SOUNDGEN

A Web Services Based Sound Generation System for the Psychoacoustics Laboratory

by
Michael R. Naber
S.B. 2007

Submitted to the Department of Electrical Engineering and Computer Science
in Partial Fulfillment of the Requirements for the Degree of
Master of Engineering in Electrical Engineering and Computer Science
at the Massachusetts Institute of Technology
May 2008

Author_____
Department of Electrical Engineering and Computer Science
May 1, 2008

Certified by_____
Louis D. Braida
Henry Ellis Warren Professor of Electrical Engineering

Accepted by_____
Arthur C. Smith
Professor of Electrical Engineering
Chairman, Department Committee on Graduate Theses

SOUNDGEN

A Web Services Based Sound Generation System for the Psychoacoustics Laboratory

by
Michael R. Naber

Submitted to the
Department of Electrical Engineering and Computer Science

May 1, 2008

In Partial Fulfillment of the Requirements for the Degree of
Master of Engineering in Electrical Engineering and Computer Science
at the Massachusetts Institute of Technology

1. ABSTRACT

Soundgen is a web services based sound generation system developed for the MIT Psychoacoustics Laboratory Course 6.182. The sounds created by Soundgen are combinations of various tones and noises, produced by a dedicated server running Linux, MATLAB, Apache, and PHP. As an example, Soundgen can generate a sound containing two tones of 500ms duration, each with its own frequency and phase, and can produce them over a broadband background noise. The characteristics of the tones and noises are passed to the Soundgen web service via a JSON object sent over HTTP. When the generation is complete, the web service replies with another JSON object containing the URL of the generated sound .wav file, along with some related information.

Accompanying the Soundgen web service is a small JavaScript library, easing the web service's use in JavaScript. This library allows JavaScript programmers to simply call a `soundgen()` function, which triggers a callback function that executes when the request has been processed by Soundgen. The library and web service allow Psychoacoustics Laboratory students to quickly and easily create portable acoustics experiments as web-applications, which can be written and run on any computer with speakers and a modern web browser.

2. DESIGN

As technology regularly progresses, methods for accomplishing software objectives continue to evolve. With the advent of the Internet and the rise of distributed application architectures, software components are no longer confined to single machines. Programmers regularly divide application components across many different systems. This division allows individual components to be separately maintained and updated, and also allows for simultaneous independent application expansion by multiple isolated teams or individuals.

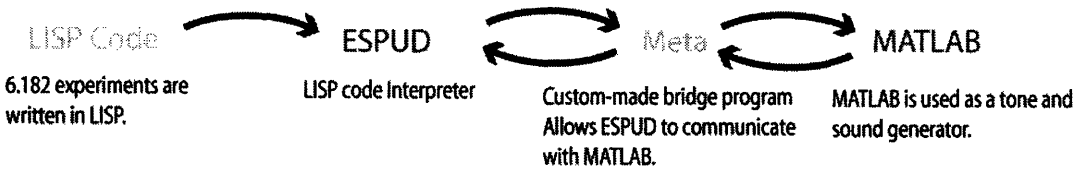
With the web services application model, individual application components can be located on different machines, but can work together by calling each other through HTTP requests over the Internet. The components can be written in any programming language that supports making HTTP requests such as Perl, Python, PHP, C, Java, HTML/JavaScript, etc. Not only can the software components be written in any programming language, they can also be deployed on completely different computing/operating system platforms. This freedom to write application components in such a variety of ways allows the programmer to work in his or her language of choice on his or her platform of choice. Furthermore, should future programming languages or development tools become available, they need only support the method of making HTTP requests to be integrated into an existing web services application.

Though it has been possible for years to design application components connected through the network using other methods, the advantage of using the web services architecture comes from the simplifications resulting from the use of standards for interoperability such as JavaScript Object Notation (JSON). Because these standards greatly assist pro-

grammers and speed up application development, it is important to emphasize not only the importance of the concepts behind distributed application architectures, but also the standards used for their implementation.

3. MOTIVATION

Students taking 6.182 write software auditory experiments, which are run on laboratory PCs in sound-proof rooms. Prior to the development of Soundgen, 6.182 experiments were written in LISP and interpreted by the LISP interpreter ESPUD, which was run locally on each of the PCs. Tones and other sounds presented during the experiments were generated by MATLAB, which also ran locally on the PCs. The ESPUD LISP interpreter communicated with MATLAB through a custom-made bridge program, Meta, written by Ray Cheng. This communication between ESPUD and MATLAB was necessary because the tones and other sounds generated by MATLAB needed to change as dictated by the application logic written in LISP and interpreted by ESPUD. The old system is well visualized by a diagram:



This LISP-based system, while adequate for writing and running experiments, had been quite problematic to maintain. Troubleshooting bugs had become a particularly tedious process. Common problems varied among the individual laboratory PCs, and included path variable inconsistencies between Windows user accounts, incompatibilities arising from different versions of MATLAB, and failures in Meta. In addition to these problems, there were

inefficiencies inherent to the system. For example, in order for students to test experimental code written in LISP, they had to restart both ESPUD and then Meta. Meta takes about fifteen seconds to start, and so when revising code, students had to wait fifteen seconds to test their changes. These limitations were unacceptable given the vast number of alternative ways for accomplishing the same software objectives that do not suffer from such difficulties. The problems and limitations were significant enough that there was compelling reason to abandon the old system and write a new one from scratch. Doing so has resulted in a much more standards-based development environment and does not suffer from the problems arising from using obscure application components.

There were many reasons for bringing web services to the 6.182 development process. Foremost, many difficulties were resolved when the 6.182 software experiments were switched to a web services architecture and the application logic migrated from LISP to HTML/JavaScript:

List of Difficulties Resolved:

1. The system for writing 6.182 experiments required students to know LISP.

   Because 6.182 experiments were written in LISP, students had to know LISP in order to modify or create experiments using the 6.182 tone generation API. While the choice to write the 6.182 experiments in LISP made sense at the time decided, with the retirement of 6.001, students are no longer taught LISP as part of the MIT curriculum. By switching the 6.182 experiments to a web services architecture, the application logic can be migrated from LISP to HTML/JavaScript. Since MIT will be

teaching Java and not LISP, and since JavaScript borrows most of its syntax from Java, students taking 6.182 will be well prepared to work on experiment code written in JavaScript.

2. The system for writing 6.182 experiments used many components, each of which had multiple points of failure.

   The 6.182 laboratory machines had many problems running the 6.182 software experiments due to system path variable inconsistencies between Windows user accounts, incompatibilities arising from different versions of MATLAB, and failures in Meta for unknown reasons. By switching the 6.182 experiments to a web services architecture and migrating the 6.182 software to HTML/JavaScript, these difficulties were eliminated.

3. The 6.182 development system required that students wait about fifteen seconds when testing code revisions

   In order for students to test their code, they had to restart Meta and ESPUD. Since META takes about fifteen seconds to start, students had to wait at least that long between testing successive iterations when revising code.

In addition to resolving problems with the old system, migrating the 6.182 experimental protocols to a web services architecture has brought about the following benefits to the 6.182 laboratory.

