Software for Psychoacoustic Experiment Design

by

Ray C. Cheng

Submitted to the Department of Electrical Engineering and Computer Science

in Partial Fulfillment of the Requirements for the Degrees of

Bachelor of Science in Electrical Engineering and Computer Science

and Master of Engineering in Electrical Engineering and Computer Science

at the Massachusetts Institute of Technology

May 27, 2003

Author_____
Department of Electrical Engineering and Computer Science
May 27, 2003

Certified by___
Louis D. Braida
Thesis Supervisor

Accepted by___
Arthur C. Smith
Chairman, Department Committee on Graduate Theses

Software for Psychoacoustic Experiment Design
by
Ray C. Cheng

Submitted to the
Department of Electrical Engineering and Computer Science

May 27, 2003

In Partial Fulfillment of the Requirements for the Degree of
Bachelor of Science in Electrical Engineering and Computer Science
and Master of Engineering in Electrical Engineering and Computer Science

ABSTRACT

Runexp is a distributed software system allows users to design psychoacoustic experiments in Lisp and that is configured for audio laboratory hardware. The first program, Rxp03, runs on the Espud Lisp interpreter and controls the operation of a real-time experiment. The program communicates via Simple Sockets with Meta, a server program that generates stimuli and collects subject responses using Matlab. Rxp03 models the trials in an experiment as a sequence of stimulus intervals and maintains a queue of commands that describe the state of the experiment. Meta also maintains an internal model of the current trial as a set of waveform parameters. Rxp03 contains a set of Espud functions to communicate with the experiment model. To run an experiment trial, users set the trial parameters, update the model on the Meta server, send the command to the server to begin generating stimuli, and send the command to collect a response from the subject. The response interaction is handled by a Matlab graphical user interface which the user must design for their own experiment. Runexp supports most of the experiment protocols used in auditory research.

Thesis Supervisor: Louis D. Braida
Title: Henry E. Warren Professor of Electrical Engineering

Table of Contents

1 Introduction **6**

2 Architecture **13**

2.1 Rxp03 and Meta 13

2.2 Laboratory Hardware 14

3 User Guide **16**

4 Implementation **20**

4.1 Communication 20

4.2 Generating Stimuli 22

4.3 Response GUIs 23

4.4 Debugging and Error-Checking 28

4.5 Future Work 29

A Rxp03 Function Specification **32**

B Experiment Messages **37**

C Sample Experiments **38**

D Runexp Code **56**

List of Figures

1-1 Model of a psychoacoustic experiment based on stimuli intervals. 7

1-2 A diagram of the timing of the tone and noise burst envelopes. 10

2-1 Implementation scheme for generating waveform stimuli. 15

2-2 Audio laboratory physical setup. 15

3-1 Runexp stimuli generation block diagram. The diagram shows the function blocks and the Espud functions that control them. 17

4-1 Interaction diagram between Runexp99, Meta, and Matlab. 21

4-2 Matlab GUI operation. 24

4-3 Matlab GUI M-file execution path created using GUIDE. 24

Chapter 1

Introduction

This thesis describes a software system called Runexp that has been implemented for the Sensory Communication Group at the MIT Research Laboratory of Electronics (RLE) for designing psychoacoustic experiments. The purpose of this project is to develop a laboratory tool for students in the MIT undergraduate course 6.182 – Psychoacoustics Project Laboratory – who have a working knowledge of Lisp. The software is designed to run on Espud, a Lisp-like interpreter designed by Joseph A. Frisbie et al. of the Sensory Communication Group [3]. Runexp is configured for the hardware setup in the audio laboratory.

Psychoacoustic experiments are a subset of psychophysical experiments which study the relationship between acoustic stimuli and their effects on the mind. In a typical psychophysical experiment, the subject is presented with a series of stimuli and then asked to respond to a question pertaining to the stimuli. The response is then recorded and depending on the experiment, the stimuli may or may not be adjusted and the procedure is repeated. For instance, in an adaptive experiment, the stimulus changes based on the subject's previous response. The process is repeated for a predefined number of trials or iterations, or until some terminating condition is satisfied.

A trial in a psychoacoustic experiment is made up of a sequence of stimulus intervals.

Each interval contains a single stimulus consisting of an additive tone burst and noise burst. An example of an N-interval experiment using this model is provided in Figure 1-1.

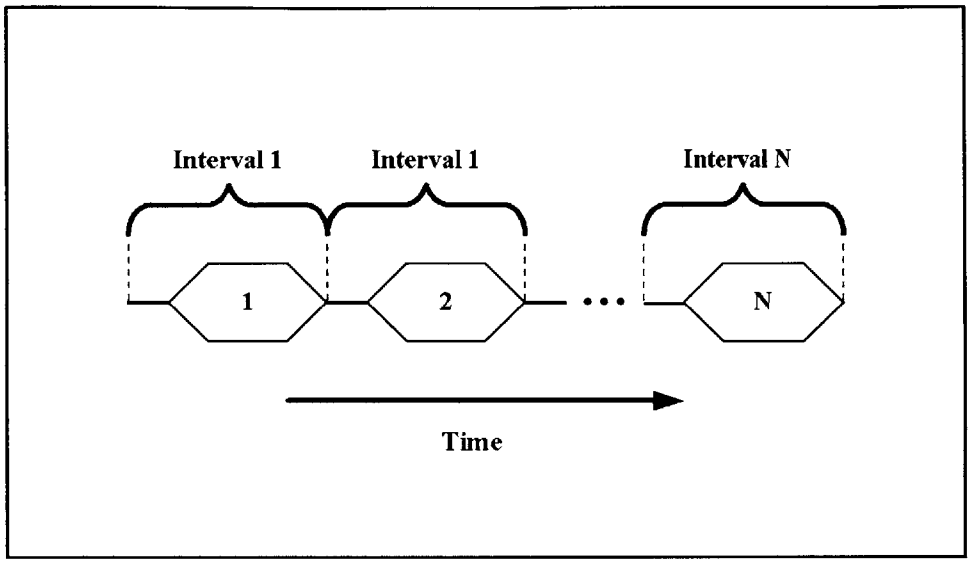

Figure 1-1: Model of a psychoacoustic experiment based on stimulus intervals. In a Runexp experiment, N=4 by default.

The trial is described mathematically by the equations

$$\vec{\omega} = [\vec{\omega}_1, \vec{\omega}_2, \ldots, \vec{\omega}_n]$$
$$\vec{\omega}_i = \vec{\tau}_i + \vec{\eta}_i$$

where ω is a vector of the sampled stimuli, and τ_i and η_i are the tone and noise bursts in interval i. The discrete-time representation of the stimuli ω is what is ultimately played via audio hardware in a system. To allow for experiments involving differing binaural stimuli, a separate $\omega - \omega_L$ and ω_R – is calculated and presented in each ear.

Tone bursts in an experiment are composed of pure tone (sinusoidal) waveforms. The tone burst τ_i can be simple, containing a single Fourier component frequency, or complex, consisting of multiple Fourier component frequencies summed together. In addition, each of the tones may be amplitude modulated. The equations for an unmodulated and modulated pure tone sine wave are

$$\sin(2\pi f_c t + \phi_c)$$
$$[1 + m(\sin 2\pi f_m t + \phi_m)]\sin(2\pi f_c t + \phi_c)$$

where f_c and f_m denote the carrier and modulation frequency, φ_c and φ_m are the carrier and modulation phase, m is the modulation index, and t is time.

Different types of noise may be used in a psychoacoustic experiment. The noise can be filtered or unfiltered and can bear different spectral shapes. Runexp generates Gaussian white noise, which is commonly used in auditory research. Gaussian white noise has a constant power density for all frequencies. Power density refers to the power – the energy per unit time – in a unit bandwidth of frequencies. Sounds that have continuous spectra, such as noise, can be described by their spectrum level or the intensity density – the energy per unit time per unit area in a unit bandwidth – expressed as a sound pressure level (SPL). Sound pressure level is defined as the sound level in decibels with respect to the reference intensity of 10^{-12} W/m^2, or equivalently a pressure of 20 uPa (N/m^{-2}). In addition, Gaussian white noise sample values have a Gaussian probability distribution function (PDF), which is equal to the probability distribution function of a normally distributed random variable with a mean μ of zero and variance σ^2

of one. The equations below show the PDFs of a random variable *Y* which is normally distributed and a random variable Z which has a Gaussian distribution.

$$Y \sim N(\mu,\sigma^2) \rightarrow f_Y(y) = \frac{1}{\sqrt{2\pi}\sigma} e^{-\frac{1}{2}(\frac{y-\mu}{\sigma})^2}$$

$$Z \sim N(0,1) \rightarrow f_Z(z) = \frac{1}{\sqrt{2\pi}} e^{-\frac{z^2}{2}}$$

Runexp uses by default a 10^{th}-order digital Butterworth filter to process the discrete-time Gaussian white noise sample to generate filtered noise. The specified Butterworth filter can be either lowpass, highpass, bandpass, or bandstop. This particular filter is chosen for its maximally flat magnitude response in the passband by providing the best Taylor series approximation to the ideal filter response. The Butterworth filter has an overall monotonic response from the analog frequencies $\omega=0$ to $\omega=\infty$. The magnitude response of the digital Butterworth filter at the Nyquist frequency (π radians/sample) is given by the equation

$$|H(j\Omega)| = \sqrt{\frac{1}{2}}, \Omega = \pi$$

The experimenter also specifies the timing of the stimuli in a psychoacoustic experiment. Runexp uses eight parameters to describe the timing of the tone and noise bursts. Figure 1-2 shows the parts of the tone and noise that are described by the interval pretime (IP), start difference (DS), noise risetime (RN), tone risetime (RT), tone ontime

(OT), tone falltime (FT), finish difference (DF), and noise falltime (FN) variables. The same tone and burst envelopes are used in Runexp to produce the stimuli in the left and right ears, ω_L and ω_R.

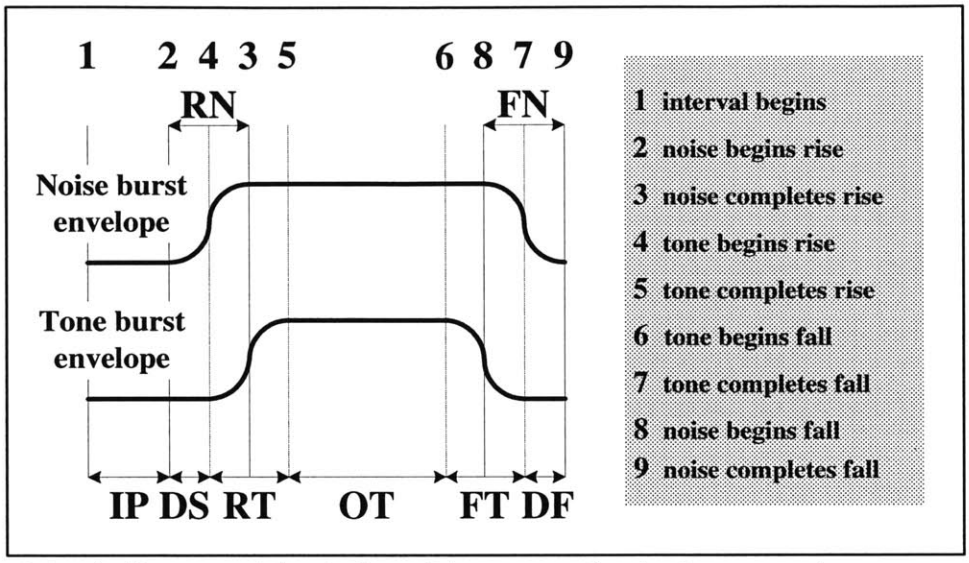

Figure 1-2: A diagram of the timing of the tone and noise burst envelopes presented in both ears.

The equations below define the discrete-time representations of the unscaled component tone bursts $T_{i,c}$ and noise burst N_i in interval i of an experiment given these timing parameters; the timing parameters are all expressed in lengths in samples given the sampling frequency f_s. The vector T_i is composed of a pre-tone period P_i followed by the component tone waveform S_i multiplied by the envelope w_i and a post-tone period Q_i. Similarly, the vector N_i consists of the pre-noise period P_i followed by the noise waveform Z_i multiplied by the envelope v_i. Runexp implements the risetime and falltime in the w_i and v_i envelopes using a Hanning window. The component tone waveform $S_{i,c}$ given by the index c is described by the equation for a sine wave which may or may not be modulated. The noise waveform Z_i contains samples of the random variable Z which

has a Gaussian distribution.

$$\overrightarrow{\mathrm{T}_{i,c}} = [\overrightarrow{P_i}, \overrightarrow{w_i}' \overrightarrow{S_{i,c}}, \overrightarrow{Q_i}]$$

$$P_i[t] = 0 \text{ for } t = 1,2,\ldots,(t_{i,ip} + t_{i,ds})$$

$$w_i[t] = [\overrightarrow{w_{i,rt}}, \overrightarrow{w_{i,ot}}, \overrightarrow{w_{i,ft}}]$$

$$w_{i,rt}[k] = 0.5(1 - \cos(\pi \frac{k-1}{t_{i,rt}})) \text{ for } k = 1,2,\ldots,t_{i,rt}$$

$$w_{i,ot}[k] = 1 \text{ for } k = 1,2,\ldots,t_{i,ot}$$

$$w_{i,ft}[k] = 0.5(1 + \cos(\pi \frac{k-1}{t_{i,ft}})) \text{ for } k = 1,2,\ldots,t_{i,ft}$$

$$S_{i,c}[t] = [1 + m(\sin \frac{2\pi g_{i,c}(t-1)}{f_s} + \varphi_{i,c})]\sin(\frac{2\pi f_{i,c}(t-1)}{f_s} + \phi_{i,c})$$

$$\text{for } t = 1,2,\ldots,(t_{i,rt} + t_{i,ot} + t_{i,ft})$$

$$Q_i[t] = 0 \text{ for } t = 1,2,\ldots,t_{i,df}$$

$$\overrightarrow{\mathrm{N}_i} = [\overrightarrow{P_i}, \overrightarrow{v_i}' \overrightarrow{Z_i}]$$

$$v_i[t] = [\overrightarrow{v_{i,rn}}, \overrightarrow{v_{i,on}}, \overrightarrow{w_{i,fn}}]$$

$$v_{i,rn}[k] = 0.5(1 - \cos(\pi \frac{k-1}{t_{i,rn}})) \text{ for } k = 1,2,\ldots,t_{i,rn}$$

$$v_{i,on}[k] = 1 \text{ for } k = 1,2,\ldots,(t_{i,ds} + t_{i,rt} + t_{i,ot} + t_{i,ft} + t_{i,df}) - (t_{i,rn} + t_{i,fn})$$

$$v_{i,fn}[k] = 0.5(1 + \cos(\pi \frac{k-1}{t_{i,fn}})) \text{ for } k = 1,2,\ldots,t_{i,fn}$$

$$Z_i[t] = Z \text{ for } t = 1,2,\ldots,(t_{i,ds} + t_{i,rt} + t_{i,ot} + t_{i,ft} + t_{i,df})$$

The final parameter in a psychoacoustic experiment is stimulus intensity. The intensity of a complex tone burst is specified in Runexp by β_i, the overall tone intensity in the given interval, $\alpha_{i,c}$, the attenuations of the particular Fourier components in the

interval, and ε, the gain in the experiment. Similarly, the intensity of a noise burst is specified by ρ_i, the intensity of the noise in the given interval, and ε. The following equations are then used to convert the intensities to their corresponding voltages; δ denotes the intensity supplied by a 1 volt peak-to-peak sinusoid and depends on the sensitivity of the hardware used.

$$\overrightarrow{\tau_i} = \sum_{c=1}^{m} v_{\text{tonescale},i,c} \times \overrightarrow{\mathrm{T}_{i,c}} \qquad \overrightarrow{\eta_i} = v_{\text{noisescale},i} \times \overrightarrow{\mathrm{N}_i}$$

$$v_{\text{tonescale},i,c} = v_{\text{ref}} \times 10^{\frac{\beta_i - \alpha_{i,c} + \varepsilon}{20}} \qquad v_{\text{noisescale},i} = v_{\text{ref}} \times 10^{\frac{\rho_i + \varepsilon}{20}}$$

$$v_{\text{ref}} = 10^{-\frac{\delta}{20}}$$

To obtain the overall tone burst τ_i in an experiment interval, each of the component bursts $T_{i,c}$ are scaled by the user-specified intensity $v_{tonescale,i,c}$ in volts and the resulting scaled vectors are summed together. Likewise, the scaled noise burst η_i is obtained by multiplying the unscaled burst N_i by $v_{noisescale,i}$. The τ_i and η_i are then summed to produce ω_i. Runexp allows the user to specify β_i, ρ_i, and ε separately for each ear, uses the appropriate values to compute ω_L and ω_R.

Chapter 2

Architecture

2.1 Rxp03 and Meta

Runexp consists of two programs, `Rxp03` and `Meta`. `Rxp03` is an Espud program that controls the psychoacoustic experiment; in addition, the program contains a set of Espud functions for communicating with `Meta`. `Meta` is a server program that generates acoustic waveforms and response interfaces via Matlab.

`Rxp03` and `Meta` communicate using the Simple Sockets Library (SSL), an open-source application programming interface developed by Charles E. Campbell Jr. and Terry McRoberts for performing socket communication [1]. During an experiment, `Rxp03` opens a client connection to `Meta` and sends commands to update trial parameters, generate trial stimuli, and obtain subject responses. `Meta` obtains a response from a subject by opening a graphical user interface (GUI) which is typically designed in Matlab using the Graphical User Interface Design Environment (GUIDE).

`Rxp03` maintains a model of the experiment as a queue of commands. These commands are sent to `Meta` at points determined by the user in the user program. `Meta` maintains its own model of the current trial of an experiment as a set of user-defined waveform parameters; the model is initialized when the `Meta` program is run.

2.2 Laboratory Hardware

The audio laboratory contains sound booths which are each equipped with a personal computer (PC) running Microsoft Windows and Athena Linux. The PC contains a LynxOne audio interface card that generates waveforms using a 24-bit delta-sigma digital-to-analog (D/A) converter manufactured by Crystal Semiconductor. The electrical signal is then fed through a Tucker-Davis Technologies PA4 programmable attenuator and HB6 headphone buffer into the sound booth. Inside the booth, the stimuli are presented via Sennheiser 580 headphones.

Under the previous hardware configuration in the lab, the stimuli were generated using a DSP 96000 board on a DOS PC. A set of programs called `moda` and `modb` set the PC up to communicate with Runexp running on Espud. Subjects responded via a response terminal in the booth connected serially to a Datability Vista VCP-1000 server. Espud was compiled for the VAX VMS operating system and ran on a terminal at the lab workstation. An earlier version of `Rxp03` called `Runexp99` was developed by David Lum of the Sensory Communication Group for VMS [4].

Figures 2-1 and 2-2 show the scheme for generating stimuli and the current setup in the audio laboratory.

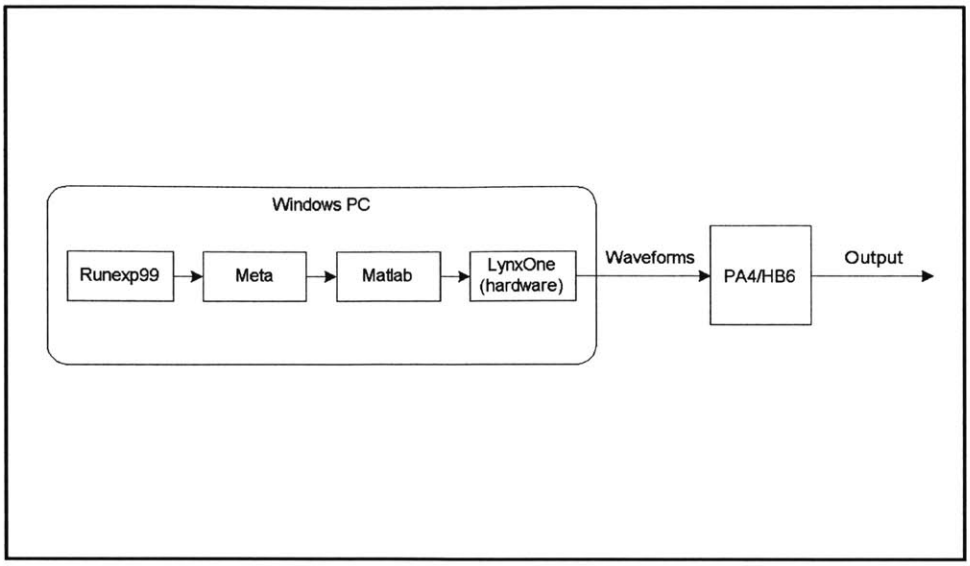

Figure 2-1: Implementation scheme for generating waveform stimuli.

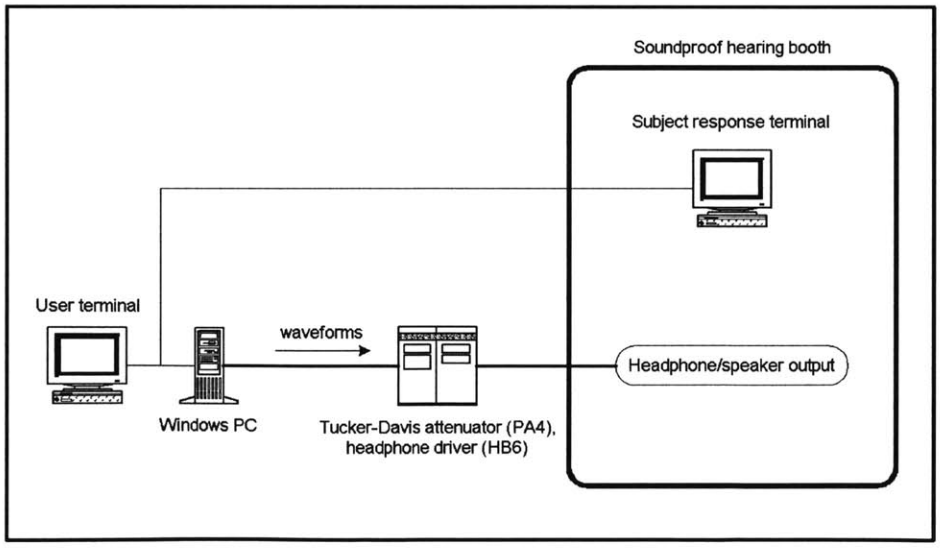

Figure 2-2: Audio laboratory physical setup.

Chapter 3

User Guide

To run the Runexp software, the user must run both Espud and `Meta`. `Rxp03` must also be loaded explicitly to Espud. To run `Meta`, the user should double-click on the meta.exe icon in Windows.

The basic plan for running a Runexp experiment is to make a connection to `Meta` using Simple Sockets, modify the trial parameters, and play the stimuli intervals. Figure 3-1 shows the system for generating stimuli and the associated Espud functions for controlling various function blocks. A complete listing of the Espud functions can be found in the Appendix A.

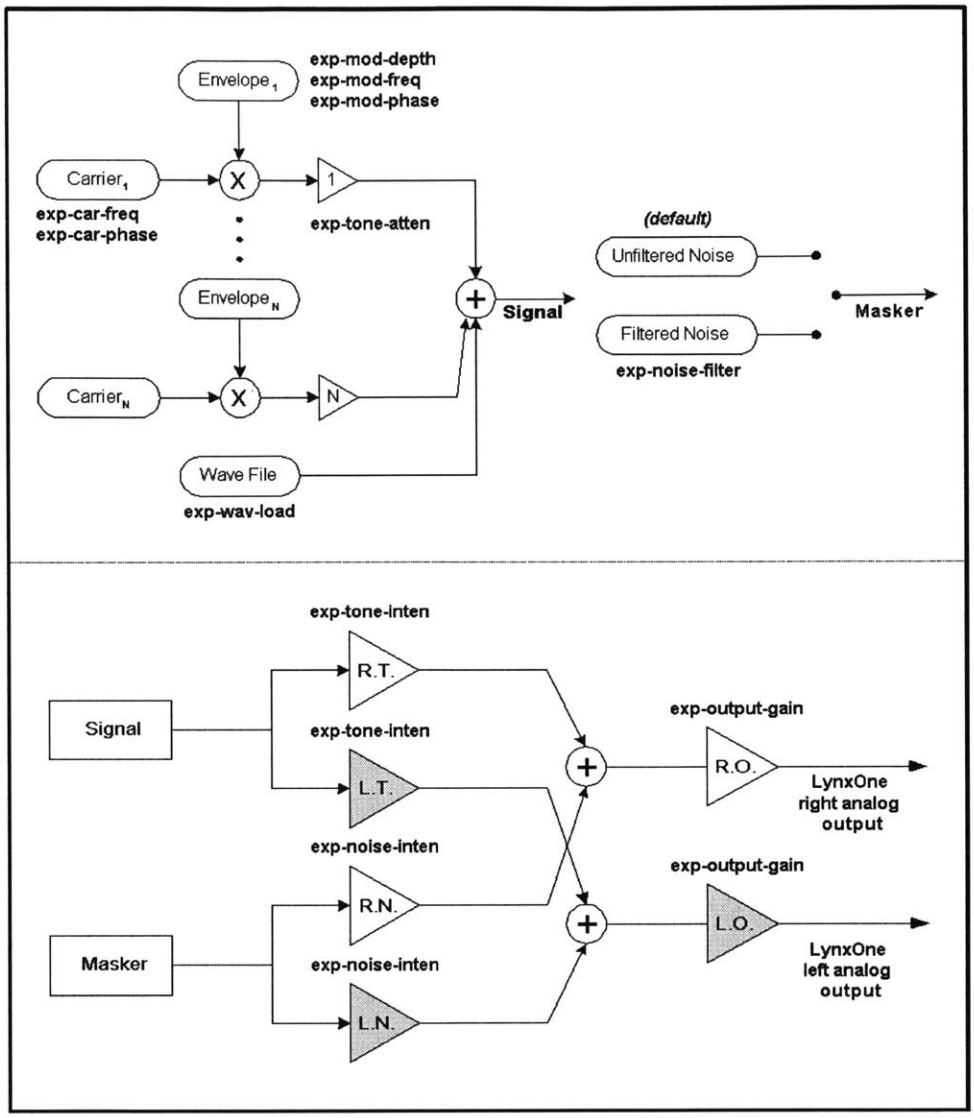

Figure 3-1: Runexp stimuli generation block diagram. The diagram shows the function blocks and the Espud functions that control them.

When starting an experiment, the user program first calls `exp-intervals` and `exp-tones` to set the number of intervals and Fourier tones per interval. If no call is made to these two procedures, `Rxp03` uses default values for the experiment. After setting the intervals and tones, the user calls `exp-connect` to open a connection to `Meta`. At this point, `Meta` is ready to receive commands from `Rxp03`. The user then sets the parameters

for the current trial of the experiment. Once the parameters have been set, the user calls `exp-send-changes` to update the model in `Meta` and then `exp-go` to actually play the stimuli. The subject's response is obtained by calling `exp-get-response`.

The example user program below illustrates a single two-interval trial in Runexp. The first interval contains a complex tone burst and the second interval contains a pure tone burst with unfiltered noise.

```
; Set the experiment model
(exp-intervals 2)
(exp-tones 1)

; If no connection already exists, connect to the runexp server on
the local machine
(cond ((not exp-connection) (exp-connect "")))

; Set the timing of the stimuli
(exp-ip 100.0 1)
(exp-rt 50.0 1)
(exp-ot 500.0 1)
(exp-ft 50.0 1)
(exp-ip 100.0 2)
(exp-rn 50.0 2)
(exp-ds 25.0 2)
(exp-rt 50.0 2)
(exp-ot 500.0 2)
(exp-ft 50.0 2)
(exp-df 25.0 2)
(exp-fn 50.0 2)

; The interval 1 complex consists of a 70 dB 1-kHz tone and a 60
dB 1.4-kHz tone
(exp-tone-inten 70.0 1 'both)
(exp-car-freq 1000 1 1)
(exp-car-freq 1400 1 2)
(exp-tone-atten 10.0 1 2)

; Set the single tone in interval 2 to 1.7-kHz, 65 dB SPL
(exp-tone-inten 65.0 2 'both)
(exp-car-freq 1700 2 1)

; Set the noise burst level in interval 2 to 55 dB SPL
(exp-noise-inten 55.0 2 'both)

; Commit these settings and begin generating the stimuli
```

```
(exp-send-changes)
(exp-go)
```

Chapter 4

Implementation

4.1 Communication

When `Meta` is first started, it starts a Matlab engine process for later use. It then creates a Simple Sockets server named `runexp` and registers the server with the Simple Sockets PortMaster. The PortMaster is a utility that maps server names to dynamically generated port numbers. The Simple Sockets Library determines the port number with the PortMaster transparently, so a user only has to specify the server name. `Meta` then waits for `Rxp03` to open a Simple Sockets client connection request to the `runexp` server on the host machine. Once `Rxp03` does this, `Meta` immediately accepts the request, and the socket connection is established.

After the connection is created, `Meta` runs in a loop processing messages from `Rxp03` as they arrive. The program remains in the loop until `Rxp03` terminates and closes the socket connection. When `Meta` terminates, it closes its current socket connection with `Rxp03`, stops the socket server, and halts the Matlab engine session before exiting.

