Audio Engineering Society

Convention e-Brief 301

Presented at the 141st Convention
2016 September 29 – October 2, Los Angeles, CA, USA

# JSAP: A Plugin Standard for the Web Audio API with Intelligent Functionality

Nicholas Jillings[1], Yonghao Wang[1], Joshua Reiss[2], and Ryan Stables[1]

[1]*Digital Media Technology Lab, Birmingham City University, Birmingham, UK*
[2]*Centre for Digital Music, Queen Mary University of London, London, UK*

Correspondence should be addressed to Nicholas Jillings (nicholas.jillings@mail.bcu.ac.uk)

**ABSTRACT**

In digital audio, software plugins are commonly used to implement audio effects and synthesizers, and integrate them with existing software packages. Whilst these plugins have a number of clearly defined formats, a common standard has not been developed for the web, utilising the Web Audio API. In this paper, we present a standard framework which defines the plugin structure and host integration of a plugin. The project facilitates a novel method of cross-adaptive processing where features are transmitted between plugin instances instead of audio routing, saving on multiple calculations of features. The format also enables communication and processing of semantic data with a host server for the collection and utilisation of the data to facilitate intelligent music production decisions.

## 1 Introduction

The Web Audio API, introduced by the World Wide Web Consortium's (W3C) HTML 5, defines a cross-browser interface for building real time audio processing, controlled by the JavaScript environment [1]. It is supported on every major browser[1] and has led to a wide range of web applications being developed to leverage the new flexibility. These include playable additive synthesisers [2], an interactive music player based on MPEG-A [3] to digital audio workstations[2].

The Web Audio API eXtension (WAAX)[4] proposes a method for building audio effects along with Tuna[3]. In both cases audio effects are loaded by a loading mechanism and are built using an internal constructor. They also use a method of control which, whilst in keeping with the Web Audio API, make it difficult for an application to understand the parameters without explicitly knowing the effect. Web Audio Modules (WAM) [5] attempt to define a more robust environment, similar to traditional desktop plugins by separating the audio processor and controller. However this assumes that the entire processing chain is a customisable script to be called, rather than using the available low-level audio nodes.

In this paper we introduce a new plugin format specifically defining both the plugin and the host requirements. We introduce the core modules for building, deploying and interacting with plugins. The paper also describes the intelligent and semantic aspects attached to the environment to aid Intelligent Music Production (IMP).

[1]At time of writing, only Opera Mini is not supported, see http://caniuse.com/#feat=audio-api
[2]Soundtrap uses the Web Audio API, available at https://www.soundtrap.com/
[3]Available at https://github.com/Theodeus/tuna

## 2 Architecture

The core modules of the standard are the Base Plugin and the Plugin Factory. These define the host interface and the plugin structure respectively. The Base Plugin wraps up an audio graph within its object prototype, enabling multiple instances of the same design to be repeated with ease. The Plugin Factory enables advanced host side interaction including multi-track support, semantic interaction and feature sharing.

### 2.1 Base Plugins

The Web Audio API enables the design of complex processing chains, where the individual audio processors are linked together to form an audio graph from source nodes to the audio output of the device. The graph is built in JavaScript and simply holds a reference to the lower-level processors to send in audio control parameters, configure the routing and control playback. Therefore it is easy to prototype this graph and store the steps to build such a graph in an object prototype (constructor).

```
// First we create the web audio API
    gain node
var gain = this.context.createGain();

// We have only one input and one output
    , which is the gain node:
_inputList[0] = gain;
_outputList[0] = gain;

// Next we must create the parameter,
    again there is only one:
var gainParam = new PluginParameter(1, "
    Number", "Volume", 0, 2, this);
// This creates a parameter called
    Volume, which uses a real number, an
     acceptable range of 0 to 2 and
    defaults at 1
gainParam.bindToAudioParam(gain.gain);
// We have bound the parameter to the
    web audio gain node parameter gain.

// We can add the gainParam to the
    parameter list presented to the
    plugin by:
_parameters.push(gainParam);
```

**Listing 1:** Extract of volume plugin showing gain node creation, connection and parameters

The sub-graph is a term used to describe the prototype audio graph held by the prototype object. For instance, a 3 band parametric equaliser may use three BiquadFilterNodes together with an input and output gain node. This will give three, parametric EQ's on variable bands for use. In the example given in listing 1, where a single gain node is being created, the sub-graph is simply that one node.