List of Benefits Gained:

1. New laboratory computers will require minimal configuration for 6.182.

   Since the 6.182 experiments are written in HTML/JavaScript, they can be run on any machine with a web browser and adequate sound card. This means that when the machines in the sound lab booths are upgraded or replaced, the 6.182 experiments will continue to run on the new hardware.

2. Students can debug and work on 6.182 projects on any computers they choose.

   With the 6.182 experiments rewritten in HTML/JavaScript, 6.182 students can debug and work on 6.182 projects on any computer with a web browser and adequate sound capabilities. (Of course, students would still need to be in the lab to use sound-proof booths when running experiments.)

3. Students can write 6.182 experiments in the programming language of their choice.

   Previous development protocols required students to write experiment code in LISP. Now that the 6.182 sound generation system has been made available as a web service, students can write this code in any programming language of their choice, so long as that programming language supported making HTTP requests over the Internet.

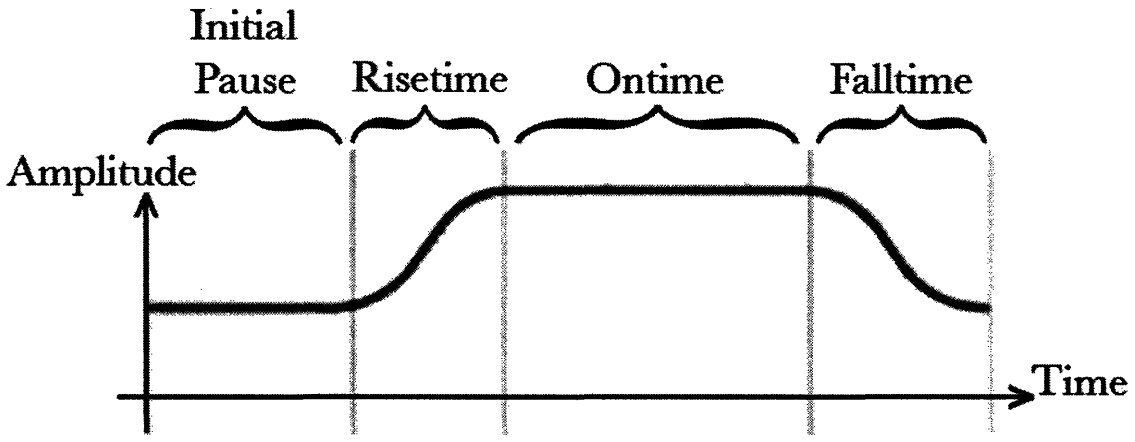
Switching the 6.182 experiments to a web services architecture has benefitted the 6.182 laboratory by solving the problems and yielding the benefits aforementioned. It has also directly benefitted students technical abilities by giving them experience working with a distributed application architecture. This experience is an important part of a software

engineer's education, as it is very likely that he or she will encounter such distributed architectures in future careers.

4. USING SOUNDGEN

Overview

The JSON object received by the web service is passed over HTTP GET, and is structured as an array of sound objects. Each sound object contains the attributes of a single sound, either a tone or a noise. The Soundgen web service computes each of these sound objects separately, sums them together, and then saves the .wav file containing their sum. The amplitude of each tone or noise in the sound is visualized by the following diagram:



This is a diagram of amplitude as a function of time for one tone or noise component of the sound. The sound component diagramed is silent during its initial pause and is at its

constant maximum amplitude during its ontime. During its risetime and falltime the amplitude of the sound component varies as sin(time)^2.

Soundgen JavaScript Library

Although the Soundgen web service can be used by any programming language capable of interacting with a JSON web service, all of the experiments currently written for the class use the JavaScript Soundgen library. This library can be included in standard HTML/JavaScript web pages and allows ordinary JavaScript on the page to use Soundgen to generate tones and noise.

The easiest way to write experiments using Soundgen is through the sound generation JavaScript library. Using the Soundgen JavaScript library, experiment interfaces can be written in HTML, and their program logic written in JavaScript. The library provides a simple interface consisting of a single function: `soundgen()`.

The `soundgen()` function accepts two arguments: an array of sound objects and a sample frequency rate. A sound object is either a tone object or a noise object, and contains all the relevant parameters of the tone or noise such as rise time, fall time, on time, frequency, etc. The sounds are defined by their constructors, `Tone()` and `Noise()` and the order of arguments for the constructors is:

Tone object:

- Initial Pause - duration of silence before the begin of the rise in ms
  between 0 and 60000 inclusive
- Risetime - duration of the amplitude rise in ms
  between 0 and 60000 inclusive

•Ontime - duration of of constant maximum amplitude in ms
between 0 and 60000 inclusive
•Falltime - duration of the amplitude fall in ms
between 0 and 60000 inclusive
•dB - decibel level of max. amplitude relative to the maximum output
between 0 and -300 inclusive
i.e. a dB level of 0 would produce a high maximum output and a dB level of -300 would be essentially undetectable
•Frequency - the frequency of the tone in Hz
between 10 and 20000 inclusive
•Phase - the phase of the tone sin wave
between $-\pi$ and π inclusive
•Channel - which channel (left, right or both) the tone is played through
either 'l' 'r' or 'b'

Noise object:

•Initial Pause - duration of silence before the begin of the rise in ms
between 0 and 60000 inclusive
•Risetime - duration of the amplitude rise in ms
between 0 and 60000 inclusive
•Ontime - duration of of constant maximum amplitude in ms
between 0 and 60000 inclusive
•Falltime - duration of the amplitude fall in ms
between 0 and 60000 inclusive
•dB - decibel level of max. amplitude relative to the maximum output
between 0 and -300 inclusive
i.e. a dB level of 0 would produce a high maximum output and a dB level of -300 would be essentially undetectable
It is important to note that the decibel level of a noise is also affected by the highpass and lowpass filters.
•Lowpass - frequency cutoff of lowpass filter in Hz
minus one means no lowpass filter, otherwise between 1 and 20000 inclusive
•Highpass - frequency cutoff of highpass filter in Hz
minus one means no highpass filter, otherwise between 1 and 20000 inclusive
•Loworder - order of the lowpass Butterworth filter

an integer between 1 and 100 inclusive

- Highorder - order of the highpass Butterworth filter
  an integer between 1 and 100 inclusive
- Seed - seed for the random number generator which creates the noise
  integer can be either minus 1 (for randomly selected seed) or between 0 and 2^32-1 inclusive
- Channel - which channel (left, right or both) the tone is played through
  either 'l' 'r' or 'b'

Note: For noises, the amplitude envelope is applied after the Butterworth filters.

