



Audio Engineering Society

Convention Express Paper 264

Presented at the AES 157th Convention
2024 October 8-10, New York, NY, USA

This Express Paper was selected on the basis of a submitted synopsis that has been peer-reviewed by at least two qualified anonymous reviewers. The complete manuscript was not peer reviewed. This Express Paper has been reproduced from the author's advance manuscript without editing, corrections, or consideration by the Review Board. The AES takes no responsibility for the contents. This paper is available in the AES E-Library (<http://www.aes.org/e-lib>) all rights reserved. Reproduction of this paper, or any portion thereof, is not permitted without direct permission from the Journal of the Audio Engineering Society.

Implementation of ADM-OSC v1.0

M. Delquignies¹, J.Gaudin², S.Thevenard³, H. Larin⁴, M. Zbyszynski⁵ and L.Zwicker⁶

¹ d&b audiotechnik, mathieu.delquignies@dbaudio.com

² Meyer Sound Laboratories, jgaudin@meyersound.com

³ Adamson, sylvain.t@adamson.ai

⁴ FLUX::, huqo.larin@harman.com

⁵ L-Acoustics, michael.zbyszynski@l-acoustics.com

⁶ Lawo, lucas.zwicker@lawo.com

ABSTRACT

Spatial and immersive audio has become increasingly mainstream, presented in concert halls and more recently through music streaming services. There is a diverse ecosystem of hardware and software, controllers, and renderers, in both live and studio productions, that would benefit from a real time standardized communication protocol to interoperate. By leveraging Audio Definition Model (ADM) and Open Sound Control (OSC), we defined ADM-OSC: a real time, lightweight and interoperable protocol that can be easily implemented in audio software, plugins, game engines, consoles, and controllers.

Now we see a consensus that brings us to the first publication of ADM-OSC specification v1.0 that can be found in this document, which is also intended as a guide for developers interested in implementing it.

1 Introduction

ADM-OSC has been designed to solve real problems for live and broadcast sound producers. Since 2019, a growing workgroup of industry stakeholders from live music and broadcast domains has gathered to exchange needs and experiences from real life production cases.

Those companies have already expressed interest or have implemented ADM-OSC: Atlas, Adamson, Amadeus Acoustics GmbH, BBC, d&b audiotechnik, DiGiCo, Dolby, FLUX::SE, FollowMe, Grapes3D, Holophonix, L-Acoustics, Lawo, Merging Technologies, Meyer Sound, Naostage, Radio-France, Steinberg etc.

This paper begins with a brief description of ADM and OSC, and why these standards were chosen. It will then cover basic recommendations for

communication ports and frame rates, before going into detail about ADM-OSC 1.0. Message types and coordinate conventions are described, along with reasoning and typical use cases that inform specific messages. We will then report on the first ADM-OSC Plug Fest and status of current implementations. Finally, ideas for future versions of the standard are discussed and a summary table of ADM-OSC 1.0 messages is presented in Appendix 1.

1.1 What is Audio Definition Model (ADM)

The Audio Definition Model (ADM) was first published by the European Broadcast Union (EBU) in 2015 as a standard representation of audio metadata [1]. The goal of ADM is to support a broad range of use cases that include spatial and immersive audio, as well as interactive personalization and accessibility features [2]. ADM can be used to represent channel-based, scene-based, and object-based audio. It is defined by the EBU in ITU-R BS.2076 [3]

It is an open standard that all content producers can use to describe Object Based Audio (OBA, cf [4]) content. As an open model, it can be interpreted by anyone as encoders evolve, like Dolby's AC-4 and Fraunhofer's MPEG-H.

More specifically, to remain agnostic from the technology, the focus of ADM-OSC is on the OBA contents. The reference documentation of the ADM defined Objects metadata can be found here [5].

1.2 What is S-ADM

ADM is not intended for live production and streaming applications, where we need a method to transmit metadata about an audio program in a real-time streaming manner.

So, the European Broadcasting Union (EBU) added the S-ADM protocol, established in 2019 as open standard [6], that serialized the ADM metadata in time and can be carried alongside the related PCM audio tracks. The metadata defined in S-ADM can be static for an entire program or it can be dynamic, adding or changing dialog and/or AD tracks for the viewer to choose from, or varying the position or level of various audio objects during production.

S-ADM generator will generate a series of S-ADM frames containing ADM based on a predefined template. For example, the ADM program names, and content and object elements would have been defined in the template. The template will leave some values undefined (or set to default values), that will allow them to be populated by values from the ADM-OSC messages. As the ADM-OSC messages will be generated in real-time from the production tool, this will allow the S-ADM metadata to be updated in real-time from these messages.

Again, S-ADM does not transmit rendering parameters to tell you how to do the rendering, but rather describing what the renderer ought to achieve. So, it is very different in its purpose from usual remote protocols like rtp-MIDI or OCA/AES70 for example, that transmit specific technical parameters.

