

Santa Clara University
DEPARTMENT of COMPUTER ENGINEERING

Date: June 18, 2003

I HEREBY RECOMMEND THAT THE THESIS PREPARED UNDER MY
SUPERVISION BY

Theodore Karl Schundler

ENTITLED

Audio-based Dynamic Lightshow Controller

BE ACCEPTED IN PARTIAL FULFILLMENT OF THE REQUIREMENTS FOR THE

DEGREE OF

BACHELOR OF SCIENCE IN COMPUTER ENGINEERING

THESIS ADVISOR

DEPARTMENTCHAIR

Audio-based Dynamic Lightshow Controller

by

Theodore Karl Schundler

SENIOR DESIGN PROJECT REPORT

Submitted in partial fulfillment of the requirements
for the degree of
Bachelor of Science in Computer Engineering
School of Engineering
Santa Clara University

Santa Clara, California

June 18, 2003

Abstract

Current lighting technology is largely limited to paradigms optimized for theatrical application. While it is actively used in dance type environments, there is much room for expansion to better meet that market's needs. The system presented herein provides a better solution for the dance lighting market by allowing the user to program light shows which work intelligently with the currently playing music, without prior knowledge of the music's nature or content. This is done by a graphical configuration environment, where the user may program the behavior of a real time system through various data processing and manipulation tools. Such tools include ones to process live audio in real time, extracting relevant information, and a state machine system to allow dynamic operation decisions made on the basis of such information as the results of audio processing. Through experimentation, including events with audiences, the software has shown itself to be successful in achieving the goal.

Keywords

Music, Dance, Lighting, State Machine, Real-Time Audio, Color Mixing, Color Matching, BeOS

Acknowledgments

I would like to thank the following individuals for their assistance.

Real-Environment Testing

Michael Moul and Jim Gross for assistance at Mayer Theater

David Thompson for permission to test at The Bronco

Equipment Loan

Arsen Ovanessoff

Presentation Assistance

Victor Gonzales-Vitale

Table of Contents

1 Introduction.....	9
1 Objective.....	9
2 Background.....	9
3 Problem.....	10
4 Requirements / Specific Objectives.....	10
2 System Concepts & Building Blocks.....	13
1 Components & Channels.....	13
2 Massive Multithreading.....	15
3 Operating System.....	16
4 Graphical User Interface.....	18
3 Component Collection.....	23
1 Input Sources.....	25
2 Show Control.....	29
3 Passive Processing.....	32
4 Device Abstraction.....	38
5 Output Destinations.....	40
4 Experimentation.....	45
1 Randomization.....	45
2 Color Matching.....	47
3 Beat Detection.....	50
5 Other Issues.....	54
6 Analysis and Conclusions.....	58
1 Economic Practicality Analysis.....	58
2 Sustainability Analysis.....	60
3 Future Work.....	63
Appendices.....	66
A: Channel Properties.....	66

B: Interface Details & Style Standards.....	68
C: File Format.....	71
D: Use of the Cortex Utility.....	79
E: Class Relationships.....	80
F: Source Code.....	82
Glossary.....	377

List of Figures

Figure 2.1 Effective Hierarchy Configuration.....	13
Figure 2.2 Example Component Connections.....	14
Figure 2.3 Primary Interface Window.....	19
Figure 2.4 Channel Tools GUI element.....	20
Figure 2.5 Color Picker, Hue/Saturation Only.....	21
Figure 2.6 Color Picker, Full Options.....	21
Figure 2.7 WaveView displaying Sawtooth wave.....	22
Figure 2.8 KrazyCtl switch region widget.....	22
Figure 3.1 Simple Desk GUI.....	26
Figure 3.2 Signal Generator GUI.....	28
Figure 3.3 Sample State Machine.....	29
Figure 3.4 State Machine: State Management GUI.....	30
Figure 3.6 State Machine: Managing Transition.....	31
Figure 3.7 State Machine: Events On Entry to the “Turn Light On” state.....	31
Figure 3.8 Mixer Signal Processing.....	32
Figure 3.9 Mixer Control Panel.....	33
Figure 3.10 Storage of Switch States.....	35
Figure 3.11 Switch Component Control Panel.....	36
Figure 3.12 Group of Lights Behavior Control.....	39
Figure 3.13 Lamp Management.....	39
Figure 3.14 PMX Output Flowchart.....	41
Figure 3.15 PMX Component Control Panel.....	42
Figure 3.16 Lamp Simulator Window.....	43
Figure 3.17 Simulator Component Control Panel.....	44
Figure 4.1 Color Match 1st Method for Blue.....	50
Figure 4.2 Color Match 1st Method for Very Light Blue.....	50
Figure 4.3 Color Match 2nd Method for Blue.....	50
Figure 4.4 Beat Detection Simulations.....	52
Figure D.1 Cortex GUI.....	79

Figure E.1 Overall Object Relationships.....	80
Figure E.2 Auroran Component Class Hierarchy.....	80
Figure E.3 List View and ChannelItem Inheritance Relationships.....	81

1 Introduction

1.1 Objective

The project is a new audio controlled lighting system, built for dance/rave (parties typically focused around electronica music, generally not held at a conventional establishment)/night club type environments. Rather than the traditional sequential scene (sets of specific sets of lights at specific brightness levels) based systems that are 2nd hand from the theater industry, where such systems are highly evolved, the new system allows for a more dynamic flow behind what the lights are doing. For example, rather than step through a sequence in order, what happens next is a branch to one of a variety of potential next states, determined by current audio activity, randomization, or other control variables. It may then run with minimal or no interaction from the user after initial programming, making it both advanced enough for large productions, and convenient enough for a single mobile DJ.

1.2 Background

Nearly all modern lighting control software is built on ideas from the theater industry. Prior to the advent of digital lighting control, the typical method was to send values for lights over separate wires as a voltage 0-10V (a standard still supported in many dimmer units). The analog lighting desk would mix the signals from one scene with those of another to fade between scenes. An analog value produced on one slide, would be merged with a value from another slide, and then output through a bundle of cables to the dimmer. The user could set up one scene, while another is playing out on stage, then fade to the new scene when ready using this mixing technology.

With digital advancements, the basic concept remained the same, replacing bundles of cables with a single one. Digital did bring about a key enhancement – the ability to store and recall sets of values. Rather than set up a scene every time it is needed, a lighting designer could simply recall it from the console's memory, or some external device (tape,

floppy disk). Further advancements allow the scenes to be set up into some order, and even potentially timed. This allows for somewhat complicated lighting sequences to be possible, and works perfectly for theater, where what comes next is scripted.

1.3 Problem

The traditional way of doing things can yield nice results, but it is not necessarily an ideal solution, as it is not geared to the ways lighting systems are applied in environments such as clubs and raves. On a given night, what is played next is at the whim of the DJ, not any sort of script. So, either a light show must be programmed that is generic enough to work out fairly well in most cases, or actively change the current running program during the show to best fit what is going on. A lighting solution for this specific industry should be able to work intelligently with what is most fundamental to the industry: the audio, not a script.

1.4 Requirements / Specific Objectives

To better drive the operation, a set of more specific goals were established in the initial problem statement. Here is the set of primary objectives determined to be key to providing a lighting control system more ideal for dance situations. As these goals have since been accomplished, this section also serves as a traceability table of sorts.

1.4.1 Intelligently process live audio

The music is the driving force, there is no set play list, and with vinyl records there is no practical way to know what is playing. This is addressed though real-time spectral analysis on data sampled from a sound card.

1.4.2 Not strictly sequential operation

Since what comes next in the music is unknown, the next step should not be restricted to something known. Rather than have a fixed next step, the next step should be able to be

one of a variety of next steps. This is addressed primarily through the State Machine component.

1.4.3 Capable of doing anything current technology can do

For a system to be acceptable, it should be able to allow lighting designers to still use any of their current tricks. The State Machine component allows for most normal sequencing to be done, only with more options. To allow fades between scenes, a Mixer component was created. Finally, to give the user a comfortable interface, the Simple Desk component is designed to look like a traditional lighting desk.

1.4.4 Support for some sort of hierarchy

To make things more manageable, it should be possible to control shows in a hierarchical manner. There are no specific features to enforce a hierarchy, but as described in section 2.1 on page 13, the user may establish a hierarchical type operation on their own.

