OPEN ACCESS Freely available online

M. Buffa, T. Burns, C. Ferrari, J. Kleimola, O. Larkin, J. Lebrun, S. Letz, Y. Orlarey, S. Ren, and S. Yi, "Ten Years of Web Audio Modules: Audio Plug-ins for the Web," *J. Audio Eng. Soc.*, vol. 73, no. 12, pp. 804–826 (2025 Dec.). https://doi.org/10.17743/jaes.2022.0238.

Ten Years of Web Audio Modules: Audio Plug-ins for the Web

MICHEL BUFFA,[1, 2, 3, *] **TOM BURNS,**[2, 4] **CESARE FERRARI,**[3, 5] **JARI KLEIMOLA,**[1]
(michel.buffa@univ-cotedazur.fr) (tom@burns.ca) (cesare.ferrari@gmail.com) (jari@webaudiomodules.org)

OLIVER LARKIN,[6, 7] **JEROME LEBRUN,**[1, 2] **STÉPHANE LETZ,**[8]
(contact@olilarkin.com) (jerome.lebrun@cnrs.fr) (letz@grame.fr)

YANN ORLAREY,[9] **SHIHONG REN,**[10, 11] **AND STEVEN YI**[12]
(yann.orlarey@inria.fr) (renshihong@hotmail.com) (stevenyi@gmail.com)

[1] *Université Côte d'Azur, Sophia-Antipolis, France*
[2] *Centre National de la Recherche Scientifique (CNRS), Sophia-Antipolis, France*
[3] *Institut National de la Recherche en Informatique et Automatique (INRIA), Sophia-Antipolis, France*
[4] *Sequencer Party, Ottawa, Canada*
[5] *Cmajor Software Ltd, London, England*
[6] *Ableton AG, Berlin, Germany*
[7] *iPlug2, London, England*
[8] *Générateur de Ressources et d'Activités Musicales Exploratoires (GRAME), Lyon, France*
[9] *Institut National de la Recherche en Informatique et Automatique (INRIA), Lyon, France*
[10] *Shanghai Conservatory of Music*
[11] *Shanghai Key Laboratory for Music Acoustic*
[12] *Independent audio developer*

Web Audio Modules (WAM) is a mature proposal for Web Audio plug-ins. Started in 2015, this open-source proposal reached version 2.0 in 2021. This paper reviews the background that led to the publication of the WAM standard and then details its characteristics. The second part is devoted to an analysis of the ecosystem that has grown up around the standard, focusing on notable achievements and applications, particularly those that have demonstrated their value within the web platform for which it was designed. Since its release, WAM V2.0 has been the subject of numerous publications, demonstrations, and presentations at various international conferences. As a direct result, there is now a large set of plug-ins and hosts in the WAM format, along with new online tutorials and tools to facilitate plug-in development (online IDEs, etc.), the vast majority being open source. An initiative called the WAM Community has also emerged to encourage the sharing of plug-ins, accompanied by an API and many online tools. New WAM hosts have been introduced, including two DAWs, collaborative applications, and compatible visual programming languages. Additionally, the WAM format now supports extensions, and this article describes some of them.

0 INTRODUCTION

In 2015, Jari Kleimola and Oliver Larkin created the first version of the Web Audio Modules (WAM) application programming interface (API)/Software Development Kit (SDK), allowing native audio plug-in code in C++ to be ported to run in the web browser using nascent web technologies [1]. Several years later, after the introduction of WebAssembly and the AudioWorklet, they updated the API and integrated it into an open-source audio plug-in framework iPlug2,[1] providing a solution for porting native audio plug-ins in their entirety to the web platform including digital signal processors (DSP) and user interface (UI) using the emscripten compiler [2]. This work was used to implement demonstration (demo) versions of commercial plug-ins, for example,[2] and was presented to the audio developer community at Audio Developer Conference 2018.[3] Around this time, they decided to join forces with others

*To whom correspondence should be addressed, email: michel.buffa@univ-cotedazur.fr

[1] https://iplug2.github.io
[2] https://www.surrealmachines.com/impact-demo/
[3] Oli Larkin - Bringing VirtualCZ to the web - ADC2018 (YouTube)



Fig. 1. WAM-Studio, an open-source web-based DAW.

working on interoperable web audio plug-ins and plug-in hosts to synchronize efforts toward the beginnings of an open standard. This initiative, originally called Web Audio Plug-ins (WAP) [3, 4], aimed to cover a wider range of use cases than the original WAM proposal, such as building audio plug-in–like entities using the existing Web Audio API nodes.

Since 2018, World Wide Web Consortium (W3C) web standards have matured. The appearance of WebAssembly, stabilization of WebComponents, support for AudioWorklets [5] in the Web Audio API, and continued evolution of JavaScript have all helped make professional-grade, web-based audio production a reality. There are now several commercial and open-source web-based digital audio workstations (DAWs), some of which support plug-ins such as BandLab,[4] Amped Studio,[5] SoundTrap,[6] OpenDAW,[7] and WAM-Studio[8][6, 7] (Fig. 1).

Taking into account these developments and the feedback received from developers over the years, this group further expanded and started working on a new version at the end of 2020. The project, called "Web Audio Modules v2.0" (WAM2), is open source and distributed via four GitHub repositories and as node package manager (NPM) modules (MIT License). From here, when the authors refer to "WAMs," the reference is to version 2.

Web-based audio solutions, in particular WAMs, offer several compelling advantages compared with native plug-in formats. Because WAMs are built using open web technologies and standards, such as Web Audio API, WebAssembly, and Web Components, they can run directly in the browser without requiring installation. This makes them platform-independent and instantly accessible from desktop or mobile devices. Furthermore, WAMs can be shared and deployed using just their uniform resource identifier (URI). Any compatible WAM host can dynamically load and run a plug-in by referencing its web address.

This URI-based architecture significantly lowers the barrier for plug-in distribution and encourages community-driven contribution and reuse. Developers can create and share WAMs in open repositories or representational state transfer (REST) endpoints, and users can integrate them effortlessly into existing WAM-powered applications, fostering an open and collaborative ecosystem for audio tool development.

To illustrate the practical potential of WAMs, Fig. 1 showcases WAMs operating within a web-based DAW. This example and many others presented throughout the paper (SECS. 7, 8, and 9), highlight real-world applications contributed both by the core WAM developers and a growing community of users and creators. These use cases serve to motivate and contextualize the more technical discussions that follow.

1 BACKGROUND CONTEXT AND TERMS

1.1 State of the Art: Native Plug-in Standards

Over the last 30 years, there has been a big shift in the technology used for music production and performance. One of the most significant changes is that there are now far fewer hardware devices and many software-based studios where mixing consoles, effect units, and tape machines have all been replaced by virtual equivalents.

One of the landmark moments in this technology shift was the introduction of Virtual Studio Technology (VST)[9] by Steinberg in 1996. This is a standard in which modular audio software devices (plug-ins) are deployed as dynamic libraries loaded by a host application that would typically be a DAW. This has become one of the primary means by which musicians access new sounds. In addition to a lively hobbyist community,[10] there are many companies that focus on the development of these software instruments and effects. Several other manufacturers of DAWs and operating systems developed their own APIs to have more influence over the user experience of their platform (Apple's AudioUnits, Avid's Avid Audio Extension [AAX], etc.), and the open-source community has created similar formats[11] as well (LADSPA, LADSPA version 2 [LV2], and most recently CLAP[12]).

These APIs all share common functionality, namely processing blocks of audio samples, handling parameter changes, handling MIDI, and managing state. Many third-party developers opt to use an intermediate C++ framework, such as JUCE[13] or iPlug2,[1] which facilitates compilation of a single codebase to multiple target APIs. Plug-ins typically fall into three categories: instruments, audio effects processors, and MIDI processors.

[4] https://www.bandlab.com/
[5] https://ampedstudio.com/
[6] https://www.soundtrap.com/
[7] https://opendaw.org/
[8] https://wam-studio.i3s.univ-cotedazur.fr/
[9] https://en.wikipedia.org/wiki/Virtual_Studio_Technology
[10] http://www.kvraudio.com/
[11] http://lv2plug.in/ns/
[12] CLAP (CLever Audio Plugin) https://u-he.com/community/clap/
[13] http://www.juce.com/

1.2 The Web Audio Application Programming Interface and the Lack of an Open Plug-in Standard

The Web Audio API in its 1.0 version is now a W3C recommendation[14] (a "frozen standard"). It proposes a set of unit generators called AudioNodes for graph-based realizations of audio algorithms, and it is supported in the latest versions of most popular desktop and mobile browsers. The connection of these nodes in the browser via a JavaScript API allows for a range of different applications involving real-time audio processing. The API comes with a limited set of built-in nodes for common operations, such as volume control, audio filtering, time delay/echo, reverberation, dynamics processing, spatialization, etc.

The addition in 2018 of the AudioWorklet node provides a solution for implementing custom low-level audio processing (supported by all major browsers in 2022), including DSP code compiled to WebAssembly, that will be executed in the browser's audio thread. Such nodes can be assembled into an "audio graph," which developers can use to write more complex audio effects or instruments.

Sometimes, when performing live, one needs to chain audio effects together (for example in the guitarist's pedal board), and when composing/producing music, multiple effects and instruments are often used. These are examples where the Web Audio API nodes are too low level, hence the need for a higher level unit to represent the equivalent of the "plug-ins" that are ubiquitous in native DAWs.

Multiple proposals for WebAudio plug-in standardization have appeared since 2013 [1, 3, 6, 8]. Authors of this paper published various state-of-the-art works [3, 4, 9] on this domain, which are still relevant today but omit a new entrant to the field: Web CLAP (WCLAP),[15] a proposed extension of the CLAP plug-in format tailored for WebAssembly (WASM) environments. CLAP is a modern, MIT–licensed, open-source native audio plug-in format focused on performance and flexibility, with unique features such as advanced parameter modulation. WCLAP, still in its infancy, targets WASM/WASI architectures and enables execution of cross-compiled CLAP plug-ins within web browsers. Comparisons with WAMs are currently speculative because WCLAP is a new development. Unlike WAM v2, it exclusively uses WASM and does not support plug-in entities made of Web Audio API nodes. Due to its permissive license and growing traction in the industry,[13] it may appeal to developers of native audio plug-ins who want to use the same API across platforms. CLAP's parameter modulation features, highlighted as one of the key advantages over other plug-in formats, is similar to the *ModulationTarget WAM extension* presented in SEC. 9.

1.3 What Makes the Web Platform Different?

Although designing an equivalent to native plug-ins but for the web platform aimed to bring the functionality of native audio plug-ins to the web, the unique advantages and constraints of the web environment necessitate a different approach to API design and at least guidelines on how to package plug-ins for publication on a plug-in repository/server and for their integration in a host. A web-based API should be "web aware" and use URIs as plug-in identifiers. Plug-ins are just another kind of web resource, like images, cascading style sheet (CSS) or JavaScript files and should be referenced by a relative/local or remote URI.

Host web apps should be able to discover local/remote plug-ins by querying plug-in folders/servers. Remote plug-ins should be usable without the need to download them manually, and the mixture of different JavaScript libraries and frameworks should not raise any naming conflicts or dependency problems. Before reaching version 2.0 of the WAM proposal, several iterations took place. The starting point was a comparative study of the different native plug-in standards and their similarities and differences, with the focus on optimal adaptation to the web platform and web APIs available in browsers. One of the advantages of the web platform is its rich set of built-in APIs, many of which are at the heart of the WAM standard (Web Audio and Web MIDI, but also Web Components, WebAssembly, etc.).

All native plug-in standards aim to do more or less the same things but are widely different and not always completely overlapping in what they can and can not provide. Additionally, they are not suited as-is for being applied to the web. In 2003, the Generalized Music Plug-in Interface[16] (GMPI) working group was formed to collect the best parts of each existing standard and to bring them together into a cross-platform specification[17] that has been later partially implemented by the LV2[18] open plug-in standard, mainly adopted by the Linux community.

Developing a new API for Web Audio plug-in presents an opportunity to enhance certain aspects of native APIs while considering the work done by the GMPI group and the LV2 community. These efforts aimed to capture the best features of other native standards. Despite its age, the GMPI final draft document provides a comprehensive overview of desirable qualities in an audio plug-in API. The LV2 achievement has been compared with the GMPI Final Draft Proposal in a categorized table that is published online.[19] The same process was started by the WAM group (comparing with GMPI requirements) and also published online.[20] This enabled both to identify the most important features that a web-based host/plug-in API should provide but also allowed for discarding irrelevant specifications.

The GMPI draft proposal lists 114 requirements grouped into 23 categories. In 2018, the WAM designers divided the

[14] https://www.w3.org/TR/webaudio/

[15] https://github.com/free-audio/web-clap

[16] https://github.com/JeffMcClintock/GMPI

[17] https://github.com/JeffMcClintock/GMPI/blob/main/Docs/GMPI_Specs.md

[18] http://lv2plug.in/ns/

[19] LV2 achievement of GMPI requirements: http://lv2plug.in/gmpi.html

[20] WebAudio plug-in vs LV2 vs GMPI: https://docs.google.com/document/d/1j18NSIpOJIpJNw9WGzLfNSelnlcrS0hfEBQ8KNS3OGk/edit?usp=sharing

categories further into three groups based on their relevance to the fundamental requirements of an audio plug-in.[21]

Priority one group of features include the following:

- **Host/plug-in model**: how plug-ins are loaded, instantiated, and connected together. Hosting scenarios require mechanisms for plug-in discovery and host/plug-in interface description.
- **Events** and **MIDI**: there should be a way to send and receive events to/from plug-ins and hosts, and MIDI support is obvious for instruments.
- **Parameters**: Plug-ins will need to expose their parameter set and provide getter/setters and provide support for parameter automation.
- **Plug-in files**: More generally, a way to persist current state, and to load/save presets/banks.
- **UIs**: Plug-ins can be run with a graphical UI (GUI) or headless.
- **Versioning**: How to deal with SDK updates.

