Extending Csound to the Web

Victor Lazzarini, Edward Costello, Steven Yi, John ffitch
Department of Music
Maynooth University
Ireland
victor.lazzarini@nuim.ie, edward.costello@nuim.ie, stevenyi@gmail.com,
jpff@codemist.co.uk

ABSTRACT

This paper discusses the presence of the sound and music computing system Csound in the modern world-wide web browser platform. It introduces the two versions of the system currently available, as pure Javascript code, and as portable Native Client binary module with a Javascript interface. Three example applications are presented, showing some of the potential uses of the system. The paper concludes with a discussion of the wider Csound application ecosystem, and the prospects for its future development.

Keywords

Music Programming Languages; Web Applications;

1. INTRODUCTION

In a recent paper[6], we have introduced two ports of the Csound sound and music computing system to the web-browser platform. In the first one, the Csound codebase was adapted and compiled to a subset of the Javascript language, asm.js, via the Emscripten compiler [11]. The second port employed the Portable Native Client (PNaCl)[3] technology to provide a platform for the implementation a Csound API-based Javascript frontend.

While the former is available for a wider range of internet browsers, as it is based on pure Javascript, the second project takes advantage of the near-native performance of PNaCl to provide a very efficient implementation of the system. Other significant differences between the two offerings are notable: the existence of pthread support in PNaCl versus the single-thread nature of pure Javascript; the dependence on Web Audio ScriptProcessorNode and audio IO in the Emscripten-based Csound versus the Pepper API-based audio and threading offered by PNaCl; and finally, the fact that the pure-Javascript implementation functions as a wrapper to the Csound API, whereas the PNaCl version provides a higher-level Javascript frontend to the system, with no direct access to the API.

Csound on the web browser is, therefore, an attractive option for audio programming targeting applications that run on clients (as opposed to server-side solutions). It offers an alternative to Adobe Flash (used, for instance in AudioTool[1], Patchwork[2], and Noteflight[3]), as well as standard HTML (used by the BBC Radiophonic workshop recreations[4], Gibberish[5], and WebPd[6]). It also fits with the development of an ecosystem of applications based on Csound, which allows users to easily move from one platform to another: desktop, mobile[8][10][7], small and custom computers[1], servers[4][5] (see also `http://www.researchcatalogue.net/view/55360/5536` for another application example), and now web clients. This paper is organised as follows: we will start with a brief overview of the two implementations of Csound for web browsers; this is followed by a discussion of some key example applications; we then explore the concept of the Csound application ecosystem and its significance; the final section shows the directions we intend to take the current ideas, and how they fit in the overall development of the system.

2. BROWSER-BASED CSOUND: OVERVIEW

The two implementations of Csound for web browsers use distinct technologies. The first is a Javascript-only port, created with the Emscripten compiler; the second is a C/C++-based application, which uses the PNaCl toolchain, and its runtime module, which currently exists only on Chrome and Chromium browsers (under most operating systems, however iOS and Android do not yet support it).

The two implementations also have another fundamental distinction. They operate at different levels, with regards to their use of the Csound library:

- **Emscripten Csound** works at the Csound API level, as it is effectively a translation of the original C library to the JavaScript language. This is equivalent, for instance, to the existing provision for the iOS and Android platforms, and the new OSX application SDK.

[1] `http://www.audiotool.com/`
[2] `http://www.patchwork-synth.com`
[3] `http://www.noteflight.com`
[4] `http://webaudio.prototyping.bbc.co.uk/`
[5] Available at `https://github.com/charlieroberts/Gibberish`, discussed in [9]
[6] `https://github.com/sebpiq/WebPd`

- **PNaCl Csound** works more as a frontend, providing a higher-level interface to the Csound engine. This is close to what is provided by the Csound objects in graphic programming environments (such as PD and MaxMSP), and the generator for LADSPA plugins.