1.4.5 Portable

Since this system is likely to be of interest to mobile DJs and rave coordinators, it is helpful if the system can be easily transportable and configured for new environments. Since it is software, it can be installed on a laptop and taken anywhere. It is then up to the lighting designer to tweak an existing show to fit the requirements of the new environment. They could potentially develop a state machine (Section 3.2.1) that acted intelligently based on what equipment is available using the On Fail feature.

1.4.6 Work w/ a variety of existing hardware

The system will only be practical to adopt if it works with existing lighting hardware. Because output is done through plug-in modules, a plug-in could be written for any output to lighting bus hardware that can connect to a computer. For the moment, a plug-in has only been written for the PMX protocol (Section 3.5.1), as it was the simplest to implement using standard RS-232 (serial port) signals.

1.4.7 Reliable operation over extended period

If the program is used in commercial environments, a failure could result in lost revenue. Measures have been taken to ensure smooth operation. Decisions such as the OS platform (Section 2.3, page 16) and multithreaded design (Section 2.2, page 15) have been made with this goal in mind.

1.4.8 Able to operate fully with no user interaction

Particularly in the mobile DJ market, where there is not a separate lighting designer, it would be helpful if the system could run without any user interaction once loaded. No specific portion of the program is built specifically to address this issue, but rather the program as a whole addresses it with its ability to function intelligently based on audio processing. Because of this, while running, except for some sort of manual override if desired, there is no need for the user to interact with the program.

2 System Concepts & Building Blocks

Before addressing subsystems which have a clear connection to one of the requirements in previous chapter, the underlying supports structures will be explored. The “components” discussed in the following chapter build upon this framework to provide the required end functionality. The content in this chapter is primarily resisted to high-level concepts. Greater details of the implementation can be found the in appendices.

2.1 Components & Channels

To allow maximum flexibility, since the concept of passing data along channels has between devices is well established in the audio and lighting industries, it has been adopted as the basis for this program. Users may connect different components together with virtual wires of a sort, typically carrying data representative of analog brightness values. Components may do anything involving some combination of reception, processing, and broadcast of some sort of lighting related data over the virtual “wires”. Some components for example reflect traditional lighting control equipment (traditional slide-based lighting desks, sequencers, etc.), other represent actual physical hardware (i.e. intelligent lights), and others are more similar in function to audio effects processing tools.

There is no intrinsic hierarchy in the component/channel system, but a virtual hierarchy may be established though clever use of components. Impressive dynamic shows may be accomplished by using one or more of the state machine components in a hierarchical-like configurations, such as the example below:

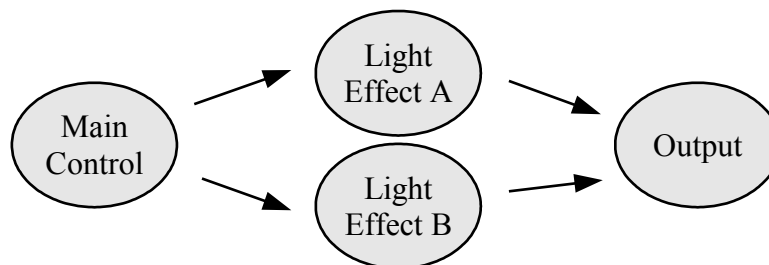


Figure 2.1 Effective Hierarchy Configuration

A component acting as a sort of high level, labeled “Main Control”, can alternate between which of two effects (component “Light Effect A” and component “Light Effect B”) are active at a given time. Both middle-level effects may share the same output. A further unified nature to the show may be achieved by sharing “Pulse” and “Activity” channel data among all components (most likely derived from the audio processor component).

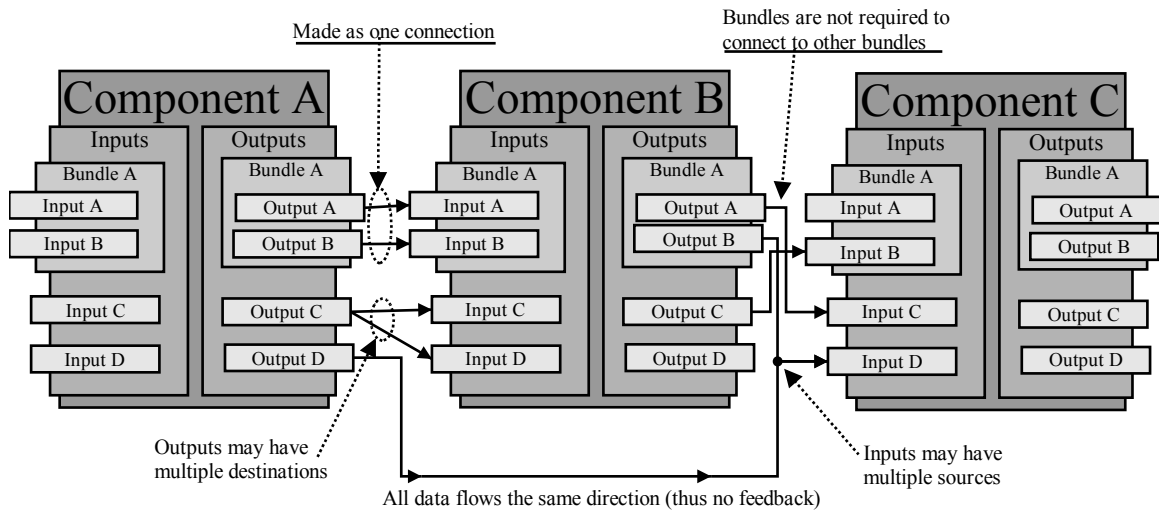


Figure 2.2 Example Component Connections

The outputs of any component may be connected to the inputs of any other component, assuming the channels are configured to carry compatible data types (see appendix A for more details on compatible channel types). The data carried may be just about anything imaginable – brightness, color, beats per minute, etc. To simply “wiring” for the end user, simple outputs may be connected to multiple destinations, and a single destination can accept data from multiple sources (essential for the sort of virtual hierarchy discussed in the previous paragraph, but can lead to odd behavior if data is received from two sources simultaneously). Additionally, to make things easier for the end user, channels may be grouped hierarchically. For example “X” and “Y” could be grouped as a “coordinate” bundle, and connections may be made using the “coordinate” bundle without needing to connect the individual channels.

All the real work of the program is performed by the components. Details of the operation of various components is explored in chapter 3. These are designed as plug-in modules (shared object files discovered and linked at run time), and more may be relatively easily added to the collection of available tools – even by users themselves.

2.2 Massive Multithreading

To help minimal latency operation, like physical pieces of hardware, this program performs many operations simultaneously. This is accomplished through massive multithreading – at a given time the program may have from 5 to over 100 simultaneously running portions. While adding to the code complexity, this design decision has many benefits:

- Tasks may be prioritized (actual processing of data passes across channels has highest priority, GUI has medium priority, and any processing whose results are not needed immediately is assigned lowest priority), so time is not wasted on unimportant things
- Better responsiveness can be achieved by doing pre-computations in background threads and taking care of that which is most important upon an event as fast as possible while leaving less essential relegated operations for lower priority processing
- Properly used threads are only active when they need to be doing work, and take no CPU when idle
- If a single thread crashes, the program may continue running, potentially only slightly debilitated and not thus hopefully not adversely affecting the audience's experience.
- Splitting up tasks help better utilize additional CPUs if they are present

In order for the various separate threads to provide some sort of unified functionality, they need to be able to talk with each other. Typically this is done using BMessages to wake up threads & give them something to process. Some cases however require that threads manage shared data (i.e. The current state of the GUI), and for this a consistent coding style was established to prevent any shared resource problems:

If a portion of code needs to access a shared resource, it must first attempt to lock access to it through use of the BeOS BLocker system. It must also attempt to gain any other relevant locks at the same time. If any attempt to attain a lock fails, all locks must be released, and a request for that operation must be re-queued for processing later. The re-queuing is done by requesting a “OneShot” BMessage from the Pulser object maintained in the AuroranWorld. By re-queuing the request for the operation, if another thread that is locking that resource also needs to lock the current thread, it may do so without causing a deadlock. Logically, critical sections are kept as short as possible. See the section of code on page 274 in appendix F for an example of this coding style.