Figure 4-1 below shows the interaction between `Rxp03`, `Meta`, and Matlab for a single trial of an experiment.

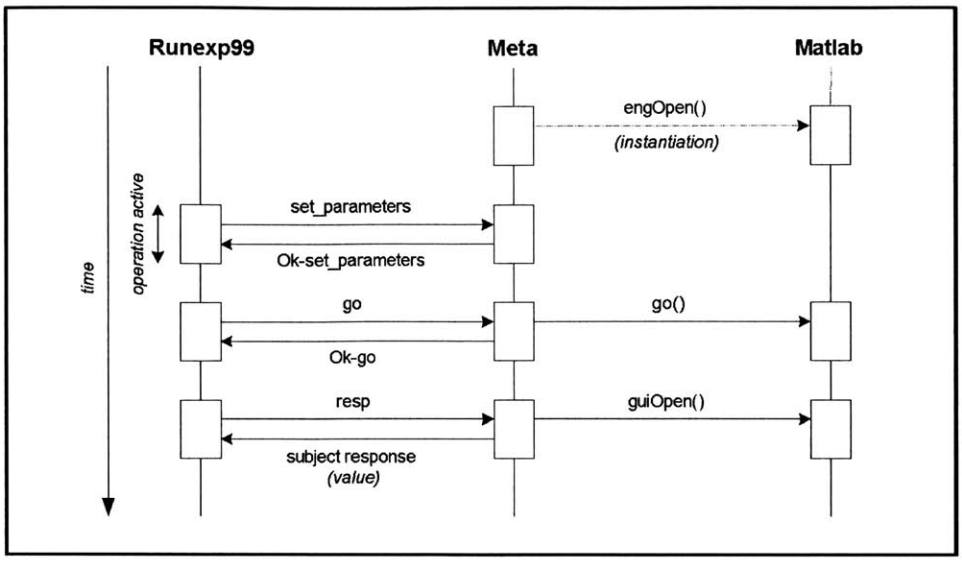

Figure 4-1: Interaction diagram between Runexp99, Meta, and Matlab.

The `Rxp03` message string syntax is provided below. Each string consists of one or more messages that are prefaced with a MSG tag. An END tag terminates the string. There are two different types of messages that `Rxp03` sends to `Meta`: 1) commands and 2) stimulus parameters. The commands do not take any arguments, while the parameters contain arguments such as value, interval, and channel, which are specific to the parameter. An example message string is shown below. A list of the `Rxp03` functions and the corresponding messages that get sent to `Meta` can be found in Appendix B.

```
<<MSG>> command <<MSG>> param1 arg1 arg2 <<MSG>> param2 arg1
arg2 … <<MSG>> set_parameters <<END>>
```

After it has received a string, `Meta` calls the parse function `fill_in_args` and processes the set of messages in the string sequentially. Parameters are stored in C array

variables that together define the waveform for the present trial. All of the parameters are defined in the header file *mod.h* and are initialized to their predetermined default values. A trial uses these default values only if `Rxp03` has not updated the corresponding parameters at any point in the experiment.

4.2 Generating Stimuli

When `Meta` receives the command to play a trial, it first translates the user-specified parameters for the trial – stored in C arrays – into the Matlab `mxArray` data type. `Meta` then binds the parameter values to variable names in the Matlab engine workspace so that they can be called to calculate the trial waveform vector described in Chapter 1. Specifically, the calculations are performed by calling the Matlab API function `engEvalString` which evaluates a string expression in the Matlab workspace. After it has constructed the trial waveform vector, `Meta` calls the Matlab `sound` function to play the stimuli.

To insert a Matlab array into the engine workspace so that it can be called as a variable in `engEvalString`, `Meta` utilizes two functions. First, it calls `mxSetName` to bind the `mxArray` type to a string name. `mxSetName` assigns the characters in the string to a fixed-width length in the memory; it can also be used to change an existing name for a Matlab array type. After it calls `mxSetName`, `Meta` then calls `engPutArray` to insert the mxArray variable into the current Matlab workspace. The string provided in `mxSetName` then becomes the handle that is used to call the variable

in Matlab using `engEvalString`.

Rather than presenting the stimuli described above consisting of tone bursts and noise bursts, `Rxp03` can alternatively load and present pregenerated audio files in the Microsoft .WAV file format as the stimuli in an experiment. To load a wave file, `Meta` reads the file specified by the `Rxp03` procedure `exp-wav-load` from the working directory in Matlab by calling the Matlab function `wavread`. When `Meta` receives the go command, it plays the sampled wave file vector via the `sound` routine using the sampling frequency and number of bits per sample obtained by `wavread`. The `wavread` function is capable of reading multi-channel data with a maximum of 16 bits per sample. After Meta plays a wave file, it waits for the file to finish being played before sending the return string back to the user program. This is intended to ensure that wave files may be loaded in succession in an experiment – for example, in the case that the user wanted to load separate wave files as alternative stimuli in a trial – without an unwanted temporal overlap of the stimuli.

4.3 Response GUIs

Graphical UIs for experiments are generated by GUIDE, the Matlab tool for building GUIs. Users must design their own response GUI for an experiment. Figures 4-2 and 4-3 show the general operation of the GUI and the execution path of the GUI application file.

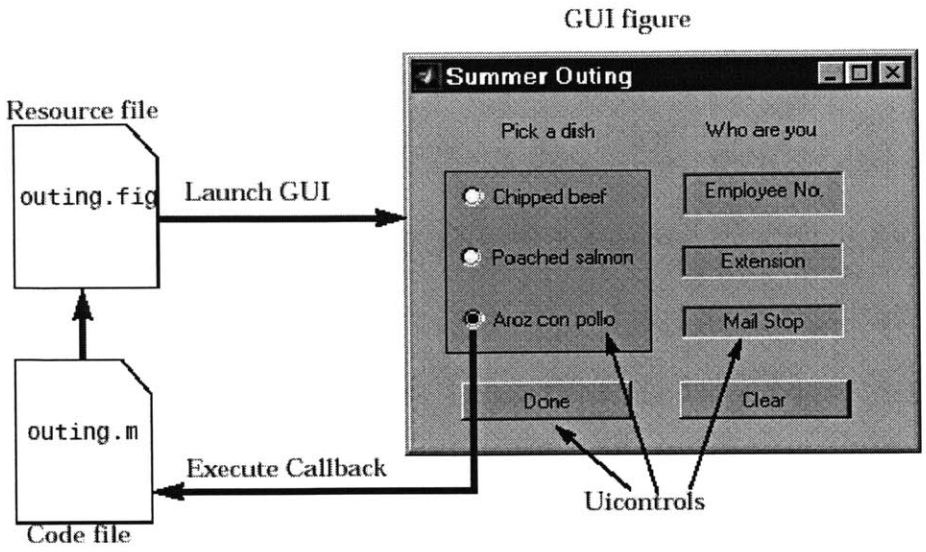

Figure 4-2: Matlab GUI operation.

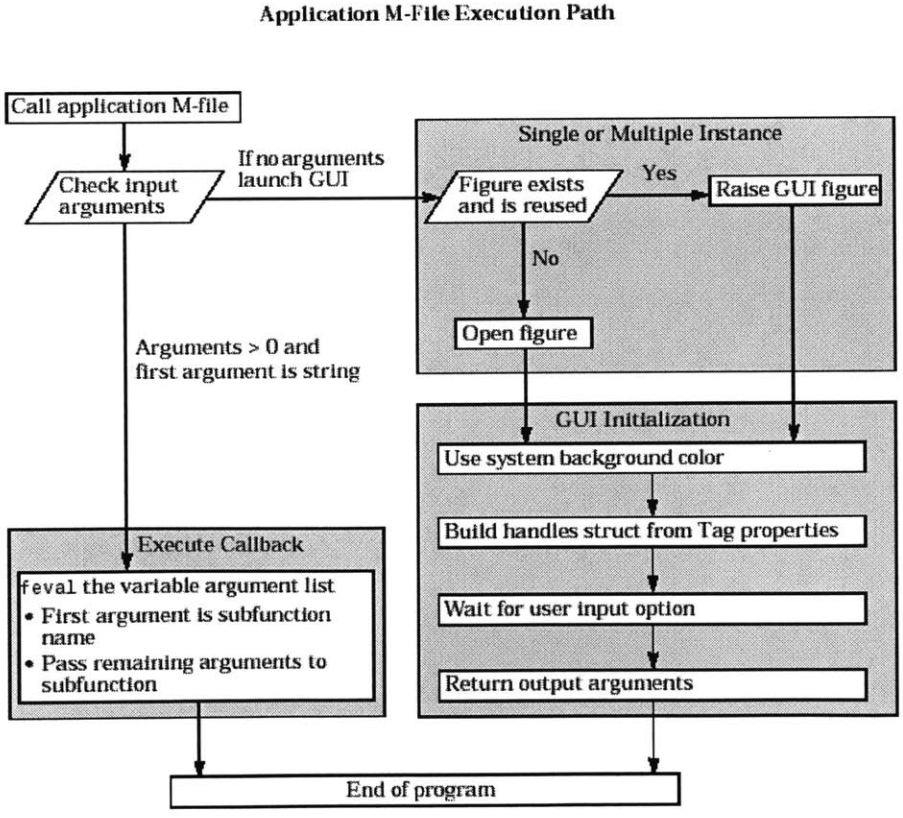

Figure 4-3: Matlab GUI M-file execution path created using GUIDE.

When Meta receives the command from Rxp03 to process a response, it first initializes the response variable resp to the empty array to indicate that there is no response for the current trial. Meta then launches the specified GUI (from the default Matlab work directory) and loops until the subject has replied. At that point, Meta returns the value of the response to Rxp03, and the transmission is complete. The object callback function in the application M-file must declare the response variable as global and set the value of this variable after the subject has submitted a reply (e.g. by clicking on a pushbutton corresponding to a particular answer, or typing the response into a text field and clicking a submit pushbutton). Meta determines whether the response variable contains a return value using the engGetArray function, which copies the variable from the Matlab engine's workspace. The example code below illustrates the response interface in Meta.

```
mxArray *resp = NULL;
double *prespr;

engEvalString(ep, "global resp;");
if (strcmp(args[0],"resp")==0) {
   engEvalString(ep, "resp=[];");
   engEvalString(ep, args[1]);
   while(1)
   {
         resp = engGetArray(ep, "resp");
         prespr = mxGetPr(resp);
         if (pxrespr != 0) break;
   }
   (return(prespr[0]));
}
```

4.4 Client-Server Protocol

For every command string that the Rxp03 sockets client sends to Meta, Meta returns a response message indicating whether or not the commands were successful. The Rxp03 command strings are sent asynchronously using the Espud primitive function s-send, which sends a message to a server, waits for a reply string, and returns the string. The s-send function is implemented by calling the Simple Sockets Sputs routine in C to send a string across a socket connection and then calling Sgets to obtain the response string. By requiring a response for every message that is sent via s-send, Espud allows network client programs to perform error-checking to determine the cause of any failed transmissions.

Communication using s-send also serializes the dialogue between a network server and the Espud client and prevents deadlock. Rxp03 is able to send a message only if it is not currently waiting for a reply to a previous message. Conversely, Meta will never have more than one message in the socket buffer waiting to be processed. Rxp03 also sets the timeout period using s-timeout so that if the program does not receive a return string by the specified time after the s-send is executed, Espud exits with an error and the experiment halts. The timeout needs to be of a long enough duration so that Matlab can generate the waveform, present the stimulus, and the user can make a response. By default, the timeout period is set to 100 seconds.

There are only three instances in which Rxp03 calls s-send to the Meta server. These instances are shown in Figure 4-1. The first is when the user calls exp-send-changes to update the model parameters in Meta. For the most part, this is the only

procedure that sends multiple commands in a single transmission (see exception below). Meta processes each of these messages separately and provides an internal return string for each. The final command in the string that Runexp sends when exp-send-changes is evaluated is "set_parameters". The purpose of this command is to invoke the actual return string by Meta to the s-send procedure that was called via exp-send-changes. If all of the commands in the sequence are processed without any errors, Meta responds with the string " Ok-set_parameters". If, however, Meta encounters an error while processing any of the commands in the sequence, the server returns an error which the user program can check.

The second instance in which Rxp03 calls s-send is when the exp-go procedure is evaluated. This procedure causes the single "go" command to get sent to Meta. The one exception to this behavior is if exp-go is immediately called after connecting to Meta without having called exp-send-changes. In this case Rxp03 prefaces the "go" command with the three setup messages consisting of the number of intervals and number of tones per interval in the experiment, followed by the command to reset the state of the model in Meta. Resetting the model causes Meta to revert to the default parameters values for the experiment and defined in the header file *mod.h*. After Meta has computed the trial waveform and presented the stimuli, the program returns the string " Ok-go".

The third instance where Rxp03 sends a message to Meta via s-send occurs when the user evaluates exp-get-response to obtain the subject's response. The return

value by Meta to this procedure – which gets bound to the external variable "exp-response" – is a string that corresponds to the subject's reply. If the subject does not respond within the timeout period specified, then Espud generates an error stating that the call to s-send in exp-get-response did not receive server verification. The same error occurs if Meta is closed prematurely.

4.4 Debugging and Error-Handling

Rxp03 contains four debugging switches that set different debug levels in the program. Setting EXP-DEBUGGING? causes Rxp03 to print all the underlying user interface procedures that get evaluated for each experiment procedure that is called. This serves as a top-level debugger. For example, Rxp03 indicates in this mode the call to exp-reset-models that is evaluated when the user attempts to connect to the SSL server named "runexp" using exp-connect. The debug mode set using EXP_DEBUGGING? prints out the changes to the internal models that are made for the Rxp03 procedures as they get called. Setting EXP_NOIO? allows the user to debug an experiment without doing physical I/O to the Meta hardware server. Finally, the EXP_NOIO_PRINT? debug mode prints to the screen the socket strings that get sent to Meta, followed by the length of the string in characters.

Meta can be compiled to run in a debug setting. When run in the debug mode, Meta displays in the program window the communication between it and Rxp03 as well as various operations that it performs. Moreover, Meta prints errors in the program window

in both the debug and normal modes. For instance, it returns an error if an improperly formatted message is sent from Rxp03, an unknown message is received, or if known message is missing any arguments. During initialization, Meta returns an error if it fails to open a sockets server.

4.5 Future Work

Using Runexp, users can design almost any type of psychoacoustic experiment of interest in auditory research [5]. For instance, Runexp is capable of generating complex pure tone stimuli such as the following:

$$l(t) = A_1 \sin(2\pi 100t) + A_2 \sin(2\pi 200t)$$
$$r(t) = A_3 \sin 2\pi(100t + \pi/2) + A_4 \sin(2\pi 400t)$$

where $l(t)$ and $r(t)$ denote the stimuli that are presented to the left and right ears, respectively, and A_1, A_2, A_3, and A_4 are differing amplitudes. However, there are a few foreseeable limitations in the system as it exists. The first is that Runexp uses the same tone and noise burst envelopes to produce binaural waveforms. In the future, it would be beneficial to have distinct timing parameters that describe the envelopes for each ear. In addition, Runexp currently only produces one type of noise and implements one type of noise filter which has been sufficient for the laboratory assignments in 6.182. However, since Runexp runs off of the Matlab engine, it would be relatively easy to implement other types of noise and filters. Future versions of this software may also take advantage

of the signal processing and statistical toolboxes in Matlab to process waveforms and provide a deeper mathematical analysis of obtained results.

Bibliography

[1] Cambell, C.E., and McRoberts, T. *The Simple Sockets Library.* 2001.

[2] Gamma, E., R. Helm, R. Johnson, and J. Vlissides. *Design Patterns: Elements of Reusable Object-Oriented Software.* Addison Wesley, Massachusetts. 1995.

[3] Lum, D.S., and Braida, L.D. *Espud manual.* 1999.

[4] Lum, D.S. *Runexp manual.* 1999.

[5] Moore, B. C. *An Introduction to the Psychology of Hearing, Fifth Edition.* Academic Press, California. 2003.

Appendix A

Rxp03 Function Specification

The Rxp03 user interface procedures for implementing a psychoacoustic experiment using Runexp are described below. None of the functions have specified return values.

Meta Server Interactions

(EXP-CONNECT name) Connects to the runexp Simple Sockets server running on Meta and initializes the state of the connection and internal data structures. This function must be called prior to any other exp- functions except exp-intervals and exp tones.

name (integer) The network name of the computer on which the server is running or "" (the empty string) if the server is running locally.

Example: (exp-connect "gloin.mit.edu")

EXP-CONNECTION Initialized to false when Rxp03 is loaded and changed to a true value when exp-connect is called. Users may test the value of this variable but should never change it except through the call to exp-connect.

(EXP-RESET-MODELS) Resets all of the experimental parameters to their initial default values, both on Espud and in Meta. The default values are comprised of intervals of zero length, frequencies of zero, and inaudible waveforms. Calling exp-connect implicitly calls exp-reset-models.

(EXP-SEND-CHANGES) Updates Meta with the stimulus changes. This function must be called after all the changes to the stimulus have been made and before exp-go is called.

(EXP-GO) Sends the command to Meta to begin playing the stimuli.

(EXP-GET-RESPONSE file) Sends the command to Meta to obtain a response from the subject by launching a Matlab GUI. This function returns after the subject has submitted a response.

file (string) The path and name of the GUI file to load.

Example: (exp-get-response "c:/lab1_test")

EXP-RESPONSE Value of the most recent subject response in string format. The value of the response is equal to the response value that is set by the user in the Matlab GUI.

(EXP-INTERVALS ints) Sets the number of intervals in the experiment. The default number of intervals is 4. This function must be called before calling exp-connect.

ints (integer) The number of intervals. This cannot exceed the maximum given by

EXP_MAXX_INTVLS.

Example: `(exp-intervals 2)`

`(EXP-TONES itones)` Sets the number of Fourier components per interval in the experiment. The default number of tones is 8. This function must be called before calling `exp-connect`.

tones (integer) The number of tones. This cannot exceed the maximum given by EXP_MAXX_TONES.

Example: `(exp-tones 3)`

`(EXP-REPETITIONS nreps)` Sets the number of times a trial is repeatedly played. The default number of repetitions is 1.

nreps (integer) The number of repetitions.

Example: `(exp-repetitions 5)`

`(EXP-SAMPLE-RATE srate)` Sets the sampling rate of the digital-to-analog converters that will be used in the trial. The default sampling rate is 11.025 kHz.

srate (float) The sampling rate in kHz.

Example: `(exp-sample-rate 22.05)`

Tone Characteristics

`(EXP-CAR-FREQ freq interval tone)` Sets the carrier frequency of a component tone in a tone burst.

freq (float) The carrier frequency in Hertz. No check is made to ensure that the frequency satisfies the Nyquist condition.

interval (integer) The index of the interval to affect. Setting the index to 0 causes all the interval in the trial to be affected.

tone (integer) The index – beginning with 1 – of the tone in the interval to affect.

Example: `(exp-car-freq 1200 1 2)`

`(EXP-CAR-PHASE phase interval tone)` Sets the carrier phase of a component tone in a tone burst.

phase (float) The carrier phase in radians.

interval (integer) The index of the interval to affect. Setting the index to 0 causes all the interval in the trial to be affected.

tone (integer) The index – beginning with 1 – of the tone in the interval to affect.

Example: `(exp-car-phase pi 2 1)`

`(EXP-MOD-FREQ freq interval tone)` Sets the amplitude modulation frequency of a component tone in a tone burst.

freq (float) The modulation frequency in Hertz. No check is made to ensure that the frequency satisfies the Nyquist condition.

interval (integer) The index of the interval to affect. Setting the index to 0 causes all the interval in the trial to be affected.

tone (integer) The index – beginning with 1 – of the tone in the interval to affect.

Example: `(exp-mod-freq 2400 1 2)`

`(EXP-MOD-PHASE phase interval tone)` Sets the amplitude modulation phase of a component tone in a tone burst.

phase (float) The modulation phase in radians.

interval (integer) The index of the interval to affect. Setting the index to 0 causes all the interval in the trial to be affected.

tone (integer) The index – beginning with 1 – of the tone in the interval to affect.

Example: `(exp-mod-phase (/ pi 2) 2 3)`

`(EXP-MOD-DEPTH depth interval tone)` Sets the depth of amplitude modulation of a component tone in a tone burst.

depth (float) The depth of modulation from 0 to 1.

interval (integer) The index of the interval to affect. Setting the index to 0 causes all the interval in the trial to be affected.

tone (integer) The index – beginning with 1 – of the tone in the interval to affect.

Example: `(exp-mod-depth 0.05 2 1)`

`(EXP-TONE-INTEN inten interval chan)` Sets the intensity of all the tones in a tone burst and in a particular channel to a specified level in dB SPL. The true output level of a component tone is equal to the intensity specified by this function minus the specified attenuation for the particular tone by `exp-tone-atten` (refer to Figure 3-1).

inten (float) The intensity level in dB SPL.

interval (integer) The index of the interval to affect. Setting the index to 0 causes all the interval in the trial to be affected.

chan (symbol or string) The channel to affect. This must be either 'left, 'right, or 'both, or equivalently, "left", "right", or "both".

Example: `(exp-tone -inten 60 1 'both)`

`(EXP-TONE-ATTEN atten interval tone)` Sets the attenuation of a component tone in a complex tone burst to a specified level in dB. The attenuation is applied to the left and right channels. The true output level of the component tone is equal to the intensity specified by `exp-tone-inten` minus the attenuation specified by this function (refer to Figure 3-1).

atten (float) The attenuation level in dB.

interval (integer) The index of the interval to affect. Setting the index to 0 causes all the interval in the trial to be affected.

tone (integer) The index – beginning with 1 – of the tone in the interval to affect.

Example: `(exp-tone-atten 10 2 1)`

Noise Characteristics

`(EXP-FILTER-NOISE filtertype . freqs)` Produces filtered Gaussian white noise stimuli.

filtertype (symbol or string) The type of filter to use. This must be either 'lowpass,

'highpass, 'bandpass, or 'bandstop, or equivalently, "lowpass", "highpass", "bandpass", or "bandstop".
freqs (float) The cutoff frequencies in Hertz.
Example: `(exp-noise-filter 'lowpass 5000)`

`(EXP-NOISE-INTEN inten interval chan)` Sets the intensity of the noise bursts in a particular channel to a specified level in dB SPL.
inten (float) The intensity level in dB SPL.
interval (integer) The index of the interval to affect. Setting the index to 0 causes all the interval in the trial to be affected.
chan (symbol or string) The channel to affect. This must be either 'left, 'right, or 'both, or equivalently, "left", "right", or "both".
Example: `(exp-noise-inten 50 1 'both)`

WAV-file Stimuli

`(EXP-WAV-LOAD file)` Loads waveform samples from a wave file.
file (string) The path and name of the wave file to load.
Example: `(exp-wav-load "c:/demo.wav")`

Timing

`(EXP-IP time interval)` Sets the length of the pretime of the tone burst envelope in a specified period (refer to Figure 1-2).
time (float) The length of time in milliseconds.
interval (integer) The index of the interval to affect. Setting the index to 0 causes all the interval in the trial to be affected.
Example: `(exp-ip 100.0 0)`

`(EXP-RN time interval)` Sets the length of the risetime of the noise burst envelope in a specified interval (refer to Figure 1-2).
time (float) The length of time in milliseconds.
interval (integer) The index of the interval to affect. Setting the index to 0 causes all the interval in the trial to be affected.
Example: `(exp-rn 25.0 0)`

`(EXP-DS time interval)` Sets the length of the difference in the starting time of the tone burst and noise burst envelopes in a specified interval (refer to Figure 1-2).
time (float) The length of time in milliseconds.
interval (integer) The index of the interval to affect. Setting the index to 0 causes all the interval in the trial to be affected.
Example: `(exp-ds 10.0 0)`

`(EXP-RT time interval)` Sets the length of the risetime of the tone burst envelope in a specified interval (refer to Figure 1-2).
time (float) The length of time in milliseconds.

interval (integer) The index of the interval to affect. Setting the index to 0 causes all the interval in the trial to be affected.

Example: `(exp-rt 50.0 0)`

`(EXP-OT time interval)` Sets the length of the ontime of the tone burst envelope in a specified interval (refer to Figure 1-2).

time (float) The length of time in milliseconds.

interval (integer) The index of the interval to affect. Setting the index to 0 causes all the interval in the trial to be affected.

Example: `(exp-ot 500.0 0)`

`(EXP-FT time interval)` Sets the length of the falltime of the tone burst envelope in a specified interval (refer to Figure 1-2).

time (float) The length of time in milliseconds.

interval (integer) The index of the interval to affect. Setting the index to 0 causes all the interval in the trial to be affected.

Example: `(exp-ft 50.0 0)`

`(EXP-FN time interval)` Sets the length of the falltime of the noise burst envelope in a specified interval (refer to Figure 1-2).

time (float) The length of time in milliseconds.

interval (integer) The index of the interval to affect. Setting the index to 0 causes all the interval in the trial to be affected.

Example: `(exp-fn 25.0 0)`

`(EXP-DF time interval)` Sets the length of the difference in the finishing time of the burst and noise burst envelopes in a specified interval (refer to Figure 1-2).

time (float) The length of time in milliseconds.

interval (integer) The index of the interval to affect. Setting the index to 0 causes all the interval in the trial to be affected.

Example: `(exp-df 10.0 0)`

Channel Gains

`(EXP-OUTPUT-GAIN gain chan)` Sets the gain applied to the left and right digital output channels before they are converted to voltages at the analog outputs of the LynxOne card.

gain (float) The amount of gain to apply to the specified output channel in dB SPL.

chan (symbol or string) The channel to affect. This must be either 'left, 'right, or 'both, or equivalently, "left", "right", or "both".

Example: `(exp-output-gain 5.0 'left)`

Appendix B
Experiment Messages

Runexp user procedure	Meta command string
exp-interval *ints*	nints *ints*
exp-tones *itones*	ntones *itones*
exp-tone-inten *inten interval channel*	left_scales tone *scale* right_scales tone *scale*
exp-noise-inten *inten interval channel*	left_scales noise *scale* right_scales noise *scale*
exp-tone-atten *atten interval tone*	ft_car_atts *interval atten_1 ... atten_maxtones*
exp-mod-depth *depth interval tone*	ft_mod_depths *interval depth_1 ... depth_maxtones*
exp-freq *freq interval <num_tone> <car-or-mod>*	ft_car_freqs/ft_mod_freqs *interval freq_1 ... freq_maxtones*
exp-phase *phase interval <num_tone> <car-or-mod>*	ft_car_phases/ft_mod_phases *interval phase_1 ... phase_maxtones*
exp-car-phase *phase interval tone*	ft_car_phases *interval phase_1 ... phase_maxtones*
exp-mod-phase *phase interval tone*	ft_mod_phases *interval phase_1 ... phase_maxtones*
exp-ip time interval exp-ip-i samples interval	ip_list *sample_1 ... sample_maxintvls*
exp-ip exp-rn exp-ds exp-rt exp-ot exp-fn exp-df exp-ft	ip_list rn_list ...
exp-output-gain *gain channel*	output_scales
exp-reset-models exp-send-changes exp-go	reset_state ... set_parameters go
exp-sample-rate	sample_rate
exp-noise-filter	filter *filtertype cutoff_freqs*

Table B-1: Runexp user interface procedures, parameters, and their associated commands in `Meta`.

Appendix C
Sample Experiments

Listed below are four experiment protocols for threshold detection that are taken from 6.182 Laboratory Assignment 1. The first experiment, `test.spd`, implements a method of limits protocol. The second experiment, `updown.spd`, uses the adaptive Levitt up-down procedure in a one-interval two-alternative forced-choice experiment. The third experiment, conv.spd, uses the method of constant stimuli for five fixed tone levels. The final experiment, fixtest.spd, implements the method of constant stimuli for a single tone level. The definition and utility files for these four experiments – `defaults.spd`, `vardef.spd`, `tools.spd`, and `defin.spd` – are also provided.

