Overview of NeuroDreamer^{TM} Sleep Mask Software

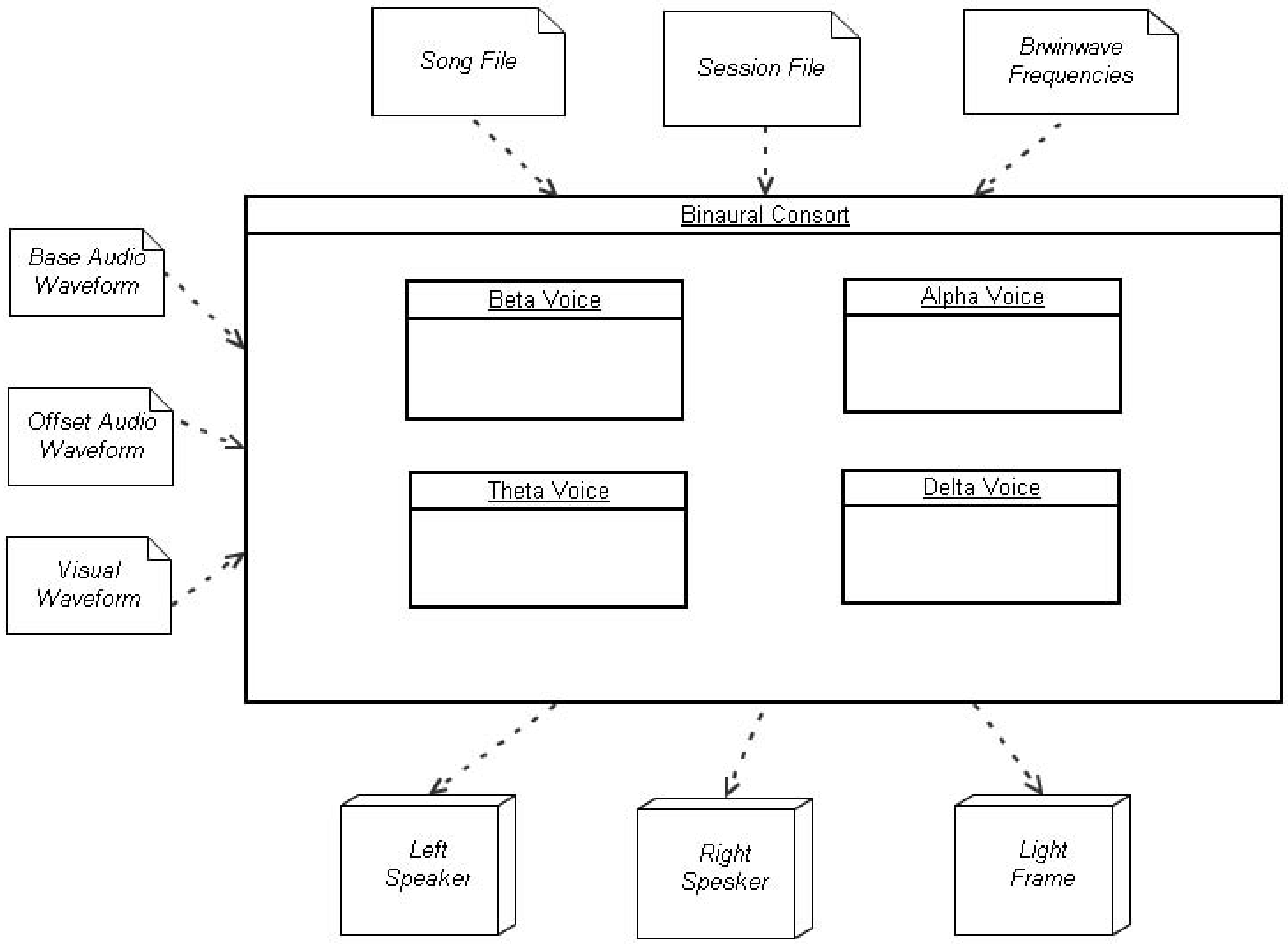
The challenge in developing the 1st version of the NeuroDreamer sleep mask was to fit an 8 voice tone generator and synchronized light frame together with 4 songs, 2 sleep sessions, ambient noise generation, and real time encoding of brainwaves into a 20Mhz 8 bit processor with 8K of ROM, 256 bytes of RAM, an 8 cycle multiply and no sound chip. To accomplish this the firmware is entirely data driven via tables that have been compressed and pre-computed by an authoring system. The firmware might be described as a clock that does not know the time it is telling While the firmware is written in z8 assembly language, the authoring system is written in C++.