2.3 Operating System

The final unifying element of the program is its target platform. This software runs under BeOS (developed for x86, with potential ability to run on PowerPC architectures), and should run on BeOS spin-off projects (i.e. ports of the BeOS API as a toolkit for Linux, and the Zeta & OpenBeOS projects). The use of such an unconventional OS certainly warrants justification. Windows is probably the obvious choice, and Linux is the alternative of choice these days. Any one could justify almost any approach, here is the reasoning for mine:

Windows aims itself at 3 primary targets: Home users (WinXP Home), Office Users (WinXP Pro), and Servers (Win2k server). For home users its goal is to be a jack of all trades – it supports a lot of hardware pretty well, and has fairly intuitive interface, and a wide base of applications. For office users, it adds better support for multiple users, and has enhanced options for use of shared network resources (i.e. remote home directories). For servers it makes setting up and maintaining a server fairly easy for the impatient who want to be up and running quickly in an acceptable configuration without having to learn much.

The users of a system like this would probably fall into the home user category. “Jack of all trades” however comes hand in hand with “master of none.” As a side effect of its effort to support all possible hardware configurations and a huge software base, it is inherently unstable. The system is also tremendously complicated due to its age (concepts from DOS are still there), and attempts to add new features and implement new idea while hanging on to the old. While it is important to keep compatibility with older programs to cater to the broad user base, it has the side effect of often kludge-y and inconsistent APIs, and such confusion lends to further instability. These incompatibilities are likely to result in system crashes at inopportune times. The nature of the system lends itself to use in environments where such failures are simply unacceptable. Even in a stable Windows environment, the task scheduler often cannot make any guarantees, resulting in side effects such as slow performance while loading another application.

The typical route of those fed up with Windows is the GPL world, specifically Linux. It has a definite record of greater stability, and Linux task scheduling can be tweaked to acceptable performance levels required in real time systems. But despite what some may like to believe, I do not believe it is a suitable system for the average desktop user. Someone putting on a light show does not want to be concerned with setting up user accounts, picking which of a dozen window managers they want, or deciding which of the plethora of distributions to use. Additionally there is the problem than UNIX was never meant for this sort of thing. There are some multimedia standards, but they are hardly universally supported, and an added hack to a system with completely different intentions.

Another potential route is MacOS. MacOS X has the power and stability expected from unix, with a fairly high level of user friendliness, and multimedia is not an afterthought. Macs are also frequently used to drive the video displays associated with light shows, so they are familiar to at least a large portion of the industry. Macs however have two related problems: fairly small user base, and relatively expensive hardware.

BeOS fills all the gaps above:

- It is a reliable system with nearly all operations running in user space (even many device drivers), thus even poorly written 3rd party drivers can't bring down the entire system.
- Its task scheduler is oriented towards highly responsive interfaces, and the OS itself is massively multithreaded. Combined with a well established set of standard priorities, unessential activity will not take CPU time from where it is needed most.
- It is also built to handle stream type data very well, and have very low latency when performing operations on media streams.
- It was built to be the *media* OS, and so the OS's API has been carefully tailored to provide a logically consistent tools to build media oriented applications.
- It can run on (most) PC hardware, which most users will have already, and it is also available for Macs, which are also frequently found in the industry.

As for distribution, there is a free version of the BeOS available (BeOS Personal Edition). It can be installed on computers running Windows or Linux without needing to engage in the risky disk partitioning. It simply installs its file system to a large file within the FAT or ext2fs file system. Additionally, BeOS can be used without installing anything, by running it off a bootable CD (though there may be some licensing questions with that). Finally, there are the open source BeOS clone projects, such as OpenBeOS, Zeta, and ports of the API for AtheOS, which provide further alternatives for running the program.

2.4 Graphical User Interface

2.4.1 Main Configuration Window

The GUI was developed using widgets from the standard BeOS Interface Kit. The primary window acts as a container. It holds a list of the currently loaded component

instances to the left, in the top left region it has space for a custom control panel area based on what component is selected, and at the bottom right there is an area to connect channel sources to their destinations.

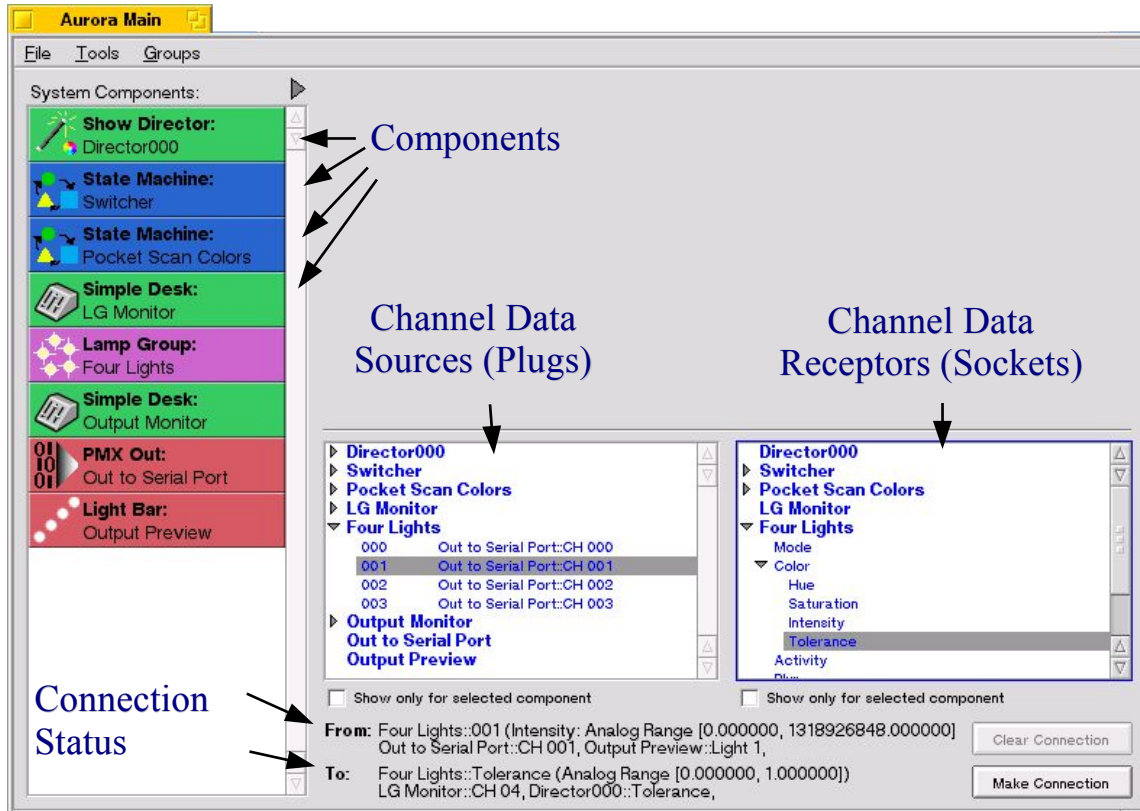


Figure 2.3 Primary Interface Window

The list of loaded components on the left, features an icon and colored background for easy identification. Also displayed is the name of the type of component, as well as a name for that particular instance of the component. The user may add components to groups to help more easily manage large collections, and then choose only to display components from a particular group. Deletion, duplication, saving (of a particular component, not the whole group), and handling of group memberships may be done though a menu reached by using the right mouse button to click on a component. Components may be re-ordered by dragging on them, and dropping them between two other items. For more in depth details of the menu structures, coloring schemes, etc. see

Appendix B. Most of the source code for the main window is located in `configandpatchwindow.cpp`, which is located in Appendix F.

2.4.2 Custom GUI Elements

While standard BeOS Interface Kit widgets were sufficient for most UI building, due to the unique nature of the program, several widgets had to be designed custom for this application. The BeOS API makes the creation of new widgets fairly simple by inheriting functionality from existing widgets thanks to the OS's object-oriented API. Most custom controls are simply standard widgets drawn in a specialized type – for instance, the components themselves are inherited from the `BStringItem` class for items in lists. The drawing method is rewritten to produce list elements like those on the left in Figure 2.3 on page 19. Some of the more advanced custom widgets are presented here. Sections are named by the name of the corresponding C++ object.

