JS-Plugin An audio plugin format for Web Audio API

1 Introduction

This project aims to build a common standard of plugin specification for distributed plugins using the Web Audio API. Custom audio DSP is supported in the API through the use of the custom ScriptProcessorNode, to be replaced with the new AudioWorker. However these, along with the rest of the nodes, do not have a standardised way of having their parameters exposed, nor do they support semantic features or cross-adaptive processing without extensive user programming.

This standard will help to bring a common approach to building and deploying custom processors. It formally defines a class for the plugin DSP (BasePlugin), parameters and user interface. We also have modules to perform feature extraction and a plugin factory to ease cross-adaptive effects building.

2 Base Plugin

The Base Plugin is the fundamental class of the function and every JS-plugin must use the prototpye functions defined. There are also several other functions which must be used which are applied on the local scope, therefore we have included these in a template.

As with most plugins on traditional DAWs, external objects must not be able to modify the internal objects directly, only through the exposed parameters that the programmer specifies. This allows the designed plugins to operate their own space and abstract controls like traditional plugins.

To aid this there is the included PluginParameter class which has several variables applied to it on construction to define it. Using these objects it is possible to send control information to the plugin. See 2.1 for more information.

The Base Plugin has several key functions to enable seamless integration with existing Web Audio API projects. Each node has the traditional connect and disconnect functions which map onto the private internal web audio objects. There are further member functions for interacting with the defined parameters and several

member variables for detecting the number of inputs, outputs and parameters and accessing these directly.

2.1 Plugin Parameter

The PluginParameter object gives a method for controlling the plugin. The objects take the following variables in its constructor:

- Default value: The initial value to set the parameter to.
- Data Type: The data type, can be Number, String or Array
- Name: String of the friendly name to give the parameter. These must be unique within each parameter.
- Minimum: Lower bound (can be undefined)
- Maximum: Upper bound (can be undefined)

These can be read directly, for instance for providing data into a scriptProcessorNode, or they can be bound to a Web Audio AudioParam object. This would follow the AudioParam value and change the underlying parameter directly.

Alternatively each parameter can have it's "onchange" function modified to run a script each time the parameter is updated allowing one parameter to control multiple nodes and/or convert the incomming parameter to different values.

2.2 Writing a plugin

A blank template is provided to help start building a plugin along with several working example plugins. Each plugin has several items that must be changed by the User. Firstly the plugin name at the top must be modified to ensure the plugin is correctly loaded. This should be as unique and descriptive as possible. The same name must be used to replace the BlankPlugin variables at the bottom of the page as well.

Your code goes between the commented lines near the top. At least one web audio node must be appended to the _input array and another to the _output array. The nodes in these lists are accessible outside this list so the plugin can work with the rest of the audio nodes. If only one node is used, or you are doing a passthrough

effect, the same node can be used for both the input and output arrays.

A parameter is then created using the in-built PluginParameter node. If a parameter is to be externally available, the node must be pushed onto the _parameters array. Only parameters in this external parameter node are available for modification using the external parameter object nodes.

3 Plugin Factory

The plugin factory enables several plugin prototypes to be loaded into one parent object. This object can then co-ordinate plugin creation as well as plugin communication. The Factory keeps a list of all the plugins that are created in the session, allowing plugins to communicate. This is important for the creation of cross-adaptive effects and operating closer to 'host' environments (such as sending events and triggers to plugins for start/stop effects).

4 Feature Extraction

5 Cross-Adaptive Effects