The plugin must also define the input and output points of the sub-graph. Each plugin can have multiple input and output points but must have at least one of each. These can feed either audio elements or parameters through conversion functions. In listing 1, these are shown as `_inputList[0] = gain;` and `_outputList[0] = gain;`.

Each plugin instance reports its current inputs and outputs by calling the getters `inputs` and `outputs` respectively, returning an array of the inputs and output nodes. By default, the plugin chain will always connect to and from the inputs and outputs at index zero. To ease integration with other audio elements, it defines a new prototype function bound onto every AudioNode called `getInputs()` which returns the first audio input. On normal web audio nodes it simply returns itself, creating a safe way of linking nodes easily.

The main interaction with plugins is through their parameters, providing bounded inputs mapped onto audio effects. Parameters are generated through the PluginParameter. The created object defines the initial value, the data type (Number, Text, Boolean or Event), parameter name, minimum and maximum values. Listing 1 shows the parameter `gainParam` being created, with a default value of 1, of type "Number", called "Volume" with an inclusive range of 0 to 2. The parameters can be bound directly to an AudioNode's own AudioParam object. For instance, the web audio gain node has one AudioParam object: `gain`. By binding it directly onto an AudioParam, any interactions are immediately translated to the node.

### 2.2 Plugin Factory

The host is defined to ensure as seamless an integration as possible into projects. The PluginFactory holds the prototype objects for creation into an audio chain as well as providing a single point of reference for the rest of the application. The plugin instances are inserted into a chain, as found in traditional DAW environments. These individual chains are managed by SubFactories, where each SubFactory on creation is given it's start and stop nodes for the chain. In an empty chain (as when first initiated) the two nodes are directly connected.

When a plugin is created it is placed into the chain and connected between the start and stop nodes. Individual plugin instances can be moved around within the chain, destroyed or moved to other SubFactories.

The PluginFactory also provides links to data stores, either for data collection or requesting. In doing so it enables plugins to directly communicate with connected web and semantic web databases. This will be discussed further in section 4. The factory also manages inter-plugin communications, enabling complex cross-adaptive processing by feature-driven modulation. This is discussed further in section 3.

### 2.3 Host Requirements

All the parameters generated by the plugins are accessible by calling `getParameters()` which returns an object with the parameter values (name, data type, value and range). This defined object enables the automatic generation of Graphical User Interface (GUI) for the parameters, matching the style or constraints of the host.

Each plugin can also generate a HTML structure to present a customised GUI , instead of the host automated interface. This can support more complex elements including feedback as well as branding or other graphical elements. Non parametrised items can also be represented, such as frequency response maps or meters. If the custom HTML GUI is not supported, then the host must automatically generate a suitable interface.

## 3 Cross-Adaptive processing

Cross-adaptive audio effects are a class of audio effect, in which a plugin's instantaneous parameter value is determined by another audio signal's features [6], whilst auto-adaptive effects use the same audio [7]. Early cross-adaptive effects used analog processing to control the volume of microphones in a conference environment [8]. Modern systems use similar methods for real-time or live environment processing, including [9] and [10], using a dedicated audio signal for determining the venue loudness and mix features. In [11], [12], [13] and [14] the audio features are derived from the incoming channels and fed into a computation engine without using an external signal and can be considered a multi-channel auto-adaptive effect.