The second group of features comprised these categories: **host services, time, latency, copy protection, localization, API issues,** and w**rappers**.

Finally, the third group consists of categories that are irrelevant for web environments, or already defined in the lower-level APIs such as Web Audio and WebRTC. These include real-time threading, sample rate, audio input/output, control input/output, and results.

The web browser environment also means that extra considerations must be added, that are of no concerns for native plug-ins. As detailed further, the most important features have been integrated, whereas other important ones, such as copy protection, will be discussed at the end of this paper.

The following sections present the WAM2 standard, emphasizing new features it offers, and compares it to existing plug-in proposals.

1.4 Glossary of Terms

The following glossary (Table 1) defines key terms used throughout the paper, to help readers navigate the terminology associated with WAM and related technologies.

1.5 Simple Host Example

Before going further, here is a short illustration: a typical host code for loading a WAM plug-in from its URI, instantiating it, and displaying its GUI. The example is available online.[22]

Fig. 2 shows a simple host loading a WAM, connecting it to the audio graph, and displaying its GUI. A JavaScript dynamic import (line 9) loads a WAM instance using its URI. The instance's `AudioNode` property connects to the audio graph (line 15), and `createGUI` returns an HyperText Markup Language (HTML) element for the plug-in GUI (line 16), which is then added to the Document Object Model (line 18). The host groups plug-ins on initialization (line 7), and each plug-in has a HTML identification (ID) attribute (`instance.instanceId`). These IDs manage plug-in creation (within a group) and host/plug-in communication. Fig. 3 shows the result: a phaser effect plug-in, coded in Faust, compiled to WebAssembly in an AudioWorklet, with a Web Component GUI.

```
1 const pluginURI =
'https://www.webaudiomodules.com/…/stonephaser/index.js';
2 const sdkURI =
'https://www.webaudiomodules.com/sdk/…/initializeWamHost.js';
3
4 (async () => {
5   // Init WamEnv
6   const { default: initializeWamHost } = await import(sdkURI);
7   const [hostGroupId] = await initializeWamHost(audioContext);
8   // Import WAM
9   const { default: WAM } = await import(pluginURI);
10   // Create a new instance of the plugin
11   const instance = await WAM.createInstance(hostGroupId,
12                                             audioContext);
13   window.instance = instance;
14   // Connect the audionode to the host
15   connectPlugin(instance.audioNode);
16   const pluginDomNode = await instance.createGui();
17
18   mountPlugin(pluginDomNode);
19   player.onplay = () => {
20     audioContext.resume();
21   };
22 })();
```

Fig. 2. Excerpt of code for importing and displaying a simple WAM plug-in.



Fig. 3. Host web page that corresponds to the execution of the code from Fig. 2.

2 WEB AUDIO MODULE 2.0 ARCHITECTURE

2.1 Project Organization

"WebAudioModules," an organization at GitHub,[23] was created to gather the related repositories and projects. The current work is grouped into four repositories, all under Massachusetts Institute of Technology license. They correspond to the WAM API (detailed in SEC. 3), SDK and Param Manager SDK (SEC. 4), and examples of plug-ins

[21] In addition, it is important to take into account that the GMPI document that was used at the time of early design is now outdated. GMPI is still developed by Jeff Mclintock and is used in his Synthedit application.

[22] https://codepen.io/w3devcampus/pen/gbOMEgV?editors=1000

[23] https://github.com/webaudiomodules

Table 1. Glossary of terms.

Term	Definition
A WAM or a WAM plug-in	An instance of a WAM (i.e., a WAM plug-in)
Descriptor (descriptor.json)	A metadata file that describes each plug-in's identity, version, capabilities, and other properties
WAM instanceId property	A unique ID for the plug-in
WAM AudioNode property of a WAM instance	Where the DSP processing occurs; a WAM can be instantiated without its GUI and still be used with its AudioNode only
WAM Host	A web application (e.g., WAM–Studio, Sequencer Party) that can load and interact with WAM plug-ins
hostGroupId property returned when host is initialized	Useful for host/plug-in communication; all WAM plug-ins belong to a host group and MIDI events can be sent to a whole group at once
WAM API	The interface specification that defines how plug-ins and hosts communicate in the WAM ecosystem
WAM SDK	A reference implementation of the WAM API, providing utility classes for plug-in and host development (e.g., ParamMgr, CompositeNode)
ParamMgr	A class provided by SDK_ParamMgr that is useful for plug-ins made of multiple AudioNodes that should behave like a single AudioNode; provides the CompositeNode class and defines "exposed parameters" and links them to internal Web API nodes
CompositeNode	A class that allows multiple Web Audio nodes to be wrapped as a single plug-in–compatible node
WamEnv/WamGroup	Internal infrastructure that manages plug-in–to–host communication across threads in the browser's execution model

and hosts (SEC. 5). The API and the SDKs are also published as NPM packages[24] under @webaudiomodules namespace (scope) and available in content delivery networks (CDNs). The home page of the project that gathers all kinds of resources (documentation, tutorials, links to SDKs, plug-in community server, and plug-in gallery, etc.) is located at webaudiomodules.com. See SEC. 6.2 for details.

2.2 Real-time Audio Processing With AudioWorklet

Real-time audio processing is an example of a "soft real-time" task, meaning that all computations must complete within a more or less fixed, finite amount of time determined by a number of factors tied to hardware capabilities and configurations, usually on the order of several milliseconds to avoid perceptually noticeable delays or inconsistency. If the processing cannot finish in the allocated time slot, audible glitches will corrupt the output. A critical component of such systems is a high-priority thread, where short blocks of samples can be processed before being fed to the system's digital-to-analog converter and played over speakers or headphones. At present, AudioWorklets[25] are the only entry point for developers to access the high-priority audio thread that runs the Web Audio API's processing graph, and thus it is a central component of the proposed plug-in architecture.

The Web Audio API defines the AudioNode interface, which serves as the foundation for both built-in audio nodes (e.g., oscillators, filters, gain nodes, etc.) and custom nodes implemented via AudioWorkletNode. Although built-in nodes handle standard audio processing tasks, AudioWorkletNode, along with AudioWorkletProcessor, enables developers to define custom audio processing logic that runs in the audio thread for low-latency performance. WAM builds on this architecture by introducing the WamNode and WamProcessor, which extend AudioWorkletNode and AudioWorkletProcessor, respectively [5]. This separation reflects the multithreaded nature of Web Audio: WamNode operates on the main thread, managing parameters and communication with the host, whereas WamProcessor executes in the audio thread, handling real-time audio processing. Although some WAMs function as a single node, others may use a subgraph combining multiple Web Audio nodes, including both built-in nodes and custom AudioWorklets. To the host, however, WAM presents a unified abstraction, a plug-in appearing as a single AudioNode, regardless of its internal complexity. This abstraction ensures seamless interoperability between hosts and plug-ins while maintaining efficient, real-time performance across both threads.

2.3 Facilitating Interaction on the Audio Thread

Because the Web Audio API provides no "official" way to communicate with processors on the audio thread, the WAM framework relies on a singleton object attached to the audio thread's global scope called the WamEnv to facilitate interactions between hosts and processors.

Processors are organized into WamGroups, managed by the WamEnv, where each WamGroup contains plug-ins created by a particular host or subhost (a plug-in that acts as a host, a pedalboard plug-in for example).

Great care was taken to minimize the API's assumptions about host implementation, and the WamEnv and WamGroup are the only objects that are expected to be provided by the host.

[24] https://www.npmjs.com/search?q=keywords:webaudiomodules

[25] https://developer.mozilla.org/en-US/docs/Web/API/AudioWorklet

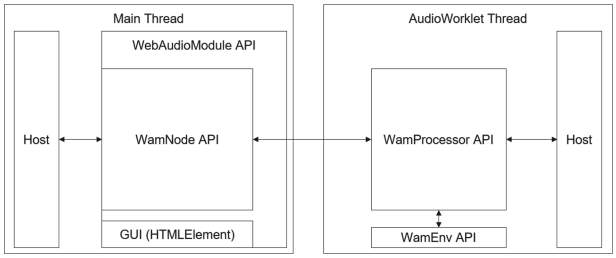


Fig. 4. Each WAM is composed of a WamNode-WamProcessor pair.

Fig. 4 illustrates the relationships between the host, WamNode, WamProcessor, and WamEnv. The host exists in both threads because some web-based DAWs (such as Amped Studio or WAM-Studio) process and play audio buffers directly in the audio thread to ensure accuracy. The API is designed for these hosts, facilitating communication between each entity on each thread.

3 APPLICATION PROGRAMMING INTERFACE OVERVIEW

3.1 Interoperability

The new WAM API[26] includes definition files for plug-ins and hosts written in TypeScript and abstract classes that conform to the API written in JavaScript. The API is designed for making web-based modular audio plug-ins and using them in compatible hosts. Relevant parts of the WAM API should be implemented by each WAM plug-in or host. Plug-ins and hosts that conform to the WAM API are guaranteed to be compatible regardless of their underlying implementations.

3.2 Primary Application Programming Interface Features

Getting the WAM's information by fetching a JavaScript Object Notation (JSON) file: Each WAM has a descriptor (Fig. 5) that contains metadata, such as name, version, thumbnail image, description, etc.

Loading the WAM plug-in constructor by fetching an ECMAScript module file: WAMs must provide a primary `index.js` module file. By loading this file from the plug-in URI, a host can create plug-in instances.

Getting a WebAudio AudioNode-compatible processor that can be inserted into an existing audio graph: WAMs appear to hosts as single Web Audio nodes, even if they are composed of multiple Web Audio nodes. The SDK comes with a CompositeAudioNode class that from which WAMs can inherit and that can wrap a subgraph of multiple nodes such that it behaves like a single node.

Saving and restoring the plug-in's state: WAMs can have many different parameters and thus many possible states. If a WAM is loaded into a DAW and its settings are manipulated by the user, it is mandatory to be able to recall and reapply those settings (i.e., values of the different parameters) when the user saves/loads a project in the host application.

```
{
  identifier: "com.wimmics.stonephaser",
  name: "StonePhaser",
  vendor: "Wimmics",
  description: "This phaser…",
  version: "1.0.0",
  apiVersion: "1.0.0",
  thumbnail: "default.png",
  keywords:["phaser"],
  isInstrument: false,
  website: "",
  hasAudioInput: true,
  hasAudioOutput: true,
  hasMidiInput: false,
  hasMidiOutput: false
}
```

Fig. 5. descriptor.json file from the example WAM plug-in in Fig. 3.

Agnostic of host/plug-in implementation: The standard is designed to provide full support for hosts that run in Javascript's main thread or on the audio thread (via AudioWorklet). Furthermore, care has been taken to ensure that the interoperability guaranteed by the API makes as few assumptions as possible about the underlying implementation of both hosts and plug-ins. Although the SDK provides concrete implementations, developers are not required to use them to build and distribute a valid WAM if they wish to implement some or all of the API themselves. Because WAM hosts and plug-ins can conceivably be designed using standard WebAudio nodes (in situations where access to the audio thread is not desired) or using AudioWorklets (in situations where running custom code and interacting with WAM processors synchronously on the audio thread is desired), it is necessary for WAM plug-in developers to implement a subset of the API on both the main thread and audio thread. The WAM SDK provides a number of convenience classes to help make it easier for developers to meet this requirement.

Getting parameter information, scheduling automation, transport, MIDI, and Open Sound Control (OSC) events from the both threads: By default, the Web Audio API handles MIDI input and parameter automation on the main thread, requiring messages to be sent across the thread barrier to reach the audio thread. This introduces latency and potential timing issues, especially for real-time workflows. WAM 2.0, however, introduces a novel approach by allowing bidirectional access to key plug-in functions from both the main and audio threads.

- Parameter access: Hosts can retrieve WAM-specific parameter information (e.g., names, values, and ranges) directly from either thread, reducing the need for cross-thread messaging.
- Automation scheduling: Unlike standard Web Audio AudioParams, which require automation to be scheduled from the main thread, WAM plug-ins allow hosts to

[26] https://github.com/webaudiomodules/api

schedule parameter changes from either thread, improving precision for real-time control.
- Transport, MIDI, and OSC event scheduling: Hosts can schedule MIDI, OSC, or transport-related events directly from either thread, enabling tighter synchronization and reducing event timing inconsistencies.

This dual-thread access model is a major innovation in WAM 2.0. It unlocks advanced capabilities, such as just-in-time (JIT) MIDI event playback and high-performance parameter automation, providing a level of control and responsiveness that surpasses the default Web Audio API behavior.

Passing events between WAM plug-ins on the audio thread: Hosts can make (nonaudio) connections between WAMs for broadcasting/receiving events. Like audio connections between nodes, these event connections are managed from the main thread. However, the events are actually transmitted on the audio thread, a task facilitated by the WamEnv/WamGroup. Such connections can only be made between plug-ins residing in the same WamGroup.