The authoring system currently compiles as a native DOS program (and can be run in a Windows command window) using the Watcom C++ compiler, or as a Linux console program, using the.GNU C++ compiler. It uses a commanf line interface, so that all user interface actions can be replicated in batch mode via text command files. The inputs to the authoring system are given via command line switches and parameters, indirect command files, text files that describe songs, sessions, and auxillary data structures, and .wav files that define one or more audio waveforms. The outputs of the authoring system are z8 assembler files which are then assembled together with the firmware proper to create an instantiation of the firmware that can be run in the NeuroDreamer sleep mask.

To understand how the firmware works it is necessary to have a conceptual overview of the software components and algorithms that comprise a sound and light entrainment device in the abstract. The authoring system implements these in a full working prototype; and then uses the software objects in this prototype to author the tables and operating constants that drive the firmware.

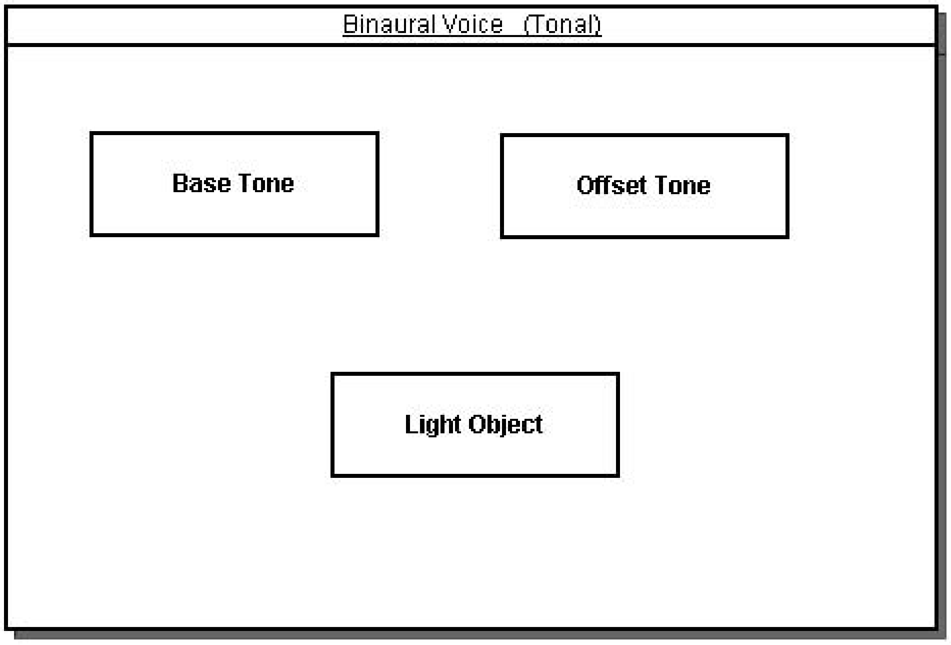
Entrainment device system diagram

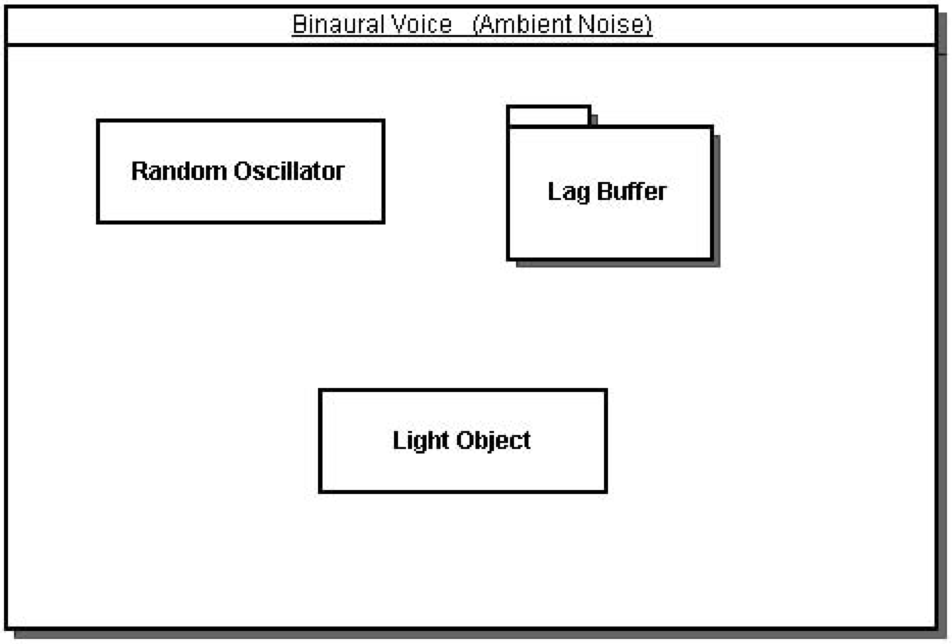
The central software object in the entrainment device is the binaural consort, which is composed of four binaural voice objects, one for each of the brainwave ranges: beta, alpha, theta, and delta. It processes input from song and session files, a brainwave frequency table, base and offset audio waveforms, a visual waveform, and produces audio and synchronized light output.



Binaural voice diagram

Binaural voice objects can be of 2 different types, a tonal voice comprised of 2 tone objects or an ambient noise voice comprised of a noise oscillator and an accompanying lag buffer. Each voice type contains a light object.





Description of inputs to the entrainment device

The brainwave frequency table species the frequencies for each brainwave range: beta, alpha, theta, and delta. These will be applied as inputs to the corresponding voices in the binaural consort.