1.3 What is OSC

Open Sound Control (OSC) is a data transport specification [7] that was proposed in 1997 for sharing music performance data (gestures, parameters and note sequences) between musical instruments (especially electronic musical instruments such as synthesizers), computers, and other multimedia devices. OSC is sometimes used as an alternative to the 1983 MIDI standard, when higher resolution and

a richer parameter space is desired. OSC is "transport agnostic", so messages can be transported for example by SLIP message over USB, or across the internet and within local subnets using UDP/IP and Ethernet.

There are several open-source implementations that simplify developers' adoption. The OSC 1.0 specification has been published in 2002 [8]. It is not a standard, but it defines type tags, semantics, address patterns etc.

It is interesting to note that in 2007 a standardized namespace within OSC called SYN, for communication between controllers, synthesizers, and hosts, was proposed [9], but missed OBA messages.

Specifically, in live applications and musical controllers, we see **UDP OSC** messages datagrams in use, because of its real time feature, and ability to use Unicast, Multicast or Broadcast.

UDP is much simpler and faster than TCP, the trade-off is it doesn't guarantee that all data is successfully transferred. With UDP, data is sent to any device that happens to be listening, but it doesn't care if some of it is lost along the way. This is one of the reasons why UDP is also known as the "fire-and-forget" or "best effort" protocol.

ADM-OSC can also be sent over TCP when an emphasis on security of transmission supersedes speed. With mostly live environments, this publication focuses on continuously changing parameters in this real time interoperable specification. Typically, objects positions that are integrated in the musical writing or follow the actor's movements in the staging are sent several times by second, making UDP datagrams the preferred choice.

1.4 Why ADM-OSC

The OSC specification does not define specific messages; these vary depending on the implementation. There are currently many audio devices implementing immersive data in OSC, but there is not necessarily agreement on how similar data is represented.

As immersive audio gains traction across various industries, from music streaming to gaming, and from live sound to broadcasting, the Audio Definition Model (ADM) is becoming a popular standard for

metadata. This includes S-ADM for broadcast and ADM BWF or XML files for studio use.

The ADM-OSC workgroup was formed to bridge the gap between immersive live and studio ecosystems. It includes leading developers and manufacturers who aim to facilitate the sharing of audio object metadata across different environments, from studios to broadcasts to live performances.

Since its initial draft implementation, ADM-OSC has been supported by various audio industry tools, including live sound rendering engines, digital audio workstations (DAWs), controllers, live tracking systems, audio mixing consoles, and media server solutions. It is currently being deployed in both live and studio productions, with an increasing interest from technology developers wanting to join and implement this standard.

We clearly saw that most real time metadata needs were positional, so ADM-OSC 1.0 aims to provide a basic interoperability layer between Object Editors and Object Renderers for objects positions as the main focus.

2 Conventions

2.1 Default port

As ADM-OSC typically uses UDP protocol, we found it useful to define some default ports that we recommend. As it is sent in parallel to the audio signal, it is assumed that the ADM-OSC data follows the same transmission direction.

The proposal is to use **4001** [10] for one way communication (so, default for senders and receivers), **4002** for two ways (default return port). Those ports should be user editable if needed.

2.2 Maximum messages rate

Some renderers may have a limited rate to avoid overload, so it is recommended to implement a feature to set the sender's rate.

S-ADM rate is usually half video frame rate, so approximately 20 ms or 50 Hz.

Also, we recommend strategies to manage rate in renderers:

- too high: avoid flooding by decimating,
- too low: add interpolation to smooth changes.

The master profile for Dolby Atmos ADM [11] recommends that the sampling period be “less than 20 ms”, although sampling is optional if the parameter does not change. ADM provides messages to specify interpolation times, but the Dolby profile does not support them.

Interpolation has not been implemented as part of ADM-OSC 1.0.

2.3 Queries and bi-directional communication

The OSC protocol is unidirectional, so the commands should be considered as SET from a sender to a receiver. A particular device might also be interested to GET the state of a particular parameter in another device. To do so, it should send a command without any arguments. The receiver should answer back to this IP with the data.

Example: sending `/adm/obj/4/xyz` to a device should trigger a reply like `/adm/obj/4/xyz -0.9 0.15 0`.

2.4 Non supported commands

It is very likely that a specific implementation of ADM-OSC will not support every message in the specification. For example, a controller might send only (x,y) positions.

As third parties do not have to implement all the ADM-OSC commands, a receiver should also be able to handle unsupported messages nicely, ignoring them without issues. The same would apply if a device receives renderer-specific commands, not in the ADM-OSC namespace: it should not hinder its operation.

In any case, OSC Type tag should be respected to avoid type mismatch error handling.

Examples: receiving only `/adm/obj/x` and `/adm/obj/y`, not `/adm/obj/xyz` nor `/adm/obj/aed`.