Test.spd

```
; The purpose of this procedure is to quickly obtain a gross estimate
; of a subject's threshold (in dB SPL) for a 1000 Hz tone.

(load "rxp03.spd")
(load "/durin/rcheng/lab0/defaults.spd")

(cond ((not exp-connection) (exp-connect "")))
(exp-reset-models)

(define run 0)
(define intstep 0)
(define step 0.0)
(define InitHigh 85.0)   ; starting high test level in dB SPL
(define InitLow 0.0)     ; starting low test level in dB SPL
(define thresh 0)
(define noisespl 0)
(define freq 0)
(define frequency 0.0)
(define noiselevel 0.0)
(set! ext_atten 0.0)

; Setup for one interval of tone at 1000 Hz for 500ms after 100ms delay
(define (setup)
        (exp-intervals 1)
        (exp-freq frequency 1)
        (exp-ip 100.0 1)
        (exp-ot 500.0 1)
        (exp-noise-inten noiselevel 0 "both"))

; Finds threshold estimate by averaging the "up" and "down" thresholds
(define (end-exp)
        (set! thresh (/ (+ InitLow InitHigh) 2))
        (print (strcat (strcat "Your descending threshold is "
                               (format "%f" InitHigh)) " dB SPL"))
        (print (strcat (strcat "Your ascending threshold is "
                               (format "%f" InitLow)) " dB SPL"))
        (print (strcat (strcat "Your threshold at 1000 Hz is "
                               (format "%f" thresh)) " dB SPL")))

; Decides what level tone to play (if the subject has responded "No" on ; the first run,
; the second run begins else the stimulus level is decreased by a randomly chosen
; increment between 1 and 5 dB SPL. "Yes" on the second run ends experiment, else the
; stimulus level is increased by a randomly chosen increment between 1 and 5 dB SPL.)
(define (set-stimulus)
        (set! intstep (urand 1 6))
        (set! step (+ 0.0 intstep))
        (cond ((= run 0) (exp-tone-inten inithigh 0 "both") (set! run 1))
              ((= run 1) (cond ((equal? exp-response "Y")
                                (set! inithigh (- inithigh step))
```

```
                                    (exp-tone-inten inithigh 0 "both"))
                                   ((equal? response "N")
                                     (set! run 2)
                                     (set! initlow (- inithigh 10.0))
                                     (exp-tone-inten initlow 0 "both"))))
              ((= run 2) (cond ((equal? exp-response "Y")
                                     (set! run 3))
                                   ((equal? exp-response "N")
                                     (set! initlow (+ initlow step))
                                     (exp-tone-inten initlow 0 "both"))))))

; Main program
(define (test-exp noisespl freq)
        (define (loop)
                (set-stimulus)
                (print ext_atten)
                (printf "\n\n")
                (print left_tones)
                (printf "\n\n")
                (print right_tones)
                (printf "\n\n")
                (print left_noise)
                (printf "\n\n")
                (print right_noise)
                (printf "\n\n")
                (exp-send-changes)
                (exp-go)  ;play stimulus
                (exp-get-response "lab1_test")
                (if (< run 3) (loop) (end-exp)))
        (set! frequency (+ freq 0.0))
        (set! noiselevel (+ noisespl 0.0))
        (print noiselevel)
        (print frequency)
        (setup)
        (loop))
```

Updown.spd

```
; The purpose of this program is to use the Levitt up-down method to
; find a subject's absolute threshold.

(load "rxp03.spd")
(load "pdefaults.spd")
(load "vardef.spd")
(load "tools.spd")

(cond ((not exp-connection) (exp-connect "")))

; Intervals one and three contain warning tones of 50 ms duration,
; intervals two and four contain test tones of 500 ms duration
(define (setup)
        (exp-intervals 4)
        (exp-freq frequency 0)
         ;exp-ip for burst #1 is a random delay (see set-stimulus)
        (exp-ip  50.0 2)
        (exp-ip 250.0 3)
        (exp-ip  50.0 4)
        (exp-rt  10.0 0)
        (exp-ft  10.0 0)
        (exp-ot  20.0 1)
        (exp-ot 500.0 2)
        (exp-ot  20.0 3)
        (exp-ot 500.0 4)
        (exp-tone-inten warnlevel 1 "both")
        (exp-tone-inten warnlevel 3 "both")
        (exp-noise-inten 0.0 1 "both")
        (exp-noise-inten noiselevel 1 "both")
        (exp-noise-inten noiselevel 2 "both")
        (exp-noise-inten noiselevel 3 "both")
        (exp-noise-inten noiselevel 4 "both"))

; sets intensity of tone
(define (set-stimulus)
        (set! interval (urand 1 3))
        (set! delay (+ 0.0 (urand 250 500)))
        (exp-ip delay 1)
        (cond ((= interval 1) (exp-tone-inten tonelevel 2 "both")
                              (exp-tone-inten -200.0 4 "both"))
              ((= interval 2)(exp-tone-inten -200.0 2 "both")
                              (exp-tone-inten tonelevel 4 "both"))))

(define (get-next-level)
        (set! trial (+ trial 1.0))
        (set! step (/ step trial))
        (if (< step 1.0) (set! step 1.0) ())
        (if (= dir up) (set! tonelevel (+ tonelevel step))
                       (set! tonelevel (- tonelevel step)))
        (printf "\nstep is %f" step)
        (printf "\ntrial is %f" trial)
        (printf "\ntonelevel is %f" tonelevel))

(define (give-feedback)
  (cond ((= interval 1)
                (cond ((equal? response "1")
                              (printf "Correct--first\n")
                              (set! correct? true))
                      ((equal? response "2"))
                              (printf "Incorrect--first\n")
                              (set! correct? false))))
        ((= interval 2)
                (cond ((equal? response "1")
                              (printf "Incorrect--second\n")
                              (set! correct? false))
                      ((equal? response "2")
                              (printf "Correct--second\n")
                              (set! correct? true))))))
```

```
;if a turning point was reached, it is saved in a list
;and added to the cumulative total
(define (save-turnpoint)
        (if (= dir up) (set! dir down) (set! dir up))
        (set! run (+ run 1))
        (set! data-list (cons tonelevel data-list))
        (get-next-level))

(define (output)
        (printf (strcat (strcat "Your threshold is "
                (format "%3f" (compute-threshold data-list 3)))
                                                    " dB SPL\n")))

;save data in file determined in updown.com
(define (save-data dfile name date)
        (define datafile (fopen dfile "w"))
        (fputs "THRESHOLD DETECTION RESULTS: ADAPTIVE PROCEDURE (UP-DOWN)\n\n"
                        datafile)
        (fputs (strcat "\nName is " name) datafile)
        (fputs (strcat "\nDate is " date) datafile)
        (fputs "\n\nThe turnaround points are the following\n" datafile)
        (fputs (list-to-string data-list) datafile)
        (fputs "\n\n(first 2 turnaround points discarded to compute threshold)"
                datafile)
        (fputs (strcat "\nYour threshold is "
                (format "%3f" (compute-threshold data-list 3))) datafile)
        (fputs (strcat "\nNumber of trials is "
                (format "%d" num)) datafile)
        (fputs (strcat "\nNumber of stimulus changes is "
                (format "%d" trials)) datafile)
        (fclose datafile))

(define (new-level? dir correct? second+)
        (OR (AND (AND (= second+ 1)(eq? correct? true))
                                        (= dir down))
                        (AND (= dir up) (eq? correct? false))))

(define (turn? dir correct? second+)
        (OR (AND (= dir down) (eq? correct? false))
                        (AND (AND (eq? correct? true)(= second+ 1))
                                                        (= dir up))))
;main program
(define (loop)
        (set! num (+ 1 num))
        (set-stimulus)
        (exp-send-changes)
        (exp-go)                ;play stimulus
        (exp-get-response "lab1_updown")
        (if (= feedback 1) (give-feedback) ())
        (cond (correct? (if (= second+ 0) (set! second+ 1) (set! second+ 0)))
                (t (set! second+ 6)))
        (cond ((new-level? dir correct? second+) (get-next-level))
                ((turn? dir correct? second+) (save-turnpoint)))
        (print data-list)
        (if (< run (+ turns 1)) (loop) (output)))

;remcommended initstep==20dB, recommended initlvl>40dB for normal
;hearing individuals in quiet
(define (updown-exp noisespl freq initlvl initstep fdback tpoints warn)
        (set! turns tpoints)
        (set! feedback fdback)
        (set! step (+ 0.0 initstep))
        (set! tonelevel (+ 0.0 initlvl))
        (if (= warn 1)(set! warnlevel (+ 0.0 initlvl))(set! warnlevel 0.0))
        (set! noiselevel (+ 0.0 noisespl))
        (set! frequency (+ 0.0 freq))
        (setup)
        (set! data-list '())
        (loop))
```

Conv.spd

```
;This program is designed to run a conventional experiment--that which has
;predetermined, fixed testing levels.
;
;Basic algorithm--> start with gross threshold estimation (GTE) and test at
; fixed intervals (say 5 dB SPL) with range centered at given GTE
;estimation.  The experiment is two alternative forced choice with the two
;alternatives being tone or not tone.  Roughly 50% of the stimuli will be tone
; (Stimuli: GTE+10dB, GTE+5dB, GTE+0dB. GTE-5dB, GTE-10dB), 50% will be no tone.
; Output will be saved in a two-by-six confusion matrix.
;The test consists of 10 runs of 50 trials.

(load "rxp03.spd")
(load "defaults.spd")
(load "defin.spd")

(set! ext_atten 0.0)

(cond ((not exp-connection) (exp-connect "")))

(define resp 0)
(define run 0)
(define num 0)
(define GTE 0)
(define stim1 0)
(define stim2 0)
(define stim3 0)
(define stim4 0)
(define stim5 0)
(define tt '(1 1 1 1 1 1))    ; testing, tone present
(define tn '(1 1 1 1 1 1))    ; testing, no tone
(define templist '(1 1 1 1 1 1))  ;temporary data structure for save-response
(define tone 0)
(define level 0)
(define noisespl 0)
(define freq 0)
(define frequency 0.0)
(define noiselevel 0.0)

;setup for one interval of tone at 1000Hz for 500ms after 500ms delay
(define (setup)
        (exp-intervals 1)
        (exp-freq frequency 1)
        (exp-ip 500.0 1)
        (exp-ot 500.0 1)
        (exp-noise-inten noiselevel 0 "both"))

(define (get-thresh)
        (KEY-WRITE ID "Please enter your threshold estimate: \n")
        (set! GTE (atoi (mget-answer)))
        (key-write id "\n")
        (set! stim1 (+ 10.0 GTE))
        (set! stim2 (+ 5.0 GTE))
        (set! stim3 (+ 0.0 GTE))
        (set! stim4 (- GTE 5.0))
        (set! stim5 (- GTE 10.0))
        (PRINT GTE)
        (PRINT STIM1)
        (PRINT STIM2)
        (PRINT STIM3)
        (PRINT STIM4)
        (PRINT STIM5))

(define (change-vars)
        (set! tone (urand 1 3))
        (set! level (urand 1 6)))

;set intensity of tone  left-- left attenuator
(define (set-stimulus)
```

```
        (cond ((= tone 1) (exp-tone-inten -200.0 0 "both"))
              ((= tone 2)
                 (cond ((= level 1) (exp-tone-inten stim1 0 "both"))
                       ((= level 2) (exp-tone-inten stim2 0 "both"))
                       ((= level 3) (exp-tone-inten stim3 0 "both"))
                       ((= level 4) (exp-tone-inten stim4 0 "both"))
                       ((= level 5)(exp-tone-inten stim5 0 "both"))))))

;asks the subject to hit a key when ready to receive stimulus
(define (wait-for-ready num trials)
        (KEY-WRITE ID (strcat (strcat (strcat (strcat
                       "You have completed " (format "%d" num))
                       " of ") (format "%d" trials)) " trials."))
        (KEY-WRITE ID "Press any key when ready...\n")
        (KEY-GET ID))

;play stimulus (calls lab:runexp.spd)
(define (play-stimulus) (exp-go) (printf "Stimulus given.\n"))

;obtain response to stimulus Y=yes tone, N=no tone
(define (get-response num)
        (KEY-WRITE ID (STRCAT (STRCAT (STRCAT "TRIAL #" (FORMAT "%d" num))
                         "  Was there a tone? ")  " V=YES, N=NO"))
        (set! response (KEY-GET ID))
        (cond ((OR (= response (ichar "v"))(= response (ichar "V")))
                           (PRINTF "Response received."))
               ((OR (= response (ichar "n"))(= response (ichar "N")))
                           (PRINTF "Response received."))
              (else
        (key-write ID "Response invalid. Please type V (yes) or N (no) now.")
                       (get-response num))))

;during practice-->give feedback
(define (give-feedback)
        (cond ((= tone 1)(cond ((OR (= response (ichar "n"))
                                    (= response (ichar "N")))
                                    (key-write ID "Correct--NO TONE\n"))
                               ((OR (= response (ichar "v"))
                                    (= response (ichar "V")))
                                    (key-write ID "Incorrect--NO TONE\n"))))
              ((= tone 2)(cond ((OR (= response (ichar "n"))
                                    (= response (ichar "N")))
                                    (key-write ID "Incorrect--TONE\n"))
                               ((OR (= response (ichar "v"))
                                    (= response (ichar "V")))
                                    (key-write ID "Correct--TONE\n"))))))

;entries in table : row=response columns=stimulus level
(define (table-reset)
        (set! templist '(0 0 0 0 0 0))
        (set! tt '(0 0 0 0 0 0))
        (set! tn '(0 0 0 0 0 0)))

;this procedure saves the response in a 2-by-6 confusion matrix
(define (save-response)
        (cond ((OR (= response (ichar "v"))(= response (ichar "V")))
                                                (set! templist tt))
              ((OR (= response (ichar "n"))(= response (ichar "N")))
                                                (set! templist tn)))
        (cond ((= tone 1) (set-sixth templist (+ 1 (sixth6 templist))))
              ((= tone 2)
                 (cond ((= level 1)(set-first templist (+ 1 (first1 templist))))
                   ((= level 2)(set-second templist (+ 1 (second2 templist))))
                   ((= level 3)(set-third templist (+ 1 (third3 templist))))
                   ((= level 4)(set-fourth templist (+ 1 (fourth4 templist))))
                   ((= level 5)(set-fifth templist (+ 1 (fifth5 templist))))))))

;data calculations--(in variables: t=tone, n=no tone, p=percent, c=correct
; tot=total)
(define pct 0)   ; percent correct tones
(define pcn 0)   ; percent correct notones
```

```
(define pctot 0) ; total percent correct
(define miss 0) ; miss = total tones - number tones detected
(define falsalrm 0) ; false alarm = total notones - number notones detected
(define hit 0) ; hit = number of tones detected
(define correject 0) ; correct rejection = number of notones detected
(define pclvl1 0)      ;percent correct at level 1
(define pclvl2 0)      ;percent correct at level 2
(define pclvl3 0)      ;percent correct at level 3
(define pclvl4 0)      ;percent correct at level 4
(define pclvl5 0)      ;percent correct at level 5
(define totn 1)               ;total notones
(define tott 1)               ;total tones
(define total 1)       ;total stimuli--tones + notones

(define (percents tt tn)
       (set! correject (sixth6 tn))
       (set! totn (+ (sixth6 tt) (sixth6 tn)))
       (set! hit (+ (first1 tt) (second2 tt) (third3 tt) (fourth4 tt)
              (fifth5 tt)))
       (set! tott (+ hit (+ (first1 tn) (second2 tn) (third3 tn)
              (fourth4 tn) (fifth5 tn))))
       (set! total (+ totn tott))
       (if (= tott 0) (set! pct 0)
       (set! pct (* (/ (float hit) (float tott)) 100)))
       (if (= totn 0) (set! pcn 0)
       (set! pcn (* (/ (float correject) (float totn)) 100)))
       (if (= total 0) (set! pctot 0)
       (set! pctot (* (/ (+ (float hit) (float correject))(float total)) 100)))
       (set! miss (- tott hit))
       (set! falsalrm (- totn correject)))

(define (save-data dfile)
       (define datafile (fopen dfile "w"))
       (fputs "     SIGNAL DETECTION RESULTS: CONVENTIONAL PROCEDURE\n\n"
              datafile)
       (fputs (strcat (strcat (strcat (strcat (strcat (strcat " "
              (format "%3d" (round stim1)))
              (strcat " " (format "%3d" (round stim2))))
              (strcat " " (format "%3d" (round stim3))))
              (strcat " " (format "%3d" (round stim4))))
              (strcat " " (format "%3d" (round stim5)))) "  NONE\n") datafile)
       (fputs (list-to-string tt) datafile)
       (fputs "       Response: TONE\n" datafile)
       (fputs (list-to-string tn) datafile)
       (fputs "       Response: NONE\n" datafile)
       (fputs "\n" datafile)
       (fputs (strcat "Number of Hits: " (format "%3d\n" hit)) datafile)
       (fputs (strcat "Number of Misses: " (format "%3d\n" miss)) datafile)
       (fputs (strcat "Number of Correct Rejections: "
                      (format "%3d\n" correject)) datafile)
       (fputs (strcat "Number of False Alarms: "
                      (format "%3d\n" falsalrm)) datafile)
       (fputs (strcat "Percent of tones detected: " (format "%3f\n" pct))
                                                    datafile)
       (fputs (strcat "Percent of tone-absence detected: "
                           (format "%3f\n" pcn)) datafile)
       (fputs "\n\n" datafile)
    (if (< pclvl1 75)
  (fputs (strcat (format "%3f" stim1) " dB SPL is below threshold\n") datafile)
  (fputs (strcat (format "%3f" stim1) " dB SPL is above threshold\n") datafile))
    (if (< pclvl2 75)
  (fputs (strcat (format "%3f" stim2) " dB SPL is below threshold\n") datafile)
  (fputs (strcat (format "%3f" stim2) " dB SPL is above threshold\n") datafile))
    (if (< pclvl3 75)
  (fputs (strcat (format "%3f" stim3) " dB SPL is below threshold\n") datafile)
  (fputs (strcat (format "%3f" stim3) " dB SPL is above threshold\n") datafile))
    (if (< pclvl4 75)
  (fputs (strcat (format "%3f" stim4) " dB SPL is below threshold\n") datafile)
  (fputs (strcat (format "%3f" stim4) " dB SPL is above threshold\n") datafile))
    (if (< pclvl5 75)
  (fputs (strcat (format "%3f" stim5) " dB SPL is below threshold\n") datafile)
```

```
  (fputs (strcat (format "%3f" stim5) " dB SPL is above threshold\n") datafile))
       (fclose datafile))


(define (print-data)
       (printf "     SIGNAL DETECTION RESULTS: CONVENTIONAL PROCEDURE\n\n"
                )
       (printf (strcat (strcat (strcat (strcat (strcat (strcat " "
              (format "%3d" (round stim1)))
              (strcat " " (format "%3d" (round stim2))))
              (strcat " " (format "%3d" (round stim3))))
              (strcat " " (format "%3d" (round stim4))))
              (strcat " " (format "%3d" (round stim5)))) "  NONE\n") )
       (printf (list-to-string tt) )
       (printf "      Response: TONE\n" )
       (printf (list-to-string tn) )
       (printf "      Response: NONE\n" )
       (printf "\n" )
       (printf (strcat "Number of Hits: " (format "%3d\n" hit)) )
       (printf (strcat "Number of Misses: " (format "%3d\n" miss)) )
       (printf (strcat "Number of Correct Rejections: "
                            (format "%3d\n" correject)) )
       (printf (strcat "Number of False Alarms: "
                            (format "%3d\n" falsalrm)) )
       (printf (strcat "Percent of tones detected: " (format "%3f\n" pct))
                                                      )
       (printf (strcat "Percent of tone-absence detected: "
                                     (format "%3f\n" pcn)) )
       (printf "\n\n" )
    (if (< pclvl1 75)
  (printf (strcat (format "%3f" stim1) " dB SPL is below threshold\n") )
  (printf (strcat (format "%3f" stim1) " dB SPL is above threshold\n") ))
    (if (< pclvl2 75)
  (printf (strcat (format "%3f" stim2) " dB SPL is below threshold\n") )
  (printf (strcat (format "%3f" stim2) " dB SPL is above threshold\n") ))
    (if (< pclvl3 75)
  (printf (strcat (format "%3f" stim3) " dB SPL is below threshold\n") )
  (printf (strcat (format "%3f" stim3) " dB SPL is above threshold\n") ))
    (if (< pclvl4 75)
  (printf (strcat (format "%3f" stim4) " dB SPL is below threshold\n") )
  (printf (strcat (format "%3f" stim4) " dB SPL is above threshold\n") ))
    (if (< pclvl5 75)
  (printf (strcat (format "%3f" stim5) " dB SPL is below threshold\n") )
  (printf (strcat (format "%3f" stim5) " dB SPL is above threshold\n") ))
       )


;This procedure computes percentages and approximates threshold and
;outputs the results.  Threshold is approximated at 75% correct. 50% correct
;is chance (two possibilities--tone or no tone) so 50% is the baseline.
;The absolute threshold is defined as the point at which a stimulus elicits
; a psychological response 50% of the time.  Finally, 50% above the
; 50% baseline is 75% correct.
(define (compute-threshold)
       (if (AND (= (first1 tt) 0) (= (first1 tn) 0)) (set! pclvl1 0)
       (set! pclvl1 (* (/ (float (first1 tt))
                          (float (+ (first1 tt) (first1 tn)))) 100)))
       (if (AND (= (second2 tt) 0) (= (second2 tn) 0)) (set! pclvl2 0)
       (set! pclvl2 (* (/ (float (second2 tt))
                          (float (+ (second2 tt) (second2 tn)))) 100)))
       (if (AND (= (third3 tt) 0) (= (third3 tn) 0)) (set! pclvl3 0)
       (set! pclvl3 (* (/ (float (third3 tt))
                          (float (+ (third3 tt) (third3 tn)))) 100)))
       (if (AND (= (fourth4 tt) 0) (= (fourth4 tn) 0)) (set! pclvl4 0)
       (set! pclvl4 (* (/ (float (fourth4 tt))
                          (float (+ (fourth4 tt) (fourth4 tn)))) 100)))
       (if (AND (= (fifth5 tt) 0) (= (fifth5 tn) 0)) (set! pclvl5 0)
       (set! pclvl5 (* (/ (float (fifth5 tt))
                          (float (+ (fifth5 tt) (fifth5 tn)))) 100))))

(define (give-results)
```

```
    (if (< pclvl1 75)
  (key-write id (strcat (format "%3f" stim1) "dB SPL is below threshold\n"))
  (key-write ID (strcat (format "%3f" stim1) "dB SPL is above threshold\n")))
    (if (< pclvl2 75)
  (key-write id (strcat (format "%3f" stim2) "dB SPL is below threshold\n"))
  (key-write ID (strcat (format "%3f" stim2) "dB SPL is above threshold\n")))
    (if (< pclvl3 75)
  (key-write id (strcat (format "%3f" stim3) "dB SPL is below threshold\n"))
  (key-write ID (strcat (format "%3f" stim3) "dB SPL is above threshold\n")))
    (if (< pclvl4 75)
  (key-write id (strcat (format "%3f" stim4) "dB SPL is below threshold\n"))
  (key-write ID (strcat (format "%3f" stim4) "dB SPL is above threshold\n")))
    (if (< pclvl5 75)
  (key-write id (strcat (format "%3f" stim5) "dB SPL is below threshold\n"))
  (key-write ID (strcat (format "%3f" stim5) "dB SPL is above threshold\n"))))

(define (data-analysis)
       (percents tt tn)
       (compute-threshold)
       (give-results))

;main program
(define (conv-exp trials noisespl freq)
       (define (loop num)
              (change-vars)
              (setup)
              (set-stimulus)
              (print ext_atten)
              (printf "\n\n")
              (print left_tones)
              (printf "\n\n")
              (print right_tones)
              (printf "\n\n")
              (print left_noise)
              (printf "\n\n")
              (print right_noise)
              (printf "\n\n")
              (exp-send-changes)
       ;      (wait-for-ready num trials)
              (play-stimulus)
              (get-response num)
              (give-feedback)
              (save-response)
              (if (< num (- trials 1)) (loop (+ 1 num))
                     (KEY-WRITE ID (strcat (strcat "You have completed "
                                   (format "%d" (+ 1 num))) " trials."))))
       (table-reset)
       (set! frequency (+ 0.0 freq))
       (set! noiselevel (+ 0.0 noisespl))
       (setup)
       (get-thresh)
       (loop 0)
       (data-analysis))

;(conv-exp trials noisespl freq)
```

Fixtest.spd

```
;This program is designed to run a single fixed level test.
;
;Basic algorithm--> tests threshold determined in previous experiment

; Booth        Machine
; front        arsenic
; back         rear
; mid          oin
; white        lear
; brown        oldlace

;(pcdef 'front)
;(pcdef 'back)
;(pcdef 'white)
;(pcdef 'brown)

;setup subject's terminal
(define ID 1)
(define termname "RESP")
(define termtype "V")
(key-setup ID termname termtype)

;loading necessary code
;(key-write id "Loading...\n")
(printf "Loading...\n")
;(load "lab0:defaults.spd")
(set! ext_atten 0.0)
;(load "lab0:runexp.spd")
;(load "lab0:defin.spd")

(load "/durin/rcheng/lab0/defaults.spd")
(load "/durin/rcheng/runexp/runexp99f.spd")
(load "/durin/rcheng/lab0/defin.spd")

;(cond ((not exp-connection) (exp-connect "arsenic")))
;(cond ((not exp-connection) (exp-connect "rear")))
;(cond ((not exp-connection) (exp-connect "oin")))
;(cond ((not exp-connection) (exp-connect "lear")))
;(cond ((not exp-connection) (exp-connect "oldlace")))

;defaults
(define resp 0)
(define run 0)
(define num 0)
(define tone 0)
(define level 0.0)
(define tresp '(0 0))
(define ntresp '(0 0))
(define templist '(0 0))
(define noiselevel 0.0)
(define noisespl 0)
(define freq 0)
(define frequency 0.0)
(define thresh 0)
(define example? 0)
(define examp-tone 0)
(define true t)
(define false nil)

;setup for one interval of tone at 1000Hz for 500ms after 500ms delay
(define (setup)
        (exp-intervals 1)
        (exp-freq frequency 1)
        (exp-ip 500.0 1)
        (exp-ot 500.0 1)
        (exp-noise-inten noiselevel 0 "both"))

;set intensity of tone
```

```
(define (set-stimulus)
        (if example? (set! tone examp-tone) (set! tone (urand 1 3)))
        (cond ((= tone 1) (exp-tone-inten -200.0 0 "both"))
              ((= tone 2) (exp-tone-inten level 0 "both"))))

;asks the subject to hit a key when ready to receive stimulus
(define (wait-for-ready num trials)
        (KEY-WRITE ID (strcat (strcat (strcat (strcat
                "You have completed " (format "%d" num))
                 " of ") (format "%d" trials)) " trials."))
        (KEY-WRITE ID "Press any key when ready...\n")
        (KEY-GET ID))
;asks the subject to hit a key when ready to receive stimulus
(define (wait-for-ready2)
        (KEY-WRITE ID "Press any key when ready...\n")
        (KEY-GET ID))

;play stimulus (calls lab:runexp.spd)
(define (play-stimulus) (exp-go) (printf "Stimulus given.\n"))

;obtain response to stimulus V=yes tone, N=no tone
(define (get-response num)
        (KEY-WRITE ID (strcat (strcat "Trial #" (format "%d" num))
                        "  Was there a tone?   V=YES N=NO"))
        (set! response (KEY-GET ID))
        (cond ((OR (= response (ichar "v"))(= response (ichar "V")))
                                (printf "Response received."))
                ((OR (= response (ichar "n"))(= response (ichar "N")))
                                (printf "Response received."))
              (else (key-write ID "Response invalid. Please type V or N now.")
                        (get-response num))))

;during practice-->give feedback
(define (give-feedback)
        (cond ((= tone 1)(cond ((OR (= response (ichar "n"))
                                    (= response (ichar "N")))
                                    (key-write ID "Correct--NO TONE\n"))
                               ((OR (= response (ichar "v"))
                                    (= response (ichar "V")))
                                    (key-write ID "Incorrect--NO TONE\n"))))
              ((= tone 2)(cond ((OR (= response (ichar "n"))
                                    (= response (ichar "N")))
                                    (key-write ID "Incorrect--TONE\n"))
                               ((OR (= response (ichar "v"))
                                    (= response (ichar "V")))
                                    (key-write ID "Correct--TONE\n"))))))

;this procedure saves the response in a 2-by-2 confusion matrix
(define (save-response)
        (cond ((OR (= response (ichar "v"))(= response (ichar "V")))
                 (cond ((= tone 1) (set-car! (cdr tresp) (+ 1 (cadr tresp))))
                       ((= tone 2) (set-car! tresp (+ 1 (car tresp))))))
              ((OR (= response (ichar "n"))(= response (ichar "N")))
                (cond ((= tone 1) (set-car! (cdr ntresp) (+ 1 (cadr ntresp))))
                       ((= tone 2) (set-car! ntresp (+ 1 (car ntresp))))))))

;data calculations--(in variables: t=tone, n=no tone, p=percent, c=correct
; tot=total)
(define pct 0)  ; percent correct tones
(define pcn 0)  ; percent correct notones
(define pctot 0) ; total percent correct
(define miss 0) ; miss = total tones - number tones detected
(define falsalrm 0) ; false alarm = total notones - number notones detected
(define hit 0) ; hit = number of tones detected
(define correject 0) ; correct rejection = number of notones detected
(define totn 1)               ;total notones
(define tott 1)               ;total tones
(define total 1)       ;total stimuli--tones + notones

(define (percents tresp ntresp)
        (set! correject (cadr ntresp))
```

```
        (set! totn (+ (cadr tresp) (cadr ntresp)))
        (set! hit (car tresp))
        (set! tott (+ (car tresp) (car ntresp)))
        (set! total (+ totn tott))
        (if (= tott 0) (set! pct 0)
        (set! pct (* (/ (float hit) (float tott)) 100)))
        (if (= totn 0) (set! pcn 0)
        (set! pcn (* (/ (float correject) (float totn)) 100)))
        (if (= total 0) (set! pctot 0)
        (set! pctot (* (/ (+ (float hit) (float correject))(float total)) 100)))
        (set! miss (- tott hit))
        (set! falsalrm (- totn correject)))

(define (save-data dfile)
        (define datafile (fopen dfile "w"))
        (fputs "     SIGNAL DETECTION RESULTS: SINGLE FIXED LEVEL\n\n"
                 datafile)
        (fputs (strcat (strcat "The level tested was " (format "%f" level))
                " dB SPL\n\n") datafile)
        (fputs "                STIMULI\n" datafile)
        (fputs "               TONE    NO TONE\n" datafile)
        (fputs (strcat (strcat (strcat (strcat
               "   Response=tone:" (format "%d" (car tresp))) "     ")
               (format "%d" (cadr tresp))) "\n") datafile)
        (fputs (strcat (strcat (strcat (strcat
               "Response=no tone:" (format "%d" (car ntresp))) "    ")
               (format "%d" (cadr ntresp))) "\n") datafile)
        (fputs "\n" datafile)
        (fputs (strcat "Number of Hits: " (format "%3d\n" hit)) datafile)
        (fputs (strcat "Number of Misses: " (format "%3d\n" miss)) datafile)
        (fputs (strcat "Number of Correct Rejections: "
                       (format "%3d\n" correject)) datafile)
        (fputs (strcat "Number of False Alarms: "
                       (format "%3d\n" falsalrm)) datafile)
        (fputs (strcat "Percent of tones detected: " (format "%3f\n" pct))
                                                          datafile)
        (fputs (strcat "Percent of tone-absence detected: "
                                 (format "%3f\n" pcn)) datafile)
        (fputs "\n\n" datafile)
        (fclose datafile))

(define (print-data)
        (printf "     SIGNAL DETECTION RESULTS: SINGLE FIXED LEVEL\n\n"
                 )
        (printf (strcat (strcat "The level tested was " (format "%f" level))
                " dB SPL\n\n") )
        (printf "                STIMULI\n" )
        (printf "               TONE    NO TONE\n" )
        (printf (strcat (strcat (strcat (strcat
               "   Response=tone:" (format "%d" (car tresp))) "     ")
               (format "%d" (cadr tresp))) "\n") )
        (printf (strcat (strcat (strcat (strcat
               "Response=no tone:" (format "%d" (car ntresp))) "    ")
               (format "%d" (cadr ntresp))) "\n") )
        (printf "\n" )
        (printf (strcat "Number of Hits: " (format "%3d\n" hit)) )
        (printf (strcat "Number of Misses: " (format "%3d\n" miss)) )
        (printf (strcat "Number of Correct Rejections: "
                       (format "%3d\n" correject)) )
        (printf (strcat "Number of False Alarms: "
                       (format "%3d\n" falsalrm)) )
        (printf (strcat "Percent of tones detected: " (format "%3f\n" pct))
                                                          )
        (printf (strcat "Percent of tone-absence detected: "
                                 (format "%3f\n" pcn)) )
        (printf "\n\n" )
       )

;gives an example of each condition before formal experimentation to
;familiarize the subject with the stimuli
(define (example)
```

```
        (define ex 0)
        (set! example? true)
        (set! examp-tone 2)
        (set-stimulus)
        (exp-send-changes)
        (key-write ID "Here is an example of the condition to be tested.")
        (key-write id "This interval has a tone with masker noise:\n")
        (wait-for-ready2)
        (exp-go)
        (set! examp-tone 1)
        (set-stimulus)
        (exp-send-changes)
        (key-write id "This interval has no tone with masker noise:\n")
        (wait-for-ready2)
        (exp-go)
        (key-write id "Would you like to hear the example again?")
        (key-write id "(Y=yes)")
        (set! ex (key-get id))
        (if (OR (= ex (ichar "y"))(= ex (ichar "Y"))) (example)
             (key-write id "You are about to begin the formal experiment.\n"))
        (set! example? false))

;main program
(define (fixtest-exp trials thresh noisespl freq)
        (define (loop num)
                (set-stimulus)
                (print ext_atten)
                (printf "\n\n")
                (print left_tones)
                (printf "\n\n")
                (print right_tones)
                (printf "\n\n")
                (print left_noise)
                (printf "\n\n")
                (print right_noise)
                (printf "\n\n")
                (exp-send-changes)
        ;       (wait-for-ready num trials)
                (play-stimulus)
                (get-response (+ num 1))
                (give-feedback)
                (save-response)
                (print tresp)
                (print ntresp)
                (if (< num (- trials 1)) (loop (+ 1 num))
                        (KEY-WRITE ID (strcat (strcat "You have completed "
                                (format "%d" (+ 1 num))) " trials."))))
        (set! level (+ 0.0 thresh))
        (set! noiselevel (+ 0.0 noisespl))
        (set! frequency (+ 0.0 freq))
        (printf "The signal level is %f\n" level)
        (printf "The noise level is %f\n" noiselevel)
        (setup)
        (example)
        (loop 0)
        (print tresp)
        (print ntresp)
        (percents tresp ntresp))

;(fixtest-exp trials thresh noisespl freq)
```