The session file is a list of records, implicitly ordered by time, describing the intensity of the brainwaves and how the intensity of each should vary over time. For example, an initial record of (Init 0, 20, 10, 30, 50) means “set the initial intensity of the alpha brainwave to 20%, the beta brainwave to 10%, the delta brainwave to 30%, and the theta brainwave to 50%.” A subsequent record of (Fade 15, 5, 20, 10, 40) means “over the next 15 seconds fade the intensity of the alpha brainwave to 5%, the beta brainwave to 20%, the delta brainwave to 10%, and the theta brainwave to 40%.” .

A song file is a list of variable-length records, implicitly ordered by time, describing the pitch and duration of a note, and which voice to ascribe it to. The pitch is described in relative terms, as the distance in semi-tones and micro-tones from the song tonic.There are also repeat, strain, and loop records which allow for the building up of traditional song structures.. If the binaural consort employs noise oscillators instead of tones, the song file is ignored.

An audio waveform is as an array of signed values, generally a digitized sample of an instrument sounding a constant note. This sample is then digitally pre-processed to create a seamless aural connection between the beginning and end of the waveform. The sample rate used to record the instrument, and the pitch of the note sounded by the instrument are kept as the operating constants *sample_rate*, and *original_pitch*. In general there are two audio waveform inputs to the entrainment device, one used by base tones, and one by offset tones (as explained further down). If the binaural consort employs noise oscillators instead of tones, the waveforms are ignored.

The visual waveform, or signal table, is an array of positive values, ranging from maximum luminosity to zero, which describe a single wavelength of the visual pulse.

The basic entrainment algorithm

The main algorithm iterates the consort once per tick (one tick equals one moment of audio output), and then uses the consort's newly iterated state values to produce an audio stream and synchronized light frame that pulsates at prescribed brainwave frequencies. The manner in which the audio is made to "pulsate" depends on whether the consort is composed of voices that employ tones, or voices that employ ambient noise oscillators.