Cleaning up when a plug-in instance is destroyed: The API accounts for the full life cycle of a WAM, ensuring that plug-in resources can be released when they are no longer needed.

3.3 Key Application Programming Interface Constructs

3.3.1 Interfaces and Abstract Classes

The API is composed of abstract interfaces and class definitions that fully define the expected behavior of a plug-in and the environment it in which it operates. The primary interfaces are listed in Table 2.

3.3.2 Plug-in Parameters

Because the WAM standard aims to allow maximum flexibility for developers, it was decided that Web Audio's AudioParams should not appear explicitly in the API. Some WAM design patterns do not make use of any stock Web Audio nodes, and although it is possible to define custom Web Audio AudioParams for AudioWorklet, in many cases it would be too heavy and cumbersome to expose the potentially hundreds of parameters residing in WebAssembly code via that API. Furthermore, the parts of the Web Audio API having to do with AudioParams were conceived before developers had any direct access to the audio thread, forcing parameter updates to be scheduled asynchronously from the main thread far enough in advance to account for crossing the thread barrier. This aspect of the Web Audio API is not compatible with the authors' goal to support synchronous, JIT interaction between hosts and plug-ins on the audio thread as in native plug-in environments. To get around these limitations it was chosen to create a WamParameter API to handle parameter updates, designed to facilitate many different WAM design patterns and interaction between hosts and plug-ins on either thread.

3.3.3 Event Scheduling

Sample-accurate event scheduling is a critical requirement for professional audio applications. The Web Audio API does allow for sample-accurate scheduling of AudioParams. This feature is leveraged in the WAM SDK to facilitate WAM designs that incorporate one or more built-in Web Audio nodes. However, as mentioned previously, the automation API for AudioParams is not useful for many WAM design patterns, and the authors also wish to support sample-accurate processing of other kinds of events such as MIDI messages. The WAM API facilitates event scheduling through a unified interface that is mirrored on both the main thread and audio thread. Hosts operating entirely on the main thread will still be required to schedule events with some look-ahead to ensure that they are processed at the intended time because these messages must still cross the thread barrier. *However, hosts with a presence on the audio thread can schedule events at the beginning of the rendering block in which the events should occur.*

4 SOFTWARE DEVELOPMENT KIT OVERVIEW

The SDK provides reference implementations covering the full functionality of the WAM API and utility classes to help developers build WAMs using several different design patterns. However, if developers wish to use their own implementations, there is no need to use the SDK as long as the alternative implementation conforms to the API.

Parameter Manager and CompositeNode: The WAM SDK's parameter manager (ParamMgr) and CompositeNode[27] classes provide scaffolding for plug-ins that use multiple audio nodes. ParamMgr automatically wraps AudioParams as WamParameters, whereas CompositeNode wraps a subgraph such that it can be treated as a single audio node. Additionally, nonaudio parameters or multiple AudioParams can also be controlled through these WamParameters by specifying callbacks in the ParamMgr.

As described in the WAM API, the developer should declare and configure via WamParameterInfo every parameter that is controllable or automatable by the host application, and make them accessible via WamNode's methods such as getParameterInfo. In the ParamMgr, these parameters are considered as the WAM's "exposed parameters."

By automating or controlling these exposed parameters, the host can indirectly change the WAM's internal state. The variables to be changed in the internal state, which are called "internal parameters" in this study, can be an AudioParam or an event handler that will be called when the values change. In some use cases the plug-in needs to control multiple internal parameters via a single exposed parameter and with different value scalings or mappings for each internal parameter. (Figs. 6 and 7).

[27] A CompositeNode extends the GainNode class from the Web Audio API. All Web Audio API nodes implement the AudioNode interface.

Table 2. Important interfaces in the WAM API.

WebAudioModule	Main entry point of a WAM plug-in instance
WamDescriptor	Contains general information about the plug-in
WamNode	Extends Web Audio's AudioNode; can be inserted into the host's audio graph
WamProcessor	Extends Web Audio's AudioWorkletProcessor; processes signals in the audio thread
WamParameterInfo	Provides parameter metadata and convenience methods
WamParameter	Maintains parameter state information
WamEvent	Provides information for scheduling or emitting WAM events (e.g., automation, MIDI messages)
WamGroup	Maintains event graph information for hosts and subhosts on the audio thread
WamEnv	Manages WamGroups, registers WamProcessors, and stores plug-in dependencies on the audio thread

```
createAudioNode(initialState) {
  // pingPongDelayNode is a CompositeNode
  const pingPongDelayNode = new
                          PingPongDelayNode(this.audioContext);
  // Exposed plugin parameters (WamParameters) config
  const paramsConfig = {
    feedback: {minValue: 0, maxValue: 1, defaultValue: 0.5},
    time: {defaultValue: 0.5},
    mix: {defaultValue: 0.5},
    enabled: {defaultValue: 1},
};
// Internal parameters (AudioParams or callbacks) config
const internalParamsConfig = {
   delayLeftTime: pingPongDelayNode.delayNodeLeft.delayTime,
   delayRightTime: pingPongDelayNode.delayNodeRight.delayTime,
   dryGain: pingPongDelayNode.dryGainNode.gain,
   wetGain: pingPongDelayNode.wetGainNode.gain,
   feedback: pingPongDelayNode.feedbackGainNode.gain,
   enabled: {
     onChange: (value) => {
       pingPongDelayNode.status = !!value;
     }
   },
};
// Mapping for one-to-many controls
const paramsMapping = {
   time: {
     delayLeftTime: {},
     delayRightTime: {}
   },
   mix: {
      dryGain: {
        sourceRange: [0.5, 1],
        targetRange: [1, 0],
      },
      wetGain: {
        sourceRange: [0, 0.5],
        targetRange: [0, 1],
      },
   },
};

const optionsIn = { internalParamsConfig, paramsConfig,
                    paramsMapping };
const paramMgrNode = await ParamMgrFactory.create(this,
                     optionsIn);
pingPongDelayNode.setup(paramMgrNode);
if (initialState) pingPongDelayNode.setState(initialState);
   return pingPongDelayNode;
}
```

Fig. 6. Code example of CompositeNode and parameter mapping with ParamMgr for a delay effect. Notice how the "mix" parameter is mapped to two internal gain parameters.

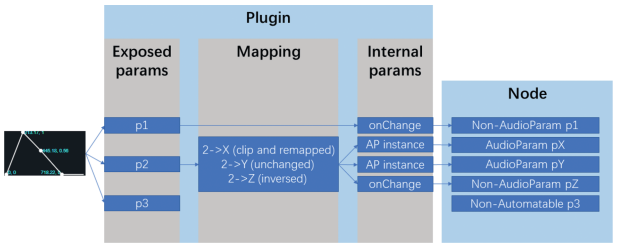


Fig. 7. Use of parameter mappings in the ParamMgr.

5 USING DOMAIN-SPECIFIC LANGUAGES FOR WEB AUDIO MODULE DEVELOPMENT

5.1 Faust

Faust is a popular functional programming language for DSP programming; thousands of source code for audio effects, instruments, and more generally DSP algorithms, filters etc. are available from the open-source community, or included in the distribution and in the online integrated development environment (IDE) [10, 11].

The Faust compiler supports exporting to a variety of platforms and standards, including WAMs. Since 2014, Faust DSPs can be compiled to JavaScript-compatible binary code and dynamically run the DSP within the browser [12]. In 2022, a new version of the Faust WebAssembly compiler named `faustwasm`, which provides TypeScript and JavaScript wrappers for Faust DSPs, was released [13]. It allows for generation of static self-contained HTML pages or JavaScript modules (including the Faust code as a WebAssembly module and various additional resources) or even to integrate the libfaust compiler in applications that need to dynamically compile and deploy Faust DSP programs.

Synthesizer and effect mono nodes and polyphonic poly nodes can be created. MIDI support is activated as soon as MIDI metadata is used in the DSP code for mono nodes and always in polyphonic mode. Sensors (accelerometer and gyroscope) are supported, as are the Progressive Web Application model (it can be used offline).

The library can be used either in Node.js-based projects or in web browsers and is published on NPM. Furthermore, an official online Faust IDE has been developed since 2019 using modern web technologies, such as WebAssembly and AudioWorklet, offering various testing, debugging, and audio visualization features, allowing connection to different kinds of audio/MIDI inputs and outputs, making easier the development of wasm WAM plug-ins, with a standard CSS-based GUI (auto-generated, Fig. 8) or with a custom GUI that can be designed with an embedded GUI builder (Fig. 9) [14, 15]. A step by step tutorial about how to build WAM plug-ins is available online.[28] Starting from an existing Faust code, it takes a few seconds to execute in the

[28] https://docs.google.com/document/d/1HDEm4m_cD47YBuDilzGYiANYQDktj56Njyv0umGYO6o/edit?usp=sharing

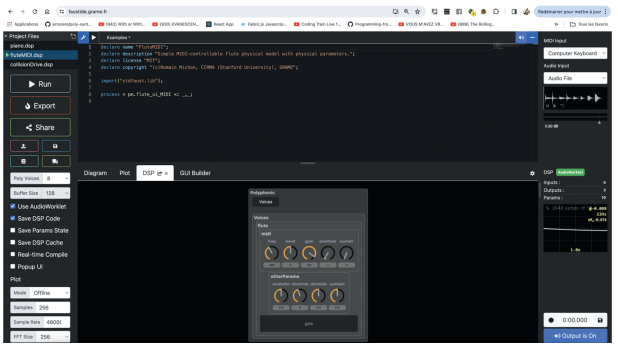


Fig. 8. A physical model of flute in the FAUST online IDE.

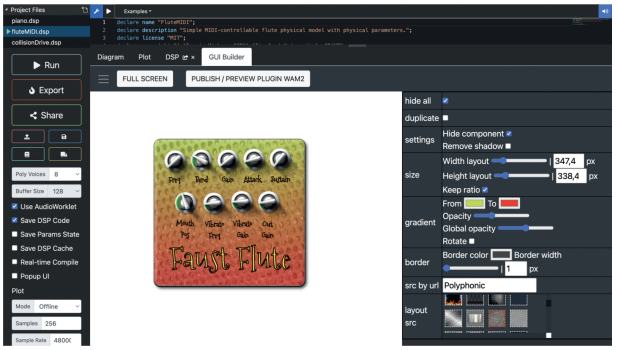


Fig. 9. The same flute with a custom GUI made with the embedded GUI builder.

IDE, verify, export (download as a zip file), publish online, and run a WAM plug-in (Fig. 10).

Based on the `faustwasm` module, `faust2wam`[30] has been recently developed; it is a JavaScript tool that can generate self-contained Faust WAMs within the Node.js environment or dynamically within the browsers. Additionally, support has been added for polyphonic instruments and Faust-based spectral processors. These new generation targets (web/wam2-ts, wam2-poly-ts, and wam2-fft-ts) are now available in the Faust IDE in the export window.

5.2 Cmajor

Cmajor[31] is a new domain-specific language (DSL) for signal processing, with support for JIT compilation and exporting to various other targets (e.g., C++, JUCE, WAM2). The language is a C-based procedural language, but unlike C and C++, it has specifically been designed for real-time use; the language does not allow memory allocation or out of bounds access, so it is implicitly safe and ideal for signal processing.

Rather than classes, it uses a graph node encapsulation pattern to facilitate code reuse. The language and runtime are open source, and there is a growing community developing and sharing code. Cmajor plug-ins are developed as patches that include the Cmajor source files for the DSP and JavaScript for the UI components.

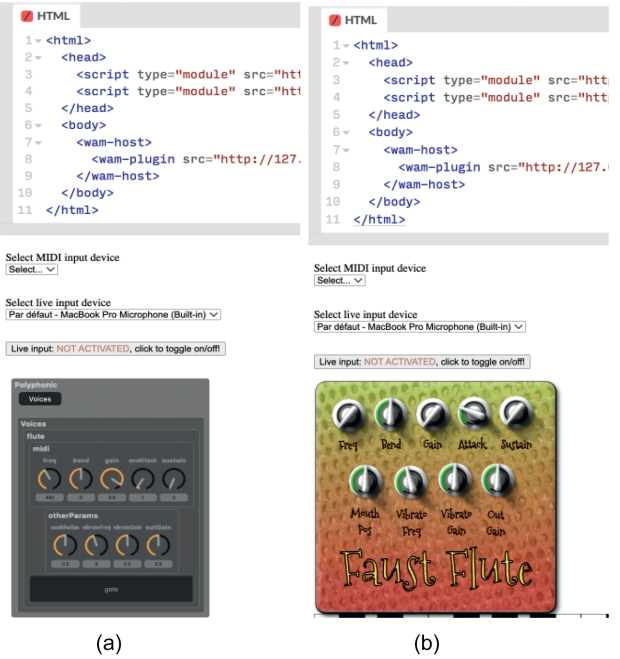


Fig. 10. The exported WAM can be used by an external host, here in codePen.[29] (a) With standard CSS-based generated UI. (b) With a custom GUI.



Fig. 11. Running a Cmajor patch in VSCode.

For DSP development, Cmajor is available as a VSCode extension, with VSCode providing a development and runtime environment for Cmajor patches (Fig. 11).

Once a patch is complete, VSCode supports an export action to create a WAM plug-in. The exporter generates wasm for the DSP and bundles the wasm with a wrapper supporting the WAM interface. The exported WAM directory includes a lightweight hosting index.html that allows the directory to be served to a browser using any web-hosting tool, for example, `python -m http.server`. This allows the plug-in to be tested locally (Fig. 12).

