JSAP Tutorial: Build and deploy web audio effects

Nicholas Jillings Ryan Stables

Digital Media Technology Lab
Birmingham City University
Curzon Street, Birmingham, UK

3rd Web Audio Conference (WAC-2017), August 21–23, 2017

Introduction

Aims of this tutorial:

- Learn how the JSAP standard works
- How to deploy JSAP into your projects
- Build your first JSAP plugin (start thinking of ideas).
- Learn how to load plugins into the host and operate

JSAP Overview

Each JSAP plugin is made up of several objects:

- `BasePlugin` - The inherited object which defines every required interaction of the JSAP instance.
  - `ParameterManager` - Built and accessed as `this.parameters`, holds all the constructors for parameters and interface for manipulating them.
  - `PluginFeatureInterface` - Allows for sharing of audio features between plugins (not going to cover today).
  - `PluginUserInterface` - For building of plugin GUI. Can hold several suitable options and interfaces.

These are all built for you on construction.

Get JSAP

Please clone the following GitHub resource:
`https://github.com/nickjillings/jsap-wac-tutorial`
This will give you the latest JSAP, jsap-plugins and jsap-sandbox environments.
Then open the file `helloworld.js` in your editor.

Get JSAP

```
var HelloWorld = function (factory, owner) {

    // This attaches the base plugin items to the Object
    BasePlugin.call(this, factory, owner);

    /* USER MODIFIABLE BEGIN */

    /* USER MODIFIABLE END */
};

// Also update the prototype function here!
HelloWorld.prototype = Object.create(BasePlugin.prototype
    );
HelloWorld.prototype.constructor = HelloWorld;
HelloWorld.prototype.name = "HelloWorld";
HelloWorld.prototype.version = "1.0.0";
HelloWorld.prototype.uniqueID = "JSHW";
```

Next Steps

HelloWorld is just the shell, right now it is empty! We need to fill it with:

1. A graph! What Web Audio Nodes should we add? What effect shall we create?
2. Parameters! How shall an end user interact with the plugin? What are the parameter names? How do they map onto our nodes?
3. GUI! How shall our plugin look? (We'll cheat today and let the sandbox host manage that).
4. I/O. What is the audio insert point and exit point.

Defining the graphs

The audio context is given to all plugins as `this.context`.
Can easily build a simply `GainNode` by `var node = this.context.createGain();`
Inside the plugin, encapsulation allows us to build our nodes in private.

```
/* USER MODIFIABLE BEGIN */
var node = this.context.createGain();
/* USER MODIFIABLE END */
```

Adding Parameters

JSAP supports several parameter types:

- `NumberParameter`
  - Standard numerical range parameter control
  - Can set min/max ranges
  - `this.parameters.createNumberParameter`
- `StringParameter`
  - Send / Receive a string with the plugin
  - Can set maximum string length
  - `this.parameters.createStringParameter`
- `ButtonParameter`
  - Trigger Parameter
  - Same behaviour as HTML`<button>` element
  - `this.parameters.createButtonParameter`
- `SwitchParameter`
  - Iterable parameter
  - Similar to dropdown. Fixed number of interactions
  - Can pass through (up/down, next/prev etc) or set to specific value
  - `this.parameters.createSwitchParameter`

Adding Parameters

All parameters have a similar constructor:

```
createNumberParameter(name, default, min, max);
createStringParameter(name, default, maxLength);
createButtonParameter(name, default);
createSwitchParameter(name, default, minState, maxState);
```

Must define the name, a unique string to identify the parameter, and the default value (what to initialise it to).

Adding Parameters

Let's add our parameter to our gain node:

```
/* USER MODIFIABLE BEGIN */
var node = this.context.createGain();
var gain_parameter = this.parameters.
    createNumberParameter("gain", 0, -12, 12);
/* USER MODIFIABLE END */
```

Adding Parameters

Now we must bind the parameter. It must do something! Two ways to do this:

1. If a parameter maps to one node parameter, use `bindToAudioParam`
   - For instance: `gain_parameter.bindToAudioParam(node.gain)`.
2. If one parameter to many node parameters, or sharing a node parameter, use `trigger` to write function
   - For instance:

```
gain_parameter.trigger = function(v) {
    node.gain.value = v;
}
```

Adding Parameters

Some parameters may not directly map onto a parameter. In the example, the gain parameter has a range of -12 to +12dB, but the Gain Node is a linear parameter.
JSAP has inbuilt conversion functions: `translate` and `update`.

