EXPANDING THE POWER OF CSOUND WITH INTEGRATED HTML AND JAVASCRIPT

Michael Gogins
michael.gogins@gmail.com
https://michaelgogins.tumblr.com
http://michaelgogins.tumblr.com/

This paper presents recent developments integrating Csound [1] with HTML [2] and JavaScript [3, 4]. For those new to Csound, it is a "MUSIC N" style, user-programmable software sound synthesizer, one of the first yet still being extended, written mostly in the C language. No synthesizer is more powerful.

Csound can now run in an interactive Web page, using all the capabilities of current Web browsers: custom widgets, 2- and 3-dimensional animated and interactive graphics canvases, video, data storage, WebSockets, Web Audio, mathematics typesetting, etc. See the whole list at HTML5 TEST [5].

Above all, the JavaScript programming language can be used to control Csound, extend its capabilities, generate scores, and more. JavaScript is the "glue" that binds together the components and capabilities of HTML5. JavaScript is a full-featured, dynamically typed language that supports functional programming and prototype-based object-oriented programming. In most browsers, the JavaScript virtual machine includes a just-in-time compiler that runs about 4 times slower than compiled C, very fast for a dynamic language. JavaScript has limitations. It is single-threaded, and in standard browsers, is not permitted to access the local file system outside the browser's sandbox. But most musical applications can use an embedded browser, which bypasses the sandbox and accesses the local file system.

HTML Environments for Csound

There are two approaches to integrating Csound with HTML and JavaScript. *Csound runs the browser:* The Csound front end embeds a Web browser, and the Csound Structured Data (.csd) file contains HTML5 code in an `<html>` element of the .csd file. The front end uses its embedded browser to display this `<html>` element as a Web page. *The browser runs Csound:* A Web browser runs Csound, and the Csound orchestra and score are embedded in the HTML of the Web page. The currently available environments are:

- ***Csound for Android*** [6]. Csound runs an embedded Android WebView [7] browser.
- ***CsoundQt*** [8] (Windows). Csound runs an embedded Chromium Embedded Framework (CEF) [9] browser.
- ***csound.node*** for NW.js [10] (Windows, Linux). A CEF browser embedded in NW.js runs Csound, and with the csound_editor application, Csound runs the browser embedded in NW.js.
- ***Csound for PNaCl*** [11, 12, 13] (Google's Chrome browser on Windows, Linux, and OS X). The Chrome desktop browser compiles and runs Csound, using a runtime embedded in Chrome.
- ***Csound for Emscripten*** [11, 14, 15] (all JavaScript-enabled Web browsers). Any desktop browser runs Csound, which has been compiled to the efficient ASM.js [16] subset of JavaScript.

The technical details of these environments vary, but there are common themes. First, Web browsers enforce tight security, otherwise thieves and hackers will steal, spy, and take over computers. So browsers run code in a sandbox with limited permissions, *e.g.* no access to the local file system, no cross-domain scripting. Where there *is* a local file system, it's mounted inside the sandbox as a private file system. If the browser is embedded in a program running on a local computer, that browser is granted access to the local file system. So for complete control of Csound pieces enhanced with HTML5, the best choice is csound.node or CsoundQt, where Csound can write soundfiles, read samples, *etc.* on the local file system. A useful variation is Csound for PNaCl, created by Victor Lazzarini, which has methods for copying soundfiles and such into and out of the sandbox. Second, most browsers run JavaScript in a separate process that communicates with the browser's user interface via asynchronous messages. Third, in all environments, there is a csound proxy object in the JavaScript context of the Web page, so code running on that page can call core functions of the Csound API [17] to control Csound. Fourth, CsoundQt, csound.node, Csound for PNaCl, and Csound for Emscripten can all pop up Chrome's "Developer Tools" to debug the JavaScript used by any Csound piece.