An example use of the soundgen() function is given in the following HTML file:

test.html

```
<head>
   <script type="text/javascript"
       src="http://tuliptree.mit.edu/soundgen/soundgen2.js"></script>
   <script type="text/javascript">
      function begin() {
        var soundArray = new Array();
        soundArray[0] = new Noise(0,1000,1000,1000, -20, 10000,5000, 2, 2, -1, 'b');
        soundArray[1] = new Tone(0,1000,1000,1000, -20, 800, 0, 'b');
        soundgen(soundArray, 44100);
      }

      function soundgen_callback(obj) {
        /*
        After the RPC executes, it automatically calls this function.
        The RPC defines several variables within obj:
          obj.sound - contains a path to the sound
          obj.image - contains a path to the image
          obj.result - either "success" or a message indicating why the failure
                       occurred
          obj.statistics.version - the version of soundgen (2.0 in this case)
          obj.statistics.gentime - the time in ms the server spent processing the
                                   request

        We can use these variables in any javascript code.
        */

        //This bit of JavaScript plays a sound.
        document.getElementById("soundspan").innerHTML="<embed src='"+obj.sound+
          "'hidden=true autostart=true loop=false>";
      }

   </script>
```

```
</head>

<body>
   <span id="soundspan"></span>
   Click <a href="#" onclick="javascript:begin()">here</a> for sound!
</body>
```

This is a simple example of how to use the Soundgen JavaScript library. From the HTML/JavaScript above, it can be seen that the Soundgen JavaScript library resides on a server called tuliptree.mit.edu, and can be included in any HTML/JavaScript document by writing:

```
<script type="text/javascript" src="http://tuliptree.mit.edu/soundgen/soundgen2.js"></script>
```

With the Soundgen library included in the JavaScript document, the `soundgen()` function can be called. The sample frequency, which is passed to the `soundgen` function along with the sound array, is input in Hz and accepts valid values of 11025, 16000, 22050, 32000, 44100, and 88200.

Upon receipt of input, the server processes the request and replies with a JSON object (called obj in the example `soundgen_callback()` function) having the following structure:

- obj.sound - the HTTP URL where the .wav file can be accessed, or a failure message
- obj.image - the HTTP URL where the .jpg representing the waveform can be reached, or a failure message
- obj.result - either "success" or a message indicating why a failure has occurred
- obj.statistics.version - the version of Soundgen run on the server

•obj.statistics.gentime - the amount of server time spent processing the request

The Soundgen JavaScript library will automatically call the `soundgen_callback()` function when the server has processed the request. In the HTML/JavaScript code given on the previous page, contents of the JSON object returned by the server are available in the `obj` variable passed into the `soundgen_callback` function. In the example, JavaScript plays the sound returned by the server by embedding the URL of the .wav file into an empty HTML element. Most web browsers require Apple QuickTime or equivalent plugin to implement this playback.

Using Soundgen Without JavaScript

Soundgen can be called directly without the use of JavaScript by creating a JSON object and passing it to the RPC over HTTP GET. The RPC is located at:

http://tuliptree.mit.edu/soundgen/soundgenapi_rpc2.php

The structure of the JSON object passed to the RPC is an array of objects of type stdtone and stdnoise, exemplified by:

```
[
    {
        type:stdtone
        ip:
        riisetime:
        ontime:
        falltime:
        db:
        freq:
        phase:
```

```
        channel:
    },
    {
        type:stdtone
        ip:
        risetime:
        ontime:
        falltime:
        db:
        freq:
        phase:
        channel:
    }
]
```

The acceptable input values for each of these properties are the same for the RPC as for using the JavaScript library.

When this JSON object is passed to the Soundgen RPC, the RPC will reply with another JSON object structured as:

```
{
    sound:
    image:
    result:
    statistics.version:
    statistics.gentime:
}
```

The Soundgen RPC may be called by any language capable of processing JSON over HTTP. A current list of languages that support JSON is provided on http://www.json.org.

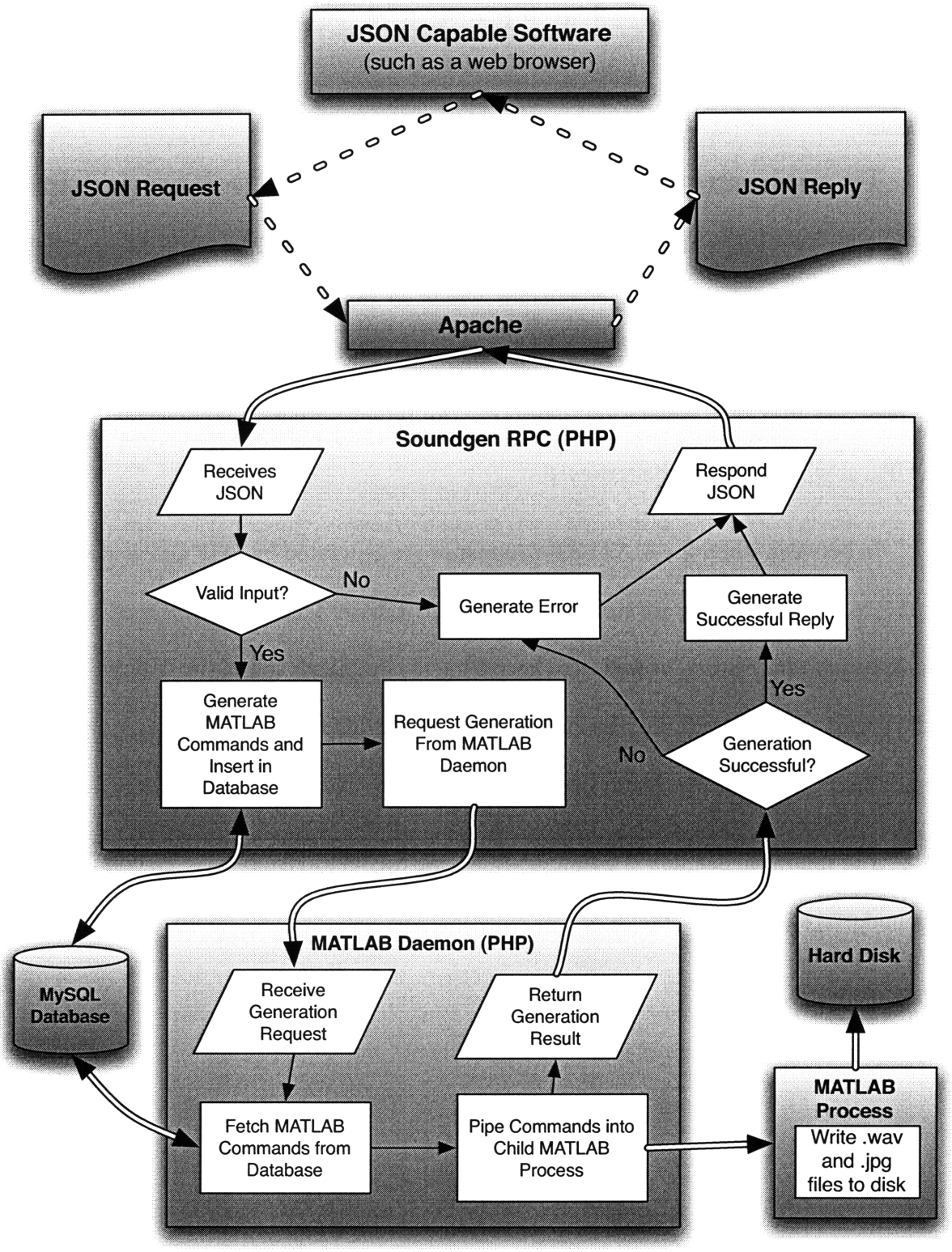
5. Technical Implementation

Overview

The sound generation web service is deployed on an Apple Mac Mini running Fedora Core Linux, MATLAB, Apache, and PHP. The PHP scripting language turns JSON requests into MATLAB sound generation commands, which are piped into a waiting MATLAB process held open by a PHP MATLAB daemon script. The MATLAB process generates the .wav files, and after their generation the PHP script replies with a JSON object containing the URL at which they can be retrieved.