In the case of a binaural consort that is composed of voices that employ tones, the following method is used to iterate the individual voices and induce audio entrainment:

> The binaural voice object is used to produce a binaural beat while playing a single part in a four part song. Each binaural voice has an associated brainwave: alpha, beta, delta, or theta.
>
> The binaural voice consists of two tone objects, the *Base_tone* and the *Offset_tone*, and a light object. A state variable *beat_frequency* is set at startup to the frequency of the voice's associated brainwave. When the voice is instructed to play a given note ("A" for example), the pitch of the *Base_tone* is set to the corresponding frequency (440 Hz), and the pitch of the *Offset_tone* is set to this frequency plus the frequency of the associated brainwave (440Hz + *beat_frequency*).
>
> When the part changes notes the process is repeated.
>
> Each tone mainrtains a moving index into it's reference waveform. This index is incremented, with wrap, once per iteration. By changing this increment, or *step*, any desired pitch can be produced. In general:
>
> ```
> step = (sample_rate/playback_rate) * (desired_pitch/original_pitch)
> ```
>
> Since a change of pitch in a tone object consists merely in computing a new *step* value without disturbing the value of *index*, the note being sounded by the binaural voice can be freely changed without affecting the phase or continuity of the binaural beat.
>
> The *Offset_tone* is scaled by the current intensity of the associated brainwave, as derived from the ongoing processing of the session file.

In the case of a binaural consort that is composed of voices that employ ambient noise oscillators, the following method is used to iterate the individual voices and induce audio entrainment:

> The binaural voice object is used to phase shift a noise source at an associated brainwave frequency: alpha, beta, delta, or theta. The voice object maintains a state variable *bw_cycle_position*, which is updated in such a way that it oscillates from 1.0 to –1.0 and back, in the period of one wavelength of the brainwave.
>
> Within the voice object there is a random noise oscillator (in this case a brownian oscillator), a circular buffer of prior oscillator values (the lag buffer), and a light object.
>
> On each iteration a new value for the noise oscillator is calculated and saved. For a brownian oscillator this is done by adding a pseudo-random value (constrained to lie within a certain range) to the last value of the oscillator. The last value of the oscillator is saved in the lag buffer and the buffer's head/tail indices are updated. The *bw_cycle_position* is then consulted. If it is positive the current value of the oscillator is output to the left audio channel; if it is negative the current value is output to the right audio channel. The absolute value of *bw_cycle_position* is then multipled by the length of the lag buffer, and scaled by the current intensity of the associated brainwave frequency. This is then used as a relative index from the head of the lag buffer, the value at that index in the lag buffer is fetched and output to the opposite audio channel from that in which the current oscillator value was output. This acheives a phase shift at the brainwave frequency and intensity.

Both types of binaural voices contain a light object. This object mainains state information used to pulsate a light regularly at the rate of its associated brainwave frequency. It does this by referencing a signal table of values giving the amplitude at each point for one wavelength of a light pulse, with an *index* which is incremented each iteration by *step*. The value of *step* is initially calculated from the brainwave frequency associated with the light object, and insures that the signal table will be traversed once per brainwave period. The value of the signal table element, once scaled by the current intensity of the associated brainwave, is considered to be an illumination percentage, which is converted into a pulse-width value and used to drive the Light Object's respective LED in the Light Frame.

Description of the firmware

The z8 firmware can be viewed structurally as a foreground task loop, an ISR, and a number of semaphores which help off-load the processing of non time-critical events to the foreground. The "meat" of the firmware exists in the ISR, which runs at the audio rate, and is responsible for iterating the consort, producing the instantaneous audio and LED values, maintaining the timing of periodic update events, processing song events, and initiating the parsing of session records. Because there is not enough RAM to buffer output this is all done in real time – the audio rate is only as good as the worst case ISR. To mitigate the worst case ISR an event deferral mechanism is implemented for distributing the processing of simultaneous and time-consuming song and session events amongst consecutive ISRs.

The algorithmic processing of the session table is the same for both tone sequences and ambient noise sequences. A session table is comprised of variable length records and is generated by the authoring system from a text desciptor file. These records are of various types, and control the activity and duration of the sequence. For purposes of this overview we will discuss only two record types: HOLD, and FADE. . The firmware maintains a down-counter, denominated in seconds, which indcates when the next session record should be processed. A HOLD record, when processed, will bump this counter by a specified number. A FADE record, when processed, will cause the firmware to incrementally alter the intensity of one or more brainwaves by given amounts over a given number of updates, and also bump the down-counter. The incremental amounts in the FADE record are pre-computed by the authoring system and are a function of the intensities for the brainwaves as given in the original text desciptor records, and the *fade update rate,* as configured via the authoring sytem. For example, if the original text descriptor for a session record was 'FADE 20 15" (which means "Fade over 20 seconds the beta brainwave from it's current intensity level to a 15% intensity level"),. and the current intensity of the beta brainwave was 2%, then the authoring system would have pre-computed an incremental fade value of (15-2) / (20*_fade_update_rate_) One of the tasks in the firmware task loop is to update ongoing fades at the *fade_update_rate*. The periodicity is maintained via a semaphore initiated by the ISR.