2.1 Javascript Csound

Csound can now be run natively within any major web browser as a Javascript library using Emscripten. Emscripten can translate from LLVM bitcode into Javascript enabling programs written in a language supported by the LLVM compiler, such as C, to be compiled into Javascript and executed on a web page. Emscripten translates the LLVM bitcode into a strict subset of Javascript called asm.js[7]. By restricting some features of the language, Javascript engines can perform optimisations not possible using standard Javascript, which can result in improved performance.

As it is written entirely in C and has only one required external dependency, Csound makes an ideal codebase for adding Javascript as a build target using Emscripten. The only external library required to build Csound is libsndfile. This library is used by some of Csound's built-in opcodes and the core system for saving and opening various sound file formats. In order to build and run Csound successfully it is first necessary to compile libsndfile into a Javascript library. Emscripten comes with a number of python scripts which set the necessary environmental variables for the build configuration and compilation of software projects into Javascript. These scripts can be used to invoke the libsndfile *configure* script and *make* file which compile the libsndfile source code into an asm.js library. The resulting Javascript library can be linked to Csound during the build process.

Csound uses the CMake build system to manage the compilation of binaries for supported platforms. Fortunately, Emscripten provides support for using CMake and comes with a toolchain file which sets the required toolchain variables for project compilation using Emscripten's compiler.

There were some minor changes which had to be made to Csound's codebase in order to get it to compile successfully with Emscripten. Csound has the option of using threads for a number of operations during runtime, but as Emscripten does not support trans-compiling code bases which make use of threads, this functionality is now removed during the build configuration step. Additionally, many of the features available in the Desktop build of Csound are also disabled in the Javascript library which do not currently make sense within a web page context such as JACK support. The plugin opcodes such as Fluidsynth and STK are also unavailable at this time but may be included in future releases. These build configuration changes have been added to the main Csound codebase and are enabled when building Csound as a Javascript library.

The Csound library is controlled in Javascript using the provided C API. This allows external applications to compile instruments, send control signals and access Csound's audio input and output sample buffers. Emscripten provides wrapper functions which allow Javascript variables to be used as arguments to Emscripten compiled C functions, for instance, when using a Javascript string type as input to a C function taking a character array as an argument. This makes it possible to use Csound's C API functions directly within Javascript; however, an interface to a number of API functions has been created which greatly simplifies using API calls in a web page context. The interface consists of a Javascript class *CsoundObj*, which offers similar functionality to the *CsoundObj* classes found in the Android and iOS SDKs.

The following HTML creates a new instance of Csound, sends an orchestra string for compilation and plays the compiled instrument for one second.

```
<!DOCTYPE html>
<head>
<title></title>
<script src="javascripts/libcsound.js"></script>
<script src="javascripts/CsoundObj.js"></script>
</head>
<body>
<script>
var csound = new CsoundObj();
csound.compileOrc("ksmps=256\n" +
                  "nchnls=2\n" +
                  "0dbfs=1\n" +
                  "instr 1\n" +
                  "a1 vco2 0.2, 440\n" +
                  "outs a1, a1\n" +
                  "endin\n");
csound.startAudioCallback();
var scoreString = "i1 0 1"
csound.readScore(scoreString);
</script>
</body>
</html>
```

The *CsoundObj* class also contains methods for sending control messages using HTML and audio input to the running Csound instance via the Web Audio API. As Emscripten also provides a virtual file system that compiled C code can access, it is possible for Csound to write and play back audio files. A number of examples demonstrating the functionality provided by the Csound Javascript API can be found at `http://eddyc.github.io/CsoundEmscripten/`.

2.2 PNaCl Csound

Native Client is a recent technology developed by the Chromium project, which provides a sandboxing environment for applications running on browsers. It exists in two basic forms: one that works with natively-compiled modules (hardware-dependent, for i386, x86_64, arm, mips, etc); and another that is hardware independent, PNaCl. The former is currently only enabled for Chrome-store supplied applications, while the latter can be offered on the open web. The Csound port for Native Client has targeted the PNaCl platform, as it provides a flexible environment for the development of audio-based web applications.