In between when the MATLAB commands are generated by PHP and when they are piped into MATLAB, they are stored in a MySQL database. This database is used to both resolve concurrency issues, as well as to keep a log of commands so that in case the MATLAB process were to crash, the last commands to enter the process could be retrieved by a server administrator (using the mysqldump command) to aid in debugging.

The following diagram is useful in visualizing the relationship between components of Soundgen residing on the server:


JSON Capable Software
(such as a web browser)
JSON Request
JSON Reply
Apache
Soundgen RPC (PHP)
Receives JSON
Respond JSON
Valid Input?
No
Generate Error
Generate Successful Reply
Yes
Yes
Generate MATLAB Commands and Insert in Database
Request Generation From MATLAB Daemon
No
Generation Successful?
MySQL Database
MATLAB Daemon (PHP)
Hard Disk
Receive Generation Request
Return Generation Result
MATLAB Process
Fetch MATLAB Commands from Database
Pipe Commands into Child MATLAB Process
Write .wav and .jpg files to disk

The above diagram shows the interactions between software components used by Soundgen. There are several server-side software components in the diagram, and they are all run on tuliptree.mit.edu, the server which runs Soundgen:

Apache

Soundgen RPC PHP Script

MySQL

MATLAB Daemon PHP Script

MATLAB Process

Each of these components interacts with the others as shown in the diagram. The specific function of each component is given in the details below:

Apache

Apache is the conventional web server software included with Fedora Core Linux. All client-server communications pass through this software component. It is run in standard configuration with PHP. Because Apache accepts JSON requests over HTTP GET, the httpd.conf file was modified to include the configuration directive `LimitRequestLine 1000000`. This directive sets the maximum allowable size (in bytes) of a client's HTTP request-line. The default setting is 8192. Since JSON passed over HTTP GET encodes the entire JSON request within the request-line, the maximum allowable size is set to 1000000 so that a large JSON object can be passed to the server.

Soundgen RPC PHP Script

/var/www/html/soundgen/soundgenapi_rpc2.php

The Soundgen RPC PHP script processes client JSON requests. Its principle purpose is to interact with the client, both in processing JSON requests and returning JSON encoded replies. In doing this, it must first validate the client input by verifying the structure and parameters of the request. If the input is determined to be valid, the Soundgen RPC PHP script generates MATLAB commands that will generate the desired sound. If the input is determined to be invalid, the Soundgen RPC PHP script returns an error message to the client indicating what was wrong with the request.

After the MATLAB commands are generated, the Soundgen RPC PHP script inserts them into a MySQL database along with the IP address of the client. The database server returns a unique ID associated with the set of inserted commands. This ID is then passed to a MATLAB Daemon PHP script over a TCP/IP socket connection, which retrieves the MATLAB commands from the MySQL database and passes them into the MATLAB process which it perpetually holds open.

If the MATLAB Daemon PHP script replies with a success message, the Soundgen RPC PHP script generates a JSON object containing the URL of the sound and image as a JSON object. Otherwise, it returns an error message to the client.

MySQL

Soundgen uses the standard MySQL database server included with Fedora Core Linux. When a client uses Soundgen to request a sound, the MySQL database server is used

to store the MATLAB commands used to generate that sound, along with the IP address of the client which requested it, and the time at which it was requested. The database server has two tables: commands and jobs.

Table commands structure:

Field	Type	Collation	Attributes	Null	Default	Extra
id	bigint(20)		UNSIGNED	No		auto_increment
cmdid	bigint(20)			No		
seq	int(11)			No		
cmd	varchar(50000)	latin1_swedish_ci		No		

This table stores the commands input to MATLAB. The fields store:

id - a unique id for every stored command

cmdid - an id unique to every set of commands

seq - the sequence number for which a set of commands having the same cmdid should be input to MATLAB

cmd - the MATLAB command

Table jobs structure:

Field	Type	Collation	Attributes	Null	Default	Extra
id	bigint(20)			No		auto_increment
ipaddr	varchar(15)	latin1_swedish_ci		No		
time	timestamp			No	CURRENT_TIMESTAMP	

This table stores the requesting client IP address and the time at which that client made the request. The fields store:

id - a unique id for every row

ipaddr - the IP address of the requesting client

time - the date/time at which the client made the request

MATLAB Daemon PHP Script

/root/soundgen_daemon/socket.php

The MATLAB Daemon PHP script is started by the init script /etc/rc5.d/S90soundgen when the server first starts up and remains running at all times. Its purpose is to hold a running MATLAB process open and pipe commands into it. The MATLAB Daemon PHP script listens on TCP port 44445 for connections from the Soundgen RPC.

After the Soundgen RPC has inserted the MATLAB commands into the MySQL database, it passes the cmdid associated with those commands to the MATLAB Daemon PHP Script over TCP/IP. If multiple requests are received simultaneously they are buffered by the TCP/IP socket.

Upon receiving the cmdid from the Soundgen RPC, the MATLAB Daemon PHP script immediately queries the MySQL database server for the commands, and pipes them into the MATLAB process it holds open. Once the commands are processed by the MATLAB process, the sound and image files are written to disk and available over HTTP request.

Note: The server runs PHP version 5.2.5, compiled with the `--enable-socket` option. This option allows for scripts to listen as TCP/IP socket servers. The configuration option was necessary because the MATLAB Daemon PHP script listens over TCP/IP for connections from the Soundgen RPC.

MATLAB Process

The MATLAB Process is started by the MATLAB Daemon PHP Script and stays open continually. Its purpose is to receive the MATLAB commands piped to it by the MATLAB Daemon PHP Script and execute them in the MATLAB shell. The execution of the commands results in writing a sound and image file to a web accessible location of the server's hard disk.

--

The combined task of all these software components is to receive a request for a particular type of sound, to generate that sound, and to store its .wav file in a web accessible location on the server's hard disk. Once the sound is stored on disk, the client is sent a reply with the URL location at which it can be accessed.

The server side software components described are all located on a single server which is accessed remotely by client machines via JSON encoded requests over HTTP. The Soundgen JavaScript library, which eases the use of the Soundgen RPC by JavaScript clients, contains constructors for `Tone` and `Noise` objects, as well as a function `soundgen()` for calling the Soundgen RPC. The function which calls the Soundgen RPC utilizes moo.base.js, cnet.base.js, and cnet.global.utils.js in order to allow cross-domain JSON requests.