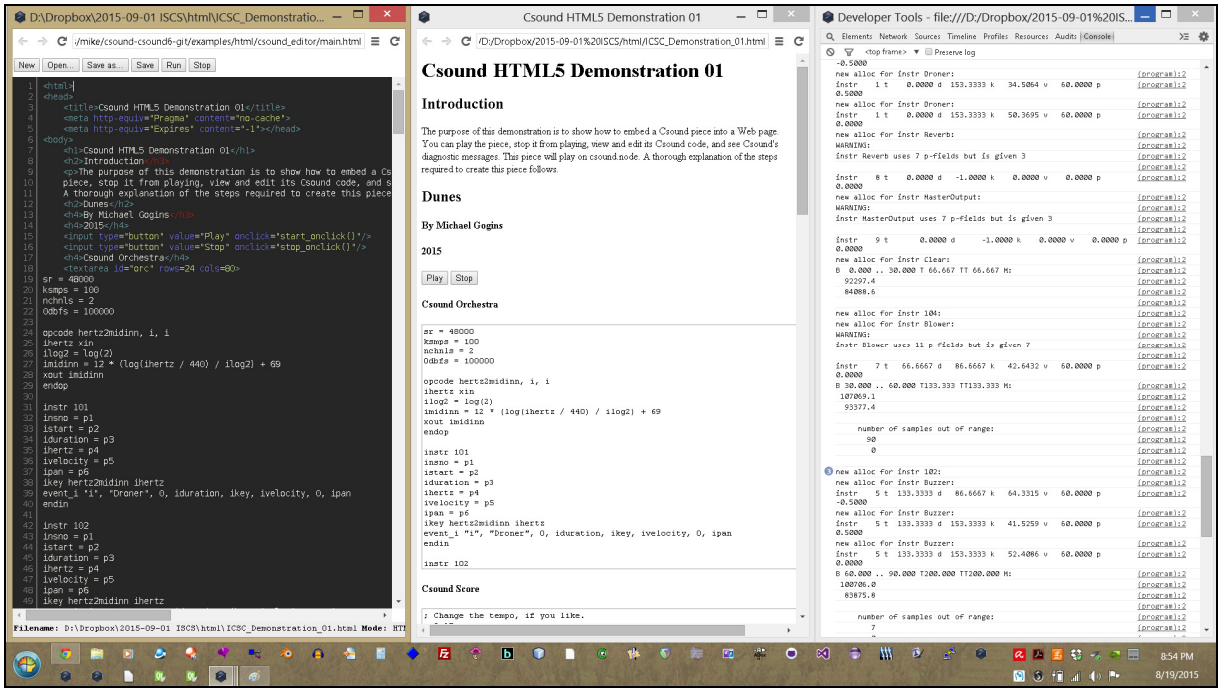
The Csound developers are working to ensure that a core subset of the Csound API has the same function names and argument types in all environments, so that a piece written, e.g., for Csound on Android can also run without editing on, *e.g.*, Csound for Emscripten. But we are not there yet.

Demonstrations

In the remainder of this paper, I demonstrate some capabilities of Csound enhanced with HTML using csound.node, Csound for Android, and CsoundQt. These demonstrations are available on my blog [18].

A Piece in a Web Page

This is a minimal piece that embeds a simple drone-style piece in a Web page that runs in csound.node. There is no user interface or score generation, the page just loads Csound and plays the piece.



In the screenshot above, the left third is csound_editor running in csound.node displaying the code for the page, the middle is the page itself, and the right is Developer Tools, showing Csound messages as Csound runs.runs. All the code for embedding Csound in csound.node and handling ***Play*** and ***Stop*** is just:

```
<input type="button" value="Play" onclick="start_onclick()"/>
<input type="button" value="Stop" onclick="stop_onclick()"/>
<script>
var csound = require('csound');
var start_onclick = function() {
    csound.setOption('-odac')
    var orc = document.getElementById('orc').value;
    csound.compileOrc(orc);
    var sco = document.getElementById('sco').value;
    csound.readScore(sco);
    csound.perform();
}
var stop_onclick = function() {
    csound.stop();
}
```

A Piece with Real-time Instrument Control

Using low-level functions to bind JavaScript event handlers to HTML elements is tedious, so almost all websites use some sort of GUI library to simplify coding. I could write these demonstrations without such a library to show the basic principles, but I feel it is more useful to show some common GUI libraries.