The PNaCl project provides a toolchain so that C/C++ applications to be easily ported to it. Code is compiled to a bytecode representation (called a pexe module). This is then further translated ahead-of-time to the target hardware as

[7] `http://asmjs.org/spec/latest/`

the page containing it is loaded. Web pages containing a PNaCl module need to be served over http, so for testing and debugging, a minimal http server is required.

As part of the PNaCl platform, we have the Pepper API, which fulfills three main roles here: general-purpose communication between the browser and the PNaCl code; access to the sandbox for file IO; and audio IO. In addition to Pepper, a number of basic C libraries are present in PNaCl, such as pthreads, and the C stdio library. Ports of common Unix libraries are also available (libogg, libvorbis, libpng, libopenal, libjpeg, to cite but a few).

PNaCl Csound is composed of two main elements:

1. the pexe module (csound.pexe): based on the Csound library, provides means to run and control Csound, as well as access to files in the sandbox
2. a Javascript interface (csound.js): the PNaCl Csound functionality is exposed via a simple Javascript module, which allows applications to interface with Csound programmatically, in similar a way to the other language frontends like csound6~for PD, and csound~for MaxMSP.

Each pexe module (one per page) runs one single Csound engine instance. For multiple instances, we would require separate web pages for each. A simple PNaCl Csound application to play a sine beep for 5 seconds looks like this:

```
<!DOCTYPE html>
<html>
<head>
 <title>Beep!</title>
 <script type="text/javascript" src="csound.js">
 </script>
 <script type="text/javascript">
 // this function is called by csound.js
 // after the PNaCl module is loaded
 function moduleDidLoad() {
  csound.Play();
  csound.CompileOrc(
  "schedule 1,0,5\n" +
  "instr 1 \n" +
  "a1 oscili 0.1, 440\n" +
  "outs a1,a1 \n" +
  "endin");
 }
 </script>
</head>
<body
   <!--module messages-->
  <div id="console"></div>
  <!--pNaCl csound module-->
  <div id="engine"></div>
</body>
</html>
```

There is, of course, full scope for the development of interactive controls via HTML5 tags, and to integrate other Javascript packages. A set of introductory examples and the module programming reference is found at `http://vlazzarini.github.io`

2.3 Performance

Some basic performance measurements have been made with both Emscripten Csound and PNaCl Csound. One of the earliest complete pieces written for the Csound language is *Trapped in Convert* by Richard Boulanger, from 1977 (originally written for MUSIC 11, total duration 4'50"). Since this has been used for a long time as benchmark for Csound performance, we present here some CPU measurements taken on an up-to-date Macbook running OSX (2.8GHz Intel Core i7, quad core). The performance figures are shown in table 1, with both PNaCl and Emscripten Csound running on the Chrome browser.

Table 1: CPU times for *Trapped in Convert* (4'50")

version	Native	Emscripten	PNaCl
CPU time	1.109	4.388	1.771

These figures show that PNaCl Csound performance is close to native speeds, while the pure Javascript is about four times slower. While the PNaCl results are very impressive, the Emscripten code also performs very well, indicating that the main performance issues that still remain have to do with the inadequacy of the ScriptProcessorNode to support uninterrupted processing (see [6] for a more detailed discussion of these). Such figures demonstrate that Javascript Csound will be perfectly acceptable once these issues are properly resolved.

3. SOME EXAMPLE APPLICATIONS

The following discusses a few example client-side web applications using Csound built with Emscripten or PNaCl.

3.1 Csound Notebook

The Csound Notebook[8] is an online organizer for Csound projects. Users can create Notebooks filled with Csound notes, with each note being equivalent to a Csound ORC/SCO project. The interface for note editing is designed for live coding, such that the user incrementally edits and evaluates Csound ORC and SCO code using a running Csound engine. The project is written using Ruby on Rails for the server-side, and Angular.js and PNaCl Csound for the client-side.