- `translate` - Convert the AudioNode parameter to the JSAP parameter space
- `update` - Convert the JSAP parameter value to the AudioNode parameter space

Adding Parameters

```
gain_parameter.translate = function (v) {
    return 20.0 * Math.log10(v);
};
gain_parameter.update = function (v) {
    return Math.pow(10, v / 20.0);
};
```

- `translate` - Convert the AudioNode parameter to the JSAP parameter space
- `update` - Convert the JSAP parameter value to the AudioNode parameter space

Adding Parameters

```
/* USER MODIFIABLE BEGIN */
var node = this.context.createGain();
var gain_parameter = this.parameters.
    createNumberParameter("gain", 0, -12, 12);
gain_parameter.translate = function (v) {
    return 20.0 * Math.log10(v);
};
gain_parameter.update = function (v) {
    return Math.pow(10, v / 20.0);
};
gain_parameter.bindToAudioParam(node.gain)
/* USER MODIFIABLE END */
```

Add Inputs and Outputs

The final step is to define what the input and output points of the plugin are (for the audio stream).

```
/* USER MODIFIABLE BEGIN */
var node = this.context.createGain();
var gain_parameter = this.parameters.
    createNumberParameter("gain", 0, -12, 12);
gain_parameter.translate = function (v) {
    return 20.0 * Math.log10(v);
};
gain_parameter.update = function (v) {
    return Math.pow(10, v / 20.0);
};
gain_parameter.bindToAudioParam(node.gain)
this.addInput(node);
this.addOutput(node);
/* USER MODIFIABLE END */
```

Completed Plugin

You should now have a fully operational plugin!

```
var HelloWorld = function (factory, owner) {

    // This attaches the base plugin items to the Object
    BasePlugin.call(this, factory, owner);

    /* USER MODIFIABLE BEGIN */
    var node = this.context.createGain();
    var gain_parameter = this.parameters.createNumberParameter("gain", 0, -12, 12);
    gain_parameter.translate = function (v) {
        return 20.0 * Math.log10(v);
    };
    gain_parameter.update = function (v) {
        return Math.pow(10, v / 20.0);
    };
    gain_parameter.bindToAudioParam(node.gain)
    this.addInput(node);
    this.addOutput(node);
    /* USER MODIFIABLE END */
};

// Also update the prototype function here!
HelloWorld.prototype = Object.create(BasePlugin.prototype);
HelloWorld.prototype.constructor = HelloWorld;
HelloWorld.prototype.name = "HelloWorld";
HelloWorld.prototype.version = "1.0.0";
HelloWorld.prototype.uniqueID = "JSHW";
```

Further considerations

- Always update the prototype information at the bottom! The name-version-uniqueID must make a unique identifier.
- Your plugin is identified by the prototype.name, not the object name itself, to the end users
- Never modify outside the Modifiable line markers.
- You can hold multiple plugins in one script
- You can prototype plugins on plugins!
- You can load a plugin as an object (but you should define the plugin in your script to avoid race conditions).

Building the Host

- The host is the `PluginFactory`, called as such because it holds all the prototype objects and builds them.
- Plugins can either be built directly from the factory, such as for deployment into a fluid environment, or can be build into chains.
- These chains are called `SubFactories` and hold a start and end point, where plugins are inserted in-between.

Building the Host

```
var context = new AudioContext();
var chainStart = context.createGain();
var chainEnd = context.createGain();
chainEnd.connect(context.destination);
var Factory = new PluginFactory(context);
var chain = Factory.createSubFactory(chainStart, chainEnd
    );
```

Loading the plugin

We will now load the "HelloWorld" plugin.

- Start up the python server (scripts included for 2.x and 3.x) to get a localhost
- Go to your browser and navigate to `http://localhost:8000/sandbox/jsap-sandbox.html`
- Open up your browser command interface (CMD+OPT+C on OS X, CTRL+ALT+C on Windows).

The rest of these will operate inside the command-line for speed.

Loading the plugin

The next step is to create the resource to execute your scripts. This is a simple object which tells the PluginFactory where your plugin is, its name and how to check it is ready.