Sending `/adm/env/change 12.4` should consider converting “12.4” as string.

Manufacturer's documentation should clearly state which parameters are relevant to specific renderers.

2.5 Name space convention

To lower communication bandwidth usage, we propose to use short names.

All ADM-OSC commands should start by `/adm`.

We can differentiate objects, environment and listener descriptors namespaces:

- `/adm/obj` relate to sound object descriptors
- `/adm/env` relate to the listening environment descriptors
- `/adm/lis` relate to the listener descriptors

2.6 Coordinate system convention

The positional commands allow different axes to be specified in the coordinate attribute.

A polar coordinate system, which uses azimuth, elevation and distance, and the Cartesian coordinate system, which uses x , y , and z . The azimuth and elevation angle may also be used for the equation sub-element for scene-based audio. To ensure consistency when specifying positions each of the polar axes should be based on these guidelines:

From this point, without explicit reference to listener position and orientation, the **origin is in the center (0,0,0) by default**, where the sweet-spot would be (although some systems do not have a sweet-spot, so the center of the space should be assumed).

The “**toward the front**” direction (e.g. stage or screen) is the green y vector on all figures.

a (Azimuth) – angle between -180° and 180° , in the horizontal plane with 0 degrees as straight ahead, and positive angles to the left (or anticlockwise) when viewed from above.

e (Elevation) – angle between -90° and 90° , in the vertical plane with 0 degrees horizontally ahead, and positive angles going up.

d (Distance) – a normalized distance, where 1.0 is assumed to be the default radius of the sphere. Because different rendering situations require a different expression of distance, this parameter is configurable (see below).

Cartesian coordinates, which are also used for object-based audio, are supported by using x , y and z as the coordinate attributes. It is recommended that normalized values be used here, with the origin being the center of the cube and x , y , z between -1 and 1 .

The direction of each axis should be:

- x left to right, with positive values to the right.
- y front to back, with positive values to the front.
- z top to bottom, with positive values to the top.

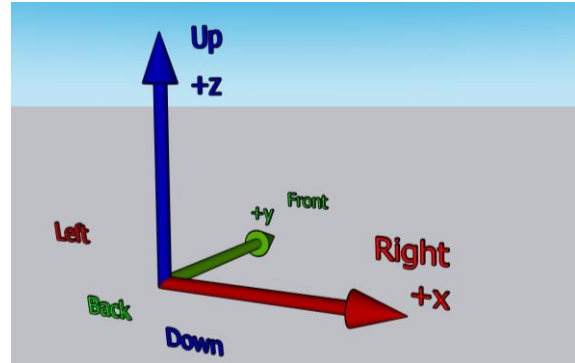


Figure 1. Cartesian 3D axis convention

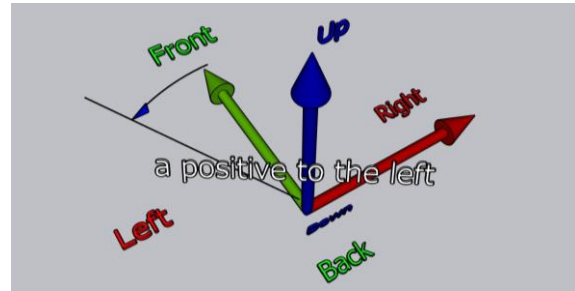


Figure 2. Azimuth angle orientation convention

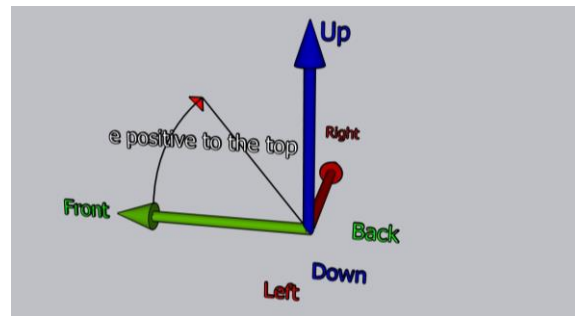


Figure 3. Elevation angle convention

To convert the coordinate system, Euler trigonometry can be used to represent the polar sphere in cartesian coordinates. The equations are provided in ITU-R BS.2127 section 6.8 [12].

$$x = -d \cdot \sin\left(\frac{\pi}{2} - e \frac{2\pi}{360}\right) \cdot \sin\left(a \frac{2\pi}{360}\right) \quad (1)$$

$$y = d \cdot \sin\left(\frac{\pi}{2} - e \frac{2\pi}{360}\right) \cdot \cos\left(a \frac{2\pi}{360}\right) \quad (2)$$

$$z = d \cdot \cos\left(\frac{\pi}{2} - e \frac{2\pi}{360}\right) \quad (3)$$

$$a = -\frac{360}{2\pi} \cdot \text{atan2}(x, y) \quad (4)$$

$$e = \frac{360}{2\pi} \cdot \text{asin}\left(\frac{z}{d}\right) \quad (5)$$

$$d = \sqrt{x^2 + y^2 + z^2} \quad (6)$$

To help conversions seamlessly, here are also open-source code examples on GitHub in SWIFT [13], in CPP [14] or JavaScript [15].