2.4.2.1 ChannelTools

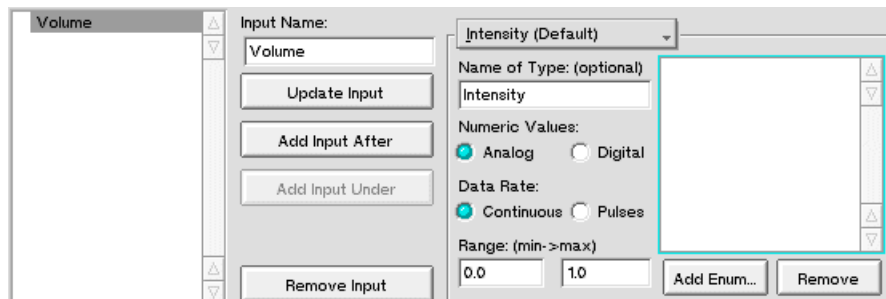


Figure 2.4 Channel Tools GUI element

This particular GUI Element is not a new widget, but rather a collection of controls organized into a useful form so that it may be used by a variety of components. It allows for a common interface for managing groups of channels. It may be configured to limit the level of control the user has over aspects of the channels, for example in the Lamp Group component (Section 3.4.1) the left half of the screen in Figure 2.4 is replaced with a color wheel. This object also has utility functions to save data about the channels it manages to disk. For the details of the saved data, see appendix x.y.z.

2.4.2.2 AColorControl

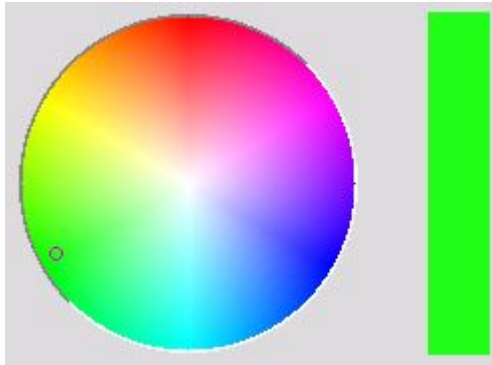


Figure 2.5

Color Picker, Hue/Saturation Only

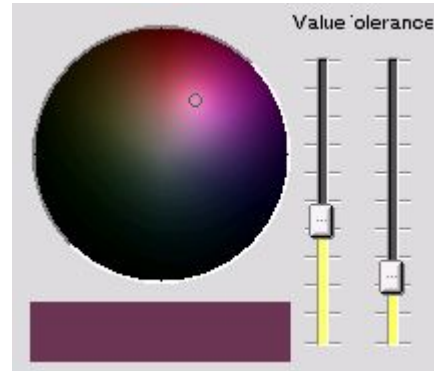


Figure 2.6

Color Picker, Full Options

The standard BeOS color picking widget only gives the user options of mixing red, green and blue, or selecting from a palette. A color wheel based color selector is preferable however because it is more natural for an end user, and it can easily generate hue (polar angle) and saturation (polar radius) values (which are what the color matching functions use). On creation, the level of control presented to the user may be specified. Two possible configurations are presented above as Figure 2.5 and Figure 2.6.

The odd brightness levels on the wheel in Figure 2.6 are to show the effects of the tolerance curve on color matching: The brightness of a color on a wheel indicates approximately how bright it will be after color matching based on the user's selected color. Due to the CPU demand needed to render the brightness levels, rendering of the color wheel itself is done in a separate thread from the main GUI thread & updated as frequently as is practical. This helps keep things feeling more responsive to the user. For more information on how the color matching is done, see section 4.2 on page 47.

2.4.2.3 WaveView

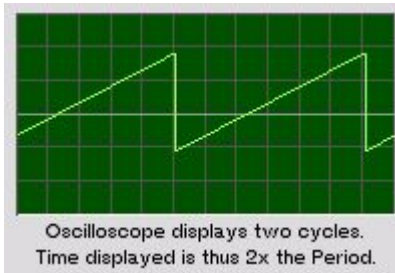


Figure 2.7 WaveView displaying Sawtooth wave

For the Signal Generator component (Section 3.1.3), this widget was created to help the user better understand the effect of the settings. It renders two periods of whatever waveform the user specifies (sin, saw, triangle, square, etc.), taking into account requested max & min values and starting offset.

2.4.2.4 KrazyCtl (switch range specifier)



Figure 2.8 KrazyCtl switch region widget

The switch component (Section 3.3.2) is a rather unique tool, and it required some creative GUI innovations. Rather than have everything operate via state machines, the component allows for a simple level of switching inputs to outputs depending upon a single control variable. This control manages the input ranges of that variable. While a region of the blue horizontal bar is selected, the user can make connections from inputs to outputs which are activated when the control variable's value is within the range specified by the size of the selection region. The size of a region may be adjusted by dragging triangles located above and below the bar. Regions may be combined to the next adjacent left or right region using the J*L or J*R buttons respectively. Regions may be selected using the arrows on either side of the bar. Regions may be split using the “Split” button. Figure 2.8 above displays a region from 0.0 to 0.5 selected.

3 Component Collection

Here are most of the more important plug-in components that make up the program. They are what provide the actual functionality of the program. Here is a brief overview of the components presented in this section and what goals they are meant to achieve (related to the goals set forth in Section 1.4 on page 10):

3.1 Input Sources

3.1.1 Audio Processor

The key to the audio portion of this project, it processes audio in real-time for use by other parts of the system.

3.1.2 Simple Desk:

This aims to provide users with an interface *similar to what they are most likely already familiar with*, for purposes of setting an monitoring channel values.

3.1.3 Signal Generator

Intended primarily for debugging purposes, this acts like a traditional electrical signal generator, parametrically producing various wave signals.

3.2 Show Control

3.2.1 State Machine

This is the key to the dynamic nature of the program. It expands greatly on the idea of scenes to allow intelligent flow between them.

3.3 Passive Processing

3.3.1 Mixer

Operates similar to a standard theatrical scene fader. It provides users with a familiar tool, and allows them to control it from state machines or any other component.

3.3.2 Switch

Allows a set of forwarding connections between inputs and outputs to be selected based on an input. It is intended to help widen the potential of what the state machines can do by letting them re-arrange inter-component wiring.

3.3.3 Polar Coordinate Converter

Some intelligent lights accept their coordinates in a polar format, others rectangular. This allows the user to operate both types of lights in sync with each other.

3.4 Device Abstraction

3.4.1 Group of Lights

Simplifies the complexity of user defined state machines by allowing them to work with a large set of lights all at once as a single unit rather than individually.

3.5 Output Destinations

3.5.1 PMX Output

Sends channel data out to a PMX bus to be received by the physical lighting hardware. At the moment this is the only lighting bus supported, but other similar plug-in components may be written fairly easily.

3.5.2 Light Bar Simulator

Simulates on screen a group of 10 lights in a horizontal row. Useful for monitoring outputs and testing shows when no hardware is immediately available.

3.1.1 Audio Processor



Processes audio, providing useful information for other components to use in directing the show.

Operation:

This component creates a subclass of BMediaConsumer node, that can attach to any standard audio BMediaProducer. What producer it connects to is determined by the user though the BeOS utility program “Cortex” (see Appendix D)

- The BMediaConsumer subclass applies a FFT to the incoming audio stream (received as buffers of 512 samples @ 44.1kHz). The FFT results are used to generate a set of 8 values. Their response is logarithmic, and ranging approximately from 40Hz-44kHz.
- “Volume Right Now” is RMS volume over the last buffer.
- “Volume Overall” is calculated by $V_n = 0.015 * \text{Vol. Right Now} + 0.985 * V_{n-1}$
- “Pulse” is triggered when a beat is detected. See Section 4.3 on page 50 for an in-depth analysis of the beat detection method used.