This project demonstrates a couple of use cases where Csound in the browser can be applied. The first use case is that a Csound user who works on the desktop or other platform wants to sketch and experiment with ideas while on the go. They can organize and experiment with their projects online and later retrieve their code to use on their desktop system. Another use case is where a Csound user wants to work with Csound but is on a computer where Csound is not installed (or can not be installed, such as at a school computer lab). With the Csound Notebook web application, users do not require any plugins or applications to be installed to the user's system and can work entirely

[8] `http://csound-notebook.kunstmusik.com`, source code available at `https://github.com/kunstmusik/csound-notebook`

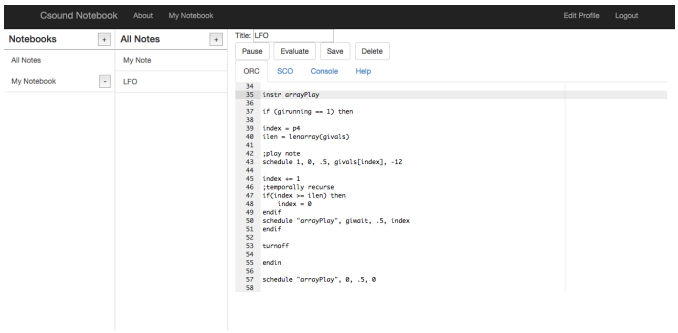

Figure 1: Csound Notebook

within a browser. While these use cases cater towards users who already know Csound and want to extend their use of the technology to the web, one can imagine that such a web application may also serve as a way for users who do not know Csound to try using it without having to pre-install any applications first.

3.2 Manual integration

As the number of opcodes within the Csound language is quite large, the Csound manual is a valuable resource for information about which opcodes are available to the language. Manual entries also provide examples of how to use opcodes within an orchestra file. Although it is available in other formats, the manual is distributed as a set of linked HTML documents. This allows the Csound Javascript library to be embedded within a manual page providing a mechanism to compile and run opcode examples directly from the manual.

In the prototype implementation shown in (fig. 2), the manual entry for the *vco2* opcode was used. Instead of static text providing an example of the opcode usage within a *csd* file, two editable text fields are provided which contain example instrument and score text. The text within each editable text field can be compiled and sent to a running instance of Csound using the provided *Send Score* and *Send Instrument* buttons. There is also an on-screen piano keyboard available, which can send score to the compiled Csound instrument along with frequency values represented by the text macro *<KEY>* within the score string.

3.3 Livecoder example

A final example of how this technology can be employed is shown in a livecoder interactive page, which is currently featured as a *Try it online!* item in the Csound community GitHub page[9] (fig. 3). This page includes, as one of its main components, a HTML5 `<textarea>` element, which can be edited to hold Csound orchestra code. The code is passed to the `csound.CompileOrc()` function, which compiles it on-the-fly. In complement to this, the page also allows users to upload files to be used by the engine, and to enable audio capture for realtime processing.

This example also highlights the educational aspects of the technology, which allow the design of online, distance/blended learning initiatives for computer music and programming. This is being incorporated in new courses such as the DSP Ear-training programme[2], developed at NTNU Trondheim, in Norway.