For full ADM compatibility, there is another recommended conversion approach in section 10.1 of ITU-R BS.2127 [12].

This conversion projects the sphere into a rectangular space that matches typical 5.1 speaker layout, where the front corners are at ± 30 degrees and the rear corners are at ± 110 degrees, so specific to this rendering context. Instead of having objects projected on sphere surface inside the cartesian cube, it projects objects on this cube surface and renormalizes distances. The reference implementation for this conversion can be found in the EBU ADM EAR renderer Python code on GitHub [16]. When converting ADM-OSC to S-ADM or ADM, this is the desired approach. However, this might not conform to user expectations in other situations, especially if the speaker layout isn't cuboid shape. The best approach is to clearly document how coordinates will be converted in a particular application and offer options if required by supported use cases.

2.7 2D only

Even if there is a trend to use more and more elevation effects, most often real life scenarios for large applications in concert venues or outdoor sound reinforcement have only one horizontal speaker layout, because of consistency thru audience, rigging possibilities, cabling, time and money to setup. The convention when the rendering is happening only in the horizontal plane is to use only (x,y) , and have the z axis, as the vertical dimension, ignored. We consider that 2D is “viewed from the top”:

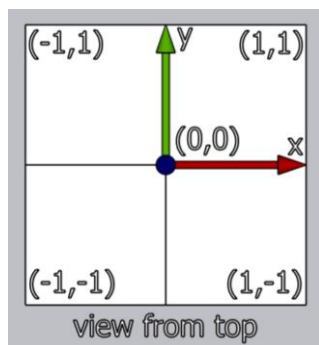


Figure 4. Cartesian 2D axis convention

Note that this convention is different from most game engine and computer graphics that use x for horizontal, y for vertical and z for depth, so as “side view”.

2.8 /obj/k/dist effect and normalization

Broadcast renderers like Dolby Atmos or MPEG-H render to a fixed, abstract volume. A distance of 1.0 is the “wall” of that volume and lower values are perceived as closer to the listener.

Other renderers, like game engines, represent a physics-based world, and the notion of source distance relates to a physical unit, such as meters. In L-ISA, for example, 0.0 is taken as the distance to the loudspeakers, and larger values are further away. Typically, the sense of distance is simulated by gain attenuation, filtering, and/or change in the object's size. These implementation details are the domain of the renderer and not specified in ADM or ADM-OSC.

One issue with distance-based effects is a singularity when distance is 0.0. Many renderers, such as Unreal and Spat Revolution, solve this by specifying a “volume of reference” around the origin where gain attenuation is not applied [17].

The proposal is:

/adm/obj/k/dRef This is the boundary where attenuation vs distance effect is applied. Below this value, there is no change in object gain. The range is 0 to 1, default is 1.0, meaning the whole volume is without gain attenuation in ADM standard unless otherwise specified.

/adm/obj/k/dMax For physics-based rendering, this is the distance in meters that corresponds to the normalized distance of 1.0.

Practically speaking, distance simulation in a binaural system has different possibilities than a large loudspeaker system. A source signal may also have distance cues incorporated into it, depending on microphone placement, etc.

Also, single listener renderers can define a relative object distance, but not a physics-based scene with several listeners positions as seen in multi-players game engines or live large audience renderers like d&b soundscape. For this reason, ADM-OSC distance parameters are not as straightforward for interoperability as (x,y,z) objects positions parameters, which leads us to 6 or 9 DOF */lis* parameters proposal.

2.9 /obj/k/gain normalisation

Gain is expressed as a linear ratio, not in dB.

So, it means that:

- 0.0 gain = silence
- 1.0 gain = unity (0 dB)
- Higher gain values are possible, but not guaranteed to be supported by a renderer. The renderer will clip to the max supported value.
- ADM does not support gain > 1.0

3 Real time production events

There is a need for some features that are not directly meaningful to ADM, but that we found helpful in the production workflow to be transmitted alongside objects positions.

3.1 /env Program Changes

This is a user interaction or show control event that describes a change in the global scene or environment. It can be useful to trigger snapshots of static parameters or even actions to or from other domains like lighting, peripheral devices etc.

This allows renderers to recall renderer specific settings, like MIDI program Changes do.

The proposal is to send this as a string argument:
`/adm/env/change s`

The corresponding query command (without argument) `/adm/env/change` should answer back the last value sent.

Examples:

`/adm/env/change day`

`/adm/env/change night`

`/adm/env/change 127` (please note that to avoid type tag issue, "127" must be sent as string)

3.2 /lis Head Tracking

Realizing there are more and more headphones renderers with headtracking use cases.