All of these processors begin by extracting certain features from the audio stream(s) to determine the internal processing. By routing audio to the individual plugins there is the potential for the same feature to be processed multiple times. For instance if channel 0 and channel 1 both require the mean of channel 2, traditionally the audio is routed from 2 to both channels 0 and 1, then both channels calculate the mean. This wastes resources as the same feature is calculated twice. Instead the plugin instances request for certain features from a different plugin. The requested features are calculated for each audio frame and returned to the PluginFactory which dispatches the features to the plugins. Therefore there are no redundant calculations of features, nor are there extra routing paths to maintain. Using this method does incur a latency in the processing, since the features extracted may not directly map frame-for-frame from the external source to the internal source due to its asynchronous nature. This difference must be accounted for, preferably without inducing any further in-line delays.

Feature extraction is performed in real-time using JS-Xtract [15]. This library uses the Web Audio API analyser node to extract the time and frequency domain information from an audio stream. It supports a wide range of features and automatically generates the objects to transfer. the library is written in a modular format, making it extensible and easy to contribute custom features. In JASP, designing auto-adaptive effects is possible by querying the previous plugin for features and using them internally, or placing an AnalyserNode inside the plugin sub-graph.

## 4 Semantic Interaction

Several ontologies have been created specifically for the audio field including the music ontology [16], studio ontology [17] and audio effects ontology [18]. By using these ontologies to describe the audio production process from inception (studio ontology) to distribution (music ontology), it is possible to create complex services by scanning databases which support these ontologies. [19] extends the music ontology to link the music performed to the location, date and details of the performers. [20] define an audio plugin recommendation service by collecting information of the transforms and types of plugins to help an end user find a suitable plugin.

[21] use semantic data to hold the data gathered from their plugins including the parameter state, audio features of the input and output signals, instrument applied to, descriptive terms ("warm" or "bright") and information regarding the engineer using the plugin. By using this extensive dataset it has been possible to generate more intuitive plugins. [22] uses these collected features terms to build a novel equaliser effect where users navigate a 2D plane, referring to two terms, allowing the ability to find settings between terms.

The PluginFactory can be fed global semantic information about the session such as tempo, sample rate and any user or personal information. The SubFactory are fed track specific terms such as the instrument used, where the audio events occur, which channels it feeds or is fed from. Most of these are described using the studio, event and timeline ontologies. Each Plugin itself is given both the global and its owner (SubFactory) semantic terms to operate with. By doing this each plugin can communicate with its own semantic store, enabling a decentralsied semantic web [23]. The plugins can automatically generate linked-data representations of themselves, such as the parameter controls, current states, presets and audio transforms for easy transmission into a data store. The system transmits both JSON-LD or RDF/XML.

## 5 Deployment & Use Cases

The code is self-contained in a single javascript file, holding all constructors for the PluginFactory, SubFactory and the Base Plugin prototype functions. The repository also contains several example plugins for building your own with extensive comment sections. The repository can be downloaded from `http://www.semanticaudio.co.uk/jsap/` along with examples and documentation.

A first use-case of the plugin standard converts three of the SAFE plugins [21], the equaliser, compressor and overdrive, into JSAP instances. Currently the site[4] enables the user to drop an example audio file onto the page for listening and browse the semantic terms associated with the plugin to load the associated parameters. These plugins will be used to extend the SAFE dataset further by gaining more participants and targetted collection of terms.

[4]Available at `http://dmtlab.bcu.ac.uk/nickjillings/safe-js/`

## 6 Conclusion

This engineering brief has introduced an audio plugin standard for building semantically linked audio plugins for the web. The standard eases integration and building of 'sub-graphs' holding prototype audio graphs built on the web audio API [1]. The standard introduces a novel way of moving features instead of audio streams in the host for building intelligent audio processors. The holding plugin factory also connects the plugins to the host semantically for bidirectional data exchange of semantic information.

## References

[1] Adenot, P. and Wilson, C., "Web audio API," 2013.

[2] Teaford, L., "Designing Synthesizers with Web Audio," in J. Freeman, A. Lerch, and M. Paradis, editors, *Proceedings of the 2nd Web Audio Conference (WAC-2016)*, Atlanta, GA, USA, 2016.

[3] Herrero, G., Kudumakis, P., Tardón, L. J., Barbancho, I., and Sandler, M., "An html5 interactive (mpeg-a im af) music player," in *Proceedings of the 10th International Symposium on Computer Music Multidisciplinary Research (CMMR), Marseille, France*, pp. 15–18, 2013.