6. MATLAB LICENSE

Soundgen uses MATLAB on the server, and as such requires that the MATLAB license be kept up to date. The currently installed MATLAB license is located on tuliptree.mit.edu at /usr/local/matlab2007b/etc/license.dat. It is used by the MATLAB li-

cense manager located at /usr/local/matlab2007b/etc/lmboot, which is loaded at system boot time by the Soundgen startup script located in /etc/rc.d/rc5.d/S90soundgen.

The currently installed MATLAB license includes both the Signal Toolbox and the Statistics Toolbox and expires August 1, 2008.

7. APPENDIX

The Soundgen JavaScript library and the code run on the Tuliptree server is attached for reference.

```
/*soundgen2.js
Soundgen JavaScript Library
Michael Naber

Description:
This JavaScript file allows the Soundgen RPC to be easily used in JavaScript code. There are two
constructors for sound objects: Tone() and Noise().

The constructors are used to generate sound objects which may be inserted into an array. The array
may be inserted passed to the soundgen() function along with a sample frequency in order to
generate a sound. When the sound is generated, the soundgen_callback() function is called.
*/

//mootools base javascript (moo.base.js), cnet.base.js, cnet.global.utils.js
//Packed JavaScript not shown.

//Function for the standard tone object
function Tone(ipVal, risetimeVal, ontimeVal, falltimeVal, dbVal, freqVal, phaseVal, channelVal)
{
    this.type = 'stdtone';
    this.ip = ipVal; //Between 0 and 60000 inclusive
    this.risetime = risetimeVal; // Between 0 and 60000 inclusive
    this.ontime = ontimeVal; // Between 0 and 60000 inclusive
    this.falltime = falltimeVal; // Between 0 and 60000 inclusive
    this.db = dbVal; //Between 0 and -300 inclusive
    this.freq = freqVal; // Between 10 and 20000 inclusive
    this.phase = phaseVal; // between -pi and pi inclusive
    this.channel = channelVal // 'l' for left, 'r' for right, or 'b' for both
}

//Function for the standard noise object
function Noise(ipVal, risetimeVal, ontimeVal, falltimeVal, dbVal, lowpassVal, highpassVal,
    loworderVal, highorderVal, seedVal, channelVal) {
    this.type = 'stdnoise';
    this.ip = ipVal; //Between 0 and 60000 inclusive
    this.risetime = risetimeVal; //Between 0 and 60000 inclusive
    this.ontime = ontimeVal; //Between 0 and 60000 inclusive
    this.falltime = falltimeVal; //Between 0 and 60000 inclusive
    this.db = dbVal; //Between 0 and -300 inclusive
    this.lowpass = lowpassVal; // -1 means no lowpass, otherwise between 1 and 20000
    this.highpass = highpassVal; //-1 means no highpass, otherwise between 1 and 20000
    this.loworder = loworderVal; //order of the Butterworth filter
    this.highorder = highorderVal; //order of the Butterworth filter
    this.seed = seedVal // Integer either -1 or between 0 and 2^32 - 1 inclusive; Seed for the
    //MATLAB random number generator. The random seed can be -1 for a randomly selected seed;
    this.channel = channelVal; // 'l' for left, 'r' for right, or 'b' for both
}

function soundgen(soundArray,sampleFreq) {
    //sampleFreq can be either 11025, 22050, 16000, 32000, 44100, or 88200
    var jsonData = {
        "soundArray": Json.toString(soundArray),
        "sampleFreq": Json.toString(sampleFreq)
    }
    new JsonP('http://tuliptree.mit.edu/soundgen/soundgenapi_rpc2.php', {
      data: {
        "soundArray": Json.toString(soundArray),
        "sampleFreq": Json.toString(sampleFreq)
      },
      onComplete: function(ret){
          soundgen_callback(ret);
      }
    }).request();
}
```

```
<?php
/*
Soundgen RPC
Michael Naber

Description:
The Soundgen RPC is responsible for receiving requests for sounds, determining whether those
requests are valid, generating MATLAB syntax, and replying to the JSON request. The Soundgen RPC
makes use of three classes to accomplish this. They are:

1)  The InputValidator
    Determines whether the user submitted a valid JSON request
2)  The MatlabGenerator
    Generates MATLAB code based on received JSON
    Requests generation from the MATLAB daemon PHP script
3)  The JsonResponder
    Generates the JSON response
*/

session_start();
include('config.php'); //sets $absoute_uri and $tempdir

//These two lines will prevent the client from caching the RPC response
//The date is an arbitrary date in the past
header("Cache-Control: no-cache, must-revalidate");
header("Expires: Mon, 26 Jul 1997 05:00:00 GMT");

$_GET = array_map('stripslashes',$_GET); //Receive the JSON data over HTTP GET
$_GET = array_map('json_decode',$_GET);  //Decode the JSON data

//Create an $input object to store all the user input
$input->soundArray = $_GET['soundArray'];
$input->sampleFreq = $_GET['sampleFreq'];
$input->callback = $_GET['callback'];

function __autoload($class_name) {
    require_once 'classes/' . $class_name . '.php';
}

//These classes are defined in separate files
$myInputValidator = new inputValidator(); //Determines whether the user submitted a valid request
$myMatlabGenerator = new matlabGenerator(); //Generates MATLAB code based on received JSON
$myJsonResponder = new jsonResponder(); //Generates the JSON response

$statistics->version = "2.0";
$statistics->gentime = "error - see obj.result";

//If our input is invalid, then fail indicating what about it is invalid
if(!$myInputValidator->validateInput($input)) {
    $myJsonResponder->echoFailure($myInputValidator->error, $statistics, $input->callback);
//If the input is valid, then generate the sound
} else {
    $time_start = microtime(true); // time the generation
    $myMatlabGenerator->generateCommands($input);
    $result = $myMatlabGenerator->sendCommandsToMatlab();
    $time_end = microtime(true);

    if($result->picname && $result->soundname) {
        $picpath = $absolute_uri . "/temp/" . $result->picname;
        $soundpath = $absolute_uri . "/temp/" . $result->soundname;
        $statistics->gentime = $time_end - $time_start;
        $myJsonResponder->echoSuccess($soundpath, $picpath, $statistics, $input->callback);
    } else {
        $myJsonResponder->echoFailure("Valid input, but MATLAB generator failed!", $statistics,
```

```
$input->callback);
    }
}
?>
```

```
<?
/*
InputValidator Class - Used by Soundgen RPC
Michael Naber

Description:
The InputValidator class has one public function validateInput, which takes a sound generation
request and returns true if the input is valid and returns false if the input is invalid.

If the input is determined to be invalid, the InputValidator will generate an error message
indicating what about the request is invalid.

*/


class inputValidator {

    //Pass a $str to be concatenated to the error message.
    private function errorproc($str) {
        if(strlen($this->error) == 0) {
            $this->error = "There was an error processing your request. The following problems wer
            found: <br>";
        }
        $this->error = $this->error . "<br>" . $str;
    }

    //Helper function to validateInput
    private function validateVal($num, $min, $max, $name, $i) {
        if(!($num <= $max && $num >= $min)) {