Pushing the exported directory to the web allows the plug-in to be hosted by any host supporting WAM (Figs. 12 and 13). The export to a WAM builds the DSP to WebAssembly, with both Single Instruction/Multiple Data and non-Single Instruction/Multiple Data code paths generated by default to support a wide range of browsers. The export

[29] https://github.com/Fr0stbyteR/faust2wam
[30] https://codepen.io/w3devcampus/pen/GgRqLvr?editors=1000
[31] https://cmajor.dev/

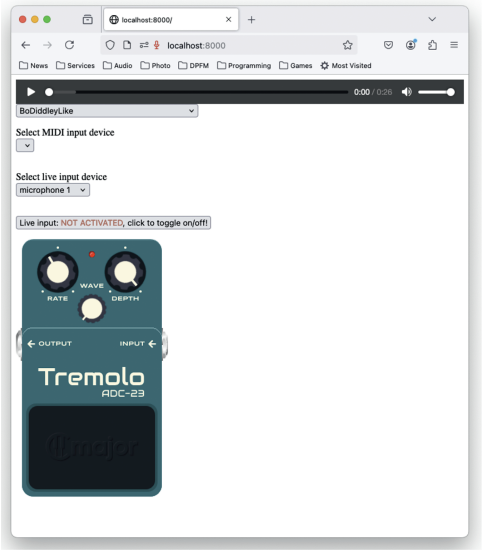


Fig. 12. Running the generated Cmajor WAM with a browser.

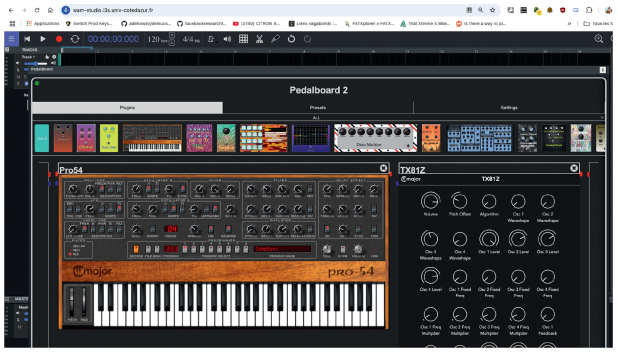


Fig. 13. Two Cmajor instruments created using the Cmajor WAM exporter, here included in the WAM-studio online DAW.

includes a WAM wrapper to bridge the calls into the generated wasm and includes the UI. If no custom UI is provided, a generic UI is generated, exposing the DSP parameters as generic controls.

Dozens of real-time effects and instruments are available on the Cmajor web website and can be exported to WAM plug-ins, including the impressive PRO-54 synth (Fig. 13), a revisited version of Native Instruments' acclaimed PRO-53. This is one of the most impressive WAM instruments available, a direct port of the original, never commercially released, with a graphical interface recreated in HTML and JavaScript, using mainly low-resolution bitmaps for authentically vintage rendering and open-source Cmajor code for the DSP part.

5.3 Csound

Csound [16] is a sound and music computing system in the MUSIC-N family of music systems. It features a large library of signal processing opcodes, a DSL for programming music and sound processing, a real-time and ahead-of-time audio engine, and an API for embedding the system within applications and plug-ins. It is cross-platform and open source and is available on desktop, server, embedded, and browser platforms.



Fig. 14. A Csound pitch shifter effect created from original Csound code exported in WebAssembly and packaged as a WAM.

Csound has been used in the browser as early as 2000,[32] and newer versions updated for modern browser technologies such as WebAssembly and AudioWorklets [18] are now supported and produced with each new release of Csound.

Using the WebAudio Csound library, an example PitchShifter WAM effect (Fig. 14) was produced[33] using Csound's phase vocoder opcodes for spectral processing for independent pitch and formant shifting. The WAM code uses a Csound AudioWorklet node, initialized with Csound audio code embedded within the plug-in, and uses Csound's API to communicate values from the GUI to Csound's channel system.

The example Csound-based WAM provides a model for Csound and WAM developers to create plug-ins using WAM's provided classes together with Csound's extensive library of signal processing opcodes. Developers can draw upon 40 years of computer music history in Csound works to the web-based music making world through WAM.

6 WEB AUDIO MODULE RESOURCES

6.1 Web Audio Module Plug-in Example Repository

In addition to the API, SDK, and ParamMgr repositories, the WAM distribution proposes a repository named "wam-examples" with a large library of plug-ins that contains source code for effects, instruments, and MIDI controllers made using different development approaches, such as JavaScript, build systems, TypeScript, FAUST, Csound, front-end frameworks, cross compiled C++/Juce plug-in ported to WAM, etc. This example repository also contains a demo host page for testing the WAMs, which is also available online[34] (Fig. 15). An audio player or a live input stream can be used at the root of the host's audio graph to audition audio effect plug-ins. In the case of plug-ins that are "MIDI controllers" (e.g., a piano keyboard), they can be

[32] This was in the form of ActiveX Csound, also known as AXCsound. A version of this is available in the CD-ROM accompaniment for [17].

[33] https://github.com/webaudiomodules/wam-examples/tree/master/packages/csoundPitchShifter/src

[34] https://mainline.i3s.unice.fr/wam2/packages/_/



Fig. 15. MIDI controller and Oberheim OB-Xd synthesizer WAM plug-ins loaded in the host.

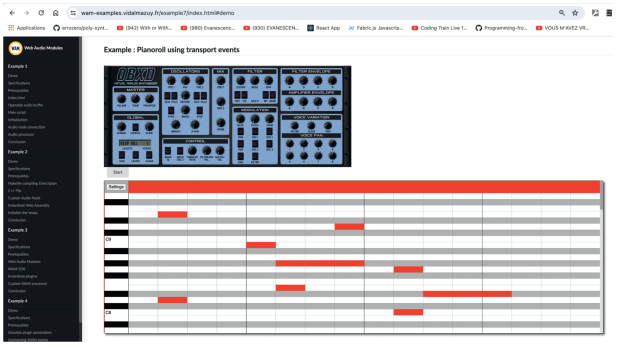


Fig. 16. One of the online tutorials showing how to use transport events with a piano roll WAM, which is sending events (MIDI notes) to a WAM instrument (Oberheim OB-Xd emulation).

loaded upstream in the processing chain to test audio synthesizers or any other plug-ins that accept MIDI input. For each loaded plug-in, the host allows one to perform some simple tasks, such as testing automation of its parameters, loading/saving states, etc. An input field can also be used to load and test any WAM using its URI. Note that there are also many open-source examples from the community of developers, in particular through the WAM Community initiative presented in SEC. 6.3.

6.2 Online Tutorials

A set of seven tutorials has been published.[35] They show how to load a WAM plug-in from JavaScript, how to attach it to a Web Audio graph, how to chain it with other WAMs, how to perform parameter automation at the frequency rate, and how to send MIDI events to a WAM instrument (using JavaScript code or by connecting a WAM virtual piano keyboard or a piano roll WAM, see Fig. 16). They also show how to write AudioWorklet-based WAMs using DSP code written in C++ and cross compiled to WebAssembly.

6.3 WEBAUDIOMODULES.COM, WEB AUDIO MODULE COMMUNITY

The webaudiomodules.org website was once the primary hub for WAM-related content but is no longer maintained,

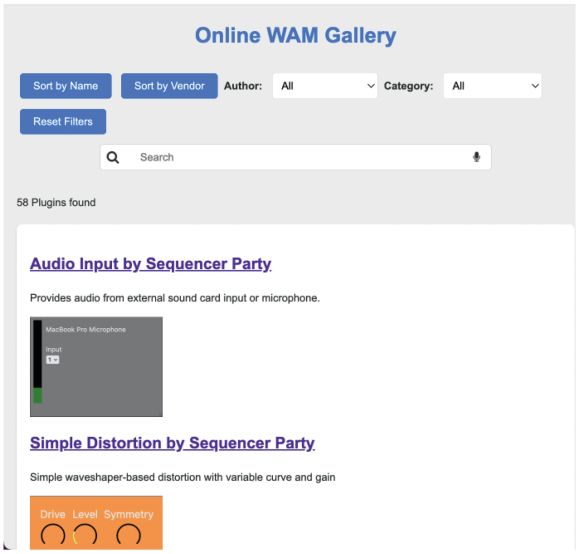


Fig. 17. Online WAM gallery web application.

and those currently involved in the WAM standard do not own the domain. To address this, a new site, webaudiomodules.com, has been created to compile key WAM resources. This new domain also hosts online SDK versions, provides an API/endpoint for querying community-published WAM plug-ins,[36] and features a WAM gallery. The gallery, powered by this API, simplifies WAM reuse by generating short source code snippets for loading and executing plug-ins.

WAM plug-ins have been designed so that they can be used in host software simply by using a JavaScript import on their URI. After including the SDK and importing the URI, one can then connect the plug-in to a Web Audio graph (any instance of WAM is seen as an AudioNode) and display its graphical interface (seen as a Web Component,[37] a.k.a. a simple HTML div with encapsulated code). This makes it very easy to reuse Web-hosted WAMs. The WAM Community initiative was created to facilitate the publication and sharing of WAMs and comprises several tools: 1) a GitHub repository[38] with prebuilt, ready-to-use WAMs, where developers can add their own WAMs by making simple pull requests; 2) the webaudiomodules.com server that hosts plug-ins previously published on the repository and provides a REST API to retrieve the list of available WAMs and their URIs[39]; and 3) an online gallery[40] (Fig. 17) to browse WAMs with a search engine with filters by category, author, etc. It is also possible to test WAMs online and copy/paste the source code of a mini-WAM host, with several possible options (Fig. 18).

[35] Online version: https://wam-examples.vidalmazuy.fr/, also on GitHub repo: https://github.com/Brotherta/wam-examples

[36] RBMO105241

[37] https://developer.mozilla.org/en-US/docs/Web/API/Web_components

[38] https://github.com/boourns/wam-community

[39] https://www.webaudiomodules.com/community/plugins.json

[40] https://mainline.i3s.unice.fr/wamGallery/public

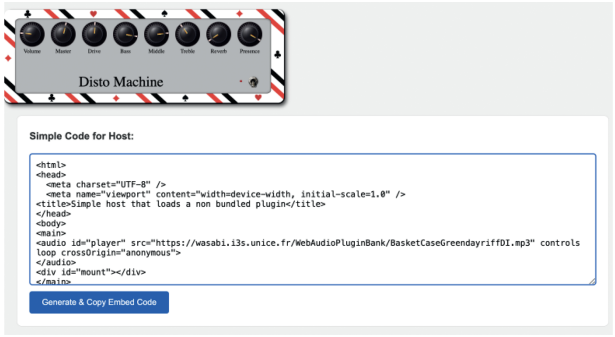


Fig. 18. The WAM gallery can run WAMs in demo pages and generate the source code for a simple host.

```
<wam-host>
    <wam-plugin src="URI_OF_WAM1">
    <wam-plugin src="URI_OF_WAM2">
    <wam-plugin src="URI_OF_WAM3">
</wam-host>
```

Fig. 19. The wam-host and wam-plugin Web Components.

7 OTHER TOOLS

7.1 `<wam-host>` and `<wam-plugin>` Web Components

These two Web Components have been developed to facilitate the embedding of WAM plug-ins in a Web page, for demo purposes or just for testing whether a WAM runs correctly.

The code from Fig. 19, when executed, will display a chain of three plug-ins. Depending on the type of the plug-ins and their location in the chain, the wam-host element may provide additional elements in the demo page such as a virtual MIDI keyboard with a MIDI input device selection menu (for WAM instruments), an audio player with a button for activating live input, or an audio input device selection menu, as shown in the examples of Fig. 10 (see the codepen for source code examples[41]).

7.2 Easy Integration With p5.js

Numerous examples of WAMs have been used with p5.js, an open-source JavaScript library for creating graphical animations and visual interactions on the web. Inspired by the Processing language,[42] it enables artists, designers, and developers to intuitively program drawings, animations, data visualizations, and interactive experiences.

The example in Fig. 20, available online,[43] integrates multiple WAMs: a piano roll for recording notes on a time grid connected to a Pro-54 synthesizer and two audio effects (reverb and delay). The synthesizer's filter parameters are modulated in real time by a 2D orbiter following a Lissajous curve. Additionally, animated graphical objects generate notes upon collision, highlighting the potential of WAMs for creating interactive environments combining sound and visuals. This example also serves as a useful tutorial for learning how to use WAMs in a host application.

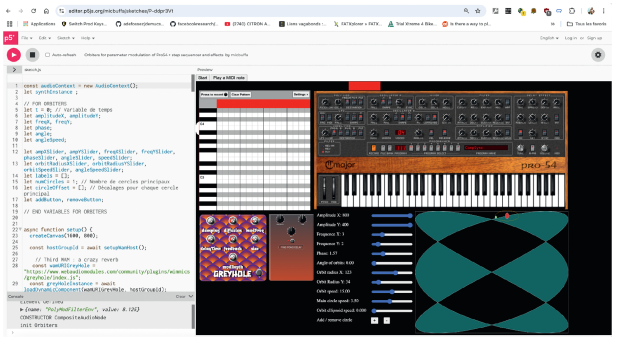


Fig. 20. Several WAM plug-ins in a p5.js sketch.



Fig. 21. The WAM-bank pedalboard WAM acts as a complete pedalboard of audio effects.