Defaults.spd

```
; Default definitions for use with the "adapt.spd" and "rxp03.spd"
; sets of adaptive psychoacoustic experimentation routines.

(define USE_FEEDBACK nil)

(define EXT_ATTEN 0.0)                ;external attenuation applied to
                                              ;the output
(define DEBUGGING? nil)                       ;flag for debugging information
(define MINSTEP 1.0)                          ;a step this small ends the
                                              ;adaptive run
(define MAXSTEP 5.0)                  ;steps are limited to this many dB
(define BASE_STEP -2.0)                       ;size and direction of first step
                                              ;to take

; The following two defines establish the SPL levels produced by the
; PC's Runexp program operating as loud as it can go (i.e. tone and
; noise outputs are multiplied by a scale factor of 1.0).
(define NOISE_DEFAULT 98.0)
(define TONE_DEFAULT 105.0)
(define STOP_FLAG nil)
(define tot_trials 0)                 ;total number of trials in run
(define tot_correct 0)                ;total number of correct trials
(define trials 0)                             ;number of trials at partic. level
(define correct 0)                    ;number of correct at partic. level
(define steplist nil)                 ;list of steps used in the past
(define stepdir '*unassigned*)        ;last direction stepped in
(define stepsize '*unassigned*)       ;last step size used
(define snr '*unassigned*)            ;signal-to-noise ratio used
(define interval '*unassigned*)

; Default filepath to send data files to
(define filepath "[]")
(define filename "output.dat")
(define output_file '*unassigned*)

(define simulation '*unassigned*)
(define subject '*unassigned*)
(define date '*unassigned*)

(define (write-defaults output_file)
  (fputs (format "EXT_ATTEN=%f\n" EXT_ATTEN) output_file)
  (fputs (format "MINSTEP=%f\n" MINSTEP) output_file)
  (fputs (format "MAXSTEP=%f\n" MAXSTEP) output_file)
  (fputs (format "BASE_STEP=%f\n" BASE_STEP) output_file)
  (fputs (format "NOISE_DEFAULT=%f\n" NOISE_DEFAULT) output_file)
  (fputs (format "TONE_DEFAULT=%f\n" TONE_DEFAULT) output_file)
  (cond ((not (eq? simulation '*unassigned*))
         (fputs (format "SIMULATION: %s\n" simulation) output_file)))
  (cond ((not (eq? date '*unassigned*))
         (fputs (format "DATE: %s\n" date) output_file)))
  (cond ((not (eq? subject '*unassigned*))
         (fputs (format "SUBJECT: %s\n" subject) output_file))))
```

Vardef.spd

```
; default variables for updown algorithms

(define frequency 0.0)
(define warnlevel 0.0)
(define noiselevel 0.0)
(define interval 0)
(define delay 0.0)
(define tonelevel 0.0)
(define trial 0)
(define step 1.0)

(define true t)
(define false nil)
(define up 1)
(define down 0)
(define dir 0)
(define run 1)
(define correct? 0)
(define second+ 6)
(define feedback 0)
(define fdback 0)
(define turns 0)
(define num 0)

(define data-list '(0))
```

Tools.spd

```
; asks the subject to hit a key when ready to receive stimulus
(define (key-wait-for-ready id)
        (key-write id "Press any key when ready...\n")
        (key-get id))

; convert the numerical representation of a character to upper case
(define (ic-upcase resp)
  (if (and (> resp 96) (< resp 123))
      (- resp 32)
      resp))

;is the end of the list reached?
(define (null? lst) (if (eq? lst nil) t nil))

; formats a list of characters into a string
(define (list-to-string lst)
        (if (not (list? (cdr lst)))
                (strcat (format "%f " (car lst)) "")
                (strcat (format "%f " (car lst)) (list-to-string (cdr lst)))))

; formats a list of integers to a string
(define (intlist-to-string lst)
        (if (not (list? (cdr lst)))
             (strcat (format "%3d " (car lst)) "")
             (strcat (format "%3d " (car lst)) (intlist-to-string (cdr lst)))))

;determines the length of a list
(define (length longlst)
        (define (iter-len lst len)
                (if (null? lst) len (iter-len (cdr lst) (+ len 1))))
        (iter-len longlst 0))


;compute the threshold of the data-list, dropping the first few turnaround
;points (determined by drop).  there's an extraneous zero at the end of the
;data-list so (compute-threshold data-list 3) computes the threshold after
;dropping the first two turnaround points
(define (compute-threshold data-list drop)
        (define total-len (length data-list))
        (define (avg lst sum len drp)
                (if (= (- total-len drp) len)
                        (if (= len 0) 0 (/ sum len))
                        (avg (cdr lst) (+ (car lst) sum) (+ len 1) drp)))
        (avg data-list 0 0 drop))


;converts lists of form (w (x (y z))) to (w x y z)
(define (clean lst)
        (if (not (list? lst)) (cons lst '())
                (cons (car lst) (clean (cadr lst)))))

(define (reverse-lst lst)
        (if (not (list? lst)) (cons lst '())
                (cons (reverse-lst (cdr lst)) (car lst))))
```

Defin.spd

```
(define (null? x)
  (if (eq? x nil) t nil))

(define (append x y)
  (if (null?  x)
    y
    (cons (car x) (append (cdr x) y))))

;adds all of the elements of the list
(define (add-all list)
   (define (iter lst n)
       (if (null? lst) n
          (iter (cdr lst) (+ n (car lst)))))
   (iter list 0))

;compute the length of a list
(define (length lst)
  (define (iter lst1 n)
    (if (null? lst1) n
      (iter (cdr lst1) (+ n 1))))
  (iter lst 0))

(define (myand x y)
        (cond ((eq? x t) (eq? y t))
                ((eq? x nil) nil)
                ((eq? y nil) nil)))

; find i'th element of list
(define (element lst i)
        (if (= i 1) (car lst) (element (cdr lst) (- i 1))))

; find elements of six-element list (non-recursively)
(define (first1 lst) (car lst))
(define (second2 lst) (cadr lst))
(define (third3 lst) (caddr lst))
(define (fourth4 lst) (cadddr lst))
(define (fifth5 lst) (car (cddddr lst)))
(define (sixth6 lst) (cadr (cddddr lst)))

;replace elements of a six-elemnt list
(define (set-first lst repl) (set-car! lst repl))
(define (set-second lst repl) (set-car! (cdr lst) repl))
(define (set-third lst repl) (set-car! (cddr lst) repl))
(define (set-fourth lst repl) (set-car! (cdddr lst) repl))
(define (set-fifth lst repl) (set-car! (cddddr lst) repl))
(define (set-sixth lst repl) (set-car! (cdr (cddddr lst)) repl))

;transform a six element list into a string
(define (list-to-string lst)
        (strcat (format "%4d" (first1 lst))
        (strcat (format "%4d" (second2 lst))
        (strcat (format "%4d" (third3 lst))
        (strcat (format "%4d" (fourth4 lst))
        (strcat (format "%4d" (fifth5 lst))
                (format "%4d" (sixth6 lst))))))))

;transforms a list into a string
;(define (list-to-string lst)
;       (if (eq? lst nil)
;               ""
;               (strcat (strcat (list-to-string (cdr lst))" ")
;                (format "%d" (car lst)))))

; add two tables and put result in table1 (t1)
(define (add-to-table t1 t2)
        (rplaca t1 (+ (first1 t1) (first1 t2)))
        (rplaca (cdr t1) (+ (second2 t1) (second2 t2)))
        (rplaca (cddr t1) (+ (third3 t1) (third3 t2)))
```

```
        (rplaca (cdddr t1) (+ (fourth t1) (fourth t2)))
        (rplaca (cddddr t1) (+ (fifth t1) (fifth t2)))
        (rplaca (cdr (cddddr t1)) (+ (sixth t1) (sixth t2))))

;(define (add-to-cumulated)
;       (add-to-table tt ctt)
;       (add-to-table tn ctn))

(define terminal 1)

(key-setup terminal "tt" "v")

(define (escape string)
  (strcat (achar 27) string))

(define (mget-answer)
  (define (list-to-string list)
    (if (eq? nil list)
        ""
      (strcat (list-to-string (cdr list) ) (achar (car list)))))

  (define (print list string)
    (key-write id (strcat (escape "[A") (strcat (list-to-string list) string))))

  (define (loop list)
    (let ((next (key-get terminal)))
      (cond ((= 13 next)
             (list-to-string list))
            ((= 8 next )
             ((lambda ()
                (if (eq? nil list)
                    (loop list)
                  ((lambda ()
                     (print list (escape "[D"))
                     (print list " ")
                     (print list (escape "[D"))
                     (loop (cdr list))))))))
            (t
             ((lambda ()
                (print list (achar next))
                (loop (cons next list))))))))
  (loop nil))
```