The proposal is to unify 6 DOF parameters for listener position and orientation.

The proposal is:

`/adm/lis/xyz f,f,f`

`/adm/lis/ypr f,f,f`

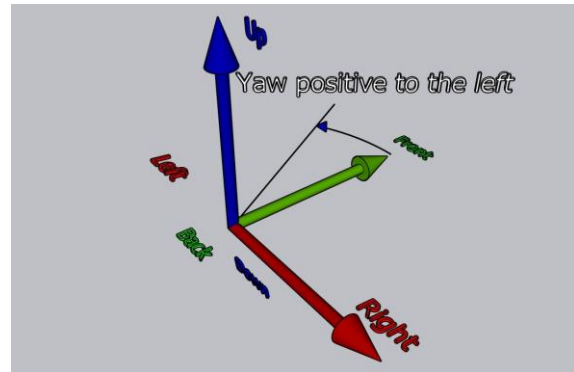


Figure 5. Yaw orientation convention

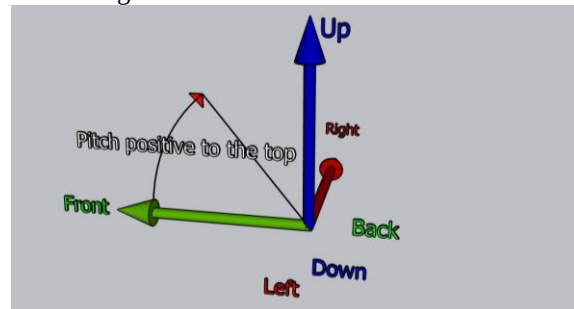


Figure 6. Pitch orientation convention

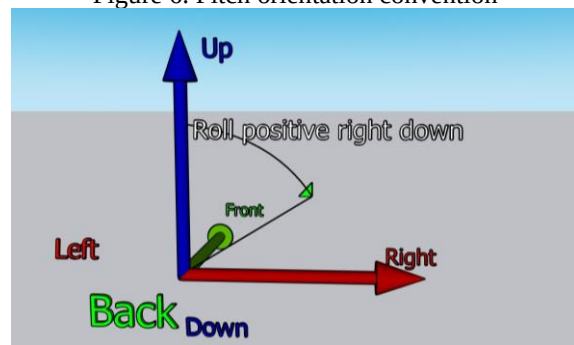


Figure 7. Roll orientation convention

The axis convention is shown above: yaw following the z axis, pitch following the x axis, and roll the y axis.

- *y* (Yaw, right -> left --- where rotating right is negative)
- *p* (Pitch, down -> up --- where rotating down is negative)
- *r* (Roll, down left -> up -> down right --- where rotating down left is negative)

It is a key feature to allow rendering a scene from allocentric (xyz physic-based position of objects) to egocentric (*aed* relative to single listener position) renderers and vice versa.

This is not currently supported in ADM, but may be in the future, as we acknowledge more and more

allocentric object-based audio in live sound, 3D games, AR and VR, and interactivity where the listener can freely move in a virtual physic-based environment.

If this information is not communicated, the default value could be that listener position is (0,0,0) and orientation (0,0,0), i.e., in the center of cube/sphere transformation and orientated forward, along the y axis (green vector in figures).

4 ADM-OSC Plug Fest

We regularly organize Plug Fests to test or demonstrate ADM-OSC functionalities as well as interoperability between devices or software from different manufacturers, or editors that implement Those hands-on sessions open discussions about implementation approaches, use cases and priorities. They are also a good opportunity to get to know each other, have a better understanding of the needs specific to different applications, tools, and markets, talking and building groups of interest and field of expertise.

The goal is to identify and understand issues, document findings and decide the next steps for this working group, but also more publicly on the ADM-OSC GitHub [18].

5 Testing

As ADM-OSC is not a standard but an open specification, there is no “compliance” or “certification” process, this helps also in beta testing

improvements and documenting third-party implementations.

The workgroup contributed also by providing functionality and performance testing tools, also on GitHub ADM-OSC repository [18].

One is a standalone application that will behave like a sender or a receiver. It features a 3D view that can be locally controlled and send data to test a receiver configuration or receive to test a sender configuration. If a message is badly formatted, the application will attempt to provide a message trying to help improve data sent or an eventual issue with the address or arguments.

Another is also a Chataigne open-source module, that may help building custom bridges to other protocols and monitoring with integrated logger.

Performance testing can also be done using provided Python Test module, these scripts will try to stress test transmission with one device.

6 Implementation matrix

We keep a centralized table with actual implementations that have been tested during the ADM-OSC Plug Fest on GitHub so you can check what any third has implemented [19].

7 Open discussions for 1.x

There are already discussions on further features in the specification. You can follow and participate in them by the GitHub Issues [20].