8 AVAILABLE WEB AUDIO MODULE PLUG-INS AND HOSTS

8.1 Notable Web Audio Module Plug-ins

The WAM-bank pedalboard (Fig. 21) is a WAM that hosts and manages chains of effect and instrument plug-ins. It includes a preset manager with around 40 plug-ins (mainly from the WAM Community initiative discussed in Sec. 6), allows filtering by category, and is easily extensible. It also offers an online standalone demo[44] and exposes its URI for embedding in external hosts,[45] such as the WAM-Studio DAW (see Sec. 8.2).

Tube guitar amp simulations: a Vox AC30 and a Marshall JCM800 (distortion machine) [19, 20] are already shipped with the WAM 2.0 distribution and available in the previous pedalboard along with many classic effect pedals for guitarists. Other WAM V1.0 amp simulations exist for metal, classic rock/blues, and clean sounds [21, 22], with a complete simulation of all the tube guitar amp stages (Fig. 22)

[41] https://codepen.io/w3devcampus/pen/GgRqLvr?editors=1000
[42] https://processing.org/
[43] https://editor.p5js.org/micbuffa/sketches/P-ddpr3Vt
[44] https://wam-bank.i3s.univ-cotedazur.fr/ (click on the blank page)
[45] See this codepen example: https://codepen.io/w3devcampus/pen/BabzVvP.



Fig. 22. Tube guitar amp simulators.

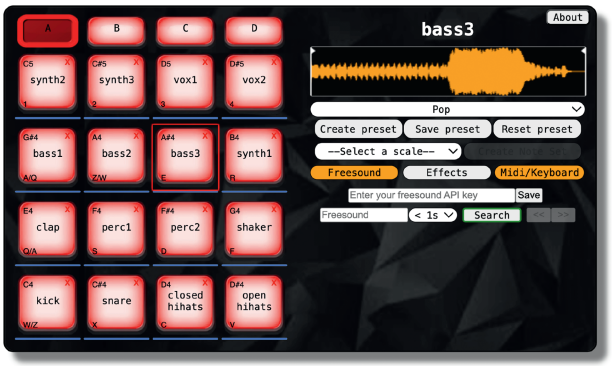


Fig. 23. A freesound.org powered sampler.

(see demos on video).[46]

WAM-sampler (Fig. 23) is an open-source creative sampler[47] that includes a collection of standard sounds (such as drums, piano, hip hop, etc.) triggered by MIDI events (with velocity support). Its main distinguishing feature is the ability to retrieve sounds from the millions available on freesound.org, which users can preview, trim, adjust the attack, decay, sustain, and release structure of the envelope, volume, panning, and apply audio effects before dragging and dropping them onto the sampler's control pads. Additionally, a powerful note set generator can automatically assign sounds made up of different notes in various

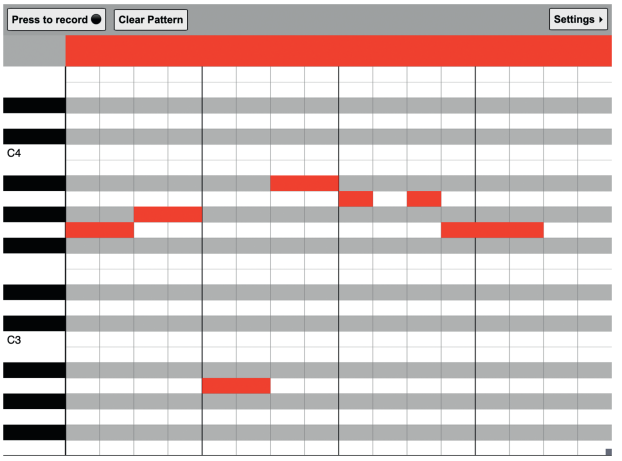


Fig. 24. A piano roll MIDI sequencer.

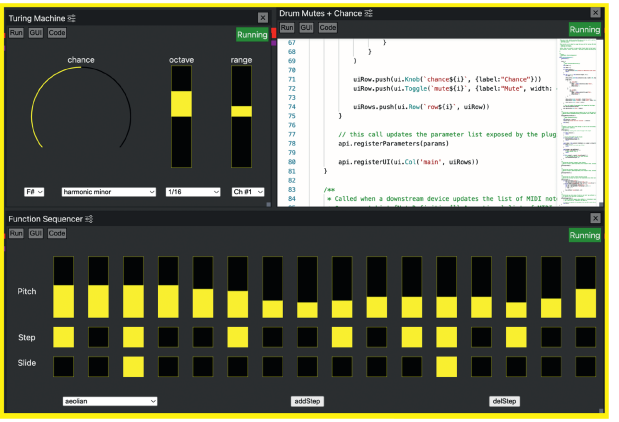


Fig. 25. A live-coding MIDI sequencer.

scales to the pads. An online demo and video presentation are available.[48]

Piano Roll (Fig. 24) is an open-source polyphonic MIDI sequence editor. In hosts that support the Patterns WAM Extension (see SEC. 9), a single piano roll WAM instance can store multiple pattern clips of varying lengths. MIDI note events are generated with sample-accurate timing by sending the WAM note events directly from the AudioWorklet, eliminating the need to schedule them in advance as is often required in similar applications.

The WAM function sequencer (Fig. 25) is an open-source live-coding MIDI sequencer environment developed to reduce the needed effort for JavaScript developers to iterate on a MIDI sequencer idea (it embeds the source code editor component from Visual Studio Code). The patch developer uses JavaScript to register a custom interface and code event handlers without the hassle of build processes or code hosting. In collaborative WAM hosts, the patch code can be edited simultaneously by multiple participants. Several examples of what is possible by programming this WAM are available in the "Sequencer Party" collaborative application described in SEC. 8.2.

[46] https://mainline.i3s.unice.fr/jaes2023/userTestsAndLatency.html

[47] https://github.com/micbuffa/WAMSampler

[48] https://www.youtube.com/watch?v=3jf2KCdwU1A

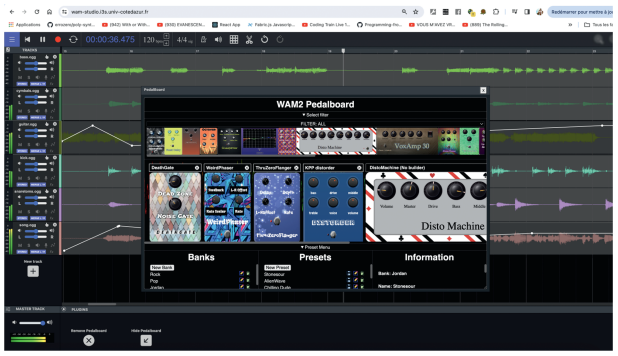


Fig. 26. WAM-Studio, an open-source DAW.



Fig. 27. Attune-specific macro editor.

8.2 Notable Web Audio Module hosts

WAM-Studio [7, 23] (Fig. 26) is an open-source, online DAW developed as a demonstrator of WAMs and recent W3C Web APIs, such as WebAssembly, Web Components, Web Midi, Media Devices etc. DAWs are feature-rich software and therefore particularly complex to develop in terms of design, implementation, performance, and ergonomics. Very few commercial online DAWs exist today, and the only open-source examples lack features (no support for inter-operable plug-ins, for example) and do not take advantage of the recent possibilities offered by modern W3C APIs (e.g., AudioWorklets/WebAssembly). See [23] for a survey of online DAWs.

In WAM-Studio, each audio track is a WAM processor (i.e., a subclass of an AudioWorkletProcessor) for playing or recording audio or MIDI buffers. The main reason is that when both host and plug-ins are implemented this way (as WamProcessor/AudioWorklet) then the WAM SDK provides under the hood, optimized host/plug-in communications that avoid crossing the audio thread. In this way, parameter automation at the sampling rate, even with hundreds of parameters, can be efficiently managed using SharedArrayBuffers. Additionally, other events can be sent without crossing the main/audio thread boundary, eliminating the need to schedule MIDI events in advance, as explained in the well-known article by Chris Wilson "A Tale of Two Clocks" [24]. Additionally, WAM-Studio is also a good implementation example of how to record audio tracks in sync with other tracks playing at the same time. In [23], the authors describe how they calibrate the input latency and perform latency compensation and how they reused Paul Adenot's ring buffer implementation[49] in a multithread architecture for a robust recording solution (i.e., no glitches if the UI is resized or the central processing unit is stressed) while recording a waveform being redrawn regularly as the recording progresses.[50]

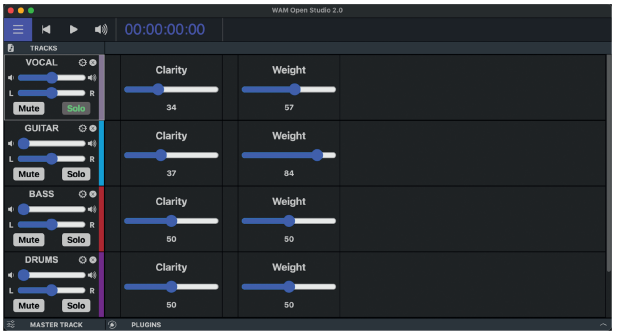


Fig. 28. Simplified view of Attune allowing DHH users to adjust sound properties.

WAM plug-ins can be associated with tracks too: a rich pedalboard handles audio effects chains, featuring a preset manager and filters for managing a large set of WAM plug-ins (standard audio effects, original ones), and can be used standalone or embedded in other WAM hosts.

The DAW is online,[51] and the GitHub repository[52] contains both front-end and back-end source code and a Docker image configuration file, making the deployment of a new instance easy. WAM-Studio project management source code (front and back end) shows how a large set of Waveform Audio files (.wav), along with the state of the different components used by a project, can efficiently be saved or restored from a remote server. WAM-Studio can render/bounce the final mix or individual tracks using OfflineAudioContext, applying effects and automation.

Attune is an open-source WAM-Studio fork[53] that is being used by researchers from the Remi group at the Center for Computer Research in Music and Accoustics (Stanford) to empower cochlear implant users. With Attune, researchers can associate individual tracks with macros that control multiple WAM plug-in parameters at once (Fig. 27). Although programming macro controls and customizing track parameters might have many applications in the music industry, they also present an opportunity to afford deaf or hard-of-hearing (DHH) users greater control over their music listening. In [25], the authors present a case study illustrating how this tool could be used by hard-of-hearing users to modify individual musical elements in a multitrack listening context to create a more enjoyable listening experience. Fig. 28 presents the simplified view for DHH users.

[49] https://blog.paul.cx/post/a-wait-free-spsc-ringbuffer-for-the-web/.

[50] Video: https://www.youtube.com/watch?v=0r6pox2eQH0

[51] https://wam-studio.i3s.univ-cotedazur.fr/

[52] https://github.com/Brotherta/wam-studio

[53] https://github.com/Brotherta/wam-studio/tree/stanford-prototype

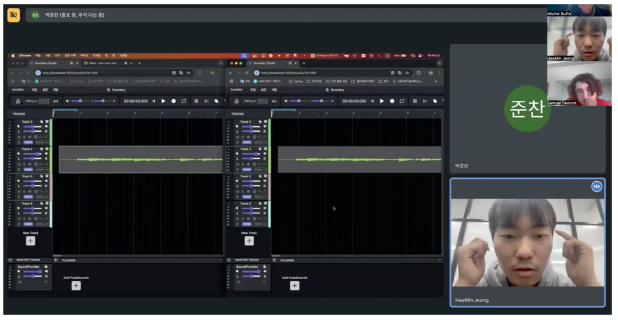


Fig. 29 The soundary.io a commercial real-time collaborative DAW, fork of the WAM-studio open-source DAW.

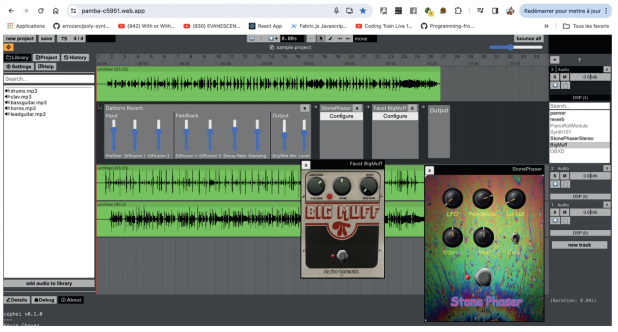


Fig. 30. Cephei: a React-based DAW that supports WAM effects and instruments.

Another WAM-Studio fork is being developed by https://soundary.io/, a Korean commercial company. Their version[54] includes real-time collaborative features, artificial intelligence (AI) tools for source separation and multitrack project generation from prompts (Fig. 29).

Cephei[55] (Fig. 30) is a React-based DAW by Kevin Chavez (@aykev). The author focused on GUI and ergonomy. At the time of writing, the DAW features audio playback, recording, bouncing, and a chain of WAMs associated with each track. The author is working on a WebGPU shader to render out the waveforms at different scales. All track elements are Document Object Model-based (no canvas, track regions are simple HTML " " tags, waveforms are CSS backgrounds, etc.), an original and efficient approach.

An open-source DAW engine that supports WAMs has also been developed by Sascha Knapp [26] and is available on GitHub.[56] The author provides a DAW prototype as a proof of concept.