[4] Choi, H. and Berger, J., "WAAX: Web Audio API eXtension." in *NIME*, pp. 499–502, 2013.

[5] Kleimola, J. and Larkin, O., "Web Audio Modules," in *Proceedings of the Sound and Music Computing 2015*, 2015.

[6] Reiss, J. D., "Intelligent systems for mixing multichannel audio," in *2011 17th International Conference on Digital Signal Processing (DSP)*, pp. 1–6, IEEE, 2011.

[7] Verfaille, V., Zolzer, U., and Arfib, D., "Adaptive digital audio effects (A-DAFx): A new class of sound transformations," *IEEE Transactions on audio, speech, and language processing*, 14(5), pp. 1817–1831, 2006.

[8] Dugan, D., "Automatic microphone mixing," *Journal of the Audio Engineering Society*, 23(6), pp. 442–449, 1975.

[9] Perez-Gonzalez, E. and Reiss, J., "Automatic mixing: live downmixing stereo panner," in *Proceedings of the 7th International Conference on Digital Audio Effects (DAFx'07)*, pp. 63–68, Bordeux, France, 2007.

[10] Perez-Gonzalez, E. and Reiss, J., "Automatic gain and fader control for live mixing," in *IEEE Workshop on applications of signal processing to audio and acoustics*, pp. 1–4, New Paltz, NY, USA, 2009.

[11] Clifford, A. and Reiss, J., "Calculating time delays of multiple active sources in live sound," in *Audio Engineering Society Convention 129*, Audio Engineering Society, 2010.

[12] Jillings, N., Clifford, A., and Reiss, J. D., "Performance optimization of GCC-PHAT for delay and polarity correction under real world conditions," in *Audio Engineering Society Convention 134*, Audio Engineering Society, 2013.

[13] Maddams, J. A., Finn, S., and Reiss, J. D., "An autonomous method for multi-track dynamic range compression," in *Proceedings of the 15th International Conference on Digital Audio Effects (DAFx-12)*, 2012.

[14] Terrell, M., Reiss, J. D., and Sandler, M., "Automatic noise gate settings for drum recordings containing bleed from secondary sources," *EURASIP Journal on Advances in Signal Processing*, 2010, p. 10, 2010.

[15] Jillings, N., Bullock, J., and Stables, R., "JS-Xtract: A Realtime audio feature extraction library for the web," in *International Society for Music Information Retrieval Conference*, 2016.

[16] Raimond, Y., Abdallah, S. A., Sandler, M. B., and Giasson, F., "The Music Ontology." in *ISMIR*, volume 422, Citeseer, 2007.

[17] Fazekas, G. and Sandler, M. B., "The Studio Ontology Framework." in *ISMIR*, pp. 471–476, 2011.

[18] Wilmering, T., Fazekas, G., and Sandler, M. B., "The Audio Effects Ontology." in *ISMIR*, pp. 215–220, 2013.

[19] Shaw, R., Troncy, R., and Hardman, L., "Lode: Linking open descriptions of events," in *Asian Semantic Web Conference*, pp. 153–167, Springer, 2009.

[20] Wilmering, T., Fazekas, G., Allik, A., and Sandler, M. B., "Audio Effects Data on the Semantic Web," in *Audio Engineering Society Convention 139*, Audio Engineering Society, 2015.

[21] Stables, R., Enderby, S., De Man, B., Fazekas, G., and Reiss, J., "SAFE: A system for the extraction and retrieval of semantic audio descriptors," in *15th International Society for Music Information Retrieval Conference (ISMIR 2014)*, 2014.

[22] Stasis, S., Stables, R., and Hockman, J., "A model for adaptive reduced-dimensionality equalisation," in *Proceedings of the 18th International Conference on Digital Audio Effects, Trondheim, Norway*, volume 30, 2015.

[23] Berners-Lee, T., Hendler, J., Lassila, O., et al., "The semantic web," *Scientific american*, 284(5), pp. 28–37, 2001.