One confronts a bewildering the array of GUI libraries and application frameworks arising from the widespread use of JavaScript to create websites. JavaScript frameworks such as AngularJS or Ember are suited to this task, and may seem very attractive to the Csound user. However, they're designed for applications where a Web browser communicates via http with a Web server that manages data and logic for the

application. For these cases, a Model-View-Controller (MVC) pattern (more or less) is well suited.

But a Csound piece that uses HTML5 runs everything in the client, even if Csound itself and/or the piece are sourced from a Web server. Csound itself is the only "Model", and the Web page that communicates with Csound via JavaScript is both "View" and "Controller."
For this reason, I prefer lower-level, "View" oriented libraries for Csound. For visual music, fractal score generators, *etc.*, graphics are more important than user controls, and a minimalistic library (*e.g.* dat.gui [19]) is suitable. For highly interactive pieces, a library that simplifies event binding (e.g. jQuery [20]) is better.

Using Sliders to Control Csound

This demonstration adds a user-controllable slider to every control channel of the piece in the previous demonstration, as well as buttons for saving and restoring the values of all controls between sessions; such persistence is essential for actually composing or performing.

The HTML `<input>` element has a variety of types, and the range type creates a slider. An HTML table lays out the sliders, labels, and output fields. All Csound instruments are encapsulated in independent blocks of code, each of which can run without any external dependencies. All real-time controls are global variables, initialized just above their `instr` block. All function tables, too, are initialized above their `instr` block. A consistent naming pattern is used to avoid name clashes, *i.e.* `"gk_" + instrument_name + "_" + control_name`. Instruments do not send sound directly to the output, but rather through audio channels to effects, which in turn send the sound to the output. Then instruments can be arranged simply by cutting and pasting, or using `#include` statements in the orchestra. Here is the block for one instrument:

```
gi_Buzzer_overlap init 20
gk_Buzzer_harmonics init 2
gi_Buzzer_sine ftgen 0, 0, 65536, 10, 1
instr Buzzer
insno = p1
istart = p2
iduration = p3
ikey = p4
ivelocity = p5
iphase = p6
ipan = p7
iamp = ampdb(ivelocity) * 4
iattack = gi_Buzzer_overlap
idecay = gi_Buzzer_overlap
isustain = p3 - gi_Buzzer_overlap
p3 = iattack + isustain + idecay
kenvelope transeg 0.0, iattack / 2.0, 1.5, iamp / 2.0, iattack / 2.0, -1.5,
iamp, isustain, 0.0, iamp, idecay / 2.0, 1.5, iamp / 2.0, idecay / 2.0, -1.5,
0
ihertz = cpsmidinn(ikey)
asignal buzz kenvelope, ihertz, gk_Buzzer_harmonics, gi_Buzzer_sine
asignal = asignal * 3
aleft, aright pan2 asignal, ipan
adamping linseg 0, 0.03, 1, p3 - 0.1, 1, 0.07, 0
aleft = adamping * aleft
aright = adamping * aright
```

```
chnmix aleft, "out_left"
chnmix aright, "out_right"
prints "instr %4d t %9.4f d %9.4f k %9.4f v %9.4f p %9.4f\n", p1, p2, p3, p4,
p5, p7
endin
```

The "effects" instruments, as well as a global "Controls" instrument, are "always on" and they are scheduled in the orchestra code, not in the score. This makes it easier to generate scores programmatically.

```
scoreline_i "i \"Reverb\" 0 -1"
scoreline_i "i \"MasterOutput\" 0 -1"
scoreline_i "i \"Controls\" 0 -1"
scoreline_i "i \"Clear\" 0 -1"
```

The "Controls" instrument simply has a `chnget` opcode for each control channel's global variable. This is much more efficient than putting `chnget` into each instrument.