[9] `http://csound.github.io`

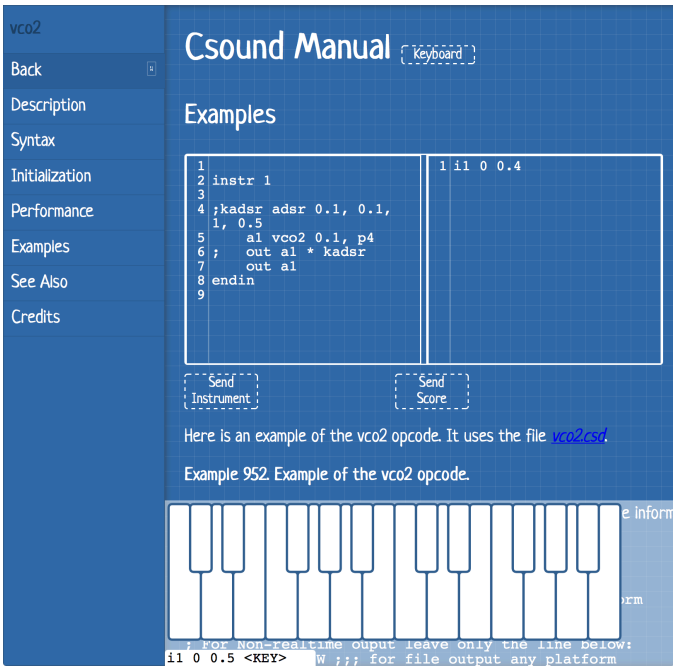

Figure 2: Csound manual integration

4. THE CSOUND APPLICATION ECOSYSTEM

The presence of Csound on the web, be it as a client or as a server application, is a part of a wider application ecosystem, which is also integrated by software running on desktop, mobile, small and embedded systems, and servers. The development of this ecosystem has been founded on the presence of an API, which has been a key feature of the Csound system since version 5, launched in 2006 (although earlier releases had already shipped with an incipient API).

Users developing multimedia applications and musical works benefit in a number of ways by using Csound. Learning one music system that can be applied to multiple musical problem spaces increases the value of that knowledge. For example, because Csound renders ORC and SCO code the same on each platform, users need only modify their projects for the platform-specific parts, such as their graphical user interfaces. This allows the user to leverage their existing framework for musical computing and focus on the unique features of each platform.

From the perspective of the existing Csound user, the web offers numerous features, such as easy deployment of applications, as well as long-term preservations of works. For example, if a Csound user creates a web-based application, they are able to share it with non-Csound users without the end user having to install Csound or other dependencies. The only requirement is that they have a browser that

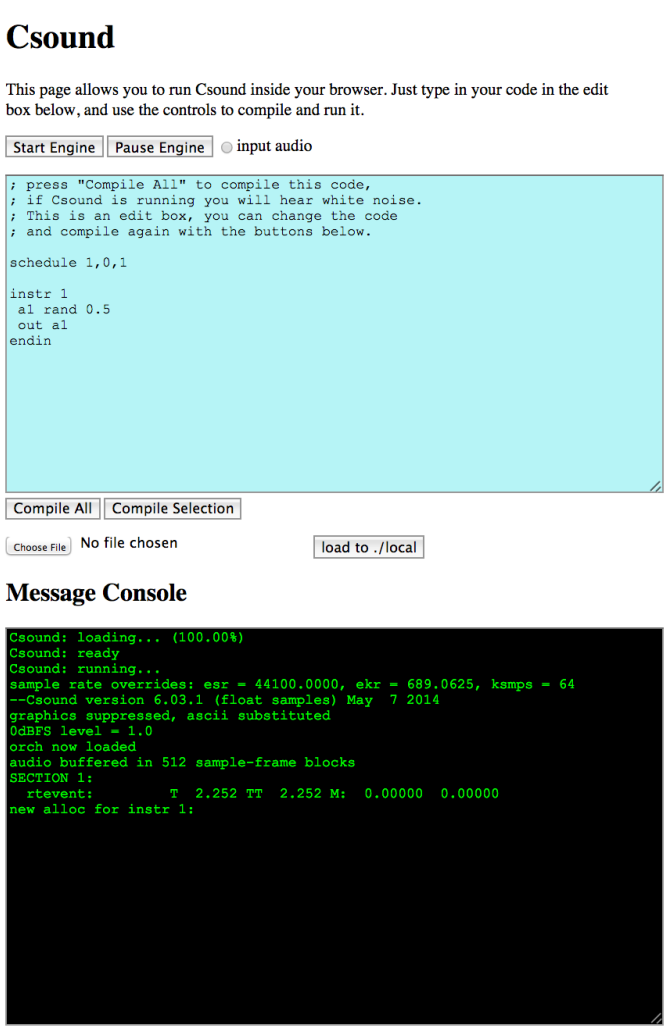

Figure 3: Csound's *Try it online!*

supports Javascript and optionally PNaCl. Having easy to reproduce projects greatly simplifies the dissemination of a work. Also, for a Csound-only project, the project can be preserved indefinitely by creating a web version of the piece. Not only is the entire project preserved, but also the specific version of Csound.