Sequencer Party[57] (Fig. 31) is a real-time collaborative audio/visual platform built entirely out of WAMs. Users work together in real-time sessions and can share WAM presets and projects publicly on the website. It comes with its own collection of open-source WAMs.[58] and users may load remote WAMs by URI. An online integrated menu

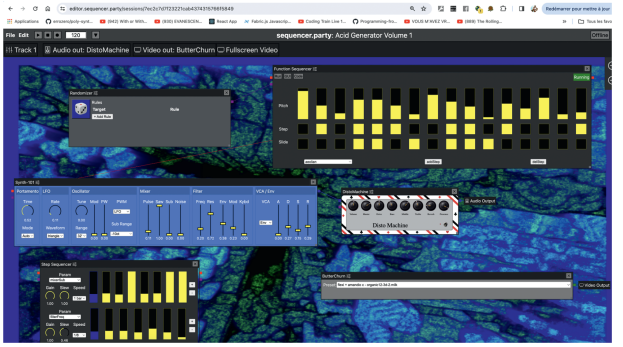


Fig. 31. Sequencer Party, a real-time collaborative musical experience.

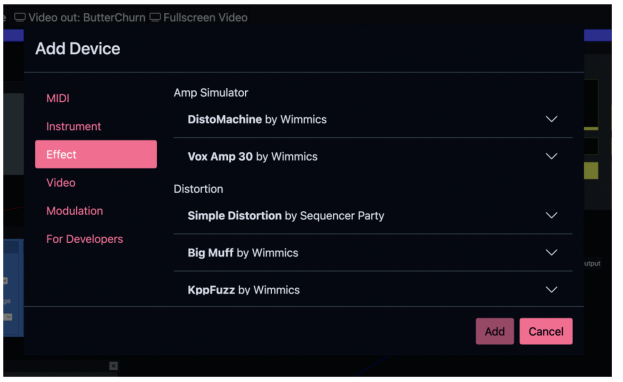


Fig. 32. Sequencer Party menu for adding a WAM plug-in (sequencer, instrument, effect, display) to a track/screen.

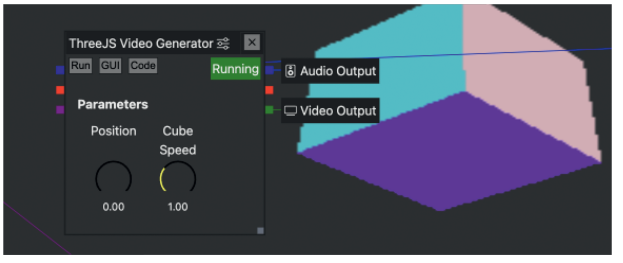


Fig. 33. A WAM in Sequencer Party that can animate 3D scenes using the Three.js library. Both host and plug-in implement the 3D WAM extension.

exposes a categorized list of plug-ins available by querying the WAM Community REST API (Fig. 32). The Sequencer Party host, along with many of the plug-ins it offers, makes extensive use of the WAM extension mechanism described in SEC. 9 (Fig. 33).

JSPatcher [27, 28] is an open-source, online visual programming language[59] in the style of Max/PureData for interactive programming, audio processing, and real-time multimedia projects. It can be used as a graph editor for WebAudio nodes and WAMs (Fig. 34). It also comes with a powerful audio buffer editor (Fig. 35) that can use WAM plug-ins to apply effects to buffer regions. Fig. 36 shows

[54] https://soundary.io/studio/
[55] https://pamba-c5951.web.app/
[56] https://github.com/KnappSas/WebAudioEngine
[57] https://sequencer.party
[58] https://github.com/boourns/burns-audio-wam
[59] https://fr0stbyter.github.io/jspatcher/dist/

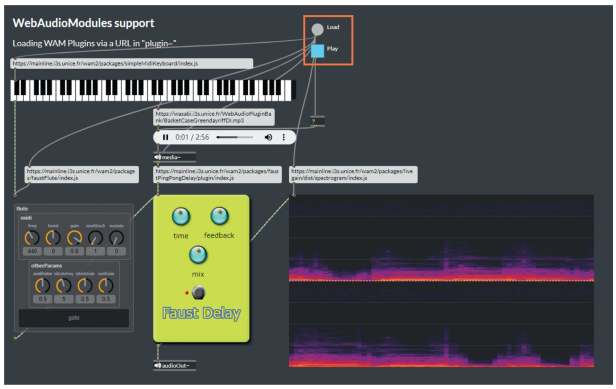


Fig. 34 WebAudioModules in a JSPatcher patch.



Fig. 35. WAM plug-ins in the JSPatcher audio editor.

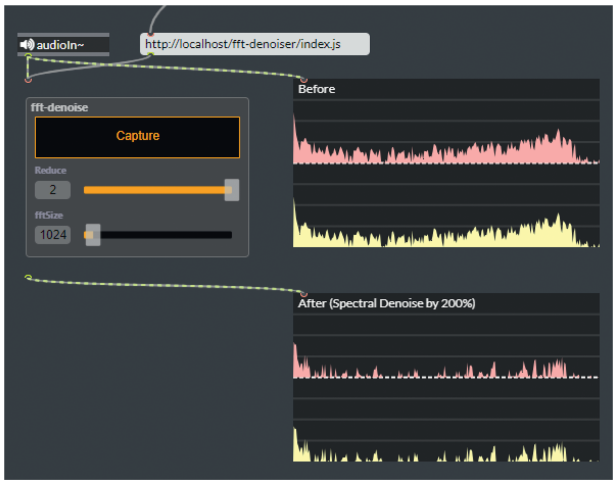


Fig. 36. JSPatcher, a Max/MSP–like host, with some WAMs (a denoiser on the left and some audio visualizations on the right).

an example in JSPatcher where a spectral denoiser WAM plug-in is loaded from a URI. The plug-in processes the microphone input signal and two WAM spectroscopes display the result.

9 WEB AUDIO MODULE EXTENSIONS

9.1 Example Web Audio Module Extensions

Extensibility is a powerful feature of some plug-in APIs. Native plug-in formats like VST3 provide the ability for DAW manufacturers to publish extensions for third parties to more tightly integrate plug-ins with their host. For example, extensions can supply extra context about the DAW project. WAM Extensions[60] provide similar functionality in the WAM2 API.

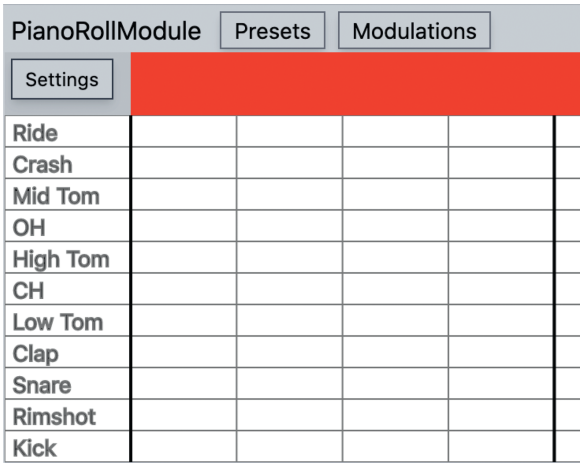


Fig. 37. Piano roll with note names, demonstrating the notes WAM extension.

The **notes extension** allows one WAM to publish a list of relevant notes with names and other connected WAMs to receive the note list. This enables a better UI when sequencing typical drum machines or samplers where only certain MIDI notes can be received (Fig. 37). The WAMSampler plug-in the GitHub repository[61] contains an example HTML host page that loads a piano roll, and the sampler and implements the Note extension. There is also a p5.js example online.[62]

The **video extension** adds video generation and processing capabilities to the WAM ecosystem. The WAM host manages a WebGL2 context, and plug-ins register render handlers to generate and process WebGL2 textures. Video WAMs can also process audio, allowing audio reactivity or simultaneous audio/video stream generation[63] (Fig. 38).

With the **asset extension**, WAM hosts control asset loading and saving. Individual WAM plug-ins can rely on the host for cloud storage, and users can manage all files related to a musical project in the host without having each WAM plug-in store files in separate cloud services or accounts (Fig. 39).

The **ModulationTarget extension** enables plug-in developers to create plug-ins whose purpose is to modulate or control the parameters of another WAM plug-in (Fig. 40). WAM Modulation plug-ins exist for parameter randomization, parameter sequencing, envelope following of an audio signal, and low-frequency oscillator control. By using modulation WAMs, hosts enable deep parameter automation similar to modular synthesis.

[60] https://github.com/boourns/wam-extensions
[61] https://github.com/micbuffa/WAMSampler.git
[62] https://editor.p5js.org/micbuffa/sketches/Kdneg0o2b
[63] Codepen example of using the WAM video extension: https://codepen.io/boourns/pen/rNbOzPY?editors=1010

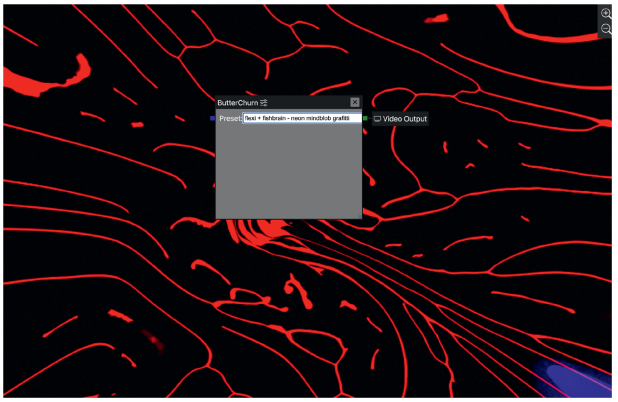


Fig. 38. Butterchurn WAM plug-in, an adaptation of the famous WinAmp Milkdrop plug-in by Ryan Geiss, uses animated WebGL shaders reacting to music.

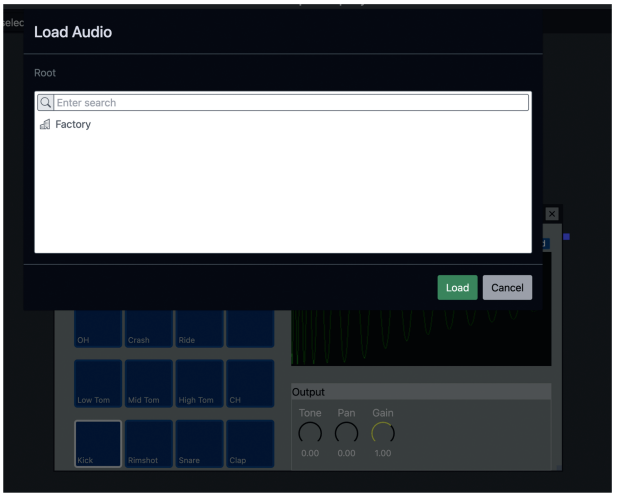


Fig. 39. Drum sampler WAM using the asset extension to allow the host to control asset load/save.

These are just a subset of the available extensions. One may check the WAM extension GitHub repository[64] for the complete list and how to use them.

10 USING THE WEB DNA OF WEB AUDIO MODULES: A MUSICAL MULTIVERSE APPLICATION

Sequencer Party, presented in SEC. 8, allows multiple participants to collaboratively build real-time music installations composed of note generators, instruments, effects, modulators, and more, all of which are WAM plug-ins. *The WAM Jam Party* project [29] is presented here as a transposition of this concept into the *Musical Metaverse* (MM), showcasing for the first time the use of WAM plug-ins in a 3D, collaborative environment.

The MM is an immersive virtual space dedicated to musical activities, expanding on the broader concept of the Metaverse [30]. It has recently gained popularity, reigniting interest in shared virtual environments within the realm of computer music. Applications include virtual concerts, educational tools, and collaborative musical performances. Transposing 2D HTML GUI elements into an immersive 3D environment and attempting to replicate traditional 2D interactions, such as rotating a knob or clicking and dragging a slider using virtual reality controllers, is generally ineffective. The interaction paradigms and interface design must be adapted to suit the specific affordances and constraints of the immersive environment [31].

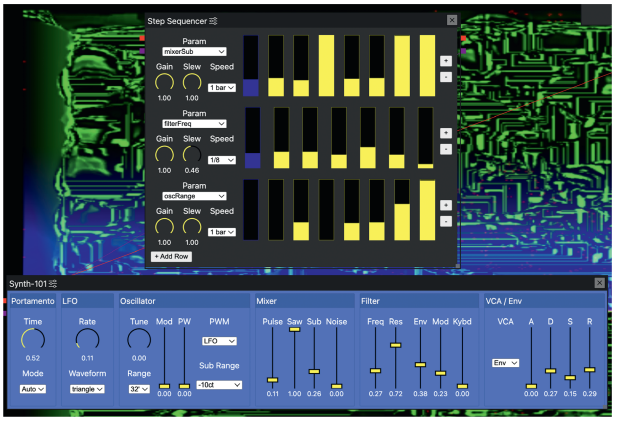


Fig. 40. The step modulator (top) uses the ModulationTarget extension to modulate the Synth-101 parameters.

WAM Jam Party uses WAM introspection to build 3D interactive GUIs from existing 2D WAM plug-ins, BabylonJS for immersive world rendering and a conflict-free replicated data type algorithm to synchronize states incrementally between clients [32, 33]. Although client states are updated three times per second, the server seamlessly saves the session to a file every second, ensuring persistence. Special care is taken in generating the 3D representation of WAM plug-ins, using an optional JSON file to map internal parameters to the 3D world. By default, each WAM is represented as a box with draggable cylinders, boxes, or push buttons for its parameters (Fig. 41). A menu proposes a large set of WAMs, enabling users to add elements in the 3D world. Additionally, a green and right sphere are located on the sides of the box, enabling users to connect WAMs together to form an audio graph. Connect a step sequencer to a synthesizer to some audio effects and what results in the first basic music setup. Any element can be moved, oriented, or have its parameters adjusted using different controllers.