            $this->errorproc('The ' . $name . " " . $i . ' is invalid. It must be between ' . $min
            . ' and ' . $max . ' inclusive.');
        }
    }

    //Function returns true if the input is valid, and false if the input is invalid. If the input
is invalid, the error variable is set with a description of the error.
    public function validateInput($input) {

        $this->error = "";

        //Check that the sample frequency is valid.
        if($input->sampleFreq != 11025 && $input->sampleFreq != 16000 && $input->sampleFreq !=
22050
        && $input->sampleFreq != 32000 && $input->sampleFreq != 44100
        && $input->sampleFreq != 88200) {
            $this->errorproc('The sample frequency value is invalid.');
        }

        if(is_array($input->soundArray)) {
            //Check the validity of the toneArray
            for($i = 0; $i < sizeof($input->soundArray); $i++) {

                $sound = $input->soundArray[$i];

                if($sound->type == 'stdtone') {
                    $this->validateVal($sound->ip, 0, 60000, 'initial pause of tone at index', $i)
                    $this->validateVal($sound->risetime, 0, 60000, 'risetime of tone at index',
$i);
                    $this->validateVal($sound->ontime, 0, 60000, 'ontime of tone at index', $i);
                    $this->validateVal($sound->falltime, 0, 60000, 'falltime of tone at index',
$i);
                    $this->validateVal($sound->db, -300, 0, 'dB of tone at index', $i);
                    $this->validateVal($sound->freq, 10, 20000, 'frequency of tone at index', $i);
                    $this->validateVal($sound->phase, -pi(), pi(), 'phase of tone at index', $i);
```

```
                    if($sound->channel != 'l' && $sound->channel != 'r' && $sound->channel != 'b')
{
                        $this->errorproc('The channel of tone ' . $i . ' is invalid. It must be
                        either "r", "l", or "b"');
                        $this->valid = false;
                    }
                }

                if($sound->type == 'stdnoise') {
                    $this->validateVal($sound->ip, 0, 60000, 'initial pause of noise at index',
$i);
                    $this->validateVal($sound->risetime, 0, 60000, 'risetime of noise at index',
$i);
                    $this->validateVal($sound->ontime, 0, 60000, 'ontime of noise at index', $i);
                    $this->validateVal($sound->falltime, 0, 60000, 'falltime of noise at index',
$i);
                    $this->validateVal($sound->db, -300, 0, 'dB of noise at index', $i);

                    if($sound->lowpass != -1 && ($sound->lowpass > 20000 || $sound->lowpass < 1))
                        $this->errorproc('The lowpass filter frequency value for noise ' . $i .
                        ' is invalid. It must be either -1 or between 1 and 20000 inclusive.');
                    }

                    if($sound->highpass !=-1 && ($sound->highpass > 20000 || $sound->highpass < 1)
{
                        $this->errorproc('The highpass filter frequency value for noise ' . $i .
                        ' is invalid. It must be either -1 or between 1 and 20000 inclusive.');
                    }

                    if($sound->loworder < 1 || $sound->loworder > 100 ||
                        $sound->loworder/ceil($sound->loworder) != 1.0) {
                        $this->errorproc('The order of the lowpass filter for noise ' . $i .
                        ' is invalid. It must be an integer between 1 and 100 inclusive.');
                    }

                    if($sound->highorder < 1 || $sound->highorder > 100 ||
                      $sound->highorder/ceil($sound->highorder) != 1.0) {
                        echo('high order ' . $sound->highorder);
                        $this->errorproc('The order of the highpass filter for noise ' . $i .
                            ' is invalid. It must be an integer between 1 and 100 inclusive.');
                    }

                    if(floor($sound->seed) != $sound->seed) {
                        $this->errorproc('The seed of noise ' . $i . ' must be an intenger');
                    } elseif(($sound->seed > 4294967295 || $sound->seed < 0) &&
                      $sound->seed != -1) {
                        $this->errorproc('The seed of noise ' . $i . ' must be either -1 or betwee
                          0 and 2^32 - 1 and 0 inclusive.');
                    }

                    if($sound->channel != 'l' && $sound->channel != 'r' && $sound->channel != 'b')
{
                        $this->errorproc('The channel of noise ' . $i . ' is invalid. It must be
                          either "r", "l", or "b"');
                    }
                }
            }
        } else {
            $this->errorproc('The soundArray is not a valid array.');
        }

        if($this->error == "") {
            return true;
```

```
        } else {
            return false;
        }

    }
}
?>
```

```
<?
/*
JsonResponder Class - Used by Soundgen RPC
Michael Naber

Description:
The JsonResponder class is used by the Soundgen RPC to encode JSON replies. After the Soundgen RPC
receives a generation request, this class is always used to either indicate successful generation
or reason for failure.

*/

class jsonResponder {
    //Returns a JSON response to the client
    function echoFailure($reason, $statistics, $callback) {
        $response->sound = "error - see obj.result";
        $response->image = "error - see obj.result";
        $response->result = $reason;
        $response->statistics = $statistics;
        if($callback != '') {
            echo($callback . "(" . json_encode($response) . ")");
        } else {
            echo(json_encode($response));
        }
    }

    function echoSuccess($sound, $image, $statistics, $callback) {
        $response->sound = $sound;
        $response->image = $image;
        $response->result = "success";
        $response->statistics = $statistics;
        if($callback != '') {
            echo($callback . "(" . json_encode($response) . ")");
        } else {
            echo(json_encode($response));
        }
    }
}

?>
```

```
<?
/*
MatlabGenerator Class - Used by Soundgen RPC
Michael Naber

Description:
If the InputValidator class determines that the sound request is valid, the Soundgen RPC uses the
MatlabGenerator class to generate MATLAB commands which, when input into the MATLAB Daemon, will
write the .wav sound file and .jpg waveform image to a http accessible directory on the server's
disk.
*/

class matlabGenerator {

    //This function sends the generated commands to the MATLAB Daemon PHP script.
    //First, it inserts the commands into the MySQL database, and next it sends the ID of those
commands to the MATLAB Daemon PHP script.
    function sendCommandsToMatlab() {
        $return->picname = false;
        $return->soundname = false;

        mysql_connect("localhost", "root", "soundlab") or die(mysql_error());
        mysql_select_db("mcmds") or die(mysql_error());
        $ip = $_SERVER['REMOTE_ADDR'];