Appendix D
Runexp Code

Rxp03.spd

```
; model variables defined (as *unassignedXYZ*) when runexp99 is first loaded
; model variable set to (exp-create-model ...) by exp-connect
; model reset ("set!" [to default value] and "initialize" [made "clean"])
;        by exp-reset-models
; model changed by calling function
; model data (var "make-clean") sent by exp-send-changes

;   This is the spud file that interfaces with the mod.asm programs running
; on the DSP-96-containing PCs.
;
; There are two types of procedures and parameters defined in RUNEXP99.SPD.
; The user must not alter the definitions of any of the following procedures.
;
;  1) User interface procedures that may be freely called by the user
;     and parameters that may be freely altered by the user. The names of
;     all such procedures/parameters begin with EXP- and contain
;     hyphens not underscores:
;
;    EXP-DEBUGGING?
;    EXP-INTERVALS EXP-REPETITIONS EXP-TONES
;    EXP-NOISE-BANDWIDTH EXP-SAMPLE-RATE
;    EXP-TONE-INTEN EXP-NOISE-INTEN
;    EXP-TONE-ATTEN EXP-FREQ EXP-PHASE EXP-CAR-PHASE
;    EXP-MOD-PHASE EXP-MOD-DEPTH
;    EXP-IP EXP-RN EXP-DS EXP-RT EXP-OT EXP-FN EXP-DF EXP-FT
;    EXP-CONNECT EXP-CONNECTION
;    EXP-RESET-MODELS EXP-SEND-CHANGES EXP-GO
;    EXP-INPUT-GAIN EXP-OUTPUT-GAIN
;
; Utility procedures that are available to the user.
;
;    EXP-SCALE2DB EXP-DB2SCALE
;    EXP-CREATE-LIST EXP-GET-ELEMENT EXP-SET-ELEMENT! EXP-FIRST
;    EXP-MIN-LST EXP-MAX-LST
;    EXP-ROUND2TENTH
;    EXP-EQ-STRINGS
;    EXP-PRINT
;
; Utility procedures for dealing with (somewhat generalized) 2-D tables.
;
;    EXP-ASSZ
;    EXP-MAKE-TABLE EXP-LOOKUP2 EXP-INSERT2!
;
; 2) Internal RUNEXP procedures that should never be called/altered by the
;     user. All names begin with EXP_ and contain underscores not hyphens:
;
;    EXP_MSG_QUEUE
;    EXP_CREATE_MODEL
;    EXP_SAMPLE_RATE EXP_SAMPL_RATE EXP_TMP
;    EXP_LATT EXP_RATT
;    EXP_IP_LIST EXP_RN_LIST EXP_DS_LIST EXP_RT_LIST
;    EXP_OT_LIST EXP_FN_LIST EXP_DF_LIST EXP_FT_LIST
;    EXP_LEFT_DB EXP_RIGHT_DB
;    EXP_FT_CAR_ATTS EXP_FT_CAR_FREQS EXP_FT_CAR_PHASES
;    EXP_FT_MODDEPTHS EXP_FT_MOD_FREQS EXP_FT_MOD_PHASES
;    EXP_CH_SCALES EXP_OUTPUT_SCALES
;
; Procedures defined in the sockets library:
;    S-CREATE
;    S-TIMEOUT
;
; Debugging switches:
```

```
;
(define EXP-DEBUGGING? #f)       ; for debugging RUNEXP Procedures
(define EXP_DEBUGGING? #f)       ; for debugging RUNEXP Procedures
(define EXP_NOIO? #f)            ; for debugging programs without doing io
(define EXP_NOIO_PRINT? #f)      ; for printing hypothetical io
;
; Internal RUNEXP parameters
;   (define EXP_NOISE_DEF100 98.0)       ; Power of a band of noise 100 Hz wide
;   (define EXP_NOISE_DEFAULT 98.0)      ; Default level for the actual noise
;   (define EXP_TONE_DEFAULT 105.6)

; EXP_MAXX_TONES & EXP_MAXX_INTVLS must agree  with MOD.H & MODSYM.ASM
;
;   (define EXP_MAXX_INTVLS 9)
;   (define EXP_MAXX_TONES 9)
;
; EXP_MAX_TONES & EXP_MAX_INTVLS must be <= EXP_MAXX_TONES & EXP_MAXX_INTVLS
;   they can be changed by exp-intervals and exp-tones,
;   before exp-connect is called.
;
;   (define EXP_MAX_INTVLS 4)
;   (define EXP_MAX_TONES 8)
;
;-----------------------------------------------------------------------
; How RUNEXP works
;
;   RUNEXP builds its own model of the experiment, and all but two of the
; functions below merely adjust this model and build up a queue of commands
; to a DSP-96 server running elsewhere.
;
;   The internal models are initialized by evaluating (EXP-RESET-MODELS)
; at a point in time when the external models are initialized.
;
;   To minimize the need for communication, only changes in the internal models
; are sent as control strings to the the PC that contains the DSP board that
; creates the waveforms, by evaluating (EXP-SEND-CHANGES). This call must
; be made after all the changes to RUNEXPs models have been made.
;
;   To command the PC to begin generating stimuli, evaluate (EXP-GO)
; (which also sends a command string).

; Proper use of RUNEXP assumes that the outputs of the DSP96 boards
; digital-to-analog converters are connected directly to the Tucker-Davis
; attenuator inputs, whose outputs are the inputs to the headphone drivers.
; This path has a nominal gain of unity. If amplification/attenuation
; is applied after the headphone driver, then an compensating
; amplification/attenuation should be applied at the output of the DSP96
; board.  For example, if 20 dB of attenuation is applied after the headphone
; driver to mask spurious noise, then 20 dB of gain should be applied
; at the output of the DSP board.  This can be done using the EXP-OUTPUT-GAIN
; function (in this example, on both output channels):
;
;                        (EXP-OUTPUT-GAIN 20.0 'both)
;
; The following two defines establish the SPL levels produced by the PC's
;  RUNEXP program operating as loud as it can go (i.e. tone and noise
;  outputs are multiplied by a scale factor of 1.0), when the DSP board plays
;  out a tone with amplitude +-10,000 digital units.  The noise level, of
;  course, will vary depending on the bandwidth of the filter being used to
;  shape it.  Ideally, SPUD would know about the parameters of the filter
;  and be able to re-calculate EXP_NOISE_DEFAULT on the fly as the filter shape
;  is changed.  This is not currently implemented, though, so if the user
;  requests a "90 dB SPL" noise burst, he/she will only *get* a 90 dB SPL
;  noise burst if the noise is being passed through a filter with a 100-Hz
;  wide pass band.

(define EXP_NOISE_DEF100 98.0)  ; Power of a band of noise 100 Hz wide
(define EXP_NOISE_DEFAULT 98.0) ; Default noise level for the actual noise
(define EXP_TONE_DEFAULT 105.6)

; These must agree with MOD.H on the PC end of things, and MOD.H must agree
```

```
;  with MODSYM.ASM

(define EXP_MAXX_INTVLS 175)
(define EXP_MAXX_TONES 9)
(define EXP_SAMPLE_RATE 11.025)
(define EXP_MAX_INTVLS 4)
(define EXP_MAX_TONES  8)

; EXP_MAX_ATTEN is the largest allowable attenuation for the RUNEXP program.
; (lest the dynamic range of the DSP-96's d2a converters may be exhausted)
;  If the user exceeds this dynamic range, RUNEXP prints a warning.
(define EXP_MAX_ATTEN 80.0)

; RCC - Default tone values. If new values are not set, then they will certainly
; remain inaudible.
(define TONE_DEF -200.0)

; Default noise values. If new values are not set, then the noise too will
; remain inaudible.
(define NOISE_DEF -200.0)

; These are lists of the current dB SPL levels for the tones in both channels.
(define LEFT_TONES
  (let ((lst nil))
    (do ((i EXP_MAX_INTVLS (- i 1)))
        ((= i 0) lst)
      (set! lst (cons TONE_DEF lst)))))

(define RIGHT_TONES
  (let ((lst nil))
    (do ((i EXP_MAX_INTVLS (- i 1)))
        ((= i 0) lst)
      (set! lst (cons TONE_DEF lst)))))

; These are lists of the current dB SPL levels for the noise in both channels.
(define LEFT_NOISE
  (let ((lst nil))
    (do ((i EXP_MAX_INTVLS (- i 1)))
        ((= i 0) lst)
      (set! lst (cons NOISE_DEF lst)))))

(define RIGHT_NOISE
  (let ((lst nil))
    (do ((i EXP_MAX_INTVLS (- i 1)))
        ((= i 0) lst)
      (set! lst (cons NOISE_DEF lst)))))

;-----------------------------------------------------------------------
; utility procedures and definitions
(define else t)  ;;; in again; commented out 28-Apr-1999 - LDB - now in lisp.l ;readded -
RCC

; converts X into 20log(X)   (base 10)
(define (exp-scale2db scale)
; (cond (EXP-DEBUGGING? (EXP-PRINT (list "EXP-SCALE2DB:" scale))))
  (* 20.0 (log. scale)))

; converts 20log(X) into X   (base 10)
(define (exp-db2scale level)
; (cond (EXP-DEBUGGING? (EXP-PRINT (list "EXP-DB2SCALE:" level))))
  (power. 10.0 (/ level 20.0)))

; round a number off to the tenths place
(define (EXP-ROUND2TENTH n)
; (cond (EXP-DEBUGGING? (EXP-PRINT (list "EXP-ROUND2TENTH" n))))
  (/ (round (* 10.0 n)) 10.0))

;;; RUNEXP99 data structures are lists, but could perhaps be vectors
;;;  hopefully enough data abstraction was used.

(define (gmap proc obj)
```

```
; (cond (EXP-DEBUGGING? (EXP-PRINT (list "GMAP" obj))))
  (cond ((pair? obj) (map proc obj))
        ((vector? obj) (list->vector (map proc (vector->list obj))))
        (else (fatalf "gmap - unanticipated structure\n"))))

(define (gfor-each proc obj)
; (cond (EXP-DEBUGGING? (EXP-PRINT (list "GFOR-EACH" obj))))
  (cond ((pair? obj) (for-each proc obj))
        ((vector? obj) (for-each proc (vector->list obj)))
        (else (fatalf "gmap - unanticipated structure\n"))))

(define (gequal? a b)
  (cond ((and (pair? a) (pair? b))
         (equal? a b))
        ((and (pair? a) (vector? b))
         (equal? a (vector->list b)))
        ((and (pair? b) (vector? a))
         (equal? b (vector->list a)))
        ((and (vector? b) (vector? a))
         (equal? (vector->list b) (vector->list a)))
        (else #f)))

; appends two data structures
;(define exp_append append)
;(define (exp_append v1 v2)
;  (letrec ((l1 (vector-length v1))
;           (l2 (vector-length v2))
;           (l (+ l1 l2))
;           (v (make-vector l)))
;          (do ((i 0 (+ i 1)))
;              ((> i l1)
;               (do ((j l1 (+ j 1)) (k 0 (+ k 1)))
;                   ((> j l) 'done)
;                   (vector-set! v j (vector-ref v2 k))))
;              (vector-set! v i (vector-ref v1 i)))
;          v))


; gets element N of list LST
; 29-Apr-1999 LDB redefined in terms of list-ref
;(define (EXP-GET-ELEMENT lst n) (list-ref lst (- n 1)))
;(define exp-get-element vector-ref)

(define (EXP-GET-ELEMENT obj n)
;  (cond (EXP-DEBUGGING? (EXP-PRINT (list "EXP-GET-ELEMENT!" obj n ))))
  (cond ((pair? obj) (EXP-GET-LIST-ELEMENT obj n))
        ((vector? obj) (vector-ref obj (- n 1)))
        (else (fatalf "EXP-SET-ELEMENT: unanticipated structure"))))

; N=1 refers to (CAR LST)
(define (EXP-GET-LIST-ELEMENT lst n)
  (define (iter newlist i)
    (cond ((= i 0) (car newlist))
          (else (iter (cdr newlist) (- i 1)))))
  (cond ((= n 1) (car lst))
        (else (iter lst (- n 1)))))

(define (EXP-SET-ELEMENT! obj n value)
;  (cond (EXP-DEBUGGING? (EXP-PRINT (list "EXP-SET-ELEMENT!" obj n value))))
  (cond ((pair? obj) (EXP-SET-LIST-ELEMENT! obj n value))
        ((vector? obj) (vector-set! obj (- n 1) value))
        (else (fatalf "EXP-SET-ELEMENT: unanticipated structure"))))

; sets element N of list LST to VALUE
; N=1 refers to (CAR LST)
(define (EXP-SET-LIST-ELEMENT! lst n value)
  (define (iter newlist i)
    (cond ((= i 0) newlist)
          (else (iter (cdr newlist) (- i 1)))))
  (cond ((= n 1) (set-car! lst value))
        (else (set-car! (iter lst (- n 1)) value))))
```

```
; create a list of length elements whose value is object
(define (exp-create-list length object)
  (define (list-iter count)
    (if (= 0 count)
       nil
       (cons object (list-iter (- count 1)))))
; (cond (EXP-DEBUGGING? (EXP-PRINT (list "exp-create-list" length object))))
  (list-iter length))

(define (exp-create-listvector length object)
 (cond (EXP_DEBUGGING?
        (EXP-PRINT (list "exp-create-listvector" length object))))
  (let ((v (make-vector length)))
    (vector-fill! v object)
    v))

(define exp-create-list exp-create-listvector)
;(define exp-create-listvector exp-create-list)
;
; LDB 27-Apr-1999
;   NO-WARNING needed: not-redefining the max and min functions
;
(define (exp-min-lst obj)
  (cond (EXP_DEBUGGING? (EXP-PRINT (list "EXP-MIN-LST" obj))))
  (cond ((pair? obj) (exp-min-lst-list obj))
        ((vector? obj) (exp-min-lst-vector obj))
        (else (faltalf "exp-min-lst: unanticipated structure"))))

; determine the minimum of the numbers in a list
; this is less than satisfactory because of the 1000000.0 parameter
(define (EXP-MIN-LST-list lst)
  (let ((MIN-VAL 1000000.0))
    (define (iter lst min-val)
      (cond ((null? lst) min-val)
            ((< (car lst) min-val) (iter (cdr lst) (car lst)))
            (else (iter (cdr lst) min-val))))
    (iter lst MIN-VAL)))

(define (exp-min-lst-vector lst)
  (let ((vl (vector-length lst))
        (z (vector-ref lst 0)))
    (do ((i 0 (+ i 1)))
        ((>= i vl) z)
        (set! z (min z (vector-ref lst i))))))

; determine the maximum of the numbers in a list
; this is less than satisfactory because of the -1000000.0 parameter
(define (exp-max-lst obj)
  (cond (EXP_DEBUGGING? (EXP-PRINT (list "EXP-MAX-LST" obj))))
  (cond ((pair? obj) (exp-max-lst-list obj))
        ((vector? obj) (exp-max-lst-vector obj))
        (else (faltalf "exp-max-lst: unanticipated structure"))))

(define (EXP-MAX-LST-LIST lst)
  (let ((MAX-VAL -1000000.0))
    (define (iter lst max-val)
      (cond ((null? lst) max-val)
            ((> (car lst) max-val) (iter (cdr lst) (car lst)))
            (else (iter (cdr lst) max-val))))
    (iter lst MAX-VAL)))

(define (exp-max-lst-vector lst)
  (let ((vl (vector-length lst))
        (z (vector-ref lst 0)))
    (do ((i 0 (+ i 1)))
        ((>= i vl) z)
        (set! z (max z (vector-ref lst i))))))

; return a list of only the first n items in the input list
; 28-Apr-1999 LDB - this aprocedure does not appear to be used by the code
```

```
;(define (EXP-FIRST n lst)
;  (define (iter lst1 lst2 n)
;    (cond ((or (null? lst1) (= n 0)) lst2)
;          (else (set! lst2 (cons (car lst1) lst2))
;                (iter (cdr lst1) lst2 (- n 1)))))
;  (cond (EXP-DEBUGGING? (EXP-PRINT (list "EXP-FIRST" n lst))))
;  (reverse (iter lst nil n)))

(define (exp-first n lst)
  (let ((v (make-vector n)))
    (vector-fill! v "*unassigned*")
    (do ((i 0 (+ i 1)))
        ((> i (min n (- (vector-length lst) 1))) v)
        (vector-set! v i (vector-ref i lst)))))

; determine whether two strings are equal
; 28-Apr-1999 this procedure does not appear to be used in the code
(define (EXP-EQ-STRINGS str1 str2)
; (cond (EXP-DEBUGGING? (EXP-PRINT (list "EXP-EQ-STRINGS" str1 str2))))
  (= (strcmp str1 str2) 0))

(define (eqs? str1 str2)
  (= (strcmp str1 str2) 0))

; 28-Apr-1999 LDB - redefined in terms of max and min
; 28-Apr-1999 LDB - min. and max. commented out - no use in code
;(define (max. a b)
;  (cond (EXP-DEBUGGING? (EXP-PRINT (list "MAX." a b))))
;  (cond ((> b a) b)
;        (else a)))

;(define max. max)
;(define min. min)

;;; 2-D table procedures adapted from SICP
;;; The table header contains a list containing the two user-specified
;;; key-comparison procedures, t1? and t2?

(define (exp-make-table t1? t2?)
  (list (list t1? t2?)))

;assz is a generalized assq

(define (exp-assz key test? lst)
  (define (loop l)
    (cond ((null? l) #f)
          ((test? (caar l) key) (car l))
          (t (loop (cdr l)))))
  (loop lst))

(define (exp-lookup2 key-1 key-2 table)
  (let ((t1? (caar table))
        (t2? (cadar table)))
    (let ((subtable (exp-assz key-1 t1? (cdr table))))
      (if (null? subtable)
          #f
        (let ((record (exp-assz key-2 t2? (cdr subtable))))
          (if (null? record) #f (cdr record)))))))

(define (exp-insert2! key-1 key-2 value table)
  (let ((t1? (caar table))
        (t2? (cadar table)))
    (let ((subtable (exp-assz key-1 t1? (cdr table))))
      (if (null? subtable)
          (set-cdr! table
                    (cons (list key-1 (cons key-2 value))
                          (cdr table)))
        (let ((record (assz key-2 t2? (cdr subtable))))
          (if (null? record)
              (set-cdr! subtable
```

```
                        (cons (cons key-2 value) (cdr subtable)))
              (set-cdr! record value))))))
  'inserted)

;zzzz
;---------------------------------------------------------------------------
;PRINTS OUT AN ARBITRARY DATA STRUCTURE (RETURNS T)
; IF INVOKED WITH AN OPTIONAL SECOND ARGUMENT WHICH IS FALSE, DOES NOT PRINT
; ANYTHING BUT RETURNS A STRING WITH WHAT WOULD HAVE BEEN PRINTED.
; note that this relies on the pun that an empty list is false
(define (EXP-PRINT obj . do-not-print?)
  (let ((result ""))
    (define (print2 obj paren)
      (cond ((not (pair? obj))
             (cond
              ((string? obj)
               (set! result (strcat result (strcat " " obj))))
              ((vector? obj)
               (set! result (strcat result " ["))
               (print2 (vector->list obj) paren)
               (set! result (strcat result " ]")))
              ((null? obj) (set! result (strcat result " )")))
              ((integer? obj)
               (set! result (strcat result
                                    (format " %d" obj))))
              ((real? obj)
               (set! result (strcat result
                                    (format " %e" obj))))
              (else (set! result (strcat result " <unknown type>")))))
            (else
             (cond (paren (set! result (strcat result " (")))
             (print2 (car obj) t)
             (print2 (cdr obj) nil))))
    ;Hey there!  Cant do debugging here with "print," thats for sure
    (print2 obj t)
    (if do-not-print? result (printf "%s\n" result))))

;
; EXP_MSG_QUEUE: An object which holds a string which can be added to.
;       Each addition also inserts the string "<<MSG>>" before the next
;       string.
;
;   THIS OBJECT IS A PROCEDURE WHICH TAKES 1 or 2 ARGS:
;
;       (exp_msg_queue "add" "string")          adds to the string.
;       (exp_msg_queue "flush")                 returns the string and resets
;
(define exp_msg_queue
  ; Define a message internal to this function, initially void
  (let ((msg "") (old-msg "*unassigned1*"))
    (lambda (command . string)
      (cond ((eqs? command "add")
             ; Adds to the internal message
             (set! string (strcat (car string) " "))
             (if (and EXP_NOIO? EXP_NOIO_PRINT?)
                 (set! msg (strcat (strcat msg "\n<<MSG>> ") string))
                 (set! msg (strcat (strcat msg "<<MSG>> ") string)))
             t)
            ((eqs? command "flush")
             ; Returns the message string and resets internal msg to ""
             (set! old-msg msg)
             (set! msg "")
             old-msg)))))

;---------------------------------------------------------------------------
;A model is a function which accepts messages as arguments.  A model has
; state which may be fetched or set.  Setting the state of a spud model causes
; the model to become "dirty" (non-current with reality) until the model
; receives the "make-clean" message, which causes it to take appropriate steps
; to ensure that it matches reality.  A model also has a name which is a
; character string.
```

```
;Currently there are 2 kinds of models, the "pc-list" variety and the
; "tdatt" variety.  A "pc-list" model has as state a list of values which it
; sends to the pc (prepended with the model name) upon "make-clean".  A "tdatt"
; model contains an attenuation which it sets the Tucker-Davis attenuator to
; upon "make-clean".
(define (exp_create_model type name . extra)
  ;Define the behavior of the "pclist" variety model: extra is the length of
  ; the data list to be sent to the PC
  (cond (EXP_DEBUGGING? (EXP-PRINT (list "EXP_CREATE-MODEL" type name extra))))
  (cond
   ((eqs? type "pclist")
    (let ((length (car extra))
          (lst (exp-create-list (car extra) "*unassigned2*"))
          (status "*unassigned3*")
          (string ""))
      (define (add-to-string item)
        (set! string (strcat string (EXP-PRINT item t))))
      (lambda (messg . args)
        (cond (EXP_DEBUGGING?
               (EXP-PRINT (list "<PCLIST MODEL>" messg args))))
        (cond
         ((eqs? messg "get-list") lst)
         ((eqs? messg "set-list!")
          (let ((new-list (car args)))
            (cond ((not (gequal? new-list lst))
                   (set! lst new-list)
                   (set! status "dirty"))
                  (else "clean"))))   ;;; not (set! status "clean") ???
         ((eqs? messg "initialize") (set! status "clean"))
         ((eqs? messg "set!")
          (cond ((eqs? status "frozen")
                 (fatalf "Can't make requested changes!")))
          (let ((index (car args))  (item (cadr args)))
            (cond
             ((eqv? (EXP-GET-ELEMENT lst index) item) "clean")
             (else
              (EXP-SET-ELEMENT! lst index item)
              (set! status "dirty")))))
         ((eqs? messg "get")
          (let ((index (car args)))
            (EXP-GET-ELEMENT lst index)))
         ((eqs? messg "make-clean")
          (cond ((eqs? status "clean") t)
                (else
                 (set! string name)
                 (gfor-each add-to-string lst)
                 (set! status "clean")
                 (exp_msg_queue "add" string))))
                ; (s-send exp-connection string))))
         ((eqs? messg "freeze")
          (set! status "frozen"))
         ((eqs? messg "make-spiffy") ; send no matter what
          (set! string name)
          (gfor-each add-to-string lst)
          (set! status "clean")
          (exp_msg_queue "add" string))
        ; (exp-print string)) - LDB
        ; (s-send exp-connection string) - old, changed by dsl
         ((eqs? messg "clean?") (eqs? status "clean"))
         ((eqs? messg "dirty?") (eqs? status "dirty"))
         ((eqs? messg "get-name") name)
         ((eqs? messg "set-name!")
          (let ((new-name (car args)))
            (set! name new-name)))
         (else (fatalf "PCLIST: unknown message passed to me"))))))

;.....................................................................
        ;Now, define the "tdatt" type model: extra should be an attenuator
        ; number (either 1 or 2)
   ((eqs? type "tdatt")
    (let ((status "*unassigned4*")
```

```
        (attenuation "*unassigned5*")
        (number (car extra))
        (max-atten 90.0))
      (lambda (messg . args)
       (cond (EXP_DEBUGGING?
              (EXP-PRINT (list "<TDATT MODEL>" messg args))))
       (cond
        ((eqs? messg "get-atten") attenuation)
        ((eqs? messg "set-atten!")
         (let ((atten (car args)))
           (set! atten (EXP-ROUND2TENTH atten))
           (cond ((> atten max-atten)
                  (set! atten max-atten)))
           (cond
            ((eqv? attenuation atten) attenuation)
            (else
             (set! attenuation atten)
             (set! status "dirty") attenuation))))
        ((eqs? messg "set-max-atten!")
         (let ((new-max (car args)))
           (set! max-atten new-max)))
        ((eqs? messg "initialize") (set! status "clean"))
        ((eqs? messg "get-max") max-atten)
        ((eqs? messg "make-clean")
         (cond ((eqs? status "clean") attenuation)
               (else
                (td-att number attenuation)
                (set! status "clean")
                attenuation)))
        ((eqs? messg "clean?") (eqs? status "clean"))
        ((eqs? messg "dirty?") (eqs? status "dirty"))
        ((eqs? messg "get-name") name)
        ((eqs? messg "set-name!")
         (let ((new-name (car args)))
           (set! name new-name)))
        (else
         (fatalf "TDATT: unknown message %s passed to me" messg))))))
;.....................................................................
        ;Define the behavior of the "dblist" variety model: extra is the
        ; length of the data list of scale levels to be sent to the PC
        ((eqs? type "dblist")
         (let ((length (car extra))
               (tonelst (exp-create-list (car extra) "*unassigned6*"))
               (noiselst (exp-create-list (car extra) "*unassigned7*"))
               (status "*unassigned8*")
               (t_pclist (exp_create_model "pclist"
                           (strcat name " tone") (car extra)))
               (n_pclist (exp_create_model "pclist"
                           (strcat name " noise") (car extra))))
           (lambda (messg . args)
             (cond (EXP_DEBUGGING?
                    (EXP-PRINT (list "<DBLIST MODEL>" messg args))))
             (cond
              ((eqs? messg "initialize") (set! status "clean"))
              ((eqs? messg "get-pclist")
               (let ((query (car args)))
                 (cond ((eqs? query "noise") n_pclist)
                       ((eqs? query "tone")  t_pclist)
                       (else
                        (fatalf
                 "DBLIST: get-pclist: must have tone or noise argument")))))
              ((eqs? messg "get-list")
               (let ((lst (car args)))
                 (cond ((eqs? lst "noise") noiselst)
                       ((eqs? lst "tone")  tonelst)
                       (else (fatalf
                 "DBLIST: get-list: Must have tone or noise argument")))))
              ((eqs? messg "set-list!")
               (let ((lst (car args)) (new-list (cadr args)))
                (cond (EXP_DEBUGGING?
                       (EXP-PRINT (list "<DBLIST set-list>" lst new-list))))
```

```
   (cond ((eqs? lst "noise")
          (cond (EXP_DEBUGGING?
                 (EXP-PRINT (list "<DBL noiselst>" noiselst))))
           (cond
            ((gequal? new-list noiselst) "clean")
            (else
             (set! noiselst new-list)
            (set! status "dirty")
            (cond
             (EXP_DEBUGGING?
                  (EXP-PRINT
                   (list "<DBL stat nlst>" status noiselst)))))))
          ((eqs? lst "tone")
           (cond ((gequal? new-list tonelst) "clean")
                 (else
                  (set! tonelst new-list)
                  (set! status "dirty"))))
          (else (fatalf
    "DBLIST: set-list!: Must have tone or noise argument")))))
((eqs? messg "set!")
 (let ((t-or-n (car args))
       (index (cadr args))  (item (caddr args)))
   (cond ((eqs? t-or-n "noise")
          (cond
           ((eqv? (EXP-GET-ELEMENT noiselst index) item) "clean")
           (else
            (EXP-SET-ELEMENT! noiselst index item)
            (set! status "dirty"))))
         ((eqs? t-or-n "tone")
          (cond
          ((eqv? (EXP-GET-ELEMENT tonelst index) item) "clean")
          (else (EXP-SET-ELEMENT! tonelst index item)
                (set! status "dirty"))))
         (else (fatalf
      "DBLIST: set!: Must have tone or noise argument")))))
((eqs? messg "get")
 (let ((lst (car args)) (index (cadr args)))
   (cond ((eqs? lst "noise") (set! lst noiselst))
         ((eqs? lst "tone") (set! lst tonelst))
         (else (fatalf
           "DBLIST: get: Must have tone or noise argument")))
   (EXP-GET-ELEMENT lst index)))
;In "make-clean", the second arguent must be an attenuator
((eqs? messg "make-clean")
 (let ((atten (car args))
       (d-range "*unassigned9*")
        (max-level "*unassigned10*")
       (min-level "*unassigned11*")
        (patten "*unassigned12*")
       (tscale-list tonelst)
        (nscale-list noiselst))
   (define (relative-tone-level db)
     (- db EXP_TONE_DEFAULT))
   (define (relative-noise-level db)
     (- db EXP_NOISE_DEFAULT))
   (cond
    ((eqs? status "clean") t)
    (else
     ; Change the status to "clean" for next time
     (set! status "clean")
     ; Change all decibel measures to dB re the default levels
     (set! tscale-list (gmap relative-tone-level tscale-list))
     (set! nscale-list (gmap relative-noise-level nscale-list))
     ;If any dB level is >0, produce an error (cant amplify!)
     (set! max-level (max (EXP-MAX-LST tscale-list)
                          (EXP-MAX-LST nscale-list)))
     (set! min-level (min (EXP-MIN-LST tscale-list)
                          (EXP-MIN-LST nscale-list)))
     (cond ((> max-level 0)
            (fatalf
           "DBLIST: make-clean: No gain-- only atten.")))
```

```
                    ; If the dynamic range of this channel exceeds the range of
                    ;  the DSP board d2a converters, produce a warning
                    (set! d-range (- max-level min-level))
                    (cond ((and (> d-range EXP_MAX_ATTEN)
                                (< d-range 150))
                           (printf "DSP cant achieve this dynamic range\n")))
                    ; Now, set the programmable attenuator to attenuate down to
                    ;  the level of the loudest tone or noise
                    ;  [produce a *positive* decibel measure]
                    (set! patten (- 0 max-level))
                    ; see how close the attenuator can come to that level
                    (set! patten (atten "set-atten!" patten))
                    ; Apply the new attenuation
                    (atten "make-clean")
                    ; Now, translate the dB measures into scales, making sure
                    ;  to subtract off the amount of attenuation from the TDs
                    ;  then, change the noise and tone pclists
                    (set! tscale-list
                          (gmap (lambda (value) (+ patten value)) tscale-list))
                    ; Only set the list if it is different than the current one
                    (set! tscale-list (gmap exp-db2scale tscale-list))
                    (cond ((gequal? tscale-list (t_pclist "get-list")) t)
                          (else
                           (t_pclist "set-list!" tscale-list)
                           (t_pclist "make-clean")))
                    (set! nscale-list
                          (gmap (lambda (value) (+ patten value)) nscale-list))
                    ; Only set the list if it is different than the current one
                    (set! nscale-list (gmap exp-db2scale nscale-list))
                    (cond ((gequal? nscale-list (n_pclist "get-list")) t)
                          (else
                           (n_pclist "set-list!" nscale-list)
                           (n_pclist "make-clean")))
                    ; Now, find out how much the PC had to attenuate
                    ;  the softest sound. If that amount is more than
                    ;  EXP_MAX_ATTEN, and less than something very soft, the
                    ;  d2a converters on the DSP board may be at their limits
                    (set! d-range
                          (- (exp-scale2db (min (EXP-MIN-LST tscale-list)
                                                (EXP-MIN-LST nscale-list)))
                             0))
                    (if (and (> d-range EXP_MAX_ATTEN)
                             (< d-range 200))
                        (printf "MAKE_CLEAN: DSP dynamic range too small.\n")
                     t)))))
                ((eqs? messg "clean?") (eqs? status "clean"))
                ((eqs? messg "dirty?") (eqs? status "dirty"))
                (else (fatalf "DBLIST: unknown message passed to me"))))))
        (else (fatalf "CREATE-MODEL: unknown model type"))))
;---------------------------------------------------------------------
;Create models for the Tucker-Davis programmable attenuators
(define EXP_LATT "*unassigned19*")
(define EXP_RATT "*unassigned20*")

; Create basic spud models
(define EXP_NINTS "*unassigned21a*")
(define EXP_NREPS "*unassigned21b*")
(define EXP_NTONES "*unassigned21c*")
;(define EXP_SAMPL_RATE "*unassigned21d*")
(define EXP_WAV_LOAD "*unassigned21e*")
(define EXP_WAV_ATTEN "*unassigned21f*")
(define EXP_NONWAV_ATTEN "*unassigned21g*")
(define EXP_WAV_POINTER "*unassigned21h*")
(define EXP_NOISE_SIGNAL "*unassigned21i*")

;Create models of the timing parameters of the noise and tone bursts in the
; intervals of the experiment
(define EXP_RESTART_LIST "*unassigned22a*")
(define EXP_INITIALIZE_NOISE_LIST "*unassigned22b*")

(define EXP_IP_LIST "*unassigned23a*")
```

```
(define EXP_RN_LIST "*unassigned23b*")
(define EXP_DS_LIST "*unassigned23c*")
(define EXP_RT_LIST "*unassigned23d*")
(define EXP_OT_LIST "*unassigned23e*")
(define EXP_FN_LIST "*unassigned23f*")
(define EXP_DF_LIST "*unassigned23g*")
(define EXP_FT_LIST "*unassigned23h*")


; Set up spud models for the decibel levels of the noise and tone bursts in
;  the left and right channels
(define EXP_LEFT_DB "*unassigned30*")
(define EXP_RIGHT_DB "*unassigned31*")

;Next, set up spud models for all of the parameters of the tone bursts.  These
; parameters include a carrier attenuation (dB), carrier frequency, carrier
; phase, modulation depth, modulation frequency, and modulation phase for
; each of the EXP_MAX_TONES which may occur in each interval:
;The data structure will be a list of models, one model for each interval, to
; contain the data on each of the 6 parameters of the tone burst.
(define EXP_FT_CAR_ATTS nil)
(define EXP_FT_CAR_FREQS nil)
(define EXP_FT_CAR_PHASES nil)
(define EXP_FT_MODDEPTHS nil)
(define EXP_FT_MOD_FREQS nil)
(define EXP_FT_MOD_PHASES nil)
(define exp_tmp "*unassigned32*")

; Now, set up SPUD models for the gains applied to the DSP boards input
;  channels before they are mixed into the generated signal-- both channels
;  are actually mixed into a *single* "right/left" channel, currently the
;  LEFT channel [as of 2-5-94]
(define EXP_CH_SCALES "*unassigned33*")

; Set up a SPUD model for the gain applied at the output of the entire DSP
;  board system, immediately before the waveforms are played out
(define EXP_OUTPUT_SCALES "*unassigned34*")

;----------------------------------------------------------------------
; EXP-CONNECT:
;
; Makes a network connection to the "runexp" server and saves it in
;  the global variable "exp_pc_connection".  Only argument is a machine
;  name (string)-- if this is "" then assumes the server is on the local
;  machine.
;
(define exp-connection #F)
(define (exp-connect name)
  (cond (exp_noio?
         (if EXP_NOIO_PRINT? (EXP-PRINT (list "EXP-CONNECT" name)) #t))
        (else (if (= 0 (strcmp name ""))
                  (set! exp-connection (s-create "runexp"))
                  (set! exp-connection
                        (s-create (strcat "runexp" (strcat "@" name)))))))
  ; Set the timeout length on this connection generously
              (s-timeout exp-connection 500000)))

  ; Now set up a bunch of global variables (sigh)
  ;Create models for the Tucker-Davis programmable attenuators
  (set! EXP_LATT (exp_create_model "tdatt" "left attenuator" 1))
  (set! EXP_RATT (exp_create_model "tdatt" "right attenuator" 2))

  ; Set the maximum attenuations achievable by the attenuators to 0 db!
  ;  This is because they will not actually come into play when using
  ;  the new system of stimulus generation.
  ; If the RUNEXP code tries to use them, they will return a value of 0 dB no
  ;  matter what level it tries to set them to.
  (EXP_LATT "set-max-atten!" 0.0)
  (EXP_RATT "set-max-atten!" 0.0)

  ; Create basic spud models
```

```
  (set! EXP_NINTS (exp_create_model  "pclist" "nints" 1))
  (set! EXP_NREPS (exp_create_model  "pclist" "nreps" 1))
  (set! EXP_NTONES (exp_create_model "pclist" "ntones" 1))
;  (set! EXP_SAMPL_RATE (exp_create_model "pclist" "sample" 1))
  (set! EXP_WAV_LOAD (exp_create_model  "pclist" "wav_load" 1))
  (set! EXP_WAV_ATTEN (exp_create_model  "pclist" "wav_atten" 1))
  (set! EXP_NONWAV_ATTEN (exp_create_model  "pclist" "nonwav_atten" 1))
  (set! EXP_WAV_POINTER (exp_create_model  "pclist" "wav_pointer" 1))
  (set! EXP_NOISE_SIGNAL (exp_create_model  "pclist" "noise_signal" 1))

  ;Create models of the timing parameters of the noise and tone bursts in the
  ; intervals of the experiment
  (set! EXP_RESTART_LIST
       (exp_create_model "pclist" "restart_list" EXP_MAX_INTVLS))
  (set! EXP_INITIALIZE_NOISE_LIST
       (exp_create_model "pclist" "initialize_noise_list" EXP_MAX_INTVLS))
  (set! EXP_IP_LIST (exp_create_model "pclist" "ip_list" EXP_MAX_INTVLS))
  (set! EXP_RN_LIST (exp_create_model "pclist" "rn_list" EXP_MAX_INTVLS))
  (set! EXP_DS_LIST (exp_create_model "pclist" "ds_list" EXP_MAX_INTVLS))
  (set! EXP_RT_LIST (exp_create_model "pclist" "rt_list" EXP_MAX_INTVLS))
  (set! EXP_OT_LIST (exp_create_model "pclist" "ot_list" EXP_MAX_INTVLS))
  (set! EXP_FN_LIST (exp_create_model "pclist" "fn_list" EXP_MAX_INTVLS))
  (set! EXP_DF_LIST (exp_create_model "pclist" "df_list" EXP_MAX_INTVLS))
  (set! EXP_FT_LIST (exp_create_model "pclist" "ft_list" EXP_MAX_INTVLS))

  ; Set up spud models for the decibel levels of the noise and tone bursts in
  ;  the left and right channels
  (set! EXP_LEFT_DB (exp_create_model "dblist" "left_scales" EXP_MAX_INTVLS))
  (set! EXP_RIGHT_DB (exp_create_model "dblist" "right_scales" EXP_MAX_INTVLS))

  ;Next, set up spud models for all of the parameters of the tone bursts. These
  ; parameters include a carrier attenuation (dB), carrier frequency, carrier
  ; phase, modulation depth, modulation frequency, and modulation phase for
  ; each of the EXP_MAX_TONES which may occur in each interval:
  ;The data structure will be a list of models, one model for each interval, to
  ; contain the data on each of the 6 parameters of the tone burst.
  (set! EXP_FT_CAR_ATTS nil)
  (set! EXP_FT_CAR_FREQS nil)
  (set! EXP_FT_CAR_PHASES nil)
  (set! EXP_FT_MOD_DEPTHS nil)
  (set! EXP_FT_MOD_FREQS nil)
  (set! EXP_FT_MOD_PHASES nil)
  (set! exp_tmp "*unassigned35*")
  (do ((i EXP_MAX_INTVLS (- i 1)))
      ((< i 1) t)
      (set! exp_tmp (format "%d" i))
      (set! EXP_FT_CAR_ATTS (cons (exp_create_model "pclist"
        (strcat "ft_car_atts " exp_tmp) EXP_MAX_TONES) EXP_FT_CAR_ATTS))
      (set! EXP_FT_CAR_FREQS (cons (exp_create_model "pclist"
         (strcat "ft_car_freqs " exp_tmp) EXP_MAX_TONES) EXP_FT_CAR_FREQS))
      (set! EXP_FT_CAR_PHASES (cons (exp_create_model "pclist"
          (strcat "ft_car_phases " exp_tmp) EXP_MAX_TONES) EXP_FT_CAR_PHASES))
      (set! EXP_FT_MOD_DEPTHS (cons (exp_create_model "pclist"
         (strcat "ft_mod_depths " exp_tmp) EXP_MAX_TONES) EXP_FT_MOD_DEPTHS))
      (set! EXP_FT_MOD_FREQS (cons (exp_create_model "pclist"
        (strcat "ft_mod_freqs " exp_tmp) EXP_MAX_TONES) EXP_FT_MOD_FREQS))
      (set! EXP_FT_MOD_PHASES (cons (exp_create_model "pclist"
         (strcat "ft_mod_phases " exp_tmp) EXP_MAX_TONES) EXP_FT_MOD_PHASES)))