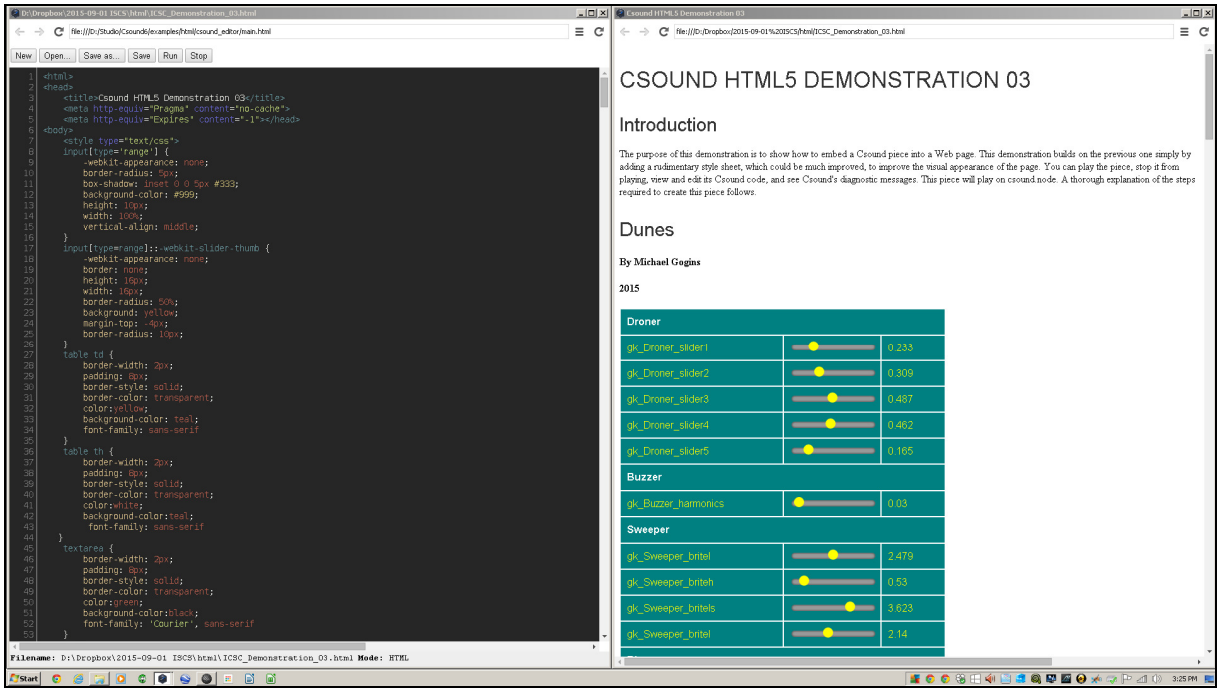
jQuery simplifies the code and gets slider values into Csound. "Event bubbling" sends events from all sliders to one event handler. The HTML IDs of all sliders are the same as the names of the corresponding Csound channels; a similar pattern links sliders to their output fields. In jQuery, `$(element)` loops over all instances of the `element` type, and `$('#id')` selects just the element named `id`. Then we use JavaScript's functional programming to create an event handler for each channel. Here is all the JavaScript for this piece:

```
<script>
var csound = require('csound');
</script>
<script
src="https://ajax.googleapis.com/ajax/libs/jquery/1.11.3/jquery.min.js">
</script>
<script>
$(document).ready(function() {
    $('#play').on('click', function() {
        csound.setOption('-odac')
        csound.compileOrc(document.getElementById('orc').value);
        csound.readScore(document.getElementById('sco').value);
        $('#restore').trigger('click');
        csound.perform();
    });
    $('#stop').on('click', function() {
        csound.stop();
    });
    $('input').on('input', function(event) {
        var slider_value = parseFloat(event.target.value);
        csound.setControlChannel(event.target.id, slider_value);
        var output_selector = '#' + event.target.id + '_output';
        $(output_selector).val(slider_value);
    });
    $('#save').on('click', function() {
        $('.persistent-element').each(function() {
            localStorage.setItem(this.id, this.value);
        });
    });
    $('#restore').on('click', function() {
        $('.persistent-element').each(function() {
            this.value = localStorage.getItem(this.id);
            csound.setControlChannel(this.id, this.value);
            var output_selector = '#' + this.id + '_output';
            $(output_selector).val(this.value);
        });
    });
    $(window).load(function() {
```