        $q_insjob = "INSERT INTO `mcmds`.`jobs` (`id` ,`ipaddr` ,`time`) VALUES ('',
'$ip',CURRENT_TIMESTAMP)";
        $r_insjob = mysql_query($q_insjob);
        $cmdid = mysql_insert_id();

        //Insert the commands into the database
        foreach ($this->mcmds as $i => $value) {
            $value = addslashes($value);
            $q_inscmd = "INSERT INTO `mcmds`.`commands` (`id` ,`cmdid` ,`seq` ,`cmd`) VALUES ('',
'$cmdid', '$i', '$value')";
            $r_inscmd = mysql_query($q_inscmd);
        }

        $host = "127.0.0.1";
        $port = 44445;  //port chosen arbitrarily
        $fp = fsockopen($host, $port, $errno, $errstr);
        if($fp) {
            fputs ($fp, $cmdid); //send the id of the commands to the MATLAB PHP Daemon Script
            $result .= fgets ($fp, 1024); //get 1024 bytes from the socket
            $result = trim($result);

            //Make sure the .wav file is on the disk before returning
            //We will wait up to $maxwaittime and we will check for the files every $waitinc
            if($result == "woohoo") {
                $maxwaittime = 30000000; // max time to wait for generator in microseconds
                $waitinc = 10000; // max time to check in microseconds
                $waitedsofar = 0; // in microseconds

                while($waitedsofar < $maxwaittime && (!file_exists($this->soundpath)) ) {
                    $waitedsofar = $waitedsofar + $waitinc;
                    usleep($waitinc);
                }

                if(file_exists($this->soundpath)) {
                    $return->picname = $this->picname;
                    $return->soundname = $this->soundname;
                }
            }
        }
```

```
        return $return;
    }

    //Generates the MATLAB commands for a valid $input sound request
    function generateCommands($input) {
        global $tempdir;

        $random = rand();
        $soundname = $random . ".wav";
        $this->soundpath = $tempdir . "/" . $soundname;
        $picname = $random . ".jpg";
        $picpath = $tempdir . "/" . $picname;

        $this->mcmds = array("sample_freq = " . $input->sampleFreq . ";");

        $soundArray = $input->soundArray;
        foreach($soundArray as $i => $sound) {
            //If the sound is a tone, generate MATLAB commands for the tone.
            if($sound->type == 'stdtone') {
                $this->mcmds = array_merge($this->mcmds, array("stdtone" . $i . "_freq = " .
$sound->freq . ";",
                    //Define the tone envelope
                    "stdtone" . $i . "_db = " . $sound->db . ";",
                    "stdtone" . $i . "_ip = " . $sound->ip . ";",
                    "stdtone" . $i . "_risetime = " . $sound->risetime . ";",
                    "stdtone" . $i . "_ontime = " . $sound->ontime . ";",
                    "stdtone" . $i . "_falltime = " . $sound->falltime . ";",
                    "stdtone" . $i . "_phase = " . $sound->phase . ";",

                    "stdtone" . $i . "_ip_sf = floor(sample_freq*stdtone" . $i . "_ip/1000);",
                    "stdtone" . $i . "_risetime_sf = floor(sample_freq*stdtone" . $i .
"_risetime/1000);",
                    "stdtone" . $i . "_ontime_sf = floor(sample_freq*stdtone" . $i .
"_ontime/1000);",
                    "stdtone" . $i . "_falltime_sf = floor(sample_freq*stdtone" . $i .
"_falltime/1000);",
                    "stdtone" . $i . "_size_sf = stdtone" . $i . "_ip_sf + stdtone" . $i .
"_risetime_sf + stdtone" . $i . "_ontime_sf + stdtone" . $i . "_falltime_sf;",

                    "if(stdtone" . $i . "_risetime_sf == 0)",
                    "    rampon = zeros(1,0);",
                    "else",
                    "    rampon = sin(linspace(0,pi/2,stdtone" . $i . "_risetime_sf)).^2;",
                    "end",
                    "if(stdtone" . $i . "_falltime_sf == 0)",
                    "    rampoff = zeros(1,0);",
                    "else",
                    "    rampoff = sin(linspace(pi/2,pi,stdtone" . $i . "_falltime_sf)).^2;",
                    "end",
                    "stdtone" . $i . "_envelope = [zeros(1,stdtone" . $i . "_ip_sf) , rampon ,
ones(1,stdtone" . $i . "_ontime_sf), rampoff];",

                    "t=[0:1/sample_freq:(stdtone" . $i . "_size_sf - 1)/sample_freq];",
                    //Define the tone as a sin wave multiplied by its envelope
                    "stdtone" . $i . " = sin(2*pi*t*stdtone" . $i . "_freq + stdtone" . $i .
"_phase) .* stdtone" . $i . "_envelope;",
                    "stdtone" . $i . " = stdtone" . $i . "*10^(stdtone" . $i . "_db/20);"));
            //If the sound is a stdnoise, generate MATLAB commands for it
            } else if($sound->type =='stdnoise') {
                if($sound->seed == -1) {
                    $seed = rand(0,429496729);
                } else {
                    $seed = $sound->seed;
                }
```

```
                $this->mcmds = array_merge($this->mcmds, array(
                    //define the noise envelope
                    "stdnoise" . $i . "_db = " . $sound->db . ";",
                    "stdnoise" . $i . "_ip = " . $sound->ip . ";",
                    "stdnoise" . $i . "_risetime = " . $sound->risetime . ";",
                    "stdnoise" . $i . "_ontime = " . $sound->ontime . ";",
                    "stdnoise" . $i . "_falltime = " . $sound->falltime . ";",
                    "stdnoise" . $i . "_lowpass = " . $sound->lowpass . ";",
                    "stdnoise" . $i . "_highpass = " . $sound->highpass . ";",

                    "stdnoise" . $i . "_ip_sf = floor(sample_freq*stdnoise" . $i . "_ip/1000);",
                    "stdnoise" . $i . "_risetime_sf = floor(sample_freq*stdnoise" . $i .
"_risetime/1000);",
                    "stdnoise" . $i . "_ontime_sf = floor(sample_freq*stdnoise" . $i .
"_ontime/1000);",
                    "stdnoise" . $i . "_falltime_sf = floor(sample_freq*stdnoise" . $i .
"_falltime/1000);",
                    "stdnoise" . $i . "_size_sf = stdnoise" . $i . "_ip_sf + stdnoise" . $i .
"_risetime_sf + stdnoise" . $i . "_ontime_sf + stdnoise" . $i . "_falltime_sf;",

                    "if(stdnoise" . $i . "_risetime_sf == 0)",
                    "    rampon = zeros(1,0);",
                    "else",
                    "    rampon = sin(linspace(0,pi/2,stdnoise" . $i . "_risetime_sf)).^2;",
                    "end",
                    "if(stdnoise" . $i . "_falltime_sf == 0)",
                    "    rampoff = zeros(1,0);",
                    "else",
                    "    rampoff = sin(linspace(pi/2,pi,stdnoise" . $i . "_falltime_sf)).^2;",
                    "end",
                    "stdnoise" . $i . "_envelope = [zeros(1,stdnoise" . $i . "_ip_sf) , rampon ,
ones(1,stdnoise" . $i . "_ontime_sf), rampoff];",
                    "randn('state'," . $seed . ");",
                    "stdnoise" . $i . " = 2*randn(1,stdnoise" . $i . "_size_sf) - 1;"));

                //apply the lowpass butterworth filter
                if($sound->lowpass != -1) {
                    $this->mcmds = array_merge($this->mcmds, array(