  ; Now, set up SPUD models for the gains applied to the DSP boards input
  ;  channels before they are mixed into the generated signal-- both channels
  ;  are actually mixed into a *single* "right/left" channel, currently the
  ;  LEFT channel [as of 2-5-94]
  (set! EXP_CH_SCALES (exp_create_model "pclist" "EXP_CH_SCALES" 2))

  ; Set up a SPUD model for the gain applied at the output of the entire DSP
  ;  board system, immediately before the waveforms are played out
  (set! EXP_OUTPUT_SCALES (exp_create_model "pclist" "EXP_OUTPUT_SCALES" 2))

; nints, & ntones, are sent only on the first transmission
  (EXP_NINTS "set!" 1 EXP_MAX_INTVLS)
```

```
  (EXP_NINTS "initialize")
  (EXP_NINTS "make-spiffy")
  (EXP_NTONES "set!" 1 EXP_MAX_TONES)
  (EXP_NTONES "initialize")
  (EXP_NTONES "make-spiffy")
;  (EXP_SAMPL_RATE "set!" 1 EXP_SAMPLE_RATE)
;  (EXP_SAMPL_RATE "initialize")
;  (EXP_SAMPL_RATE "make-spiffy")   ; change to send sample-rate later

  ; Reset the models to their default state
  (exp-reset-models)

  ; Return a throwaway truth value
  #t)

;----------------------------------------------------------------------
; EXP-RESET-MODELS: This function resets the SPUD models to some default
;                   values and tells the DSP board to reset its own
;                   parameters as well.
; !! WARNING !!  The default parameters used in this function must match the
;  default parameters in the RESET_STATE function on the DSP boards end of
;  things (in the MODB.C program).
;
(define (exp-reset-models)

  ; Set up a list EXP_MAX_INTVLS long of all float zeros
  (define (intvls-tiny) (exp-create-listvector EXP_MAX_INTVLS -200.0))
  (define (tones-to-tiny model)
    (model "set-list!" (exp-create-listvector EXP_MAX_TONES -200.0))
    (model "initialize"))
  (define (tones-to-zero model)
    (model "set-list!" (exp-create-listvector EXP_MAX_TONES 0.0))
    (model "initialize"))
  (define (intvls-to-zero model)  ; zero is now 0 rather than 0.0 LDB 5-23-99
    (model "set-list!" (exp-create-listvector EXP_MAX_INTVLS 0))
    (model "initialize"))

  (cond (EXP-DEBUGGING? (EXP-PRINT (list "EXP-RESET-MODELS"))))
  (cond ((> EXP_MAX_INTVLS EXP_MAXX_INTVLS)
         (fatalf "EXP_MAX_INTVLS > EXP_MAXX_INTVLS\n"
                 EXP_MAX_INTVLS  EXP_MAXX_INTVLS)))
  (cond ((> EXP_MAX_TONES EXP_MAXX_TONES)
         (fatalf "EXP_MAX_TONES > EXP_MAXX_TONES\n"
                 EXP_MAX_TONES  EXP_MAXX_TONES)))

  ; First, tell the PC that it should also reset all of its state.  As long
  ;  as the PCs reset parameters agree with those in this function, then
  ;  future SPUD<->PC model-versus-reality agreement is guaranteed.
  (exp_msg_queue "add" "reset_state")
  (cond (EXP_DEBUGGING?
         (EXP-PRINT (list "EXP-RESET-MODELS  exp-msg-queue"))))
  ;(s-send exp-connection "reset_state")  --- no need to bypass this line

  ; Zero out the interval times listings
  (gfor-each intvls-to-zero
             (list EXP_RESTART_LIST EXP_INITIALIZE_NOISE_LIST
                   EXP_IP_LIST EXP_RN_LIST EXP_DS_LIST EXP_RT_LIST
                   EXP_OT_LIST EXP_FN_LIST EXP_DF_LIST EXP_FT_LIST))
  (cond (EXP_DEBUGGING?
         (EXP-PRINT (list "EXP-RESET-MODELS intvls-to-zero"))))
  ; Zero out the attenuation, frequency, phase, etc. lists for each interval
  (gfor-each tones-to-tiny EXP_FT_CAR_ATTS)  ; reset state has large attenuations
  (gfor-each tones-to-zero EXP_FT_CAR_FREQS)       ; all frequencies are zero, and all
  (gfor-each tones-to-zero EXP_FT_CAR_PHASES)  ;  phases are zero
  (gfor-each tones-to-zero EXP_FT_MOD_DEPTHS)  ; modulation depth is zero
  (gfor-each tones-to-zero EXP_FT_MOD_FREQS)
  (gfor-each tones-to-zero EXP_FT_MOD_PHASES)
  (cond (EXP_DEBUGGING?
         (EXP-PRINT (list "EXP-RESET-MODELS tones-to-zero"))))
  ; Set the decibel lists for all tones and noise to -200 dB, and set the
  ;  internal tonelists all to 0.0 (yes, its an abstraction barrier violation
```

```
;  Sue me!)
(EXP_LEFT_DB "set-list!" "noise" (intvls-tiny))
(cond
 (EXP_DEBUGGING?
  (EXP-PRINT (list "EXP-RESET-MODELS exp_left_db set list noise tiny"))))
(EXP_LEFT_DB "set-list!" "tone" (intvls-tiny))
(cond
 (EXP_DEBUGGING?
  (EXP-PRINT (list "EXP-RESET-MODELS exp_left_db set list tone tiny"))))
(EXP_LEFT_DB "initialize")
(cond
 (EXP_DEBUGGING?
  (EXP-PRINT (list "EXP-RESET-MODELS exp_left_db initialize"))))
(EXP_RIGHT_DB "set-list!" "noise" (intvls-tiny))
(EXP_RIGHT_DB "set-list!" "tone"  (intvls-tiny))
(EXP_RIGHT_DB "initialize")
(cond
 (EXP_DEBUGGING?
  (EXP-PRINT (list "EXP-RESET-MODELS exp_right_db"))))
(gfor-each intvls-to-zero (list (EXP_LEFT_DB "get-pclist" "tone")
                                (EXP_LEFT_DB "get-pclist" "noise")
                                (EXP_RIGHT_DB "get-pclist" "tone")
                                (EXP_RIGHT_DB "get-pclist" "noise")))

(cond
 (EXP_DEBUGGING?
  (EXP-PRINT (list "EXP-RESET-MODELS gfor-each intvls"))))

; Actually force the Tucker-Davis attenuators to 0 dB of attenuation
;; The next line was commented out: the Tucker Davis attenuators should
;;  not really be necessary, so they are not used.  To ensure that they are
;;  never used later on, their MAX-ATTEN setting was made 0.0 during the
;;  creation of the models.  Here, we initialize their settings to 0.0
; (td-att 1 0.0)
; (td-att 2 0.0)
; Then, initialize the Spud models to agree with that
(EXP_RATT "set-atten!" 0.0)
(EXP_RATT "initialize")
(EXP_LATT "set-atten!" 0.0)
(EXP_LATT "initialize")
;  (set! exp_debugging? #t)
(cond
 (EXP_DEBUGGING?
  (EXP-PRINT (list "EXP-RESET-MODELS exp_latt"))))

(EXP_NREPS "set!" 1 1)
(EXP_NREPS "initialize")

(EXP_WAV_LOAD "set!" 1 "")
(EXP_WAV_LOAD "initialize")

(EXP_NOISE_SIGNAL "set!" 1 "")
(EXP_NOISE_SIGNAL "initialize")

(EXP_WAV_ATTEN "set!" 1 "")
(EXP_WAV_ATTEN "initialize")

(EXP_NONWAV_ATTEN "set!" 1 "")
(EXP_NONWAV_ATTEN "initialize")

(EXP_WAV_POINTER "set!" 1 "")
(EXP_WAV_POINTER "initialize")

; Change the default input channel scale factors to 0.0 (- a lot in dB)
(EXP_CH_SCALES "set!" 1 0.0)
(EXP_CH_SCALES "set!" 2 0.0)
(EXP_CH_SCALES "initialize")

(cond
 (EXP_DEBUGGING?
  (EXP-PRINT (list "EXP-RESET-MODELS exp_ch_scales"))))
```

```
  ; Set the default output scaling to 1.0 (0 dB)
  (EXP_OUTPUT_SCALES "set!" 1 1.0)
  (EXP_OUTPUT_SCALES "set!" 2 1.0)
  (EXP_OUTPUT_SCALES "initialize")

  (cond
   (EXP_DEBUGGING?
    (list (EXP-PRINT "EXP-RESET-MODELS exp_output_scales"))))

  ; Finally, make sure that all current information has been downloaded to
  ;  the DSP board inside the PC
  ;(exp_msg_queue "add" "set_parameters") ;rc
  ;(s-send exp-connection "set_parameters"])  --- no need to bypass this
  (cond
   (EXP_DEBUGGING?
    (EXP-PRINT (list "EXP-RESET-MODELS exp_msg_queue set-parameters"))))
  )

;-----------------------------------------------------------------------
; EXP-SEND-CHANGES -- for any spud models which have been changed, this
;                      function ensures that the real world is affected
;
(define (exp-send-changes)
  (define (make-clean model)
    (model "make-clean"))
  (cond (EXP-DEBUGGING? (EXP-PRINT (list "EXP-SEND-CHANGES"))))

  ; Deal with the number of repetitions FIRST
  (EXP_NREPS         "make-clean")
  (EXP_WAV_LOAD      "make-clean")
  (EXP_WAV_ATTEN     "make-clean")
  (EXP_NONWAV_ATTEN "make-clean")
  (EXP_WAV_POINTER   "make-clean")
  (EXP_NOISE_SIGNAL "make-clean")

  ; Then deal with the decibel lists -- must specify the attenuator to use
  (EXP_LEFT_DB "make-clean" EXP_LATT)
  (EXP_RIGHT_DB "make-clean" EXP_RATT)

  ; then the fourier series tone parameters
  (gfor-each make-clean EXP_FT_CAR_ATTS)
  (gfor-each make-clean EXP_FT_CAR_FREQS)
  (gfor-each make-clean EXP_FT_CAR_PHASES)
  (gfor-each make-clean EXP_FT_MOD_DEPTHS)
  (gfor-each make-clean EXP_FT_MOD_FREQS)
  (gfor-each make-clean EXP_FT_MOD_PHASES)

  ; then the interval times
  (gfor-each
   make-clean
   (list EXP_RESTART_LIST EXP_INITIALIZE_NOISE_LIST
         EXP_IP_LIST EXP_RN_LIST EXP_DS_LIST EXP_RT_LIST
        EXP_OT_LIST EXP_FN_LIST EXP_DF_LIST EXP_FT_LIST))
  ; next, the input channel gains
  (EXP_CH_SCALES "make-clean")
  ; then, the output channel gain
  (EXP_OUTPUT_SCALES "make-clean")

  ; finally, the attenuators
  (EXP_LATT "make-clean")
  (EXP_RATT "make-clean")

  ; then, make sure that the PC sends data structure changes to its
  ;  resident DSP board:
  (exp_msg_queue "add" "set_parameters")
  ;(s-send exp-connection "set_parameters") --- no need to bypass this

  ; Also, flush out the message queue
  (cond (exp_noio?
         (if EXP_NOIO_PRINT?
```

```
            (EXP-PRINT (strcat (exp_msg_queue "flush") "<<END>>")) #t))
        (else (s-send exp-connection
                      (strcat (exp_msg_queue "flush") "<<END>>")))))
;
;-----------------------------------------------------------------------
; EXP-TONE-INTEN -- set the intensity of any intervals tone bursts
;      INTEN: intensity, in dB SPL
;   INTERVAL: which experimental interval to affect (i.e. 1,2,3,4)
;              *** an argument of 0 here means "affect all intervals" ***
;       CHAN: which channel to affect, must be 'left, 'right, or 'both
;
(define (exp-tone-inten inten interval chan)
  (cond (EXP-DEBUGGING?
         (EXP-PRINT (list "EXP-TONE-INTEN" inten interval chan))))
  (cond
   ((if (symbol? chan)
        (eq? chan 'left)
        (eqs? chan "left"))
    (cond ((= interval 0)
           (EXP_LEFT_DB
           "set-list!"
           "tone"
           (exp-create-listvector EXP_MAX_INTVLS inten)))
          (else (EXP_LEFT_DB "set!" "tone" interval inten))))
   ((if (symbol? chan)
        (eq? chan 'right)
        (eqs? chan "right"))
    (cond ((= interval 0)
           (EXP_RIGHT_DB
           "set-list!"
           "tone"
           (exp-create-listvector EXP_MAX_INTVLS inten)))
          (else (EXP_RIGHT_DB "set!" "tone" interval inten))))
   ((if (symbol? chan)
        (eq? chan 'both)
        (eqs? chan "both"))
    (exp-tone-inten inten interval 'left)
    (exp-tone-inten inten interval 'right))
   (else (fatalf "EXP-TONE-INTEN: invalid channel"))))
;-----------------------------------------------------------------------
; EXP-NOISE-INTEN -- set the intensity of any intervals noise burst
;      INTEN: intensity, in dB SPL
;   INTERVAL: which experimental interval to affect (i.e. 1,2,3,4)
;              *** an argument of 0 here means "affect all intervals" ***
;       CHAN: which channel to affect, must be 'left, 'right, 'both
;
(define (exp-noise-inten inten interval chan)
  (cond (EXP-DEBUGGING?
         (EXP-PRINT (list "EXP-NOISE-INTEN" inten interval chan))))
  (cond
   ((if (symbol? chan) (eq? chan 'left) (eqs? chan "left"))
    (cond ((= interval 0)
           (EXP_LEFT_DB
           "set-list!"
           "noise"
           (exp-create-listvector EXP_MAX_INTVLS inten)))
          (else (EXP_LEFT_DB "set!" "noise" interval inten))))
   ((if (symbol? chan) (eq? chan 'right) (eqs? chan "right"))
    (cond ((= interval 0)
           (EXP_RIGHT_DB
           "set-list!"
           "noise"
           (exp-create-listvector EXP_MAX_INTVLS inten)))
          (else (EXP_RIGHT_DB "set!" "noise" interval inten))))
   ((if (symbol? chan) (eq? chan 'both) (eqs? chan "both"))
    (exp-noise-inten inten interval 'left)
    (exp-noise-inten inten interval 'right))
   (else (fatalf "EXP-NOISE-INTEN: invalid channel"))))

;-----------------------------------------------------------------------
; EXP-FREQ -- sets the frequency of any of the tones in any interval
```

```
;               SEE ALSO exp-car-freq, exp-mod-freq at the end of this
;               functions definition.
;       FREQ: the desired frequency
;   INTERVAL: the interval to be affected, i.e. 1,2,3,4
;              *** if this argument = 0, all intervals will be set ***
; <NUM_TONE>: (optional) which tone in the interval to affect
;<CAR-OR-MOD>:(optional) whether to affect the carrier or modulation freq
;              should be either 'car or 'mod
(define (exp-freq freq interval . args)
  (let ((car-or-mod "*unassigned36*") (num_tone "*unassigned37*"))
    (cond (EXP-DEBUGGING? (EXP-PRINT "EXP-FREQ" freq interval num_tone)))
    ; extract only the first optional argument if more than one.  If no
    ;  optional argument, set the tone number to 1
    (cond
     ((null? args) (set! num_tone 1) (set! car-or-mod 'car))
     (else (set! num_tone (car args))
           (if (null? (cdr args)) (set! car-or-mod 'car)
               (set! car-or-mod (cadr args)))))

    ; if the INTERVAL arg is too big, complain
    (cond ((or (< interval 0) (> interval EXP_MAX_INTVLS))
           (fatalf "EXP-FREQ: interval specified %d is out of range" interval))

          ; if the INTERVAL arg is 0, do once for each interval
          ((= interval 0)
           (do ((i EXP_MAX_INTVLS (- i 1)))
               ((= i 0) t)
             (exp-freq freq i num_tone car-or-mod)))
          ; otherwise, change the spud model of frequencies
          ((eq? car-or-mod 'car)
           ((EXP-GET-ELEMENT EXP_FT_CAR_FREQS interval) "set!" num_tone freq))
          ((eq? car-or-mod 'mod)
           ((EXP-GET-ELEMENT EXP_FT_MOD_FREQS interval) "set!" num_tone freq))
          (else (fatalf "EXP-FREQ: recevied invalid car-or-mod argument")))))

(define (exp-car-freq freq interval tone)       ;define two other ways to
  (cond (EXP-DEBUGGING? (EXP-PRINT (list "EXP-CAR-FREQ" freq interval tone))))
  (exp-freq freq interval tone 'car))           ; access this function
(define (exp-mod-freq freq interval tone)
  (cond (EXP-DEBUGGING? (EXP-PRINT (list "EXP-MOD-FREQ" freq interval tone))))
  (exp-freq freq interval tone 'mod))

;---------------------------------------------------------------------
; EXP-PHASE -- sets the phase of any of the tones in any interval
;               SEE ALSO exp-car-freq, exp-mod-freq at the end of this
;               functions definition.
;      PHASE: the desired phase, as a floating-point number
;   INTERVAL: the interval to be affected, i.e. 1,2,3,4
;              *** if this argument = 0, all intervals will be set ***
; <NUM_TONE>: (optional) which tone in the interval to affect
;<CAR-OR-MOD>:(optional) whether to affect the carrier or modulation phase
;              should be either 'car or 'mod
(define (exp-phase phase interval . args)
  (let ((car-or-mod "*unassigned38*") (num_tone "*unassigned39*"))
    (cond (EXP-DEBUGGING? (EXP-PRINT "EXP-PHASE" phase interval num_tone)))
    ; extract only the first optional argument if more than one.  If no
    ;  optional argument, set the tone number to 1
    (cond
     ((null? args) (set! num_tone 1) (set! car-or-mod 'car))
     (else (set! num_tone (car args))
           (if (null? (cdr args)) (set! car-or-mod 'car)
               (set! car-or-mod (cadr args)))))

    ; if the INTERVAL arg is too big, complain
    (cond ((or (< interval 0) (> interval EXP_MAX_INTVLS))
           (fatallf "interval specified is out of range"))

          ; if the INTERVAL arg is 0, do once for each interval
          ((= interval 0)
           (do ((i EXP_MAX_INTVLS (- i 1)))
               ((= i 0) t)
```

```
            (exp-phase phase i num_tone car-or-mod)))
          ; otherwise, change the spud model of phases
          ((eq? car-or-mod 'car)
           ((EXP-GET-ELEMENT EXP_FT_CAR_PHASES interval) "set!" num_tone phase))
          ((eq? car-or-mod 'mod)
           ((EXP-GET-ELEMENT EXP_FT_MOD_PHASES interval) "set!" num_tone phase))
          (else (fatalf "EXP-PHASE: received invalid car-or-mod argument")))))

(define (exp-car-phase phase interval tone)     ; supply some other ways to
  (exp-phase phase interval tone 'car))         ;  access this function
(define (exp-mod-phase phase interval tone)
  (exp-phase phase interval tone 'mod))
;-----------------------------------------------------------------------
; EXP-WAV-LOAD -- specifies a file to be loaded into the wav buffer
;       FILE: the name of the file to be loaded
;
(define (exp-wav-load file)
  (cond (EXP-DEBUGGING?
         (EXP-PRINT (list "EXP-WAV-LOAD" file ))))
  (EXP_WAV_LOAD "set!" 1 file))
;-----------------------------------------------------------------------
; EXP-WAV-ATTEN -- sets an attenuation in dB for the periodic waveform.
;      ATTEN: an amount of attenuation, in dB, to apply to the tone
;
(define (exp-wav-atten atten)
  (cond (EXP-DEBUGGING?
         (EXP-PRINT (list "EXP-WAV-ATTEN" atten ))))
  (EXP_WAV_ATTEN "set!" 1 (- 0.0 atten)))
;-----------------------------------------------------------------------
; EXP-NONWAV-ATTEN -- sets an attenuation in dB for the NON-periodic waveform.
;      ATTEN: an amount of attenuation, in dB, to apply to the tone
;
(define (exp-nonwav-atten atten)
  (cond (EXP-DEBUGGING?
         (EXP-PRINT (list "EXP-NONWAV-ATTEN" atten ))))
  (EXP_NONWAV_ATTEN "set!" 1 (- 0.0 atten)))
;-----------------------------------------------------------------------
; EXP-WAV-POINTER -- sets an *attenuation* in dB for the periodic waveform
;                    a particular interval.
;      INDEX: initial position of waveform buffer pointer
;
(define (exp-wav-pointer index)
  (cond (EXP-DEBUGGING?
         (EXP-PRINT (list "EXP-WAV-POINTER" index ))))
  (EXP_WAV_POINTER "set!" 1 (round index)))
;-----------------------------------------------------------------------
; EXP-NOISE-SIGNAL -- how much noise to treat as signal (not dB)
;   FRACTION: fraction of the noise to treat as signal
;
(define (exp-noise-signal fraction)
  (cond (EXP-DEBUGGING?
         (EXP-PRINT (list "EXP-NOISE-SIGNAL" fraction ))))
  (EXP_NOISE_SIGNAL "set!" 1 fraction))
;-----------------------------------------------------------------------
; EXP-TONE-ATTEN -- sets an *attenuation* in dB for a particular tone in
;                   a particular interval -- this amount will be subtracted
;                   (or added) to the value set with EXP-TONE-INTEN, which
;                   affects all of the tones in the interval.
;      ATTEN: an amount of attenuation, in dB, to apply to the tone
;   INTERVAL: the interval to affect, i.e. 1,2,3,4
;              *** an argument of 0 here means "affect all intervals" ***
;       TONE: which tone to affect
;
(define (exp-tone-atten atten interval tone)
  (cond (EXP-DEBUGGING?
         (EXP-PRINT (list "EXP-TONE-ATTEN" atten interval tone))))
  (cond ((or (< interval 0) (> interval EXP_MAX_INTVLS))
         (fatalf "EXP-TONE-ATTEN: too many or too few intervals specified"))
        ((or (< tone 1) (> tone EXP_MAX_TONES))
         (fatalf "EXP-TONE-ATTEN: too many or too few tones specified"))
        ((= interval 0)
```

```
         (do ((i EXP_MAX_INTVLS (- i 1)))
             ((= i 0) t)
           (exp-tone-atten atten i tone)))
        ; As of 1995, the attenuations are actually interpreted as
        ;  *amplifications* on the PC end, so invert the atten argument
        (else ((EXP-GET-ELEMENT EXP_FT_CAR_ATTS interval) "set!" tone (- 0 atten)))))
;-----------------------------------------------------------------------
; EXP-MOD-DEPTH -- sets the modulation depth of a particular tone in
;                  a particular interval
;      DEPTH: the depth of modulation to use (max=1.0)
;   INTERVAL: the interval to affect, i.e. 1,2,3,4
;             *** an argument of 0 here means "affect all intervals" ***
;       TONE: which tone to affect
;
(define (exp-mod-depth depth interval tone)
  (cond (EXP-DEBUGGING?
         (EXP-PRINT (list "EXP-MOD-DEPTH" depth interval tone) )))
  (cond ((or (< interval 0) (> interval EXP_MAX_INTVLS))
         (fatalf "EXP-MOD-DEPTH: too many or too few intervals specified"))
        ((or (< tone 1) (> tone EXP_MAX_TONES))
         (fatalf "EXP-MOD-DEPTH: too many or too few tones specified"))
        ((= interval 0)
         (do ((i EXP_MAX_INTVLS (- i 1)))
             ((= i 0) t)
           (exp-mod-depth atten i tone)))
        (else ((EXP-GET-ELEMENT EXP_FT_MOD_DEPTHS interval)
               "set!" tone depth))))
;-----------------------------------------------------------------------
; EXP-INITIALIZE-NOISE -- causes the noise filter state variables to be zeroed
;   INTERVAL: the interval to affect
;
(define (exp-initialize-noise interval)
  (cond (EXP-DEBUGGING?
         (EXP-PRINT (list "EXP-INITIALIZE-NOISE" interval) )))
  (cond ((> interval EXP_MAX_INTVLS)
         (fatalf "EXP-INITIALIZE-NOISE: interval %d out of range" interval))
        ((= interval 0)
         (EXP_INITIALIZE_NOISE_LIST
          "set-list!"
          (exp-create-listvector EXP_MAX_INTVLS 1)))
        (else (EXP_INITIALIZE_NOISE_LIST "set!" interval 1))))
;-----------------------------------------------------------------------
; EXP-RESTART-NOISE -- causes the noise state variables to be reset
;   INTERVAL1: the interval to affect, i.e. 1,2,3,4
;   INTERVAL2: the interval to copy < INTERVAL2
;
(define (exp-restart-noise interval1 interval2)
  (cond (EXP-DEBUGGING?
         (EXP-PRINT (list "EXP-RESTART-noise" interval1 interval2) )))
  (cond ((or (> interval1 EXP_MAX_INTVLS) (< interval1 1))
         (fatalf "EXP-RESTART-NOISE: interval1 %d out of range" interval1))
        ((or (> interval2 EXP_MAX_INTVLS) (< interval2 1))
         (fatalf "EXP-RESTART-NOISE: interval2 %d out of range" interval2))
        ((< interval1 interval2)
         (fatalf "EXP-RESTART-NOISE: interval1 (%d) >= interval2 %d"
                 interval1 interval2))
        (else (EXP_RESTART_LIST "set!" interval1 (- interval2 1)))))
;-----------------------------------------------------------------------
; EXP-IP-I -- sets the interval pre-time for a given interval
;   SAMPLES:  the number of samples to set the ip (integer)
;   INTERVAL: the interval to affect, i.e. 1,2,3,4
;             *** an argument of 0 here means "affect all intervals" ***

(define (exp-ip-i samples interval)
  (cond (EXP-DEBUGGING?
         (EXP-PRINT (list "EXP-IP-I" samples interval) )))
  (cond ((= interval 0)
         (EXP_IP_LIST "set-list!"
                      (exp-create-listvector EXP_MAX_INTVLS samples)))
        ((> interval EXP_MAX_INTVLS)
         (fatalf "EXP-IP-I: interval %d out of range") interval)
```

```
        (else (EXP_IP_LIST "set!" interval samples))))
;---------------------------------------------------------------------
; EXP-RN-I -- sets RN  for a given interval
;   SAMPLES:  the number of samples to set the rn (integer)
;   INTERVAL: the interval to affect, i.e. 1,2,3,4
;             *** an argument of 0 here means "affect all intervals" ***
;
(define (exp-rn-i samples interval)
  (cond (EXP-DEBUGGING?
         (EXP-PRINT (list "EXP-RN-I" samples interval) )))
  (cond ((= interval 0)
         (EXP_RN_LIST "set-list!"
                      (exp-create-listvector EXP_MAX_INTVLS samples)))
        ((> interval EXP_MAX_INTVLS)
         (fatalf "EXP-RN-I: interval %d out of range") interval)
        (else (EXP_RN_LIST "set!" interval samples))))
;---------------------------------------------------------------------
; EXP-DS-I -- sets DS for a given interval
;   SAMPLES:  the number of samples to set the ds (integer)
;   INTERVAL: the interval to affect, i.e. 1,2,3,4
;             *** an argument of 0 here means "affect all intervals" ***
;
(define (exp-ds-i samples interval)
  (cond (EXP-DEBUGGING?
         (EXP-PRINT (list "EXP-DS-I" samples interval) )))
  (cond ((= interval 0)
         (EXP_DS_LIST "set-list!"
                      (exp-create-listvector EXP_MAX_INTVLS samples)))
        ((> interval EXP_MAX_INTVLS)
         (fatalf "EXP-DS-I: interval %d out of range" interval))
        (else (EXP_DS_LIST "set!" interval samples))))
;---------------------------------------------------------------------
; EXP-RT-I -- sets RT for a given interval
;   SAMPLES:  the number of samples to set the rt (integer)
;   INTERVAL: the interval to affect, i.e. 1,2,3,4
;             *** an argument of 0 here means "affect all intervals" ***
;
(define (exp-rt-i samples interval)
  (cond (EXP-DEBUGGING?
         (EXP-PRINT (list "EXP-RT-I" samples interval) )))
  (cond ((= interval 0)
         (EXP_RT_LIST "set-list!"
                      (exp-create-listvector EXP_MAX_INTVLS samples)))
        ((> interval EXP_MAX_INTVLS)
         (fatalf "EXP-RT-I: interval %d out of range" interval))
        (else (EXP_RT_LIST "set!" interval samples))))
;---------------------------------------------------------------------
; EXP-OT-I -- sets OT for a given interval
;   SAMPLES:  the number of samples to set the ot (integer)
;   INTERVAL: the interval to affect, i.e. 1,2,3,4
;             *** an argument of 0 here means "affect all intervals" ***
;
(define (exp-ot-i samples interval)
  (cond (EXP-DEBUGGING?
         (EXP-PRINT (list "EXP-OT-I" samples interval) )))
  (cond ((= interval 0)
         (EXP_OT_LIST "set-list!"
                      (exp-create-listvector EXP_MAX_INTVLS samples)))
        ((> interval EXP_MAX_INTVLS)
         (fatalf "EXP-OT-I: interval %d out of range" interval))
        (else (EXP_OT_LIST "set!" interval samples))))
;---------------------------------------------------------------------
; EXP-FN-I -- sets FN for a given interval
;   SAMPLES:  the number of samples to set the fn (integer)
;   INTERVAL: the interval to affect, i.e. 1,2,3,4
;             *** an argument of 0 here means "affect all intervals" ***
;
(define (exp-fn-i samples interval)
  (cond (EXP-DEBUGGING?
         (EXP-PRINT (list "EXP-FN-I" samples interval) )))
  (cond ((= interval 0)
```

```
        (EXP_FN_LIST "set-list!"
                     (exp-create-listvector EXP_MAX_INTVLS samples)))
       ((> interval EXP_MAX_INTVLS)
        (fatalf "EXP-FN-I: interval %d out of range" interval))
       (else (EXP_FN_LIST "set!" interval samples))))
;---------------------------------------------------------------------
; EXP-DF-I -- sets DF for a given interval
;   SAMPLES:  the number of samples to set the df (integer)
;   INTERVAL: the interval to affect, i.e. 1,2,3,4
;             *** an argument of 0 here means "affect all intervals" ***
;
(define (exp-df-i samples interval)
  (cond (EXP-DEBUGGING?
         (EXP-PRINT (list "EXP-DF" samples interval) )))
  (cond ((= interval 0)
         (EXP_DF_LIST "set-list!"
                      (exp-create-listvector EXP_MAX_INTVLS samples)))
        ((> interval EXP_MAX_INTVLS)
         (fatalf "EXP-DF-I: interval %d out of range" interval))
        (else (EXP_DF_LIST "set!" interval samples))))
;---------------------------------------------------------------------
; EXP-FT-I -- sets FT for a given interval
;   SAMPLES:  the number of samples to set the ft (integer)
;   INTERVAL: the interval to affect, i.e. 1,2,3,4
;             *** an argument of 0 here means "affect all intervals" ***
;
(define (exp-ft-i samples interval)
  (cond (EXP-DEBUGGING?
         (EXP-PRINT (list "EXP-FT" samples interval) )))
  (cond ((= interval 0)
         (EXP_FT_LIST "set-list!"
                      (exp-create-listvector EXP_MAX_INTVLS samples)))
        ((> interval EXP_MAX_INTVLS)
         (fatalf "EXP-FT-I: interval %d out of range" interval))
        (else (EXP_FT_LIST "set!" interval samples))))
;---------------------------------------------------------------------
; EXP-MS-TO-SAMPLES -- converts a time spec in ms to a number of samples
;       TIME: the length of time, in msec, to set the ip (float)
; since the sample rate is stored in kHz, this is easy.

  (define (exp-ms-to-samples time)
    (round (* EXP_SAMPLE_RATE time)))

;---------------------------------------------------------------------
; EXP-IP -- sets the interval pre-time for a given interval
;       TIME: the length of time, in msec, to set the ip (float)
;   INTERVAL: the interval to affect, i.e. 1,2,3,4
;             *** an argument of 0 here means "affect all intervals" ***
;
(define (exp-ip time interval)
  (cond (EXP-DEBUGGING?
         (EXP-PRINT (list "EXP-IP" time interval) )))
  (cond ((> interval EXP_MAX_INTVLS)
         (fatalf "EXP-IP: interval %d out of range" interval))
        (else (exp-ip-i (exp-ms-to-samples time) interval))))
;---------------------------------------------------------------------
; EXP-RN -- sets the RN time for a given interval
;       TIME: the length of time, in msec, to set the rn (float)
;   INTERVAL: the interval to affect, i.e. 1,2,3,4
;             *** an argument of 0 here means "affect all intervals" ***
;
(define (exp-rn time interval)
  (cond (EXP-DEBUGGING?
         (EXP-PRINT (list "EXP-RN" time interval) )))
  (cond ((> interval EXP_MAX_INTVLS)
         (fatalf "EXP-RN: interval %d out of range" interval))
        (else (exp-rn-i (exp-ms-to-samples time) interval))))
;---------------------------------------------------------------------
; EXP-DS -- sets the DS time for a given interval
;       TIME: the length of time, in msec, to set the ds (float)
;   INTERVAL: the interval to affect, i.e. 1,2,3,4
```

```
;             *** an argument of 0 here means "affect all intervals" ***
;
(define (exp-ds time interval)
  (cond (EXP-DEBUGGING?
         (EXP-PRINT (list "EXP-DS" time interval) )))
  (cond ((> interval EXP_MAX_INTVLS)
         (fatalf "EXP-DS: interval %d out of range" interval))
        (else (exp-ds-i (exp-ms-to-samples time)))))
;-------------------------------------------------------------------------
; EXP-RT -- sets the RT time for a given interval
;       TIME: the length of time, in msec, to set the rt (float)
;   INTERVAL: the interval to affect, i.e. 1,2,3,4
;             *** an argument of 0 here means "affect all intervals" ***
;
(define (exp-rt time interval)
  (cond (EXP-DEBUGGING?
         (EXP-PRINT (list "EXP-RT" time interval) )))
  (cond ((> interval EXP_MAX_INTVLS)
         (fatalf "EXP-RT: interval %d out of range" interval))
        (else (exp-rt-i (exp-ms-to-samples time) interval))))
;-------------------------------------------------------------------------
; EXP-OT -- sets the OT time for a given interval
;       TIME: the length of time, in msec, to set the ot (float)
;   INTERVAL: the interval to affect, i.e. 1,2,3,4
;             *** an argument of 0 here means "affect all intervals" ***
;
(define (exp-ot time interval)
  (cond (EXP-DEBUGGING?
         (EXP-PRINT (list "EXP-OT" time interval) )))
  (cond ((> interval EXP_MAX_INTVLS)
         (fatalf "EXP-OT: interval %d out of range" interval))
        (else (exp-ot-i (exp-ms-to-samples time) interval))))
;-------------------------------------------------------------------------
; EXP-FN -- sets the FN time for a given interval
;       TIME: the length of time, in msec, to set the fn (float)
;   INTERVAL: the interval to affect, i.e. 1,2,3,4
;             *** an argument of 0 here means "affect all intervals" ***
;
(define (exp-fn time interval)
  (cond (EXP-DEBUGGING?
         (EXP-PRINT (list "EXP-FN" time interval) )))
  (cond ((> interval EXP_MAX_INTVLS)
         (fatalf "EXP-FN: interval %d out of range" interval))
        (else (exp-fn-i (exp-ms-to-samples time) interval))))
;-------------------------------------------------------------------------
; EXP-DF -- sets the DF time for a given interval
;       TIME: the length of time, in msec, to set the df (float)
;   INTERVAL: the interval to affect, i.e. 1,2,3,4
;             *** an argument of 0 here means "affect all intervals" ***
;
(define (exp-df time interval)
  (cond (EXP-DEBUGGING?
         (EXP-PRINT (list "EXP-DF" time interval) )))
  (cond ((> interval EXP_MAX_INTVLS)
         (fatalf "EXP-DF: interval %d out of range" interval))
        (else (exp-df-i (exp-ms-to-samples time) interval))))
;-------------------------------------------------------------------------
; EXP-FT -- sets the FT time for a given interval
;       TIME: the length of time, in msec, to set the ft (float)
;   INTERVAL: the interval to affect, i.e. 1,2,3,4
;             *** an argument of 0 here means "affect all intervals" ***
;
(define (exp-ft time interval)
  (cond (EXP-DEBUGGING?
         (EXP-PRINT (list "EXP-FT" time interval) )))
  (cond ((> interval EXP_MAX_INTVLS)
         (fatalf "EXP-FT: interval %d out of range" interval))
        (else (exp-ft-i (exp-ms-to-samples time) interval))))
;-------------------------------------------------------------------------
; EXP-GO -- tells the PC to actually play the experiment intervals
(define (exp-go)
```

```
  (cond (EXP-DEBUGGING? (EXP-PRINT (list "EXP-GO" ))))
  (exp_msg_queue "add" "go")
  (cond (exp_noio?
         (if EXP_NOIO_PRINT?
             (EXP-PRINT (strcat (exp_msg_queue "flush") "<<END>>")) #t))
        (else (s-send exp-connection
                      (strcat (exp_msg_queue "flush") "<<END>>")))))
;-----------------------------------------------------------------------
; EXP-INTERVALS -- sets the actual number of intervals
;    ints: an integer specifying the number of intervals
;
(define (exp-intervals ints)
  (cond (EXP-DEBUGGING? (EXP-PRINT (list "EXP-NINTS" ints))))
  (cond ((> ints EXP_MAXX_INTVLS)
         (fatalf
          "EXP-INTERVALS too many intervals %d > %d\n"
          ints EXP_MAXX_INTVLS)))
  (set! EXP_MAX_INTVLS ints))

;-----------------------------------------------------------------------
; EXP-REPETITIONS -- sets the actual number of repetitions
;    ireps: an integer specifying the number of repetitions
;
(define (exp-repetitions ireps)
  (cond (EXP-DEBUGGING? (EXP-PRINT (list "EXP-REPETITIONS" ireps))))
  (EXP_NREPS "set!" 1 ireps))
;-----------------------------------------------------------------------
; EXP-TONES -- sets the actual number of intervals
;    itones: an integer specifying the number of tones
;
(define (exp-tones itones)
  (cond (EXP-DEBUGGING? (EXP-PRINT (list "EXP-NTONES" itones))))
  (cond ((> itones EXP_MAXX_TONES)
         (fatalf
          "EXP-TONES too many tones %d > %d\n" itones EXP_MAXX_TONES)))
  (set! EXP_MAX_TONES itones))

;-----------------------------------------------------------------------
; EXP-SAMPLE-RATE -- lets RUNEXP use the same sample rate as the DSP board
;    SRATE: a new sample rate, in kHz (must be precise, e.g., 11.025)
;
(define (exp-sample-rate srate)
  (cond (EXP-DEBUGGING? (EXP-PRINT (list "EXP-SAMPLE-RATE" srate))))
  (set! EXP_SAMPLE_RATE srate))
;-----------------------------------------------------------------------
; EXP-INPUT-GAIN -- sets the gains (hence the scale factors) applied to the
;                   input channels of the DSP board before they are mixed
;                   into one of the generated-signal channels
;       GAIN: a gain factor, in dB, to apply to the input channel
;    CHANNEL: which input channel to affect, 'right, 'left, or 'both
(define (exp-input-gain gain channel)
  (cond (EXP-DEBUGGING? (EXP-PRINT (list "EXP-INPUT-GAIN" gain channel) )))
  (cond ((if (symbol? chan) (eq? chan 'right) (eqs? chan "right"))
         (EXP_CH_SCALES "set!" 2 (exp-db2scale gain)))
        ((if (symbol? chan) (eq? chan 'left) (eqs? chan "left"))
         (EXP_CH_SCALES "set!" 1 (exp-db2scale gain)))
        ((if (symbol? chan) (eq? chan 'both) (eqs? chan "both"))
         (exp-input-gain gain 'left) (exp-input-gain gain 'right))
        (else
         (fatalf "EXP-INPUT-GAIN: received an invalid channel"))))
;-----------------------------------------------------------------------
; EXP-OUTPUT-GAIN -- sets the gain (hence scale factor) applied at the output
;       GAIN: a gain factor, in dB, to apply to the DSP board output
;    CHANNEL: which input channel to affect, 'right, 'left, or 'both
(define (exp-output-gain gain channel)
  (cond (EXP-DEBUGGING? (EXP-PRINT (list "EXP-OUTPUT-GAIN" gain channel) )))
  (cond ((if (symbol? chan) (eq? chan 'right) (eqs? chan "right"))
         (EXP_OUTPUT_SCALES "set!" 2 (exp-db2scale gain)))
        ((if (symbol? chan) (eq? chan 'left) (eqs? chan "left"))
         (EXP_OUTPUT_SCALES "set!" 1 (exp-db2scale gain)))
        ((if (symbol? chan) (eq? chan 'both) (eqs? chan "both"))
```

```
         (exp-output-gain gain 'left) (exp-output-gain gain 'right))
        (else
         (fatalf "EXP-OUTPUT-GAIN: received an invalid channel"))))
;-----------------------------------------------------------------------
; EXP-NOISE-BANDWIDTH -- changes the default bandwidth of the noise so
;       that EXP-NOISE-INTEN sets the level properly.
;       BANDWIDTH: the bandwidth, in Hz, of the noise signal.
;
;   NOTE: this simply changes the global variable EXP_NOISE_DEFAULT.
(define (exp-noise-bandwidth bandwidth)
  (cond ((= 0.0 bandwidth)
         (error "EXP-NOISE-BANDWIDTH: bandwidth cant be 0.0!")))
  (set! EXP_NOISE_DEFAULT
        (+ EXP_NOISE_DEF100
           (* 10.0 (log. (/ bandwidth 100.0))))))
;-----------------------------------------------------------------------
;exp-get-response
(define (exp-get-response filename numresponses)
  (set! EXP-RESPONSE (s-send exp-connection (format "<<MSG>> resp %s %d <<END>>" filename
numresponses))))

;exp-phase-diff
(define (exp-phase-diff phase)
  (s-send exp-connection (format "<<MSG>> phasediff %f <<END>>" phase)))

;exp-filter-noise
(define (exp-filter-noise filtertype . args)
  (cond ((if (symbol? filtertype) (eq? filtertype 'highpass) (eqs? filtertype
"highpass"))
         (s-send exp-connection (format "<<MSG>> filter %s %d <<END>>" filtertype (car
args))))
        ((if (symbol? filtertype) (eq? filtertype 'lowpass) (eqs? filtertype "lowpass"))
         (s-send exp-connection (format "<<MSG>> filter %s %d <<END>>" filtertype (car
args))))
        ((if (symbol? filtertype) (eq? filtertype 'bandpass) (eqs? filtertype
"bandpass"))
         (s-send exp-connection (format "<<MSG>> filter %s %d %d <<END>>" filtertype
(car args) (cadr args))))))

;=======================================================================
```