The algorithmic control of the LEDs is the same for both tone sequences and ambient noise sequences. An *led update rate* is specified via the authoring system and is uniform throughout all sequences for any given instantiation of the firmware. A *led signal table,* or waveform, can be specified for each LED. This defines the shape of the visual pulse. It is a table of values giving the amplitude at each point for one wavelength of the LED signal. The number of points comprising a single wavelength is configurable via the authoring system. For best results, a sine or triangle waveform should be used. During the playing of a sequence the firmware maintains a moving index into each LED signal table. This index is incremented (at the *led update rate),* with wrap, by a *led step value* which has been pre-calculated by the authoring system to traverse the signal table once per the associated brainwave period. This lookup value is scaled by the current intensity of the associated brainwave (as derived via session table procesiing), and then by the global LED intensity coefficient (which changes depending on the user-selected light level.) This final value is then used to pulse-width modulate a GPIO pin configured to drive an LED

Audio for tone seqeunces is computed in the ISR by iterating the base and offset tones of each voice in the manner described in the section on the entrainment device. The main difference between the algorithms in the firmware and the authoring system (which implements a general entrainment device) lies in the processing of note records within the song table. In the case of the authoring system, each note record in a song table contains fields defining it's relative pitch, in semi-tones and micro-tones, from the song tonic. The resulting *step values* to be added to the index into the associated waveform are computed in real time. In the case of the firmware, the note record fields defining relative pitch have been replaced by a single field, pre-calculated by the authoring system, which gives the *step value* itself These step values are loaded into the tone structures directly from the song table as this table is processed in the ISR.

Audio for ambient noise seqeunces is computed in the ISR by first iterating each brown noise oscillator in the manner described in the section on the entrainment device. The phase shift (or lag between audio channels) for each oscillator is determined by subtracting the current index into the LED signal table of the associated LED from half the period of the LED signal table, and multiplying this value by the current intensity of the associated brainwave.

For tone sequences, an ADSR amplitude envelope can be applied to voice 0. This envelope can only be applied if there is no brainwave activity assigned to that voice throughout the sequence. By applying an ADSR envelope to voice 0 it is possible to add rhythmic texture to a sequence. The ADSR envelope parameters are specified via the authoring system..

The firmware task loop contains a task which senses when the battery charger is plugged in or out. In the production version, the sleep mask is turned off if the battery charger is sensed as being plugged in for ½ second or longer.

The firmware contains an optional *diagnostics* mode. In the production version this mode is invoked at power-on reset (when the battery charger is plugged in to recharge a "dead" battery). In this mode the LEDs are turned on and off in the order blue, green, yellow, and red. Then a tone is played momentarily first in the left ear, then in the right ear.. A firmware loop then follows, allowing the user to press buttons at will, resulting in different colored LEDs being turned on as follows: the red LED is turned on while the top button is pressed, the yellow LED is turned on while the middle button is pressed, the green LED is turned on while the bottom button pressed. The loop times out after no buttons have been pressed for several seconds. The diagnostics mode provides a built-in capability to trouble-shoot possible hardware issues.

The firmware supports two different user interface paradigms: one-touch and multi-touch. Under the one-touch paradigm all sequences are assigned to the top user button, the volume intensity is controlled by the middle user button, and the light intensity is controlled by the lower button. The top button also controls turning off the mask (this is the implicit functionality of the button once the last sequence has been reached). Under the multi-touch user interface paradigm, the simultaneous pressing of multiple buttons can be defined as a single user interface event. The programmer can assign sequences, light/volume adjustments, and mask off functionality to different buttons, or button combinations, The choice of user interface paradigm, and the specification of custom button semantics when using the multi-touch paradigm is configurable via the authoring system. The production version of the firmware uses the one-touch paradigm.