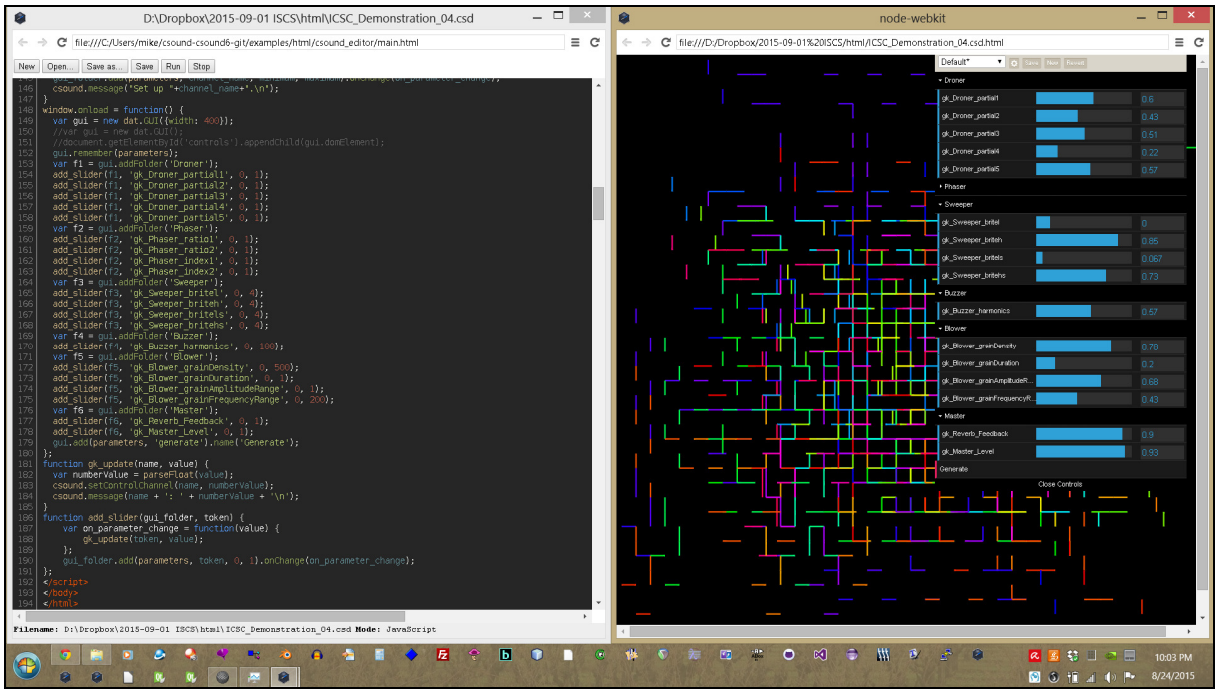
```
        $('#restore').trigger('click');
    });
});
</script>
```

A Piece with a Customized Appearance

Cascading Style Sheets (CSS) are a well-thought-out protocol for customizing the visual style of the elements on Web pages. This piece simply adds a CSS to the previous demonstration. Much more could be done to improve the page, *e.g.* using jQuery UI [21] to simplify creating and styling the elements. The screenshot shows the style sheet in the csound_editor on the left, and the effect of the styles on the piece on the right.