Meta.c

```
#include <stdlib.h>
#include <stdio.h>
#include <math.h>
#include <string.h>
#include "engine.h"
#include "matlab.h"
#include "sockets.h"
#include "include/mod.h"

/* defines */
//debug flag
#define DEBUG 1

//lab hardware sensitivity
#define TONE_DEFAULT (98.0)
#define NOISE_DEFAULT (98.0)

#define MAX_ARGS 200
#define MAX_MESSAGE_LENGTH 4096

void rpterr (char *);

#define sign(x) ((x>=0.0) ? +1.0 : -1.0)

/* globals */
Engine *ep;
//parameters
double wav_atten;
double wav_sc_l;
double wav_sc_r;
//M-statement buffers
char wav_load_eval[25];
char wav_load_eval2[25];
//play wave file flag
int wav_load;
//mxArray variables
mxArray *phsdiff = NULL;
mxArray *lpfcut=NULL, *hpfcut=NULL, *bpfcut1=NULL, *bpfcut2=NULL;
//response variables
mxArray *y = NULL;
double *pyreal;
mxArray *z = NULL;
double *pzreal;

int fill_in_args(char **, char *);
char return_string[MAXCHAR]; /* returned by "meta()" function */


char *meta(char **messagePtr) {
    int i;
    char command[MAXCHAR];
    char args[MAX_ARGS][MAXCHAR];
    int argnum;
    long tmp;

    //response variables
    mxArray *x = NULL;
    double *pxreal;

    //stimuli generation variables
    mxArray *nbw1 = NULL, *nbw2 = NULL, *nbw3 = NULL, *nbw4 = NULL;

    //debug variables
    //char resp[10];
    int numresps;

    if(strcmp("@PING",*messagePtr)==0) {
        printf("Received a @PING signal from the Espud client.\n");
```

```
    *messagePtr = NULL;
    return("@PING");
}

/* Loop through processing it until it is set to NULL by "fill_in_args" */
argnum = fill_in_args(messagePtr,&(args[0][0]));

/* Here is where the message will be interpreted */
if(argnum==0) {
    printf("command string: no arguments: \n%s\n",*messagePtr);
    return("?No command string received");
}
strcpy(command,args[0]);
if (DEBUG) printf("\nCOMMAND: %s\n",command);

/*** DEBUG ***/
if (strcmp("debug",command)==0) {
    if (sscanf(args[1],"%d",&debug) != 1) {
            printf("?ERROR: debug needs a debug level (integer number)\n");
            return("?ERROR: debug needs a debug level (integer number)");
    }
}

/*** RESET_STATE: load in the MOD.H values of the parameters ***/
if (strcmp("reset_state",command)==0) {
    //reset_state(); //obsolete
    //setup_parameters(); //obsolete
    return(" Ok");
}

/*** SET_PARAMETERS: calculate all of the experimental parameters again ***/
if (strcmp("set_parameters",command)==0) {
    //setup_parameters(); //obsolete
    //download_memory(); //obsolete
    return(" Ok-set_parameters");
}

/*** GO: cause the intervals to actually play ***/
if (strcmp("go",command)==0) {
    go_new();
    //go();
    return(" Ok-go");
}

/*** GO-FILTERED-NOISE: cause the intervals to actually play ***/
if (strcmp("go-filtered-noise",command)==0) {
    go_filtered_noise();
    return(" Ok-go-filtered-noise");
}

/*** GO_EXTENDED: cause the intervals to actually play ***/
if (strcmp("go_extended",command)==0) {
    go_new();
    return(" Ok-go_extended");
}

/*** NINTS: specifies the number of intervals in the experiment ***/
if (strcmp("nints",command)==0) {
    if (nints > MAX_DI) {
            return("?nints: exceeds maximum allowable intervals");
    }
    else sscanf(args[1],"%d",&nints);
    return(" Ok");
}

/*** NTONES: specifies the number of tones in the experiment ***/
if (strcmp("ntones",command)==0) {
    if (ntones > MAX_TONES) {
            return("?ntones: exceeds maximum allowable tones");
    }
    else sscanf(args[1],"%d",&ntones);
```

```
        return(" Ok");
    }

    /*** CH_SCALE: scale the signal in channels A & B of the DSP board by ***/
    /*** this amount before mixing in to the signal ***/
    if (strcmp("ch_scales",command)==0) {
        printf("ch_scales: empty command\n");
        return(" Ok");
    }

    /*** OUTPUT_SCALES: scale the signal at the output by this amount before ***/
    /*** sending it to the d2a converters ***/
    if (strcmp("output_scales",command)==0) {
        printf("output_scales: empty command\n");
        return(" Ok");
    }

    /*** SAMPLE RATE: change the sample rate ***/
    if (strcmp("sample", command)==0) {
        printf("sample: empty command\n");
        return(" Ok");
    }

    /*** REPORT: spud is asking that we send it our state. This function will ***/
    /*** return a string to spud telling it what it wants to know ***/
    if (strcmp("report", command)==0) {
        printf("report: empty command\n");
        return(" Ok");
    }

/*-------------------------------------------------------------------------*/

    /*** IP_LIST: spud is sending a list of times to fill the ip[] array ***/
    if (strcmp(args[0],"ip_list")==0) {
        if (argnum < (nints + 1)) {
                printf("ERROR: ip_list received too few intervals\n");
                return("?ip_list: not enough arguments");
        }
        /* fill up the ip time array with the passed floats */
        for (i=0; i<nints; i++)
                sscanf(args[i+1],"%f",&(ip[i]));
        /* print ip time array */
        if (DEBUG) {
                printf("ip:");
                for (i=0; i<nints; i++) printf(" %f",ip[i]);
                printf("\n");
        }
        return(" Ok");

    }

    /*** RN_LIST: spud is sending a list of times to fill the rn[] array ***/
    if (strcmp(args[0],"rn_list")==0) {
        if (argnum < (nints + 1)) {
                printf("ERROR: rn_list received too few intervals\n");
                return("?rn_list: not enough arguments");
        }
        /* fill up the rn time array with the passed floats */
        for (i=0; i<nints; i++)
                sscanf(args[i+1],"%f",&(rn[i]));
        /* print rn time array */
        if (DEBUG) {
                printf("rn:");
                for (i=0; i<nints; i++) printf(" %f",rn[i]);
                printf("\n");
        }
        return(" Ok");
    }

    /*** DS_LIST: spud is sending a list of times to fill the ds[] array ***/
    if (strcmp(args[0],"ds_list")==0) {
```

```
    if (argnum < (nints + 1)) {
            printf("ERROR: ds_list received too few intervals\n");
            return("?ds_list: not enough arguments");
    }
    /* fill up the ds time array with the passed floats */
    for (i=0; i<nints; i++)
            sscanf(args[i+1],"%f",&(ds[i]));
    /* print ds time array */
    if (DEBUG) {
            printf("ds:");
            for (i=0; i<nints; i++) printf(" %f",ds[i]);
            printf("\n");
    }
    return(" Ok");
}

/*** RT_LIST: spud is sending a list of times to fill the rt[] array ***/
if (strcmp(args[0],"rt_list")==0) {
    if (argnum < (nints + 1)) {
            printf("ERROR: rt_list received too few intervals\n");
            return("?rt_list: not enough arguments");
    }
    /* fill up the rt time array with the passed floats */
    for (i=0; i<nints; i++)
            sscanf(args[i+1],"%f",&(rt[i]));
    /* print rt time array */
    if (DEBUG) {
            printf("rt:");
            for (i=0; i<nints; i++) printf(" %f",rt[i]);
            printf("\n");
    }
    return(" Ok");
}

/*** OT_LIST: spud is sending a list of times to fill the ot[] array ***/
if (strcmp(args[0],"ot_list")==0) {
    if (argnum < (nints + 1)) {
            printf("ERROR: ot_list received too few intervals\n");
            return("?ot_list: not enough arguments");
    }
    /* fill up the ot time array with the passed floats */
    for (i=0; i<nints; i++)
            sscanf(args[i+1],"%f",&(ot[i]));
    /* print ot time array */
    if (DEBUG) {
            printf("ot:");
            for (i=0; i<nints; i++) printf(" %f",ot[i]);
            printf("\n");
    }
    return(" Ok");
}

/*** FN_LIST: spud is sending a list of times to fill the fn[] array ***/
if (strcmp(args[0],"fn_list")==0) {
    if (argnum < (nints + 1)) {
            printf("ERROR: fn_list received too few intervals\n");
            return("?fn_list: not enough arguments");
    }
    /* fill up the fn time array with the passed floats */
    for (i=0; i<nints; i++)
            sscanf(args[i+1],"%f",&(fn[i]));
    /* print fn time array */
    if (DEBUG) {
            printf("fn:");
            for (i=0; i<nints; i++) printf(" %f",fn[i]);
            printf("\n");
    }
    return(" Ok");
}

/*** DF_LIST: spud is sending a list of times to fill the df[] array ***/
```

```
    if (strcmp(args[0],"df_list")==0) {
        if (argnum < (nints + 1)) {
                printf("ERROR: df_list received too few intervals\n");
                return("?df_list: not enough arguments");
        }
        /* fill up the df time array with the passed floats */
        for (i=0; i<nints; i++)
                sscanf(args[i+1],"%f",&(df[i]));
        /* print df time array */
        if (DEBUG) {
                printf("df:");
                for (i=0; i<nints; i++) printf(" %f",df[i]);
                printf("\n");
        }
        return(" Ok");
    }

    /*** FT_LIST: spud is sending a list of times to fill the ft[] array ***/
    if (strcmp(args[0],"ft_list")==0) {
        if (argnum < (nints + 1)) {
                printf("ERROR: ft_list received too few intervals\n");
                return("?ft_list: not enough arguments");
        }
        /* fill up the ft time array with the passed floats */
        for (i=0; i<nints; i++)
                sscanf(args[i+1],"%f",&(ft[i]));
        /* print ft time array */
        if (DEBUG) {
                printf("ft:");
                for (i=0; i<nints; i++) printf(" %f",ft[i]);
                printf("\n");
        }
        return(" Ok");
    }

/*-------------------------------------------------------------------------*/

    /*** FT_CAR_ATTS: sending the attenuations on all of the tones ***/
    /*** in a particular interval ***/
    if (strcmp(args[0],"ft_car_atts")==0) {
        if (sscanf(args[1],"%d",&j) != 1) {
                printf("ERROR: ft_car_atts needs an interval number\n");
                return("?ft_car_atts needs an interval number");
        }
        j--;    /* interval #1 is actually array element #0 */
        for (i=0; i<ntones; i++)
                sscanf(args[i+2],"%f",&(att[j][i]));
        /* print ft_car_atts */
        if (DEBUG) {
        }
        return(" Ok");
    }

    /*** FT_CAR_FREQS: sending the carrier frequencies of the fourier ***/
    /*** series in a particular interval ***/
    if (strcmp(args[0],"ft_car_freqs")==0) {
        if (sscanf(args[1],"%d",&j) != 1) {
                printf("ERROR: ft_car_freqs needs an interval number\n");
                return("?ft_car_freqs needs an interval number");
        }
        j--;    /* interval #1 is actually array element #0 */
        for (i=0; i<ntones; i++)
        {
                sscanf(args[i+2],"%f",&(frq[j][i]));
                printf("DEBUG: ft car freq %i, interval %i = %f\n", i, j, frq[j][i]);
        }
        /* print ft_car_freqs */
        if (DEBUG) {
        }
        return(" Ok");
    }
```

```
    /*** FT_CAR_PHASES: it is sending the carrier phases of the ***/
    /*** series tones in a particular interval ***/
    if (strcmp(args[0],"ft_car_phases")==0) {
        if (sscanf(args[1],"%d",&j) != 1) {
                printf("ERROR: ft_car_phases needs an interval number\n");
                return("?ft_car_phases needs an interval number");
        }
        j--;    /* interval #1 is actually array element #0 */
        for (i=0; i<ntones; i++)
                sscanf(args[i+2],"%f",&(crz[j][i]));
        /* print ft_car_phases */
        if (DEBUG) {
        }
        return(" Ok");
    }

    /*** FT_MOD_DEPTHS: sending the modulation depths of the fourier ***/
    /*** series tones in a particular interval ***/
    if (strcmp(args[0],"ft_mod_depths")==0) {
        if (sscanf(args[1],"%d",&j) != 1) {
                printf("ERROR: ft_mod_depths needs an interval number\n");
                return("?ft_mod_depths needs an interval number");
        }
        j--;    /* interval #1 is actually array element #0 ***/
        for (i=0; i<ntones; i++)
                sscanf(args[i+2],"%f",&(mod[j][i]));
        /* print ft_mod_depths */
        if (DEBUG) {
        }
        return(" Ok");
    }

    /*** FT_MOD_FREQS: sending the modulation frequencies of the ***/
    /*** fourier series tones in a particular interval ***/
    if (strcmp(args[0],"ft_mod_freqs")==0) {
        if (sscanf(args[1],"%d",&j) != 1) {
                printf("ERROR: ft_mod_freqs needs an interval number\n");
                return("?ft_mod_freqs needs an interval number");
        }
        j--;    /* interval #1 is actually array element #0 */
        for (i=0; i<ntones; i++)
                sscanf(args[i+2],"%f",&(mfr[j][i]));
        /* print ft_mod_freqs */
        if (DEBUG) {
        }
        return(" Ok");
    }

    /*** FT_MOD_PHASES: it is sending the modulation phases of the ***/
    /*** fourier series tones in a particular interval ***/
    if (strcmp(args[0],"ft_mod_phases")==0) {
        if (sscanf(args[1],"%d",&j) != 1) {
                printf("ERROR: ft_mod_phases needs an interval number\n");
                return("?ft_mod_phases needs an interval number");
        }
        j--;    /* interval #1 is actually array element #0 */
        for (i=0; i<ntones; i++)
                sscanf(args[i+2],"%f",&(amz[j][i]));
        /* print ft_mod_phases */
        if (DEBUG) {
        }
        return(" Ok");
    }

/*-------------------------------------------------------------------------*/

    /*** RIGHT_SCALES: spud is sending us a list of scale factors to apply to ***/
    /*** the right-channel intervals ***/
    if (strcmp(args[0],"right_scales")==0) {
        if (argnum < 3) {
```

```
            printf("ERROR: right_scales received too few arguments\n");
            return("?right_scales: too few arguments");
        }
        /* fill up the srt[] or srn[] arrays with the passed floats */
        if (strcmp(args[1],"tone")==0)
            for (i=0;i<nints; i++) {
                sscanf(args[i+2],"%f",&(srt[i]));
                if (DEBUG) printf("DEBUG: tone scale %i = %g\n", i, srt[i]);
            }
        else if (strcmp(args[1],"noise")==0)
            for (i=0; i<nints; i++) {
                sscanf(args[i+2],"%f",&(srn[i]));
                if (DEBUG) printf("DEBUG: noise scale %i = %g\n", i, srn[i]);
            }
        else {
            printf("ERROR: right_scales didn't receive 'tone' or 'noise'\n");
            return("?right_scales: didn't receive 'tone' or 'noise'");
        }
        if (DEBUG) {
            /* print srt, srn arrays */
        }
        return(" Ok");
    }

    /*** LEFT_SCALES: spud is sending us a list of scale factors to apply to ***/
    /*** the left-channel intervals ***/
    if (strcmp(args[0],"left_scales")==0) {
        if (argnum < 3) {
            printf("ERROR: left_scales received too few arguments\n");
            return("?left_scales: too few arguments");
        }
        /* fill up the slt[] or sln[] arrays with the passed floats */
        if (strcmp(args[1],"tone")==0)
            for (i=0; i<nints; i++) {
                sscanf(args[i+2],"%f",&(slt[i]));
                if (DEBUG) printf("DEBUG: tone scale %i = %g\n", i, slt[i]);
            }
        else if (strcmp(args[1],"noise")==0)
            for (i=0; i<nints; i++) {
                sscanf(args[i+2],"%f",&(sln[i]));
                if (DEBUG) printf("DEBUG: noise scale %i = %g\n", i, sln[i]);
            }
        else {
            printf("ERROR: left_scales didn't receive 'tone' or 'noise'\n");
            return("?left_scales: didn't receive 'tone' or 'noise'");
        }
        if (DEBUG) {
            /* print slt, sln arrays */
        }
        return(" Ok");
    }

/*-------------------------------------------------------------------------*/

    if (strcmp(args[0],"wav_load")==0) {
        sscanf(args[1],"%s",&file);
        sprintf(wav_load_eval,"[stm,fs,nbits]=wavread('%s.wav');",file);
        //sprintf(wav_load_eval2,"siz=wavread('%s.wav','size');",file);
        engEvalString(ep,wav_load_eval);
        //engEvalString(ep,wav_load_eval2);
        wav_load=1;
        return(" Ok");
    }

    if (strcmp(args[0],"wav_atten")==0) {
        sscanf(args[1],"%f",&wav_atten);
        return(" Ok");
    }

/*-------------------------------------------------------------------------*/
```

```
    //External miscellaneous experiment commands
    //low-pass filtered noise
    if (strcmp(args[0],"noisebw")==0) {
        //printf("%s\n",args[1]);

        //sscanf(args[1],"%f",&(nbw[0]));
        //printf("%f\n",nbw[0]);
        //sscanf(args[2],"%f",&(nbw[1]));
        //printf("%f\n",nbw[1]);
        //sscanf(args[3],"%f",&(nbw[2]));
        //printf("%f\n",nbw[2]);
        //sscanf(args[4],"%f",&(nbw[3]));
        //printf("%f\n",nbw[3]);

        for (i=0;i<4;i++) {
               sscanf(args[1],"%f",&(nbw[i]));
               //printf("nbw %d = %f\n",i,nbw[i]);
        }

        mlfAssign(&nbw1, mlfScalar(nbw[0]));
        mlfAssign(&nbw2, mlfScalar(nbw[1]));
        mlfAssign(&nbw3, mlfScalar(nbw[2]));
        mlfAssign(&nbw4, mlfScalar(nbw[3]));
        mxSetName(nbw1, "nbw1");
        mxSetName(nbw2, "nbw2");
        mxSetName(nbw3, "nbw3");
        mxSetName(nbw4, "nbw4");
        engPutArray(ep, nbw1);
        engPutArray(ep, nbw2);
        engPutArray(ep, nbw3);
        engPutArray(ep, nbw4);

//  engEvalString(ep, "[b1 a1]=butter(10,nbw1/22050);");
//  engEvalString(ep, "n1f=filter(b1,a1,n1);");
//  engEvalString(ep, "noiseint1l=[maip1,nssc1l.*n1f];");
//  engEvalString(ep, "noiseint1r=[maip1,nssc1r.*n1f];");

        return(" Ok-nbw");
    }

    //all filtered noise types
    if (strcmp(args[0],"filter")==0) {
        if (strcmp(args[1],"lowpass")==0) {
               sscanf(args[2],"%f", &lpfc);
               mlfAssign(&lpfcut,mlfScalar(lpfc));
               mxSetName(lpfcut,"lpfcut");
               engPutArray(ep,lpfcut);
               return(" Ok-lpf");
        }
        if (strcmp(args[1],"highpass")==0) {
               sscanf(args[2],"%f",&hpfc);
               mlfAssign(&hpfcut,mlfScalar(hpfc));
               mxSetName(hpfcut,"hpfcut");
               engPutArray(ep,hpfcut);
               lpfc=0; //turn off lpf
               return(" Ok-hpf");
        }
        if (strcmp(args[1],"bandpass")==0) {
               sscanf(args[2],"%f",&bpfc1);
               sscanf(args[3],"%f",&bpfc2);
               mlfAssign(&bpfcut1,mlfScalar(bpfc1));
               mlfAssign(&bpfcut2,mlfScalar(bpfc2));
               mxSetName(bpfcut1,"bpfcut1");
               mxSetName(bpfcut2,"bpfcut2");
               engPutArray(ep,bpfcut1);
               engPutArray(ep,bpfcut2);
               engEvalString(ep,"bpfcut=[bpfcut1 bpfcut2];");
               lpfc=0; //turn off lpf
               hpfc=0; //turn off hpf
               return(" Ok-bpf");
        }
```

```
    }

    //phase difference in the left and right channels
    if (strcmp(args[0],"phase_diff")==0) {
        sscanf(args[1],"%f",&phasediff);
        mlfAssign(&phsdiff, mlfScalar(phasediff));
        mxSetName(phsdiff, "phasediff");
        engPutArray(ep, phsdiff);
        return(" Ok-phasediff");
    }

/*--------------------------------------------------------------------------*/

    if (strcmp(args[0],"resp")==0) {

        engEvalString(ep, "x=[];");
        engEvalString(ep, args[1]);
        while(1)
        {
                x = engGetArray(ep, "x");
                pxreal = mxGetPr(x);
                if (pxreal != 0) break;
        }
        if (pxreal[0] == 1) {
                return("1");
        }
        if (pxreal[0] == 2) {
                return("2");
        }
    }

/*--------------------------------------------------------------------------*/

    printf("ERROR: unknown command received!\n");
    printf("command: %s, args[1]: %s\n",command,args[1]);
    return("?Unknown command");

}


/* void parse1(char c , char *clarg) {} */

void rpterr (char *mess) {
    printf(mess);
    exit(0);
}

/* void get_val(char *carg, int *got, float *val) {} */
/* void get_tone_params(char *carg) {} */

explain(void) {
    printf("\nN-Interval AM-Tone/Noise Burst Waveform.\n");
    exit(1);
}

bad_value() {
    printf("\nError: Bad value passed on command line.\n");
    explain();
}

/*--------------------------------------------------------------------------*/

void reset_state() {
    printf("reset_state(): empty function\n");
}

void setup_parameters(void) {
    printf("setup_parameters(): empty function\n");
}

void download_memory(void) {
```

```
    printf("download_memory(): empty function\n");
}

/*-------------------------------------------------------------------------*/
//rc
/*
void go(void) {
    printf("Sending the RESTART command,\n");
    //cmd_m96(0,DFLT,CMD_RESTARTCOMMAND);
    if (debug >= 100) printf("Restart signal sent!\n");
}
*/


void go_() {

    char wav_play_eval[21];

    int i,t;
    int MAX_STR=100; //max string size

    char stm_length_init[100][20];
    char stm_length_eval[100][20];
    char stm_l_init[100][20];
    char stm_r_init[100][20];

    char ipstr[100][10];
    char rtstr[100][10];
    char ftstr[100][10];
    char otstr[100][10];
    char ipastr[100][10];

    char *** frqstr;
    char *** crzstr;

    char sltstr[100][10];
    char srtstr[100][10];
    char slnstr[100][10];
    char srnstr[100][10];

    char mip_convert[100][30];
    char mrt_convert[100][30];
    char mft_convert[100][30];
    char mot_convert[100][30];
    char maip_convert[100][30];

    char t_eval[100][50];
    char n_eval[100][50];
    char filter_eval[100][50];

    char tnsc_l_eval[100][50];
    char tnsc_r_eval[100][50];
    char *** tone_l_eval;
    char *** tone_r_eval;
    char nssc_l_eval[100][50];
    char nssc_r_eval[100][50];
    char noise_l_eval[100][50];
    char noise_r_eval[100][50];
    char *** stm_l_tones_eval;
    char *** stm_r_tones_eval;
    char stm_l_eval[100][50];
    char stm_r_eval[100][50];
    char stm_l_eval2[100][50];
    char stm_r_eval2[100][50];
    char stm_intvl_eval[100][50];
    char stm_eval[100][50];

    mxArray *mip[100];
    mxArray *mrn[100];
    mxArray *mds[100];
    mxArray *mrt[100];
```

```
mxArray *mot[100];
mxArray *mfn[100];
mxArray *mdf[100];
mxArray *mft[100];
mxArray *maip[100];
mxArray *mfrq[100][10];
mxArray *mcrz[100][10];
mxArray *mslt[100];
mxArray *msrt[100];
mxArray *msln[100];
mxArray *msrn[100];

if (wav_load)
{
    //wav_sc_l=pow(10,-TONE_DEFAULT/20)*slt[0];
    //wav_sc_r=pow(10,-TONE_DEFAULT/20)*srt[0];
    sprintf(wav_play_eval,"stm=[%g*stm,%g*stm];",slt[0],srt[0]);
    engEvalString(ep,wav_play_eval);
    sprintf(wav_load_eval2,"siz=wavread('%s.wav','size');",file);
    engEvalString(ep,wav_load_eval2);
    engEvalString(ep,"sound(stm,fs);");
    //wait after playing wav file
    engEvalString(ep,"pause(siz(1,1)/fs);");
    wav_load=0;
}
else
{
    //initialize mxArrays
    for(i=0;i<100;i++) {
            mip[i]=NULL;
            mrn[i]=NULL;
            mds[i]=NULL;
            mrt[i]=NULL;
            mot[i]=NULL;
            mfn[i]=NULL;
            mdf[i]=NULL;
            mft[i]=NULL;
            maip[i]=NULL;
            for(t=0;t<10;t++) {
                    mfrq[i][t]=NULL;
                    mcrz[i][t]=NULL;
            }
            mslt[i]=NULL;
            msrt[i]=NULL;
            msln[i]=NULL;
            msrn[i]=NULL;
    }

    if (DEBUG) printf("%d-interval experiment...\n",nints);

    //If no phase difference between left and right channels specified, set to 0
    if (phasediff==0) {
            mlfAssign(&phsdiff, mlfScalar(phasediff));
            mxSetName(phsdiff, "phasediff");
            engPutArray(ep, phsdiff);
    }

    //INITIALIZE STIMULUS VECTORS
    //initialize stm#(l/r) to matrix of zeroes for the length of the trial
    for(i=0;i<nints;i++) {
            //TIMING ARGUMENTS
            mlfAssign(&mrn[i],mlfScalar(rn[i]));
            mlfAssign(&mds[i],mlfScalar(ds[i]));
            mlfAssign(&mfn[i],mlfScalar(fn[i]));
            mlfAssign(&mdf[i],mlfScalar(df[i]));

            //pretimes
            mlfAssign(&mip[i],mlfScalar(ip[i]));
            sprintf(ipstr[i],"mip%d",i+1);
            mxSetName(mip[i],ipstr[i]);
            engPutArray(ep,mip[i]);
```

```
                //risetimes
                mlfAssign(&mrt[i],mlfScalar(rt[i]));
                sprintf(rtstr[i],"mrt%d",i+1);
                mxSetName(mrt[i],rtstr[i]);
                engPutArray(ep,mrt[i]);
                //falltimes
                mlfAssign(&mft[i],mlfScalar(ft[i]));
                sprintf(ftstr[i],"mft%d",i+1);
                mxSetName(mft[i],ftstr[i]);
                engPutArray(ep,mft[i]);
                //ontimes
                mlfAssign(&mot[i],mlfScalar(ot[i]));
                sprintf(otstr[i],"mot%d",i+1);
                mxSetName(mot[i],otstr[i]);
                engPutArray(ep,mot[i]);

                //pretime arrays
                mlfAssign(&maip[i],mlfZeros(mlfScalar(1),mip[i],NULL));
                sprintf(ipastr[i],"maip%d",i+1);
                mxSetName(maip[i],ipastr[i]);
                engPutArray(ep,maip[i]);

                if (lpfc!=0 || hpfc!=0 || bpfc1!=0 || bpfc2!=0) {
                        //DEBUG
                        //printf("Converting to 44.1kHz...\n");
                        engEvalString(ep,"fs=44100;");
                        sprintf(mip_convert[i],"mip%d=mip%d*4;",i+1,i+1);
                        engEvalString(ep,mip_convert[i]);
                        sprintf(mrt_convert[i],"mrt%d=mrt%d*4;",i+1,i+1);
                        engEvalString(ep,mrt_convert[i]);
                        sprintf(mft_convert[i],"mft%d=mft%d*4;",i+1,i+1);
                        engEvalString(ep,mft_convert[i]);
                        sprintf(mot_convert[i],"mot%d=mot%d*4;",i+1,i+1);
                        engEvalString(ep,mot_convert[i]);

    sprintf(maip_convert[i],"maip%d=[maip%d,maip%d,maip%d,maip%d];",i+1,i+1,i+1,i+1,i+1);
                        engEvalString(ep,maip_convert[i]);
                }
                else engEvalString(ep,"fs=11025;");

                sprintf(stm_length_init[i],"stmlength%d=0;",i+1);

    sprintf(stm_length_eval[i],"stmlength%d=stmlength%d+mip%d+mrt%d+mot%d+mft%d;",i+1,i+1,
i+1,i+1,i+1,i+1);

                engEvalString(ep,stm_length_init[i]);
                engEvalString(ep,stm_length_eval[i]);
        }
        for(i=0;i<nints;i++) {
                sprintf(stm_l_init[i],"stm%dl=zeros(1,stmlength%d);",i+1,i+1);
                sprintf(stm_r_init[i],"stm%dr=zeros(1,stmlength%d);",i+1,i+1);

                engEvalString(ep,stm_l_init[i]);
                engEvalString(ep,stm_r_init[i]);
        }
        //initialize the trial vector to the empty matrix
        engEvalString(ep,"stm=[];");

        frqstr=calloc(sizeof(char **),nints);
        crzstr=calloc(sizeof(char **),nints);
        tone_l_eval=calloc(sizeof(char **),nints);
        tone_r_eval=calloc(sizeof(char **),nints);
        stm_l_tones_eval=calloc(sizeof(char **),nints);
        stm_r_tones_eval=calloc(sizeof(char **),nints);

        for(i=0;i<nints;i++) {
                frqstr[i]=calloc(sizeof(char *),ntones);
                crzstr[i]=calloc(sizeof(char *),ntones);
                for(t=0;t<ntones;t++) {
                        // Fourier series carrier frequencies
                        mlfAssign(&mfrq[i][t],mlfScalar(frq[i][t]));
```

```
            frqstr[i][t]=malloc(MAX_STR);
            sprintf(frqstr[i][t],"f%dint%d",t+1,i+1);
            mxSetName(mfrq[i][t],frqstr[i][t]);
            engPutArray(ep,mfrq[i][t]);

            // Fourier series phases
            mlfAssign(&mcrz[i][t],mlfScalar(crz[i][t]));
            crzstr[i][t]=malloc(MAX_STR);
            sprintf(crzstr[i][t],"f%di%dcz",t+1,i+1);
            mxSetName(mcrz[i][t],crzstr[i][t]);
            engPutArray(ep,mcrz[i][t]);
        }

        //tone and noise scales
        mlfAssign(&mslt[i],mlfScalar(slt[i]));
        sprintf(sltstr[i],"mslt%d",i+1);
        mxSetName(mslt[i],sltstr[i]);
        engPutArray(ep,mslt[i]);
        mlfAssign(&msrt[i],mlfScalar(srt[i]));
        sprintf(srtstr[i],"msrt%d",i+1);
        mxSetName(msrt[i],srtstr[i]);
        engPutArray(ep,msrt[i]);
        mlfAssign(&msln[i],mlfScalar(sln[i]));
        sprintf(slnstr[i],"msln%d",i+1);
        mxSetName(msln[i],slnstr[i]);
        engPutArray(ep,msln[i]);
        mlfAssign(&msrn[i],mlfScalar(srn[i]));
        sprintf(srnstr[i],"msrn%d",i+1);
        mxSetName(msrn[i],srnstr[i]);
        engPutArray(ep,msrn[i]);

        //FILTER NOISE

        //DEBUG
        //printf("lpfc=%f\n",lpfc);
        //printf("hpfc=%f\n",hpfc);
        //printf("bpfc1=%f\n",bpfc1);
        //printf("bpfc2=%f\n",bpfc2);

        sprintf(t_eval[i],"t%d=[1:(mot%d+mrt%d+mft%d)];",i+1,i+1,i+1,i+1);
        engEvalString(ep,t_eval[i]);
        sprintf(n_eval[i],"n%d=randn(1,(mot%d+mrt%d+mft%d));",i+1,i+1,i+1,i+1);
        engEvalString(ep,n_eval[i]);

        if (lpfc!=0) {
            //DEBUG
            //printf("lowpass filtering...\n");
            engEvalString(ep,"[b1,a1]=butter(10,lpfcut/22050);");
            sprintf(filter_eval[i],"n%d=filter(b1,a1,n%d);",i+1,i+1);
            engEvalString(ep,filter_eval[i]);
        }
        else if (hpfc!=0) {
            //DEBUG
            //printf("highpass filtering...\n");
            engEvalString(ep, "[b1,a1]=butter(10,hpfcut/22050,'high');");
            sprintf(filter_eval[i],"n%d=filter(b1,a1,n%d);",i+1,i+1);
            engEvalString(ep,filter_eval[i]);
        }
        else if (bpfc1!=0 || bpfc2!=0) {
            //DEBUG
            //printf("bandpass filtering...\n");
            engEvalString(ep, "[b1,a1]=butter(5,bpfcut/22050,'stop');");
            sprintf(filter_eval[i],"n%d=filter(b1,a1,n%d);",i+1,i+1);
            engEvalString(ep,filter_eval[i]);
        }

        //CONSTRUCT STIMULI


    sprintf(tnsc_1_eval[i],"tnsc%dl=mslt%d*hannfl(mrt%d+mot%d+mft%d,mrt%d,mft%d)';",i+1,i+
1,i+1,i+1,i+1,i+1,i+1);
```

```
    sprintf(tnsc_r_eval[i],"tnsc%dr=msrt%d*hannfl(mrt%d+mot%d+mft%d,mrt%d,mft%d)';",i+1,i+
1,i+1,i+1,i+1,i+1,i+1);
                engEvalString(ep,tnsc_l_eval[i]);
                engEvalString(ep,tnsc_r_eval[i]);

                //for all tones in an interval
                tone_l_eval[i]=calloc(sizeof(char *),ntones);
                tone_r_eval[i]=calloc(sizeof(char *),ntones);
                for(t=0;t<ntones;t++) {
                        tone_l_eval[i][t]=malloc(MAX_STR);
                        tone_r_eval[i][t]=malloc(MAX_STR);

    sprintf(tone_l_eval[i][t],"tone%dint%dl=[maip%d,tnsc%dl.*sin(2*pi*f%dint%d*t%d/fs+f%di
%dcz)];",t+1,i+1,i+1,i+1,t+1,i+1,i+1,t+1,i+1);

    sprintf(tone_r_eval[i][t],"tone%dint%dr=[maip%d,tnsc%dr.*sin(2*pi*f%dint%d*t%d/fs+f%di
%dcz+phasediff)];",t+1,i+1,i+1,i+1,t+1,i+1,i+1,t+1,i+1);
                        engEvalString(ep,tone_l_eval[i][t]);
                        engEvalString(ep,tone_r_eval[i][t]);
                }


    sprintf(nssc_l_eval[i],"nssc%dl=msln%d*hannfl(mrt%d+mot%d+mft%d,mrt%d,mft%d)';",i+1,i+
1,i+1,i+1,i+1,i+1,i+1);

    sprintf(nssc_r_eval[i],"nssc%dr=msrn%d*hannfl(mrt%d+mot%d+mft%d,mrt%d,mft%d)';",i+1,i+
1,i+1,i+1,i+1,i+1,i+1);
                engEvalString(ep,nssc_l_eval[i]);
                engEvalString(ep,nssc_r_eval[i]);


    sprintf(noise_l_eval[i],"noiseint%dl=[maip%d,nssc%dl.*n%d];",i+1,i+1,i+1,i+1);

    sprintf(noise_r_eval[i],"noiseint%dr=[maip%d,nssc%dr.*n%d];",i+1,i+1,i+1,i+1);
                engEvalString(ep,noise_l_eval[i]);
                engEvalString(ep,noise_r_eval[i]);

                //eqv: engEvalString(ep,
"stm1l=(tone1int1l+tone2int1l+tone3int1l+tone4int1l+tone5int1l+tone6int1l+tone7int1l+tone
8int1l+noiseint1l)';");
                stm_l_tones_eval[i]=calloc(sizeof(char *),ntones);
                stm_r_tones_eval[i]=calloc(sizeof(char *),ntones);
                for(t=0;t<ntones;t++) {
                        stm_l_tones_eval[i][t]=malloc(MAX_STR);
                        stm_r_tones_eval[i][t]=malloc(MAX_STR);
                        //eqv: stm(i)(l/r)=stm(i)l+tone(t)int(i)(l/r)

    sprintf(stm_l_tones_eval[i][t],"stm%dl=stm%dl+tone%dint%dl;",i+1,i+1,t+1,i+1);

    sprintf(stm_r_tones_eval[i][t],"stm%dr=stm%dr+tone%dint%dr;",i+1,i+1,t+1,i+1);
                        engEvalString(ep,stm_l_tones_eval[i][t]);
                        engEvalString(ep,stm_r_tones_eval[i][t]);
                }
                sprintf(stm_l_eval[i],"stm%dl=stm%dl+noiseint%dl;",i+1,i+1,i+1);
                sprintf(stm_r_eval[i],"stm%dr=stm%dr+noiseint%dr;",i+1,i+1,i+1);
                sprintf(stm_l_eval2[i],"stm%dl=stm%dl';",i+1,i+1);
                sprintf(stm_r_eval2[i],"stm%dr=stm%dr';",i+1,i+1);

                sprintf(stm_intvl_eval[i],"stm%d=[stm%dl,stm%dr];",i+1,i+1,i+1);

                sprintf(stm_eval[i],"stm=[stm;stm%d];",i+1);

                engEvalString(ep,stm_l_eval[i]);
                engEvalString(ep,stm_r_eval[i]);
                engEvalString(ep,stm_l_eval2[i]);
                engEvalString(ep,stm_r_eval2[i]);
                engEvalString(ep,stm_intvl_eval[i]);
                engEvalString(ep,stm_eval[i]);
      }
        engEvalString(ep, "sound(stm,fs,16);");
```

```
    }
}


void FatalError(char *msg) {
    fprintf(stderr, "ERROR: %s\n", msg);
    exit(1);
}

/* This chops up a string into fields separated by spaces-- the array filled */
/*  by fill_in_args is two-dimensional, and passed as the second argument */
int fill_in_args(char **messagePtr, char *args) {
    char *message = *messagePtr;
    char *place;
    char tmpString[64];
    int i=0,j;

    /* Strip off the first "<<MSG>> " string */
    strncpy(tmpString, message, 7);
    tmpString[7] = '\0';       /* NUL-terminate it */
    if (strcmp(tmpString, "<<MSG>>") != 0)
        FatalError("meta: fill_in_args: message didn't start with <<MSG>>.");
    /* Skip to the meat of the message */
    message += 8;
    place = message;

    while (sscanf(place,"%s",(args+i*MAXCHAR))==1) {
        /* Test to see if we've hit a new message or the end of a message */
        if (strcmp((args+i*MAXCHAR), "<<MSG>>") == 0) {
                *messagePtr = place;
                break;
        }
    if (strcmp((args+i*MAXCHAR), "<<END>>") == 0) {
        *messagePtr = NULL;
        break;
    }

    /* Otherwise we got a standard message string */
    i++;
    while ((*place != ' ') && (*place != '\0'))
        place++;
    if (*place == '\0')
        break;
    while ((*place == ' ') && (*place != '\0'))
        place++;
    if (*place == '\0')
        break;
    }
    for (j=i; j<MAX_ARGS; j++)/* For each possible argument which did not */
        *(args+j*MAXCHAR) = '\0';     /*  arrive, set its string to ""(null string)*/
    return(i);
}

/*-------------------------------------------------------------------------*/

int main(int argc, char *argv[])
{
    Socket *srvr;
    Socket *skt;
    char buffer[MAX_MESSAGE_LENGTH];
    //char buffer[40960]; //DEBUG
    char *charPtr, *ret;

    //debug variables
//  char arg[10][10];
//  mxArray *x[10];
//  mxArray *y[10];
//  mxArray *z=NULL;
//  int a;
    //TEST
    //vector<mxArray *> vect;
```

```
printf("Meta build 3.2.33\n");
if (DEBUG) printf("DEBUG MODE\n");

ep = engOpen("\0"); /* open MATLAB engine */
//if in debug mode, make MATLAB engine session invisible
if (DEBUG) {engSetVisible(ep,1);}
else engSetVisible(ep,0);

sleep(1);
engEvalString(ep, "global x;"); //response variable
engEvalString(ep, "global y;");
engEvalString(ep, "global z;");

printf("\nOpening up a server socket...\n");
if ((srvr = Sopen("runexp","S")) == NULL) {
    printf("ERROR: Can't create a server-- are you sure Spm is running?\n");
    exit(1);
}
printf("\tAwaiting a connection...\n");
if ((skt = Saccept(srvr)) == NULL) {
    printf("ERROR: Can't accept.\n");
    exit(1);
}
printf("\tWaiting for \"@PING\"...\n");
if (Sgets(buffer, 1024, skt) == NULL) {
    printf("ERROR: Can't receive a string.\n");
    exit(1);
}
if (strcmp(buffer, "@PING") != 0) {
    printf("ERROR: Didn't receive the \"@PING\" signal-- instead got:\n");
    printf("%s.\n", buffer);
    exit(1);
}
/* Echo the PING signal back */
printf("\tGot the PING signal.\n");
Sputs("@PING", skt);

/* Debug: echo the socket string */
//while (Sgets(buffer,40960,skt) != NULL) {
//  printf("\n%s\n\n",buffer);
//  Sputs(buffer,skt);
//}

/* Now go into the echo loop */
while (1) {
    if (Sgets(buffer, MAX_MESSAGE_LENGTH, skt) == NULL) {
            printf("ERROR: Can't receive a string!\n");
            Sclose(skt);
            Sclose(srvr);
            exit(1);
            }
    charPtr = buffer;
    if (DEBUG) printf("\n%s\n",charPtr); /* print the message buffer from Runexp */

    while (charPtr != NULL) {

            ret = meta(&charPtr);
            if (DEBUG) printf("meta: %s\n",ret);
            if ((strcmp(ret," Ok")!=0) && (strcmp(ret,"?Unknown command")!=0))
            {
                    Sputs(ret, skt);
                    printf("%s\n",ret);
            }
            //Sputs(ret, skt); //RCC

            /* If the return value was the reserved string "@TERM" then quit. */
            if (strcmp(ret,"@TERM")==0) break;

    }
```

```
        if (strcmp(ret,"@TERM")==0) {
                printf("\tGot the \"@TERM\" message, so quitting.\n");
                break;
        }
    }

    Sclose(skt);
    Sclose(srvr);
    engClose(ep); /* close Matlab engine */
    return 0;
}
```