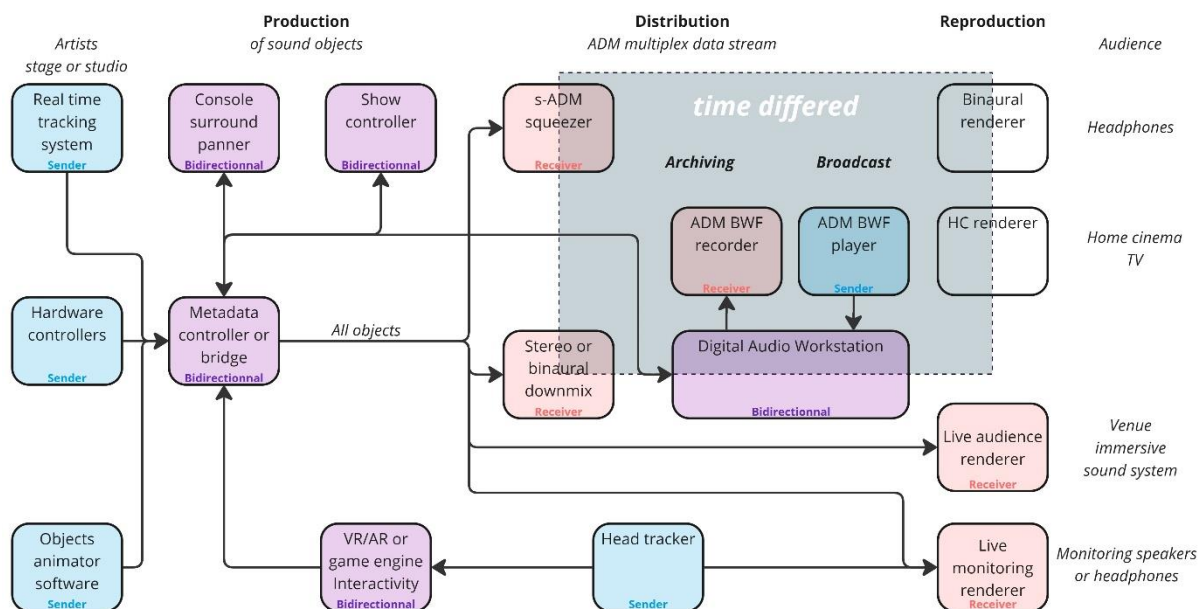


Figure 8. shows a few examples where metadata could be shared by ADM-OSC.

Here is a curation of some open topics:

7.1 */obj/k* Touch and release interactions

In ADM, there could already be several interactive parts, like changing foreground/background levels, languages, or storylines.

In production and live events, we see systematically producer's interactions on objects. Typically, a hardware controller is touched to overwrite an existing parameter, as in DAW or mixing desk automation switching to playback to record and back. Also, we often see several people or thirds that interact with metadata, like an artist on stage playing objects position automations from Ableton Live, a producer using an iPad to take control, and a sound engineer using a surround panner on a mixing desk to override those previous settings. They can all send ADM-OSC to the renderer, so we need to define a priority management.

This feature is also very helpful in bi-directional context to avoid infinite loops of metadata's changes.

A first idea was to specify "touch" and "release" specific commands.

Examples:

/adm/obj/4/touch

/adm/obj/4/release

But we need to be sure it is received along the right object property changes, and there is a risk of sending this as separate commands, as UDP OSC does not warrant they will arrive simultaneously. Also, some commands can be lost, so it would be necessary to send those in a redundant manner. One option would be to use advanced OSC features like Bundles or timestamp, but we do not see those features broadly implemented currently.

The proposal instead is to add an optional argument to the existing commands.

Example with gain, command becomes:

/adm/obj/k/gain f, i

This can be extended to any previously defined commands.

No additional parameter means 0 or lowest priority as default (i.e. "release").

Adding an argument starting to 1 set it as user interact with higher priority (i.e. "touch").

Using an integer gives a full OSC 1.0 compatibility (no Boolean type) and possibility to even define several levels of users (e.g. musician = 2, DAW = 1,

sound engineer mixing desk surround panner=3, etc.) to sort out what the renderer or squeezer should do if it receives positions from several transmitters.

Examples:

/adm/obj/4/xy -0.9 0.15

Some low priority (playback system) sends this position. Same as: */adm/obj/4/xy -0.9 0.15 0*.

/adm/obj/4/xy 0.5 0.15 1

User with priority 1 touch an XY pan controller. So, the receiver can overwrite the data and set this as a new position.

/adm/obj/5/gain 0.5 2

User with priority 2 override the object gain setting send by others with lower priority.

7.2 *Obj/k/w* or */widthDeg*?

"Width" describes the perceived size of a sound object. Maybe this parameter is more relevant with multichannel objects, as spreading a single mono sound to several loudspeakers may not lead to hearing a sound object size change because of the precedence effect. At the time of writing, the ADM-OSC group does not see much implementation of multichannel objects, but mainly objects as one single channel.

