2003, pp. 153–160.

[3] "fabb/SynOSCopy," *GitHub*. [Online]. Available: https://github.com/fabb/SynOSCopy. [Accessed: 27-Jan-2014].

[4] "openmediacontrol Home - openmediacontrol." [Online]. Available: http://openmediacontrol.wetpaint.com/. [Accessed: 06-Feb-2011] Now defunkt.

[5] J. Malloch, S. Sinclair, and M. M. Wanderley, "A Network-Based Framework for Collaborative Development and Performance of Digital Musical Instruments," in *Computer Music Modeling and Retrieval. Sense of Sounds*, R. Kronland-Martinet, S. Ystad, and K. Jensen, Eds. Springer Berlin Heidelberg, 2008, pp. 401–425.

[6] I. Bergstrom, A. Steed, and B. Lotto, "Mutable Mapping: gradual re-routing of OSC control data as a form of artistic performance," in *Proceedings of the international conference on Advances in computer entertainment technology*, Athens, Greece, 2009, pp. 290–293.

[7] E. R. Miranda and M. M. Wanderley, *New Digital Musical Instruments: Control And Interaction Beyond the Keyboard*. AR Editions, 2006.

[8] M. Wright and A. Schmeder, "A Query System for Open Sound Control," 2004. [Online]. Available: http://cnmat.berkeley.edu/publications/query_system_open_sound_control. [Accessed: 06-Feb-2011].

[9] M. Habets, "Schema System for Open Sound Control Query System," 2005.

[10] T. Place, T. Lossius, A. R. Jensenius, N. Peters, and P. Baltazar, "Addressing classes by differentiating values and properties in osc," 2008.

[11] A. Eales and R. Foss, "Service Discovery Using Open Sound Control," in *Audio Engineering Society Convention 133*, 2012.

[12] "OSC Schemas: standardized OSC Address Spaces plus their semantics | opensoundcontrol.org." [Online]. Available: http://opensoundcontrol.org/osc-schemas-standardized-osc-address-spaces-plus-their-semantics. [Accessed: 29-Jan-2014].

[13] "h e x l e r . n e t | TouchOSC." [Online]. Available: http://hexler.net/software/touchosc. [Accessed: 29-Jan-2014].

[14] "Lemur – Liine." [Online]. Available: http://liine.net/en/products/lemur/. [Accessed: 29-Jan-2014].

[15] I. Bergstrom and B. Lotto, "Mother: Making the Performance of Real-Time Computer Graphics Accessible to Non-programmers," in *(re)Actor3: The Third International Conference on Digital Live Art Proceedings*, 2008, pp. 11–12.