Mod.h

```
#define MAX_STAGES (20)
#define MAX_REPS (2)
#define MAX_DI (100)          /* maximum number of different intervals */
#define MAX_INTVLS (MAX_DI)   /* maximum number of intervals */
#define MAX_TONES (8)         /* maximum number of tones per interval */
#define N_COEFFS (5)
#define N_SVARS (4)
#define MAX_AMP (10000.0)
#define MIN_AMP (1.0)
#define NOISE_AMP (1.0)
#define MAXLINE (100)
#define PI (3.141592654)
#define TWOPI (2.0*PI)
#define QPI (PI/4.0)
#define N_FIXED_OSC (2)
#define N_RFT_OSC (2)
#define N_VAR_OSC (N_RFT_OSC+2*MAX_TONES+N_RFT_OSC)
#define N_OSCPAR (3)
#define N_ATTPAR (4)
#define N_INTPAR (11)
#define CMD_SRATE (0x90)
#define CMD_STARTCOMMAND (0x91)
#define CMD_RESTARTCOMMAND (0x92)
#define YES (1)
#define NO (0)
#define LTA (0)
#define RTA (1)
#define LNA (2)
#define RNA (3)
#define RN (0)
#define RT (1)
#define MD (0)
#define ON (1)
#define FT (0)
#define FN (1)

#define MAXCHAR (128)

void parse0 (char, char *);
void parse1 (char, char *);
void Srate(int);
void reset_state(void);
void setup_parameters(void);
void download_memory(void);
void go(void);
void go_filtered_noise(void);
void go_new(void);

/* Static data */
long rec, srindex=2;
short srmasks[12]=
{0x20,0x21,0x22,0x23,0x24,0x25,0x26,0x27,0x28,0x29,0x2a,0x2b};
float srates[12]=
{6.0,8.0,11.025,12.0,16.0,22.05,24.0,32.0,44.1,48.0,64.0,96.0};

int got_one_or_two;
int got_init, got_update, got_p0, got_p1;
int got_start, got_restart;
int got_sr, got_mn;
int got_lnatt, got_rnatt, got_cf, got_bw, got_stages;
int n_tones[MAX_DI];
int got_slt[MAX_DI], got_srt[MAX_DI];
int got_sln[MAX_DI], got_srn[MAX_DI];
int got_ip[MAX_DI],  got_rt[MAX_DI];
int got_ot[MAX_DI],  got_ft[MAX_DI];
int got_nr[MAX_DI],  got_nf[MAX_DI];
int got_ds[MAX_DI],  got_df[MAX_DI];
int got_br[MAX_DI],  got_bf[MAX_DI];
```

```
int got_infile, got_outfile, got_file;
int got_ftone, got_fnoise;
int got_nints, got_nreps, got_fint[MAX_DI];
int got_par,got_debug;
long stages=2;
float   lnatt=0.0,
        rnatt=0.0,
        lnamp=1.0,
        rnamp=1.0,
        bw=100.0,
        cf=1000.0,
        lnfact=1.0,
        rnfact=1.0;
unsigned long nints=4,nreps=1,ntones=8;
float   att[MAX_DI][MAX_TONES],              /* F.S. tone attenuations */
        amp[MAX_DI][MAX_TONES],              /* This is calculated from att[][] */
        frq[MAX_DI][MAX_TONES],              /* F.S. carrier frequencies */
        mfr[MAX_DI][MAX_TONES],              /* F.S. modulator frequencies */
        mod[MAX_DI][MAX_TONES],              /* F.S. modulation depths */
        fmf[MAX_DI],                    /* (does not appear in MODB.C) */
        fma[MAX_DI],                    /* (does not appear in MODB.C) */
        crz[MAX_DI][MAX_TONES],              /* F.S. carrier phase */
        amz[MAX_DI][MAX_TONES],              /* F.S. modulator phase */
        fmz[MAX_DI],                    /* (does not appear in MODB.C) */
        slt[MAX_DI]={0.1,0.1,0.1,0.1},       /* left intervals tone scales */
        srt[MAX_DI]={0.1,0.1,0.1,0.1},       /* right intervals tone scales */
        sln[MAX_DI]={0.1,0.1,0.1,0.1},       /* left intervals noise scales */
        srn[MAX_DI]={0.1,0.1,0.1,0.1},       /* right intervals noise scales */

        ip[MAX_DI]={20.0,20.0,20.0,20.0},    /* Next 8 items are times */
        ds[MAX_DI]={20.0,20.0,20.0,20.0},    /*  marking the durations */
        df[MAX_DI]={20.0,20.0,20.0,20.0},    /*  of the tone/noise     */
        rt[MAX_DI]={20.0,20.0,20.0,20.0},    /*  bursts in the */
        ot[MAX_DI]={300.0,300.0,300.0,300.0}, /*  intervals */
        ft[MAX_DI]={20.0,20.0,20.0,20.0},
        rn[MAX_DI]={10.0,10.0,10.0,10.0},
        fn[MAX_DI]={10.0,10.0,10.0,10.0},

        fint[MAX_DI]={1.0,1.0,1.0,1.0},      /* Scale factor applied to intervals */
        ftone=1.0,     /* scale factors applied to all tones */
        fnoise=1.0,    /* scale factors applied to all noise */

        nbw[MAX_DI]={0.1,0.1,0.1,0.1}, /* noise bandwidths */
        phasediff=0.0, /* phase difference, in radians, between the left and right
channels */
        lpfc=0.0, hpfc=0.0, bpfc1=0.0, bpfc2=0.0; /* cutoff frequencies for filtered noise
*/

FILE *fp;
char infile[MAXLINE], outfile[MAXLINE], file[MAXLINE],
     line[MAXLINE], semicolon[]=";";
```