In a multiparticipant session, each user can perform the same interactions. The sound produced by each installation is spatialized and changes as one moves their avatar in the 3D world. WAM Jam Party enables the integration of 3D WAMs into the proposed MM virtual world, and these WAMs can be positioned, oriented, connected to form complex graphs that can be manipulated by all participants, added, or removed (Fig. 42). Adding new WAMs simply requires editing a configuration JSON file, making them available to all participants through the main menu.

[64] https://github.com/boourns/wam-extensions



Fig. 41. 3D representation of the Big Muff WAM. The GUI has been generated on the fly from the WAM parameter description just after it has been dynamically imported in the web application running in the browser of the virtual reality headset.

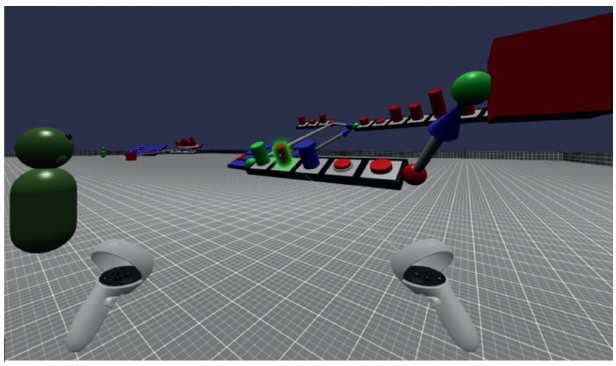


Fig. 42. A musical installation in the foreground, another participant's avatar on the left, and a third participant in the background with the installation they are currently building in the distance.

In conclusion, WAM Jam Party is an extensible construction tool that provides a solid foundation for building and experimenting in the MM. Based on WAM plug-ins and enabling the generation of 3D GUIs on the fly, its design highlights the versatility of the WAM standard by allowing the reuse of existing WAMs, using them in a headless mode, and dynamically creating 3D interfaces. The fact that each WAM has its own state further facilitates the creation of real-time multiuser applications, as demonstrated also by platforms like Sequencer Party and soundary.com. The multiparticipant creation of 3D musical installations through high-level plug-in patching sets WAM Jam Party apart from other works, offering a unique and collaborative approach to music production.

11 FUTURE WORK

Propose good practices and APIs for protecting plug-ins and assets against piracy. The computer music industry relies heavily on commercial plug-ins but faces major piracy issues, leading to revenue loss. Developers use various protection methods, such as license keys, online activation, and hardware dongles like iLok,[65] which offer strong security but reduce convenience. Some embed unique user-identifiable data in plug-ins to track leaks, and others use code obfuscation and encryption to prevent reverse engineering [34, 35], although this can impact performance. Proprietary security systems often combine multiple protective strategies, and trust-based models encourage legitimate purchases through ongoing support. Unlike traditional VST or AAX plug-ins, WAMs function within an open web environment, which makes them more vulnerable to reverse engineering and unauthorized distribution. The very nature of web standards encourages openness and interoperability, which can conflict with stringent copy protection. However, developers can enhance security using JavaScript obfuscation tools like Obfuscator.io or Terser[66] and WebAssembly compilation and encryption tools like Tigress. The WAM 3.0 standard will likely leave copy protection to individual developers, ensuring flexibility and avoiding SDK bloat. A universal security solution could slow updates and limit customization. Instead, WAM 3.0 could provide comprehensive documentation on best practices and optional utility libraries for tasks such as encryption, watermarking, and license key validation.

Use the blockchain for distributing plug-ins and assets? Thanks to its compatibility with WebAssembly and its ability to handle fast and secure transactions via blockchain [36], the NEAR protocol[67] enables developers to create solutions for protecting web-based audio software and musical works. Projects like NEAR Music [37], led by Peter Salomonsen rely on these technologies to ensure the authenticity, traceability, and monetization of musical creations, and it offers a relevant case study for exploring the potential integration of blockchain-driven protection mechanisms in WAM 3.0.

Develop new WAM extensions and/or add new APIs, in particular for AI and/or cloud-based computing. In the age of AI, web-based multimedia software is becoming increasingly relevant as cloud-based back-end processing provides access to powerful AI models running on GPUs on an industrial scale. Unlike native applications, which are limited by the constraints of on-device computing, web-based software can seamlessly exploit these large-scale models, making it the preferred choice for consumers and professionals alike. This shift is particularly important in the

[65] https://www.ilok.com/
[66] https://github.com/terser/terser
[67] https://near.org/

world of DAWs, where AI-driven tools for music production, sound design, and mixing require tremendous computing power (i.e., mastering, unmixing, generated projects, resynthesis, voice changing, all powered by AI). On the other hand, tools like ONNX Runtime Web[68] enables one to run and deploy machine learning models in web plug-ins using JavaScript APIs (this approach also enhances privacy by keeping data processing on the user's device). Interesting open-source PyTorch-based audio libraries like FLAMO [38] could be used to train models for integration in WAM plug-ins. As the adoption of WebAssembly and ONNX continues to grow, it can be expected that more web applications across various domains will be seen, leveraging these technologies to deliver powerful, client-side AI functionalities.

As a result, the long-awaited era of "WebDAW," with its "WebPlugins," is approaching, reinforcing the need for a robust, standardized ecosystem for hosts/plug-ins, such as WAMs. Soundari.io, for example, has taken the open-source DAW WAM-Studio and added collaborative and AI-powered features.

Encourage original, web-oriented projects that could not easily be realized with native applications. Sequencer Party and Wam Jam Party are good examples of real-time collaborative projects for music creation and live performance that take full advantage of WAM's design choices (plug-in states can be easily synchronized, plug-ins can be used in 3D browsers with on-the-fly GUI generation, etc.). As a follow-up to the Wam Jam Party project (a 3D web application, in the MM), the WAM team has partnered with BabylonJS developers at Microsoft to integrate WAM support into the upcoming BabylonJS release. This collaboration will add a new dimension to BabylonJS by enhancing its WebAudio capabilities, going beyond the current features to include interactive sound synthesis, advanced audio effects, and interactive note generators. Currently, the WAM 3D extension is only compatible with the Three.js library, so adaptation is needed. This effort includes refining the WAM 3D and video extensions to enable seamless interactions between audio, geometric models, and textures, bringing a richer and more immersive audio experience to BabylonJS.

Finally, push the project adoption with more educational and documentation initiatives. More comprehensive guides, interactive tutorials and examples, community knowledge bases providing best practices, troubleshooting tips, and real-life use cases would enable newcomers to understand the fundamentals of WAM design and development better and faster. Additionally, hosting workshops, hackathons, and mentorship programs would encourage collaboration and innovation, helping to refine the ecosystem and expand its capabilities.

In the near future, the authors plan to focus on making more tutorials and examples of the existing WAM extensions. There are also plans to develop new collaborative features in WAM-Studio and continue experimenting with 3D WAMs in WebXR environments. Additionally, there is an aim to strengthen the educational use cases with enhanced tutorials and documentation, particularly targeting browser-first music education platforms.

12 CONCLUSION

Today, WAMs have become an established Web Audio plug-in system. It is actively maintained and in constant evolution. The examples proposed when v2.0 was launched in 2021 showed great potential, but at that time, the lack of tutorials, documentation, development tools, third party examples, and a reference website made the first steps difficult. Today, the WAM ecosystem has evolved, and this article highlights some of the most notable contributions that benefit the Audio Web developer community. These developments further reinforce the web browser as a viable platform for computer music, expanding its capabilities for real-time audio production and interactive sound applications [39]. Tools like the Faust online IDE enable rapid and efficient design, testing, and publication of polyphonic WAM instruments and audio effects. Similarly, Cmajor offers a VSCode extension for effectively compiling and exporting fully featured WAM effects or instruments.

The WAM Community initiative is also particularly important because it facilitates the exchange and reuse of existing plug-ins, and it is hoped that other developers will contribute to its content. Several online DAWs are already WAM-compatible, and at least one is open source and uses WAM in its core design (WAM-Studio). Examining its source code is a good way to understand how to design and implement basic mechanisms such as robust recording in sync with other audio tracks, latency compensation, project rendering/bouncing, load/save projects with large audio content, plug-in states, parameter automation, etc.

Last but not least WAM's extension mechanism also opens up many new perspectives, notably by introducing the multimodal aspect (audio and video) that was missing from the initial proposal. New types of applications will surely emerge along the lines of the collaborative application Sequencer Party, the first host application to take advantage of WAM extensions, and its 3D cousin WAM Jam Party.

Between 2015 and today, the WAM standard has evolved significantly. The examples presented in this paper highlight the diversity of plug-ins and host software that have been developed using various approaches, from pure JavaScript code to WebAssembly with cross-compilation from languages like C/C++ or DSLs such as FAUST, Csound, or Cmajor.

To conclude this discussion of the WAM standard, it is worth mentioning a particularly insightful presentation entitled "A Critique of Audio Plug-In Formats - VST, AU, AAX, JUCE, and Beyond" by Fabian Renn-Giles. Presented at the Audio Developer Conference 2024, four years after the release of WAM version 2.0, this talk critically examines the strengths and weaknesses of various native plug-in standards [40]. Although WAM was designed specifically for the web, many of the design choices

[68] https://onnxruntime.ai/docs/tutorials/web/

made in 2020 for its *non–web-specific components* closely align with the best practices outlined in this Audio Developer Conference 2024 presentation. This comparison reinforces the relevance and forward-thinking nature of WAM's core architecture, independently of its web-based aspects.

The primary goal of WAM v2.0 was to first build an open-source community and to evaluate whether the proposed standard could meet common needs, particularly for noncommercial use cases. Now, five years after the release of version 2.0, it is time to start considering potential developments for a future version of the standard.

REFERENCES

[1] J. Kleimola and O. Larkin, "Web Audio Modules," in *Proceedings of the 12th Sound and Music Computing Conference (SMC)* (Maynooth, Ireland) (2015 Jul.-Aug.). https://www.cs.nuim.ie/smc15/files/FinalProceedings.pdf.

[2] O. Larkin, A. Harker, A., and J. Kleimola, "iPlug 2: Desktop Audio Plug-In Framework Meets Web Audio Modules," in *Proceedings of the 4th Web Audio Conference (WAC)* (Berlin, Germany) (2018 Sep.). https://webaudioconf.com/posts/2018_13/.

[3] M. Buffa, J. Lebrun, J. Kleimola, O. Larkin, and S. Letz, "Towards an Open Web Audio Plug-In Standard," in *Companion Proceedings of The Web Conference (WWW)*, pp. 759–766, (Lyon, France) (2018 Apr.). https://doi.org/10.1145/3184558.3188737.

[4] M. Buffa, J. Lebrun, J. Kleimola, O. Larkin, G. Pellerin, and S. Letz, "WAP: Ideas for a Web Audio Plug-In Standard," in *Proceeding of the 4th Web Audio Conference (WAC)* (Berlin, Germany) (2018 Sep.). https://doi.org/10.13140/RG.2.2.25932.69767.

[5] H. Choi, "Audioworklet: The Future of Web Audio," *in Proceedings of the International Computer Music Conference (ICMC)*, (Daegu, South Korea) (2018 Aug.). http://hdl.handle.net/2027/spo.bbp2372.2018.021.

[6] H. Choi and J. Berger, "WAAX: Web Audio API eXtension," in *Proceedings of the International Conference on New Interfaces for Musical Expression (NIME)* (Daejeon, Korea) (2013 May). https://doi.org/10.5281/zenodo.1178494.

[7] M. Buffa and S. Demont. "Can You DAW It Online?," in *Proceedings of the IEEE 5th International Symposium on the Internet of Sounds (IS2)*, (Erlangen, Germany) (2024 Sep.-Oct.). https://doi.org/10.1109/IS262782.2024.10704163.

[8] N. Jillings, Y. Wang, R. Stables, and J.D. Reiss, "Intelligent Audio Plugin Framework for the Web Audio API," in *Proceedings of the 3rd Web Audio Conference (WAC) (WAC)* (London, UK) (2017 Aug.). https://www.open-access.bcu.ac.uk/id/eprint/4964.

[9] M. Buffa, J. Lebrun, S. Ren, S. Letz, Y. Orlarey, R. Michon, and F. Fober, "Emerging W3C APIs Opened Up Commercial Opportunities for Computer Music Application," in *Proceedings of The Web Conference: Developers Track*, (Taipei, Taiwan) (2020 Apr.). https://doi.org/10.13140/RG.2.2.16456.19202.

[10] Y. Orlarey, D. Fober, and S. Letz, "Syntactical and Semantical Aspects of Faust," *Soft Comput.*, vol. 8, pp. 623–632 (2004 Sep.). https://doi.org/10.1007/s00500-004-0388-1.

[11] Y. Orlarey, D. Fober, and S. Letz, "FAUST : An Efficient Functional Approach to DSP Programming," in G. Assayag and A. Gerzo (Eds.) *New Computational Paradigms for Computer Music*, pp. 65–96 (Delatour, Sampzon, France, 2009). https://hal.archives-ouvertes.fr/hal-02159014.

[12] S. Letz, Y. Orlarey, and D. Fober, "Compiling Faust Audio DSP Code to WebAssembly," in *Proceedings of the 3rd Web Audio Conference (WAC)(WAC)* (London, UK) (2017 Aug.). https://qmro.qmul.ac.uk/xmlui/handle/123456789/26145.