Input Channels:

(none)

Output Channels:

OL	Name + Description	Type Name	Format	Rate	Range	Def	Enumeration
1	Volume Right Now	Intensity	Float	Cons	0.0-1.0	0.0	
2	Low 0	Intensity	Float	Cons	0.0-1.0	0.0	
2	Low 1	Intensity	Float	Cons	0.0-1.0	0.0	
2	Low 2	Intensity	Float	Cons	0.0-1.0	0.0	
2	Mid 0	Intensity	Float	Cons	0.0-1.0	0.0	
2	Mid 1	Intensity	Float	Cons	0.0-1.0	0.0	
2	High 0	Intensity	Float	Cons	0.0-1.0	0.0	
2	High 1	Intensity	Float	Cons	0.0-1.0	0.0	
2	High 2	Intensity	Float	Cons	0.0-1.0	0.0	
1	Pulse	Pulse	Integer	Puls	0	0	0:Thump
1	Volume Overall	Intensity	Float	Cons	0.0-1.0	0.0	

3.1.2 Simple Desk



Simulates a traditional lighting desk, with slide controls used to adjust output intensities

Operation:

Uses values generated by moving a BSlider GUI control to generate data to pass onto output channels. Additionally, it subscribes to the Pulser service so 15x per second it checks the values for what its outputs are connected to. If necessary, it corrects the position of the corresponding slider on the display. For example, if some other component is changing the value of a channel, and this is connected to the same channel, the slider corresponding to that channel will move itself to the actual output value.

Input Channels:

OL	Name + Description	Type Name	Format	Rate	Range	Def	Enumeration
(none)							

Output Channels:

OL	Name + Description	Type Name	Format	Rate	Range	Def	Enumeration
1	CH(00-18)	Intensity	Float	Cons	0.0-1.0	0.0	
	19 channels, labeled as indicated above						

Control Panel:

A row of 19 sliders, labeled by the number of its corresponding output channel.

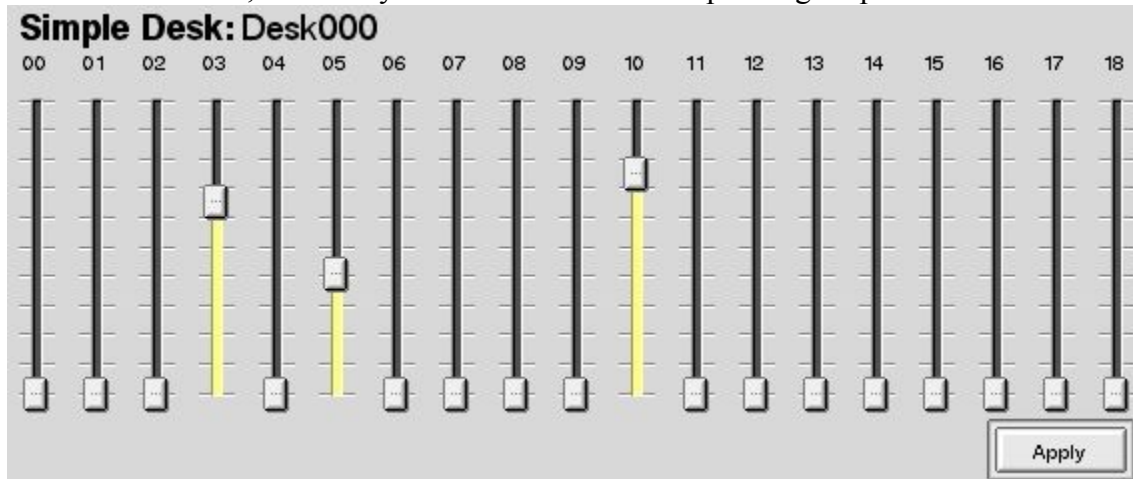


Figure 3.1 Simple Desk GUI

3.1.3 Signal Generator



Simulates an electrical signal generator. Its purpose, like its real-world counterpart is primarily for testing other components. It may however be applied for use in practical show applications, as a potential signal source.

Operation:

Waves are generated by setting output values for a channel to whatever they should be for that point in the waveform. The values are updated at a rate set by *Signal Accuracy* in the control panel, which in turn controls the rate the thread is woken up by the Pulser service. Users can set up multiple output channels, all which use the same clock source (otherwise adjusting the phase offset would be meaningless). The signal generated on a channel may be one of Sine, Cosine, Triangle, Saw, Square, and Pulse. The “Pulse” signal is unique in that it does not constantly update its output – instead it merely updates the output once every *Period* seconds, with the *Maximum* value.

Input Channels:

OL	Name + Description	Type Name	Format	Rate	Range	Def	Enumeration
(none)							

Output Channels:

OL	Name + Description	Type Name	Format	Rate	Range	Def	Enumeration
1	????	Intensity	Float	????	0.0-1.0	0.0	
Set of user defined outputs. Rate is normally continuous unless it has a pulse signal type.							

Control Panel:

To the left is a list of output channels. When selected, the selected channels name and properties may be adjusted with the controls in the middle column. “Period” is measured in seconds, and “Offset” refers to a phase offset in the signal. Other functionality is self-explanatory. Changes are only put into effect after the “Apply” button is pressed.

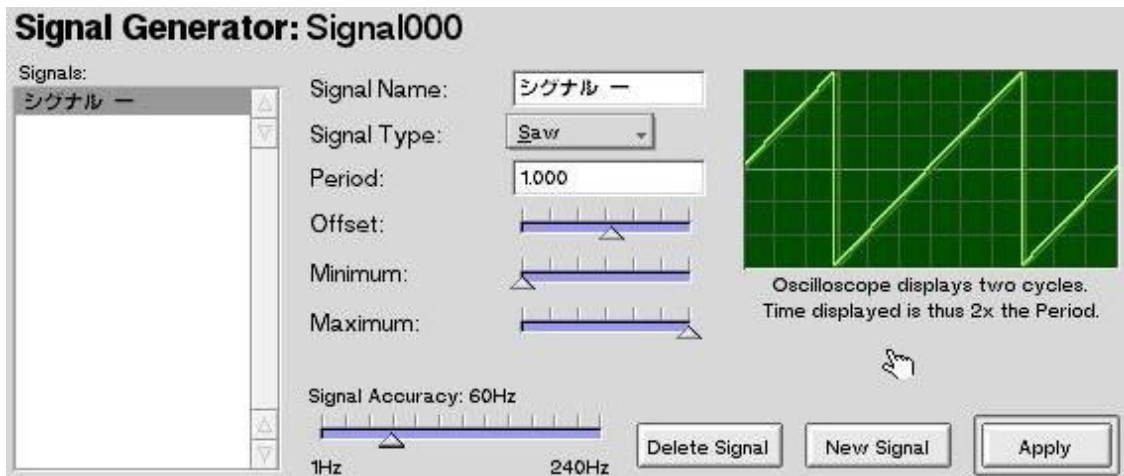


Figure 3.2 Signal Generator GUI

3.2.1 State Machine



An upgrade to the traditional sequencer, allowing non-linear operation. It allows for transitioning from one scene/state to another based on user-defined set of parameters.

Operation:

Expands on the sequencer for scenes (the basis of current lighting technology, discussed in Section 1.2 on page 9) idea by having both a set of data to load upon entering a state, and a separate set of data to load upon leaving a state (allows for hierarchy where entering a state activates a Sequence, and leaving a state deactivates it). Below is an example diagram of the sort of state machine one might implement with this component. It flashes a light in sync with the beat.