Finally, for non-Csound users looking to develop music applications for the web, using Csound offers numerous benefits. By developing a web-based music project with Csound code, users have options to create desktop, mobile, and embedded applications reusing their Csound code. Csound also offers a rich library of unit generators, giving a large foundation on which to build upon. Lastly, having a long history, users learning Csound have a wealth of examples to draw upon for inspiration for their own work.

5. FUTURE PROSPECTS

Csound on the web is an important platform for the Csound community. The current Emscripten and PNaCl builds are done using the same source code as is used for the desktop and mobile releases. Csound development currently takes into account all platforms and plans are to continue to support each system equally. As a result, improvements made in the main codebase are automatically shared with all platform builds, and the entire ecosystem progresses together.

For platform-specific code, the CsoundObj API is re-written for each platform in the native language of the platform. This API is offered to help facilitate easier cross-platform development. Future plans are to create a full CsoundObj implementation for the web that will match closely in features to the Android and iOS versions. It is also planned to explore making CsoundObj delegate to either PNaCl or Emscripten builds of Csound, depending on what is available in the user's browser. Having a unified CsoundObj API would then allow users to depend on a single API to develop against that would work across browsers.

6. CONCLUSIONS

The Csound computer music platform has been available for composition, research, and musical application development on the desktop, mobile, and embedded platforms. In this paper, we have shown two implementations of Csound for the web, one using Emscripten and another using PNaCl, that extends the existing Csound ecosystem into the browser. This research explores not only the possibilities of web-based music applications, but also the benefits of extending existing systems to the web.

7. ACKNOWLEDGMENTS

This research was partly funded by the Program of Research in Third Level Institutions (PRTLI 5) of the Higher Education Authority (HEA) of Ireland, through the Digital Arts and Humanities programme.

8. REFERENCES

[1] P. Batchelor and T. Wignall. BeaglePi: An Introductory Guide to Csound on the BeagleBone and the Raspberry Pi, as well other Linux-powered tinyware. *Csound Journal*, (18), 2013.

[2] O. Brandtsegg, S. Saue, J. P. Inderberg, A. Tidemann, V. Lazzarini, J. Tro, H. Kvidal, J. Rudi, and N. J. W. Thelle. The Development of an online course in DSP eartraining. In *Proceedings of DAFx 2012*, 2012.

[3] A. Donovan, R. Muth, B. Chen, and D. Sehr. PNaCl: Portable Native Client Executables. *Google White Paper*, 2010.

[4] J. ffitch, J. Mitchell, and J. Padget. Composition with sound web services and workflows. In S. O. Ltd, editor, *Proceedings of the 2007 International Computer Music Conference*, volume I, pages 419–422. ICMA and Re:New, August 2007. ISBN 0-9713192-5-1.

[5] T. Johannes and K. Toshihiro. „Và, pensiero!“ - Fly, thought! Experiment for interactive internet based piece using Csound6 . `http://tarmo.uuu.ee/varia/failid/cs/pensiero-files/pensiero-presentation.pdf`, 2013. Accessed: February 2nd, 2014.

[6] V. Lazzarini, E. Costello, S. Yi, and J. ffitch. Csound on the Web. In *Linux Audio Conference*, pages 77–84, Karlsruhe, Germany, May 2014.

[7] V. Lazzarini, S. Yi, and J. Timoney. Digital Audio Effects on Mobile Platforms. In *Proceedings of DAFx 2012*, 2012.

[8] V. Lazzarini, S. Yi, J. Timoney, D. Keller, and M. Pimenta. The Mobile Csound Platform. In *Proceedings of ICMC 2012*, 2012.

[9] C. Roberts, G. Wakefield, and M. Wright. The Web Browser As Synthesizer And Interface. *Proceedings of the International Conference on New Interfaces for Musical Expression*, 2013.

[10] S. Yi and V. Lazzarini. Csound for Android. In *Linux Audio Conference*, volume 6, 2012.

[11] A. Zakai. Emscripten: an LLVM-to-Javascript Compiler. In *Proceedings of the ACM international conference companion on Object oriented programming systems languages and applications*, pages 301–312. ACM, 2011.