The ADM specification has object parameters for width, height, and depth in both polar and cartesian coordinates systems, see [5]. This flexibility is not supported in the Dolby Atmos ADM profile, which states that these three values should always be equal. However, cinematic renderers might adjust the perceived size of an object according to distance; objects get larger as they get closer.

The experience of the ADM-OSC group has been that a "width" parameter may be a useful musical effect, but that effect does not need to be three-dimensional.

When mixing immersive audio for streaming broadcast, it is common to give the voice a bit more "width", for example, to help it hold its position in the mix. Or to "spread" an object on several loudspeakers to increase the coverage for large audience like in d&b soundscape, which doesn't give "width" effect.

For that reason, ADM-OSC currently only implements a */adm/obj/k/w* normalized message. This is a useful parameter, but there is no consensus across renderers what a "width" of 0.5 might mean or if it changes with distance. If interoperability is critical, width should be used cautiously.

Further discussion may lead to addition on this, typically to fully implemented and documented object size descriptors in 3D, polar and cartesian.

7.3 Dealing with multi-channel objects (and more)

A basic source/object might be one audio channel. That is how most of the renderer's work: each source is mapped to one mono audio input. However, there is the possibility to link two objects into a stereo object. Other environments might treat a whole bed as a single set. This topic came up when discussing how ADM-OSC might make it easier to control a group of objects, or higher-level objects, like stereo or multichannel.

This may lead to a new `/adm/bed/` name space addition. There are already correlated parameters defined in ADM, especially a "layout" property for "audioPackFormat" [5].

7.4 Broadcast-specific ADM parameters

The ADM standard contains many parameters that are not currently covered by ADM-OSC. The "0.5 Draft Specification" on GitHub describes several potential ADM-OSC messages relating specifically to broadcast use cases. As this information is not common in a typical sound production context, the ADM-OSC group is currently discussing whether these messages should be added to a future version of the standard or if they should be implemented elsewhere in the broadcast production chain. ADM-OSC 1.0 primarily defines "live" messages that have a high enough data rate that some packet loss would not be a problem. Defining large programs and templates is a different situation that might be better served by a more robust mechanism than OSC over UDP.

Another broadcast-specific area involves object importance. The EBU is currently finalizing an ADM Production Profile, which will provide a mechanism for specifying how the number of objects might be reduced depending on the technical limitations of a given transmission. These will include notation of object importance and hierarchy that could be supported in a future ADM-OSC standard.

8 Conclusions

Following years of development by multiple stakeholders addressing diverse, real-world use cases, ADM-OSC has matured and stabilized to the point where a 1.0 standard can be announced. The draft

standard has already been included in released products, and the implementation has been verified at plug fests in Paris (2023) and New York (2024). Plug Fest participants have included developers of audio software and plugins, hardware controllers, mixing consoles, and immersive audio renderers. Complex scenarios have been tested by Radio France broadcasts of tours and concerts, and BBC's 2023 Eurovision Broadcast [21]. There is increasing implementation in music preproduction tools, helping OBA interoperability with different renderers for musicians on tours.

As immersive audio practice continues to evolve, ADM-OSC can add more features to support future workflows of OBA productions. The ADM-OSC workgroup overlaps with the maintainers of ADM, and these two standards will develop in parallel. ADM-OSC will continue to be a set of recommendations with no certifying body. The most current documentation about the specification will be published online at:

<https://immersive-audio-live.github.io/ADM-OSC/>

New contributors to this project are welcome and invited to join the discussions on GitHub or contact the authors directly.