A Piece with a Score Generator



This piece runs in Csound for Android, or in csound_node's's csound_editor. dat.gui [19] creates the user controls. A Lindenmayer system from Silencio [22] generates notes, chords to which notes are fitted, and the graphic display. All code for creating controls is shown in the csound_editor. The `ChordSpace.LSys` class of Silencio generates notes into a `Silencio.Score`, which then sends them to Csound.

Acknowledgements

Thanks to Edward Costello, John ffitch, Victor Lazzarini, Brian Redfern, and Steven Yi for making the first integrations of HTML with Csound.

References

[1] Csound developers, Csound: A Sound and Music Computer System, https://csound.github.io/, retrieved August 12, 2015.
[2] World Wide Web Consortium, HTML 5.1: W3C Working Draft 09 July 2015, http://www.w3.org/TR/html51/, retrieved August 12, 2015.
[3] ECMA International, Standard ECMA-262, 5.1 Edition, ECMAScript Language Specification, http://www.ecma-international.org/ecma-262/5.1/Ecma-262.pdf, June 2011.
[4] Mozilla Developer Network, JavaScript, https://developer.mozilla.org/en-US/docs/Web/JavaScript, retrieved August 12, 2015.
[5] HTML 5 TEST: how well does your browser support HTML5, https://html5test.com/, retrieved August 12, 2015.
[6] Steven Yi and Victor Lazzarini, "Csound for Android," in Frank Neumann (ed.), Proceedings of the Linux Audio Conference 2012, Center for Computer Research in Music and Acoustics (CCRMA), Stanford University, California, April 2012, pp. 29-34.
[7] Android Developers, WebView, http://developer.android.com/reference/android/webkit/WebView.html, retrieved August 13, 2015.
[8] Andres Cabrera, CsoundQt, http://qutecsound.sourceforge.net/, retrieved August 13, 2015.
[9] Marshall Greenblatt et al., CEF: chromiumembedded, https://bitbucket.org/chromiumembedded/cef, retrieved August 13, 2015.
[10] Roger Wang et al., NW.js, http://nwjs.io/, retrieved August 13, 2015.
[11] Victor Lazzarini, Edward Costello, Steven Yi and John ffitch, "Extending Csound to the Web," IRCAM and Mozilla, 1st Audio Conference, Paris, January 26-28, http://wac.ircam.fr/pdf/wac15_submission_14.pdf, retrieved August 13, 2015.
[12] Victor Lazzarini, Csound for Portable Native Client, http://vlazzarini.github.io/, retrieved August 13, 2015. Please note: This link will load and play Csound pieces in Google's Chrome browser on OS X, Windows, and Linux.
[13] The Chromium Projects, Introduction to Portable Native Client, https://www.chromium.org/nativeclient/pnacl/introduction-to-portable-native-client, retrieved August 13, 2015.
[14] Edward Costello, Csound Emscripten, http://eddyc.github.io/CsoundEmscripten/#, retrieved August 13, 2015. Please note: This link will load and play Csound pieces in Web browsers that run JavaScript.

[15] Alon Zakai et al., Emscripten: An LLVM-to-JavaScript Compiler, http://kripken.github.io/emscripten-site/docs/introducing_emscripten/about_emscripten.html, retrieved August 13, 2015.
[16] asm.js, http://asmjs.org/, retrieved August 13, 2015.
[17] Csound developers, Csound API Documentation, http://csound.github.io/docs/api/index.html, retrieved August 13, 2015.
[18] Michael Gogins, Examples of Csound with HTML5 for the ICSC, http://michaelgogins.tumblr.com/post/127552774633/examples-of-csound-with-html5-for-the-icsc.
[19] Google Data Arts Team, dat.gui, https://github.com/dataarts/dat.gui, retrieved August 19, 2015.
[20] John Resig, et al., jquery, http://jquery.com/, retrieved August 19, 2015.
[21] Paul Bakaus, et al., jquery-ui, https://github.com/jquery/jquery-ui, retrieved August 23, 2015.
[22] Michael Gogins, silencio, https://github.com/gogins/silencio, retrieved August 24, 2015.