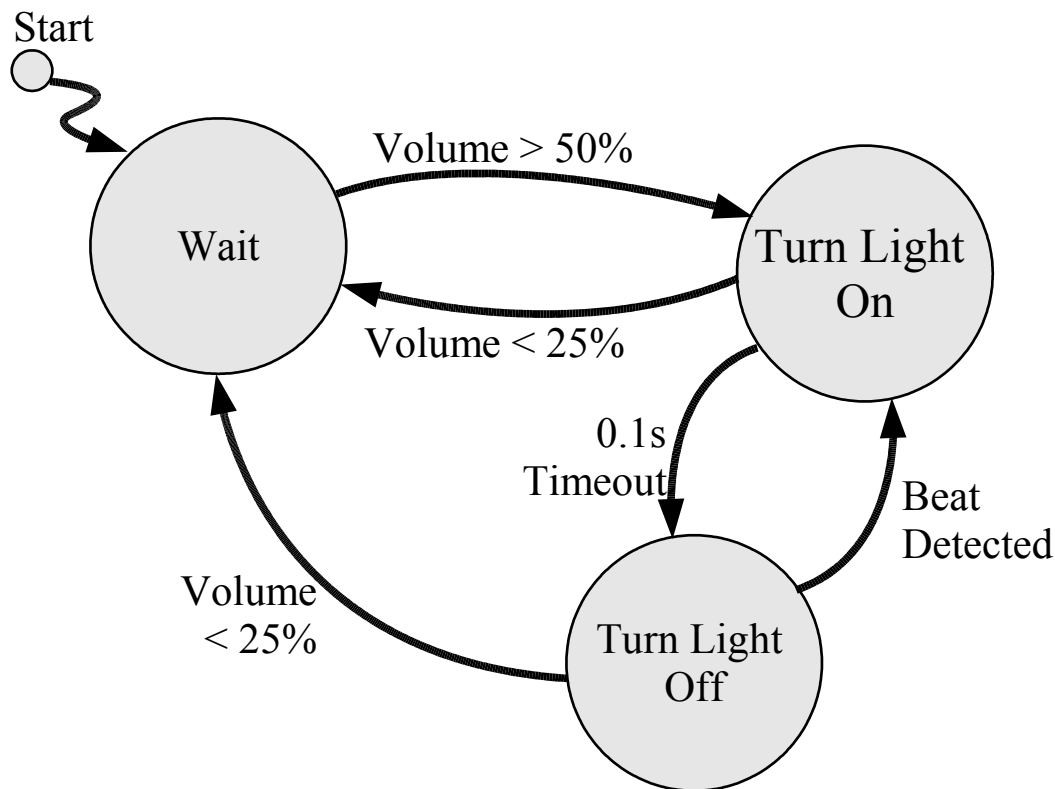


Figure 3.3 Sample State Machine

When to go to the next state is determined by a boolean logic combination of thresholds (i.e. (over 2s have elapsed AND the overall music volume is 0.2) OR (over 5s have

elapsed)), and what state to go to next is also determined by these thresholds. Though the example in the chart above uses only single events, a dependency tree for the aforementioned set of requirements would look similar to the following in the transition requirements section of the GUI:

> 2.00s Timeout

Volume > 0.2

5.00s Timeout

For the 2.00s Timeout to evaluate as true, it depends on the Volume > 0.2 requirement to also be true, effectively working as a logical AND. For a level in the hierarchy to evaluate as true, any of its members may be true, so there is a logical OR for the two timeout requirements.

Input Channels:

(user defined)

Output Channels:

(user defined)

Control Panel:

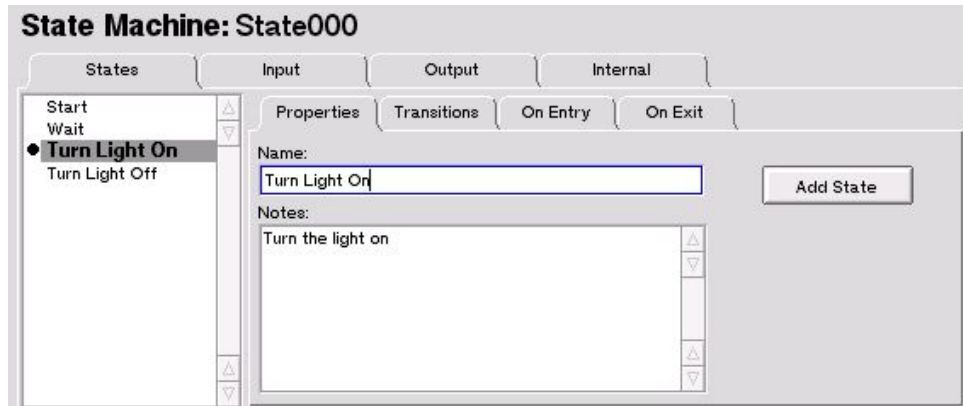


Figure 3.4 State Machine: State Management GUI

The highlighted state is the currently selected state. The state with a bullet to its left is the currently active state. Double clicking a state forces entry to that state.

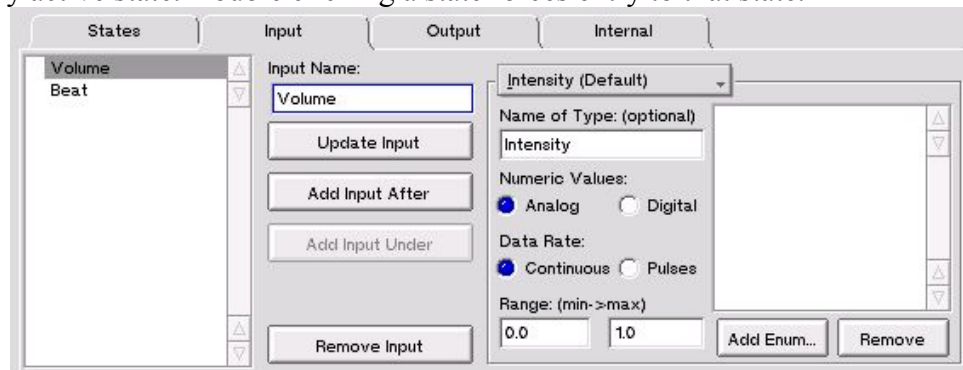


Figure 3.5 State Machine: Managing user-defined channels

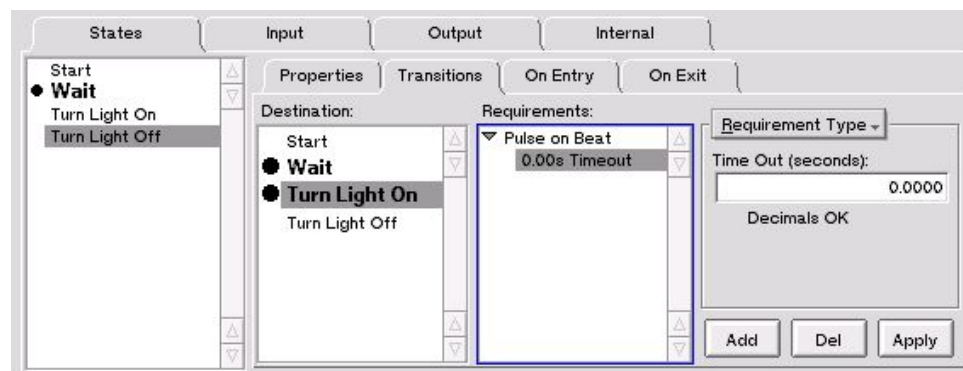


Figure 3.6 State Machine: Managing Transition

This is showing the requirements to transit from the “Turn Light Off” state to the “Turn Light On” state. Note that the 0.00s Timeout is the above screen shot has no functional value. It is there to show the nature of dependencies. For the pulse requirement to be satisfied, the zero second timeout must also be true.

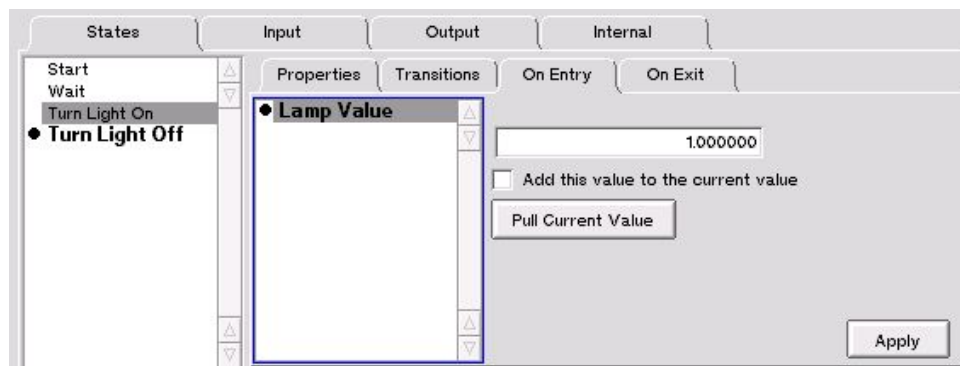


Figure 3.7 State Machine: Events On Entry to the “Turn Light On” state

3.3.1 Mixer



Acts as scene select and master brightness controls on a traditional lighting desk

Operation:

Normally if two outputs are connected to a single input, the output will constantly update with whatever it received most recently i.e. no mixing of values is done. This component provides a means of mixing channel values together. Based on its design, more than 10 channels can be controlled at a time by cascading the Online, Master Brightness, and scene fader outputs to the inputs of another mixer component.