                        "[LB,LA] = butter(" . $sound->loworder . ",[stdnoise" . $i .
"_lowpass/(sample_freq/2)],'low');",
                        "stdnoise" . $i . " = filter(LB,LA,stdnoise" . $i . ");"));
                }

                //apply the highpass butterworth filter
                if($sound->highpass != -1) {
                    $this->mcmds = array_merge($this->mcmds, array(
                        "[HB,HA] = butter(" . $sound->highorder . ",[stdnoise" . $i .
"_highpass/(sample_freq/2)],'high');",
                        "stdnoise" . $i . " = filter(HB,HA,stdnoise" . $i . ");"));
                }

                //multiply the noise by its envelope
                $this->mcmds = array_merge($this->mcmds, array(
                    "stdnoise" . $i . " = stdnoise" . $i . " .* stdnoise" . $i . "_envelope;",
                    "stdnoise" . $i . " = stdnoise" . $i . "*10^(stdnoise" . $i . "_db/20);"));
            }
        }

        $this->mcmds = array_merge($this->mcmds, array("maxsize=0;"));

        //compute the maximum duration tone or noise in the sound
        foreach($soundArray as $i => $sound) {
            $this->mcmds = array_merge($this->mcmds,array("maxsize=max([maxsize,size(" .
```

```
$sound->type . $i . ",2)]);" ));
        }

        foreach($soundArray as $i => $sound) {
            //resize the sound and set the channel
            $this->mcmds = array_merge($this->mcmds, array($sound->type . $i . " = [" .
$sound->type . $i . ", zeros(1,maxsize-size(" . $sound->type . $i . ",2))]';"));

            //Set the sound to the right channel
            if($sound->channel == 'r') {
                $this->mcmds = array_merge($this->mcmds, array($sound->type . $i . " =
[zeros(size(" . $sound->type . $i . ",1),1)," . $sound->type . $i . "];"));
            } else if($sound->channel == 'l') {
                $this->mcmds = array_merge($this->mcmds, array($sound->type . $i . " = [" .
$sound->type . $i . ",zeros(size(" . $sound->type . $i . ",1),1)];"));
            } else {
                $this->mcmds = array_merge($this->mcmds, array($sound->type . $i . " = [" .
$sound->type . $i . "," .  $sound->type . $i . "];"));
            }
        }

        foreach($soundArray as $i => $sound) {
            if($i == 0) {
                $this->mcmds = array_merge($this->mcmds, array("playme = " . $sound->type . $i .
";"));
            } else {
                $this->mcmds = array_merge($this->mcmds, array("playme = playme + " . $sound->type
. $i . ";"));
            }
        }

        $this->mcmds = array_merge($this->mcmds, array(
            "wavwrite(playme,sample_freq,32,'$this->soundpath');",
            "plot([1:maxsize]/sample_freq,playme);",
            "set(gca,'ylabel',text(0,0,'Magnitude'),'FontSize',10);",
            "set(gca,'xlabel',text(0,0,'Time (seconds)'));",
            "print -djpeg -r72 $picpath;",
            "clear all;"));

        $this->soundname = $soundname;
        $this->picname = $picname;

    }
}
?>
```

```
<?
/*
daily.php
Daily Maintenance Script
Michael Naber

This script is run daily and performs two tasks.
    1) Deletes old generated .wav and .jpg files from the temp directory
    2) Delete old matlab commands from the database
*/

include('config.php');
//sets $tempdir and $absolute_uri

//delete the old (more than 5 days old) .wav and .jpg files from the temp directory
exec('find /var/www/html/soundgen/temp -type f -mtime +5 | grep "wav\|jpg" | xargs rm -f');

//delete the old matlab commands from the database
mysql_connect("localhost", "root", "soundlab") or die(mysql_error());
mysql_select_db("mcmds") or die(mysql_error());
mysql_query("TRUNCATE TABLE `jobs`");
sleep(5);
mysql_query("TRUNCATE TABLE `commands`");

?>
```

```
<?
/*
config.php
Just a configuration file...
*/

//The http path where the library is located
$absolute_uri = "http://tuliptree.mit.edu/soundgen";
//The file path where the library is located
$tempdir = "/var/www/html/soundgen/temp";

?>
```

```
<?
/*
MATLAB Daemon PHP Script
Michael Naber

Description:
This script is run when the server boots, and remains running at all times. Its purpose is to pass
matlab commands to the MATLAB process it continually holds open. Keeping the MATLAB process
continually running prevents the MATLAB startup delay that would happen otherwise.
*/

include('/root/soundgen_daemon/config.php'); //sets $tempdir and $absolute_uri

$host = "127.0.0.1"; // listen on the the localhost
$port = 44445;       // the listening tcp port

//Make the script not time out
set_time_limit(0);

//Listen on the tcp address and port specified above
$socket = socket_create(AF_INET, SOCK_STREAM, 0) or die("Could not create socket\n");
$result = socket_bind($socket, $host, $port) or die("Could not bind to socket\n");
$result = socket_listen($socket, 3) or die("Could not set up socket listener\n");

//Descriptorspec to be used by the MATLAB process.
$descriptorspec = array(
    0 => array("pipe", "r"),  // stdin is a pipe that the child will read from
    1 => array("pipe", "$tempdir/stdout.txt", "a"),  // stdout is a pipe that the child will write
to
    2 => array("file", "$tempdir/error.txt", "a") // stderr is a file to write to
);

//Start the MATLAB process with no GUI
$process = proc_open("/usr/local/bin/matlab -nodesktop -nodisplay -nojvm", $descriptorspec,
$pipes);
echo("Matlab started\n");

do{

    $spawn = socket_accept($socket) or die("Could not accept incoming connection\n");

    //Receive the command ID
    $cmdid = socket_read($spawn, 1024);
    $cmdid = trim($cmdid);

    //Get the commands to generate from the database
    mysql_connect("localhost", "root", "soundlab") or die(mysql_error());
    mysql_select_db("mcmds") or die(mysql_error());
    $q_mcmds = "SELECT * FROM `mcmds`.`commands` WHERE `cmdid` =$cmdid ORDER BY `seq` ASC";
    $r_mcmds = mysql_query($q_mcmds);
    mysql_close();

    //Perform the generation
    if (is_resource($process)) {
        // $pipes now looks like this:
        // 0 => writeable handle connected to child stdin
        // 1 => readable handle connected to child stdout

        $time_start = microtime(true);
        $time_diff = 0;
        //10 seconds MAX to run the generation
        while($w_mcmds = mysql_fetch_array($r_mcmds)) {
            $mcmd = stripslashes($w_mcmds['cmd']);
```

```
            if($time_diff < 10) {
                fwrite($pipes[0], $mcmd . "\n");
            } else {
                fwrite($pipes[0], "clear all;" . "\n");
                break;
            }
            $time_now = microtime(true);
            $time_diff = $time_now - $time_start;
        }
    }

    //If we finished generating in less than 10 seconds return success
    if($time_diff < 10) {
        $output = 'woohoo';
    } else {
        $output = 'boohoo';
    }

    socket_write($spawn, $output, strlen ($output)) or die("Could not write output\n");
    socket_close($spawn);

} while(true);

//These last few lines should never execute since the previous while look should never end.
socket_close($socket);
fclose($pipes[0]);
fclose($pipes[1]);
proc_close($process);
?>
```