[13] S. Ren, S. Letz, Y. Orlarey, D. Fober, R. Michon, M. Buffa, L. Pottier, and Y. Yu, "Modernized Toolchains to Create JSPatcher Objects and Web Audio Modules from Faust Code," *in Proceedings of the 7th Web Audio Conference (WAC)*, (Cannes, France) (2022 Jul.). https://doi.org/10.5281/zenodo.6767596.

[14] S. Ren, S. Letz, Y. Orlarey, R. Michon, D. Fober, M. Buffa, E. Ammari, and J. Lebrun, "FAUST Online IDE: Dynamically Compile and Publish FAUST Code as WebAudio Plugins," in *Proceedings of the 5th Web Audio Conference (WAC)*, (Trondheim, Norway) (2019 Dec.). https://doi.org/10.13140/RG.2.2.28868.71042.

[15] S. Ren, S. Letz, Y. Orlarey, R. Michon, D. Fober, M. Buffa, and J. Lebrun, "Using Faust DSL to Develop Custom, Sample Accurate DSP Code and Audio Plugins for the Web Browser," *J. Audio Eng. Soc.*, vol. 68, no. 10, pp. 703–716 (2020 Oct.). https://doi.org/10.17743/jaes.2020.0014.

[16] V. Lazzarini, S. Yi, J. ffitch, J. Heintz, Ø. Brandtsegg, and I. McCurdy, *Csound: A Sound and Music Computing System*, (Springer AG, Cham, Switzerland, 2016).

[17] R. Boulanger (Ed.), *The Csound Book: Perspectives in Software Synthesis, Sound Design, Signal Processing, and Programming*, (MIT Press, Cambridge, MA, USA, 2000).

[18] S. Yi, V. Lazzarini, and E. Costello, "WebAssembly AudioWorklet Csound," in *Proceedings of the 4th Web Audio Conference (WAC)*, (Berlin, Germany) (2018 Sep.). https://webaudioconf.com/posts/2018_24/.

[19] M. Buffa and J. Lebrun, "Real Time Tube Guitar Amplifier Simulation Using WebAudio," in *Proceedings of the 3rd Web Audio Conference (WAC) (WAC)*, (London, UK) (2017 Aug.). https://doi.org/10.13140/RG.2.2.25093.83682.

[20] M. Buffa and J. Lebrun, "Web Audio Guitar Tube Amplifier vs Native Simulations," in *Proceedings of the 3rd Web Audio Conference (WAC) (WAC)*, (London, UK) (2017 Aug.). https://doi.org/10.13140/RG.2.2.11672.06409.

[21] M. Buffa and J. Lebrun, "Rocking the Web With Browser-Based Simulations of Tube Guitar Amplifiers," *J. Audio Eng. Soc.*, vol. 71, no. 11, pp. 753–768 (2023 Nov.). https://doi.org/10.17743/jaes.2022.0110.

[22] M. Buffa and J. Lebrun, "A Browser-Based WebAudio Ecosystem to Dynamically Play With Real-Time Simulations of Historic Guitar Tube Amps and Their Typi-

cal Distortions," in G. Bromham and A. Moore (Eds.), *A.in Distortion in Music Production: The Soul of Sonics*, pp 28–46 (Focal, Taylor & Francis, London, UK, 2023 Jun.), 1st ed.

[23] M. Buffa and A. Vidal-Mazuy, "WAM-Studio, a Digital Audio Workstation (DAW) for the Web," in *Companion Proceedings of The Web Conference (WWW)*, pp. 543–548 (Austin, TX) (2023 May). https://doi.org/10.1145/3543873.3587987.

[24] C. Wilson, "A Tale of Two Clocks," (2013 Jan.). https://web.dev/articles/audio-scheduling.

[25] M. Buffa, A. Vidal-Mazuy, L. May, and M. Winckler, "WAM-Studio: A Web-Based Digital Audio Workstation to Empower Cochlear Implant User," in *Proceedings of the IFIP Conference on Human-Computer Interaction (INTERACT)*, vol. pp. 101–110, (York, UK) (2023 Aug.-Sep.). https://doi.org/10.1007/978-3-031-42280-5_6.

[26] S. Knapp, *Browser-Based Engine Prototype for a Digital Audio Workstation*, Batchelor's thesis, Hamburg University of Applied Sciences, Hamburg, Germany (2024 Sep.). https://hdl.handle.net/20.500.12738/16337.

[27] S. Ren, L. Pottier, and M. Buffa, "Build WebAudio and JavaScript Web Applications Using JSPatcher: A Web-Based Visual Programming Editor," in *Proceedings of the 6th Web Audio Conference (WAC)*, (Barcelona, Spain) (2021 Jul.). https://webaudioconf.com/posts/2021_4/.

[28] S. Ren, L. Pottier, M. Buffa, and Y. Yu, "JSPatcher, a Visual Programming Environment for Building High-Performance Web Audio Applications," *J. Audio Eng. Soc.,,* vol. 70, no. 11, pp. 938–950 (2022 Nov.). https://doi.org/10.17743/jaes.2022.0056.

[29] M. Buffa, D. Girard, and A. Hofr, "Using Web Audio Modules for Immersive Audio Collaboration in the Musical Metaverse," in *Proceedings of the IEEE 5th International Symposium on the Internet of Sounds (IEEE IS2)*, pp 1–10 (Erlangen, Germany) (2024 Sep.). https://doi.org/10.1109/IS262782.2024.10704108.

[30] L. Turchet, "Musical Metaverse: Vision, Opportunities, and Challenge," *Pers. Ubiquitous Comput.*, vol. 27, no. 5, pp.1811–1827 (2023 Oct.). https://doi.org/10.1007/s00779-023-01708-1.

[31] A. Boe and L. Turchet, "Musical Metaverse Playgrounds: Exploring the Design of Shared Virtual Sonic Experiences on Web Browsers," in *Procedings of IEEE 4th International Symposium on the Internet of Sounds (IEEE IS2)(IEEE IS2)*, pp. 1–9 (Pisa, Italy) (2023 Oct.). https://doi.org/10.1109/IEEECONF59510.2023.10335297.

[32] P. Nicolaescu, K. Jahns, M. Derntl, and R. Klamm, "Near Real-Time Peer-to-Peer Shared Editing on Extensible Data Types," in *Proceedings of the ACM International Conference on Supporting Group Work (Group)*, pp. 39–49 (Sanibel Island, Florida, USA) (2016 Nov.). https://doi.org/10.1145/2957276.2957310.

[33] D. Sun, C. Sun, A. Ng, and W. Cai, "Real Differences Between OT and CRDT in Correctness and Complexity for Consistency Maintenance in Co-Editors," in *The Proceedings of the ACM on Human-Computer Interaction (HCI)*, vol. 4, no. CSCW1, pp. 1–30 (Hamburg, Germany) (2020 Apr.). https://doi.org/10.1145/3392825.

[34] J. Van den Broeck, B. Coppens, and B. De Sutter, "Obfuscated Integration of Software Protections," *Int. J. Inf. Secur.*, vol. 20, pp.73–101 (2021 Feb.). https://doi.org/10.1007/s10207-020-00494-8.

[35] S. A. Ebad, A. A. Darem, and J. H. Abawajy, "Measuring Software Obfuscation Quality–A Systematic Literature Review," *IEEE Access*, vol. 9, pp. 99024–99038 (2021 Jul.). https://doi.org/10.1109/ACCESS.2021.3094517.

[36] A. Qureshi and D. Megías Jiménez, "Blockchain-Based Multimedia Content Protection: Review and Open Challenges," *Appl. Sci.*, vol. 11, no. 1 (2021 Jan.). https://doi.org/10.3390/app11010001.

[37] P. Salomonsen, *Building and Deploying WebAssembly Apps: Building Secure, Portable, and Optimized Web Applications With WebAssembly* (BPB Publications, New Delhi, India, 2025).

[38] G. Dal Santo, G. M. De Bortoli, K. Prawda, S. J. Schlecht, and V. Välimäki, "FLAMO: An Open-Source Library for Frequency-Domain Differentiable Audio Processing," in *Proceedings of the IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*, pp. 1–5 (Hyderabad, India), (2025 Apr.). https://doi.org/10.1109/ICASSP49660.2025.10888532.

[39] L. Wyse and S. Subramanian, "The Viability of the Web Browser as a Computer Music Platform," *Comput. Music J.*, vol. 37, no. 4, pp.10–23 (2013 Dec.). https://doi.org/10.1162/COMJ_a_00213.

[40] F. Renn-Giles, "A Critique of Audio Plug-In Formats - VST, AU, AAX, JUCE and Beyond," in *Proceedings of the Audio Developer Conference (ADC)*, (Bristol, UK) (2024 Nov.). https://www.youtube.com/watch?v = nPJpX8GR9d4.

THE AUTHORS


Michel Buffa


Jerome Lebrun


Shihong Ren


Stephane Letz


Tom Burns


Steven Yi


Yann Orlarey


Cesare Ferrari


Jari Kleimola


Oliver Larkin

Michel Buffa is a professor/researcher at University Côte d'Azur, a member of the Web-instrumented Man-Machine Interactions, Communities, and Semantics (WIMMICS) research group, common to the National Institute for Research in Digital Science and Technology (Institut National de la Recherche en Informatique et Automatique [INRIA]) and to the Computer Science, Signals, and Systems laboratory of Sophia Antipolis (Centre National de la Recherche Scientifique [CNRS]). He has contributed to the development of the Web Audio research field, having he participated in all Web Audio Conferences, being part of each program committee between 2015 and 2019, and organizing the 2022 edition in Cannes, France. He actively works with the W3C Web Audio working group.

•

Jerome Lebrun, Dr.Sc., Ing.Dipl., is a full-time CNRS researcher at the University Côte d'Azur, leader of the Biomedical Signal Processing group of the Computer Science, Signals, and Systems laboratory of Sophia Antipolis, and head of the Signals, Image, and Systems (SIS) research division. His current interests and research fields include signal processing for computer music, ciphered-speech secure communications, multimodal EEG/f-NIRS/f-MRI acquisitions, and the analysis of animal vocalizations.

•

Shihong Ren, Ph.D., is a composer/researcher in computer music, currently a postdoctoral fellow at Shanghai Conservatory of Music, and a member of the Shanghai Key Laboratory for Music Acoustics. He entered the Conservatoire National Supérieur Musique et Danse de Lyon in 2011 in the electroacoustic composition class and graduated in 2016 as the youngest laureate of a Diplôme National Supérieur Professionnel du Musicien (DNSPM) and master's degree owner in the composition major. He got the Artist Diploma in 2018 and followed the course of composition in the Institute for Research and Coordination in Acoustics/Music (IRCAM) in the same year. He received his Ph.D. in computer music in 2024.

•

Stephane Letz is the head of the computer music research laboratory at the Centre national de création musicale in Lyon, France "Générateur de Ressources et d'Activités Musicales Exploratoires" and a researcher in the Emeraude Research team at National Institute for Research in Digital Science and Technology (INRIA). He is a coauthor of the Faust programming language, which was created by Yann Orlarey in 2002, and a core member of the Faust development team. He has previously been involved in the development of the JACK Audio Connection Kit (JACK), particularly on the jack2 version, which introduced multiprocessor scalability and support for operating systems other than Linux.

•

Tom Burns is a musician, event organizer, and programmer in Ottawa, Ontario, Canada. Between 2020 and 2022 he developed the multiplayer Web Audio Module platform Sequencer Party. Today he co-runs Catalyst Ottawa, a monthly electronic music open mic night.

•

Steven Yi is a composer, performer, and programmer that works on open-source music technology and multimedia applications. He is one of the core developers of Csound, the author of the Blue music composition environment, and author of the Csound-live-code live coding system.

•

Yann Orlarey is a composer and researcher in computer music. He cofounded the Centre national de création musicale "Générateur de Ressources et d'Activités Musicales Exploratoires" (GRAME-CNCM), a center for musical creation and research in Lyon, France, where he served as scientific director until the end of 2022. Along with Stéphane Letz and colleagues at GRAME, he has authored or coauthored several music software programs, most notably the Faust programming language. He is currently at National Institute for Research in Digital Science and Technology (INRIA; Emeraude team), where his research focuses on advancing the Faust programming language and its compiler.

•

Cesare Ferrari is a programmer who has specialized in audio signal processing and real-time programming. He is the cocreator of the Cmajor programming language, designed for real-time audio processing and digital signal processing (DSP), offering a safe and efficient alternative to C++ for developing audio software and plug-ins.

•

Jari Kleimola is a researcher and developer specializing in web technologies, sound synthesis, and audiovisual interaction. He earned his Doctor of Science (Technology) degree from Aalto University in the department of media technology. In 2015, Kleimola coauthored the initial proposal for Web Audio Modules and significantly contributed to integrating native digital audio workstation (DAW) and plug-ins into web environments. In 2015 he cocreated Amptrack Technologies and created one of the first commercial online DAWs (Amped Studio).

•

Oliver Larkin is a software engineer at Ableton and a prolific contributor to open-source audio projects over the last 20 plus years. He has developed his own successful audio plug-in products and has worked with other well-known brands such as Arturia, Focusrite and Roli. Oli maintains the liberally licensed iPlug2 C++ audio plug-in framework that many other developers use to release their projects, including as Web Audio Modules.