When new values for one of the channel inputs is received, if the component is active, the following operation is performed for that channel, n :

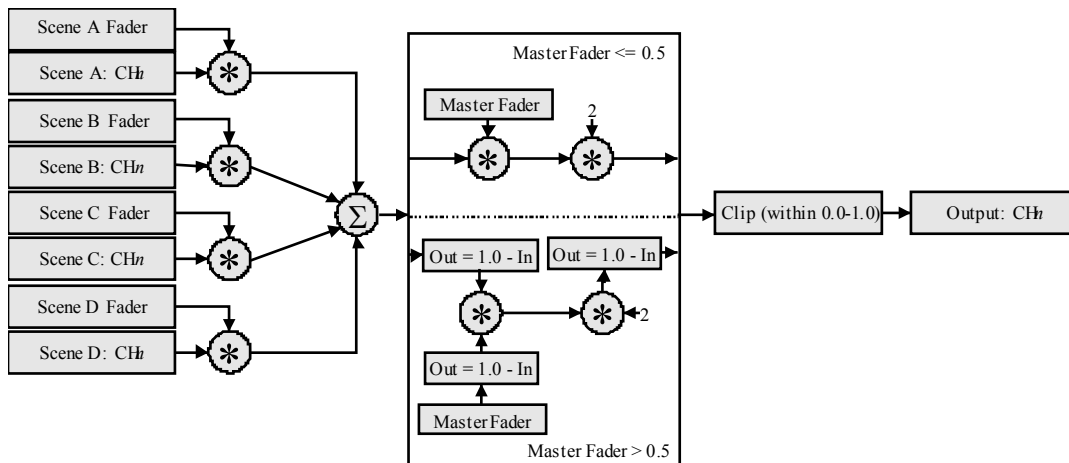


Figure 3.8 Mixer Signal Processing

If new values are received for any of the fader inputs, the above operation is applied for all output channels.

Input Channels:

OL	Name + Description	Type Name	Format	Rate	Range	Def	Enumeration
1	Online	Activator	Integer	Puls	0,1	1	0:Off, 1:On
	Determines if the mixer should be running or not. Could be helpful for enforcing a hierarchy, where a master sequencer or other control switches other components on and off.						
1	Master Brightness	Intensity	Float	Cons	0.0-1.0	0.5	

	Acts as a master fader for all scenes. Normal 100% value is at 0.5, for values > 0.5, scenes are made brighter up to full brightness for all channels when this is 1.0 (see signal processing diagram above)						
1	Scene A	Bundle					
2	Scene A Fader	Intensity	Float	Cons	0.0-1.0	1.0	
2	Scene A CH(0-9)	Intensity	Float	Cons	0.0-1.0	0.0	
Each scene has 10 channels, numbered 0-9.							
This one entry in this table is to hold the place for all 10 actual channels.							
1	Scene B	Bundle					
2	Scene B Fader	Intensity	Float	Cons	0.0-1.0	1.0	
2	Scene B CH(0-9)	Intensity	Float	Cons	0.0-1.0	0.0	
1	Scene C	Bundle					
2	Scene C Fader	Intensity	Float	Cons	0.0-1.0	1.0	
2	Scene C CH(0-9)	Intensity	Float	Cons	0.0-1.0	0.0	
1	Scene D	Bundle					
2	Scene D Fader	Intensity	Float	Cons	0.0-1.0	1.0	
2	Scene D CH(0-9)	Intensity	Float	Cons	0.0-1.0	0.0	

Output Channels:

OL	Name + Description	Type Name	Format	Rate	Range	Def	Enumeration
1	Online	Activator	Integer	Puls	0,1	1	0:Off, 1:On
1	Master Brightness	Intensity	Float	Cons	0.0-1.0	0.5	
1	Scene A Fader	Intensity	Float	Cons	0.0-1.0	1.0	
1	Scene B Fader	Intensity	Float	Cons	0.0-1.0	1.0	
1	Scene C Fader	Intensity	Float	Cons	0.0-1.0	1.0	
1	Scene D Fader	Intensity	Float	Cons	0.0-1.0	1.0	
1	Output Scene	Bundle					
2	CH(0-9)	Intensity	Float	Cons	0.0-1.0	0.0	

note: Fader outputs are simply passed through from the inputs, and are intended for cascading mixers. No extra processing is done on them.

Control Panel:

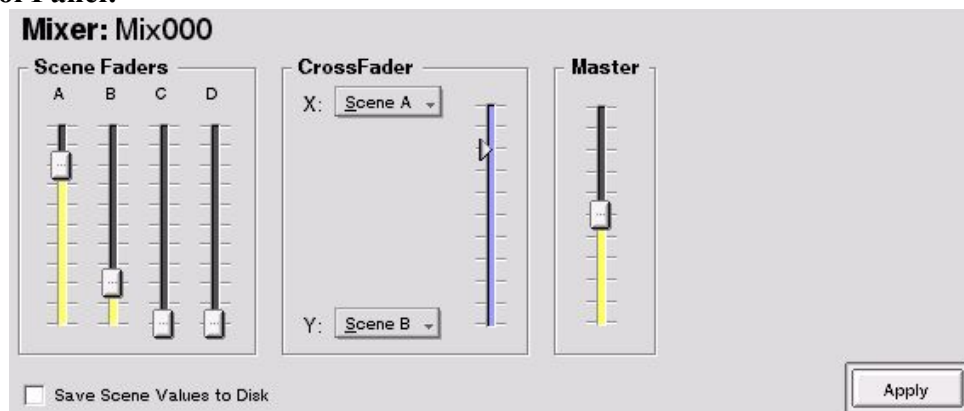


Figure 3.9 Mixer Control Panel

This component has no configuration options, however to make it practical for use with no external control inputs (faders / activation), the control panel has a set of faders. The scene and master faders correspond to their respective input channels. The cross fader allows to fade smoothly between two select scenes at once. X is the scene fully on when the cross fader is at the top, Y is the scene on when the cross fader is at the bottom. Drop down list options for X and Y are any of the 4 scenes. The fader's operation is managed when the component's interface message handler receives interface messages.

Like the “Simple Desk” component (Section 3.1.2, page 32), the system subscribes to a Pulser service, and updates faders, except cross fader, with their actual values. The check box at the bottom may be used to save the values at all the inputs to the disk, which is useful if fader values are not controlled by other components (i.e. State Machines), but rather defined manually by the user.

3.3.2 Switch



Provides a means of changing connections on the fly during a show, by way of a control channel.

Operation:

Upon receiving a new value for a channel, it retrieves a list of where that value should be sent based on the current state of the switch (set by the SwitchValue input), and sends the value to the requested outputs.

There is a maximum of 256 states. States are stored in a 256 element array, pointed to from their start to stop values (i.e. If a state is active from SwitchValue == 0 to SwitchValue == 3/255, then it has pointers from three places, as show in the figure below). This facilitates rapidly looking up the current state. The state contains a BList of Input and Output pairs. Once the state is found, the outputs must be found by stepping through the list. Example mapping:

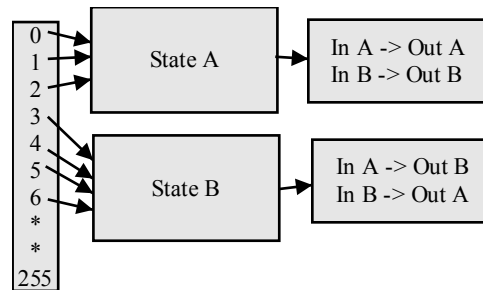


Figure 3.10 Storage of Switch States

Input Channels:

OL	Name + Description	Type Name	Format	Rate	Range	Def	Enumeration
1	SwitchValue	Intensity	Float	Cons	0.0-1.0	0.0	(user defined)
1	???? Set of user defined inputs.	Intensity	Float	Cons	0.0-1.0	0.0	

Output Channels:

OL	Name + Description	Type Name	Format	Rate	Range	Def	Enumeration
1	???? Set of user defined outputs.	Intensity	Float	Cons	0.0-1.0	0.0	