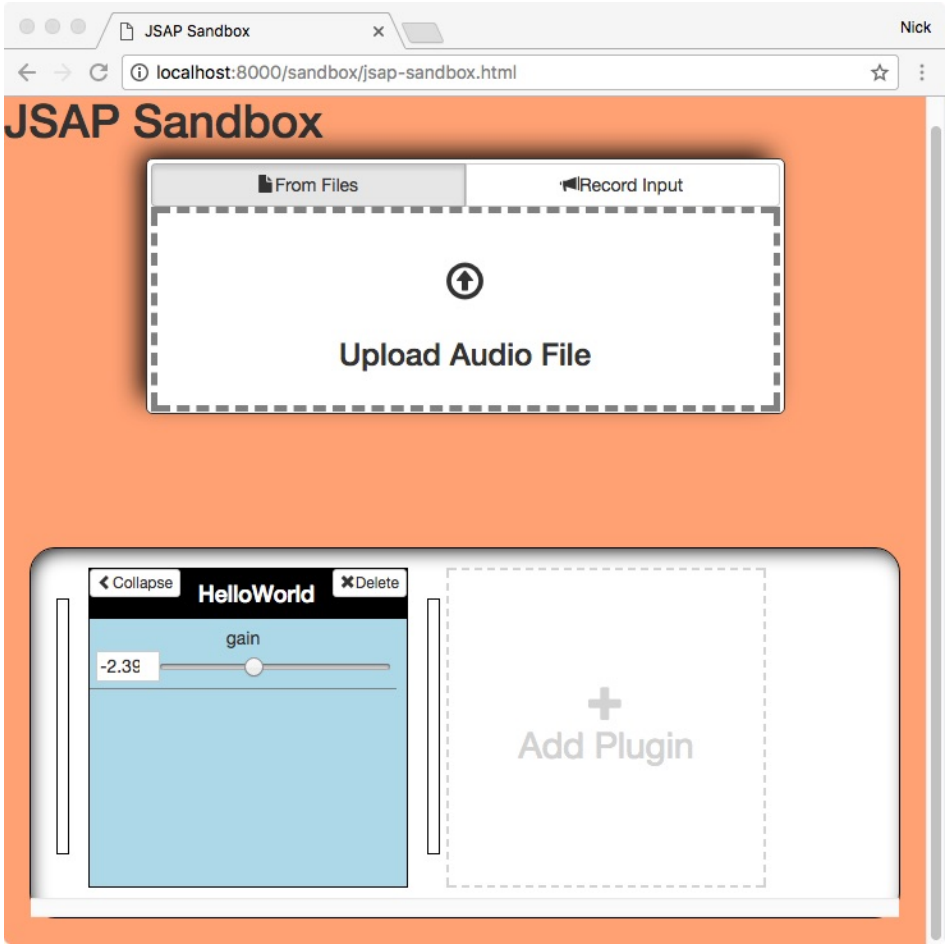
```
var loader = {
    "url": "/helloworld.js",
    "type": "JavaScript",
    "returnObject": "HelloWorld",
    "test": function() { return typeof HelloWorld === "
        function";}
}
```

Loading the plugin

The object can then be used by the PluginFactory to load the page

```
Factory.loadPluginScript(
    loader);
```

Your plugin should now be loaded, create it from the interface and see



Loading the plugin

1. Get the plugin prototypes
2. Select the prototype you want from that list by the name
3. Call on the chain to load this script into the end of the chain

```
var prototypes = Factory.
    getProtoypes()
var protoype = prototypes.find(
    function(a){return a.name == "
    HelloWorld";});
chain.createPlugin(prototype);
```

Mission accomplished!

You've now succesfully built and deployed a JSAP instance! Go and make more!
You can push any plugins that you make to a reposity of open-source effects at `https://github.com/nickjillings/jsap-plugins`.

Mission accomplished!

Further resources:

- Repositories
  - JSAP: `https://github.com/nickjillings/jsap` - Latest versions, issue/buglist
  - JSAP-Plugins: `https://github.com/nickjillings/jsap-plugins` - Latest versions, issue/buglist
- Documentation: `http://dmtlab.bcu.ac.uk/nickjillings/docs/index.php?src=jsap/index.md`
- News: `http://www.semanticaudio.co.uk/projects/jsap/`