9 References

- [1] S. Füg, D. Marston, and S. Norcross, "The audio definition model—a flexible standardized representation for next generation audio content in broadcasting and beyond," in *Audio Engineering Society Convention 141*. Geneva, Switzerland, 2016.
- [2] "What is ADM ?," EBU, [Online]. Available: https://adm.ebu.io/background/what_is_the_adm.html.
- [3] "BS.2076 Audio Definition Model," ITU, 2019. [Online]. Available: <https://www.itu.int/rec/R-REC-BS.2076-2-201910-I/en>.
- [4] N. Tsingos, *Immersive Sound*, Ch. 8 Object-based audio, Routledge, 2017.
- [5] "ADM Guidelines - audioChannelFormat - Objects," EBU, [Online]. Available: https://adm.ebu.io/reference/adm_elements/audio_channel_format_objects.html.
- [6] "BS.2125 s-ADM," ITU, 2022. [Online]. Available: <https://www.itu.int/rec/R-REC-BS.2125/en>.
- [7] "Open Sound Control (OSC)," Stanford University, 2002. [Online]. Available: <https://opensoundcontrol.stanford.edu/index.html>.
- [8] "OpenSoundControl Specification 1.0," Stanford University, 2002. [Online]. Available: https://opensoundcontrol.stanford.edu/spec-1_0.html.
- [9] "SynOSC Wiki," 2007. [Online]. Available: <https://github.com/fabb/SynOSCopy/wiki>.
- [10] "ADMix Player," Ircam, March 2019. [Online]. Available: <https://forum.ircam.fr/projects/detail/admix/>.
- [11] "Dolby Atmos ADM Profile specification," Dolby, Jan 2022. [Online]. Available: <https://professionalsupport.dolby.com/s/article/Dolby-Atmos-ADM-Profile-specification>.
- [12] "BS2127 Audio Definition Model renderer for advanced sound systems," ITU, 2019. [Online]. Available: <https://www.itu.int/rec/R-REC-BS.2127/en>.
- [13] "SWIFT polar<>cartesian code examples," [Online]. Available: <https://github.com/Daniel-Higgott/ADMCoordinateConversion>.
- [14] "CPP polar<>cartesian code example," [Online]. Available: <https://github.com/ChristianAhrens/RemoteProtocolBridgeCore/blob/master/Source/ProcessingEngine/ProtocolProcessor/OSCProtocolProcessor/ADMOSCProtocolProcessor.cpp>.
- [15] "Javascript polar<>cartesian code example," [Online]. Available: <https://github.com/madees/ADM-OSC-Chataigne-Module>.
- [16] "Python polar<>cartesian code examples," [Online]. Available: https://github.com/ebu/ebu_adm_renderer/blob/master/ear/core/objectbased/conversion.py.
- [17] J.-M. Jot, R. Audfray, M. Hertensteiner, and B. Schmidt, "Rendering spatial sound for interoperable experiences in the audio metaverse: from Architecture to Automotive," I3DA, 2021.
- [18] "ADM-OSC GitHub," [Online]. Available: <https://immersive-audio-live.github.io/ADM-OSC>.
- [19] "ADM-OSC GitHub Implementation matrix," [Online]. Available: https://immersive-audio-live.github.io/ADM-OSC/html/implementation_matrix.html.
- [20] "ADM-OSC GitHub Issues for discussions," [Online]. Available: <https://github.com/immersive-audio-live/ADM-OSC/issues>.
- [21] M. Zbyszyński, H. Déjardin, D. Marston, and G. Le Nost, "ADM-OSC: an industry initiative for communicating object-based audio data," in *Immersive and 3D Audio: from Architecture to Automotive (I3DA)*, 2023.

Appendix 1 - ADM-OSC v1.0 Messages

osc address*		Type*	units	min	max	default	description	example
/adm/obj/k	/azim	f	degrees	-180.0	180.0	0.0	azimuth of sound object location.	/adm/obj/4/azim -22.5
	/elev	f	degrees	-90.0	90.0	0.0	elevation of sound object location	/adm/obj/4/elev 12.7
	/dist	f	normalized	0.0	1.0	1.0	distance from listener	/adm/obj/4/dist 0.9
	/aed	f, f, f	normalized	-1.0	1.0	0.0	3D compact format	/adm/obj/4/aed -22.5 12.7 0.9
	/w	f	normalized	0.0	1.0	0.0	object width	/adm/obj/4/w 0.5
	/x	f	normalized	-1.0	1.0	0.0	left/right sound object location	/adm/obj/4/x -0.9
	/y	f	normalized	-1.0	1.0	0.0	front/back sound object location	/adm/obj/4/y 0.15
	/z	f	normalized	-1.0	1.0	0.0	top/bottom sound object location	/adm/obj/4/z 0.7
	/xy	f, f	normalized	-1.0	1.0	0.0	2D compact format	/adm/obj/4/xy -0.9 0.15
	/xyz	f, f, f	normalized	-1.0	1.0	0.0	3D compact format	/adm/obj/4/xyz -0.9 0.15 0.7
	/gain	f	linear	0.0	1.0	1.0	apply a gain to the audio in the object.	/adm/obj/4/gain 0.707
	/dref	f	normalized	0.0	1.0	1.0	distance where start gain attenuation	/adm/obj/4/dref 0.707
	/dmax	f	meters	0.0			distance in meters that correspond to 1.0	/adm/obj/k/dmax 25
	/mute	i	integer	0	1	0	1 means “true” so muted	/adm/obj/2/mute 0
	/name	s	string		128 char.		object nice name	/adm/obj/1/name “kickdrum”
/adm/env	/change	s	string		128 char.	program changes	/adm/env/change “day”	
/adm/lis	/xyz	f, f, f	normalized	-1.0	1.0	0.0	listener position	/adm/lis/xyz 0 0.5 0
	/ypr	f, f, f	degree	-180	180	0.0	listener orientation	/adm/lis/ypr 0 30 0

* k is object number. Type tag are defined as OSC 1.0 specification: i=int32, f=float32, s=OSC-string