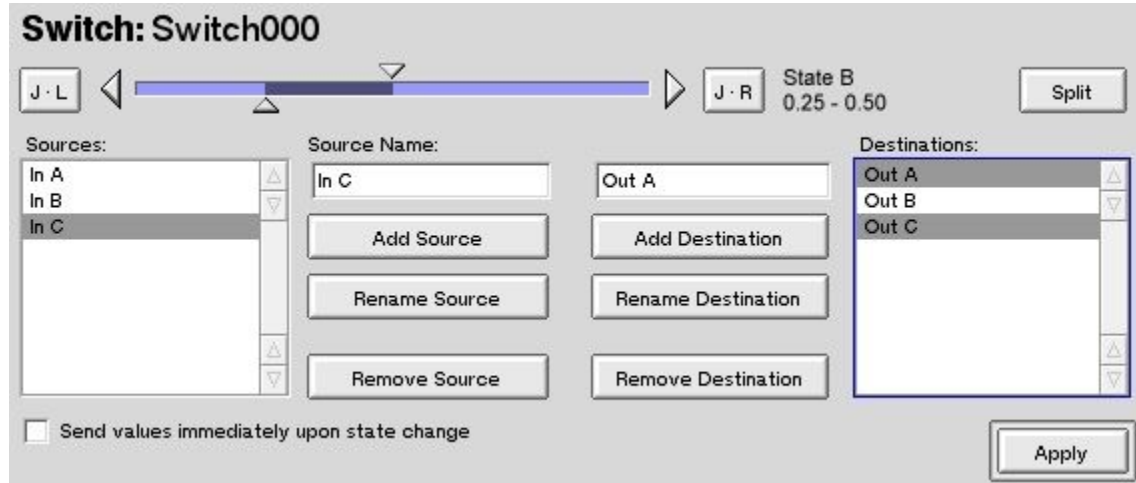
Control Panel:

Figure 3.11 Switch Component Control Panel

The upper portion is for managing the switch's states. Since the switch's states are designed to correlate to analog values, they are managed as portions of a horizontal bar. For details of the operation of this unique control, see Section 2.4.2.4 on page 22.

The lower portion is used for managing input and output channels. They can be created, renamed, or deleted. Newly created channels are always added to the end of the list. To rename or delete a channel, they may select it then press the applicable button. to connect inputs to outputs. the user selects an input, and the outputs it is connected to are highlighted, changing the selection changes the output's destinations. Multiple destinations may be selected by holding now the shift on control keys while making the selection.

Debugging Notes:

To simplify development and testing there is a “Fork” component, which acts as a switch with only one state, and no complicated state selection and management control. This allows the basic functionality of looking up a state and finding requested destinations to be tested without the possibility of the switching functionality inducing other problems.

3.3.3 Polar to Rect



For positioning intelligent lights, translates polar to rectangular coordinates. There also exists a Rect to Polar which is the converse of this function.

Operation:

Translates coordinates. The full circle's angular positions are 0.0-1.0. The Radius value is centered at 0.5, to keep it compatible with 0.0-1.0 standard channels (rather than sending the radius as (-0.5)-(+0.5)).

$$X = \cos(\text{Angle} * 2\pi) * (\text{Radius} - 0.5) + 0.5 \quad (3.1)$$

$$Y = \sin(\text{Angle} * 2\pi) * (\text{Radius} - 0.5) + 0.5 \quad (3.2)$$

Input Channels:

OL	Name + Description	Type Name	Format	Rate	Range	Def	Enumeration
1	Angle	Angle	Float	Cons	0.0-1.0	0.0	
1	Radius	Radius	Float	Cons	0.0-1.0	0.5	

Output Channels:

OL	Name + Description	Type Name	Format	Rate	Range	Def	Enumeration
1	X	Position	Float	Cons	0.0-1.0	0.0	
1	Y	Position	Float	Cons	0.0-1.0	0.5	

Control Panel:

no options

3.4.1 Lamp Group



Simple abstraction for a group of dimmed lamps. Simplifies higher level operations by allowing the control of many lights as a single unit.

Operation:

There are 6 modes of operation (or offline, in which case nothing is done):

- **Random:** Lights are given random values, then are masked based on the requested Hue & saturation levels. An attempt is made to make results average approximately the requested color. (see Section 4.1, page 45, for more details)
- **Color:** Sets lights to mix to the currently requested color. (see Section 4.2, page 47, for more details)
- **Chase:** each step the current light is switched off, and the next in sequence is switched on @ the *Intensity* value. When the final light is reached, it starts from the beginning again.
- **1-Hot:** Like chase, except the one light that is on "next" is selected randomly.
- **Solo Test:** Intended to assist programming the current gel color, this will set the current select light in to channel management tab to 100%, and all others to 0%.
- **Blackout:** All channels set to 0.0.

Motion Blur factor: When a new set of values is determined by one of the methods above, the actual new values used = new values + previous values * 0.9 * *Motion Blur*.. Values greater than 1.0 are clipped. The Scaling by 0.9 ensures the old values will eventually fade out.

The new values generated are not sent to the outputs immediately. Rather they are faded linearly from the previous values in over a time period = $2 * \log_{10}(0.01 + 0.9 * \text{Activity})$ seconds, using subscription to the Pulser service as a time base, updating the output 20x per second.

If no pulses have been received in $((1.01 - \text{Activity}) * 8)$ seconds, a pulse is triggered internally. This way pulses are not needed, additionally, if the requested activity level is high, a wild strobing effect is created.

Input Channels:

OL	Name + Description	Type Name	Format	Rate	Range	Def	Enumeration
1	Mode	Mode	Float	Puls	0.0-1.0	0	0 Offline; 0.2 Randomize; 0.4 Color; 0.7 Chase; 0.9 1-Hot
1	Color	Bundle					
2	Hue	Hue	Float	Cons	0.0-1.0	0.0	
2	Saturation	Saturation	Float	Cons	0.0-1.0	0.0	
2	Tolerance	Intensity	Float	Cons	0.0-1.0	1.0	
2	Intensity	Intensity	Float	Cons	0.0-1.0	0.33	
1	Activity	Activity	Float	Cons	0.0-1.0	0.0	
1	Motion Blur	Intensity	Float	Cons	0.0-1.0	0.0	
1	Pulse	Pulse	Integer	Puls	0	0	0:Thump

Output Channels:

OL	Name + Description	Type Name	Format	Rate	Range	Def	Enumeration
1	???? Set of user defined outputs.	Intensity	Float	Puls	0.0-1.0	0.0	

Control Panel:

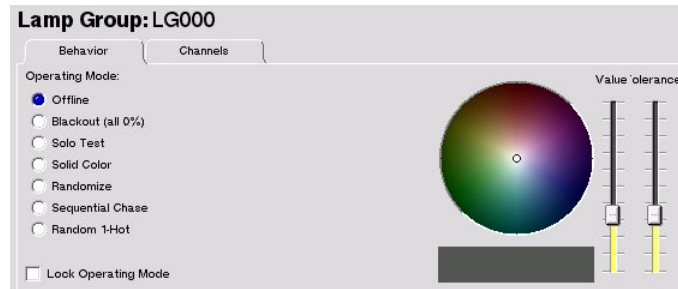


Figure 3.12 Group of Lights Behavior Control

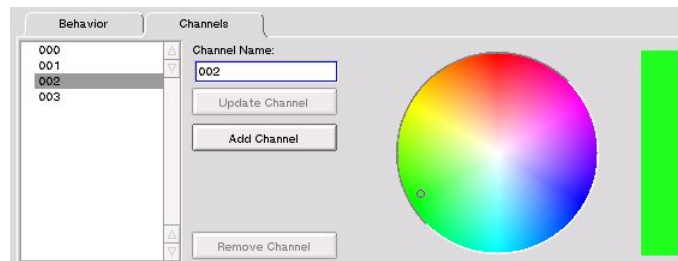


Figure 3.13 Lamp Management

3.5.1 PMX Output



Sends data to a PMX serial bus. Channels are named with the number of their corresponding output channel.

Operation:

The floating-point values it receives are translated to a PMX value: a 14 bit number ranging 0 (off) - 16384 (100% on). Data is outputted as the channel number + 128, followed by the upper 7 bits of the value, then the lower 7 bits of the value:

1 st Byte		2 nd Byte		3 rd Byte	
Bit 7	Bit 6-0	Bit 7	Bit 6-0	Bit 7	Bit 6-0
1	Channel #	0	Upper 7 bits	0	Lower 7 bits

To keep data flowing as smoothly as possible, operation is split into two threads, one for getting data to be sent, another responsible for sending data to the serial port. Chart on the left is for the component's event processor thread. The chart on the right is for the RS232 output thread.

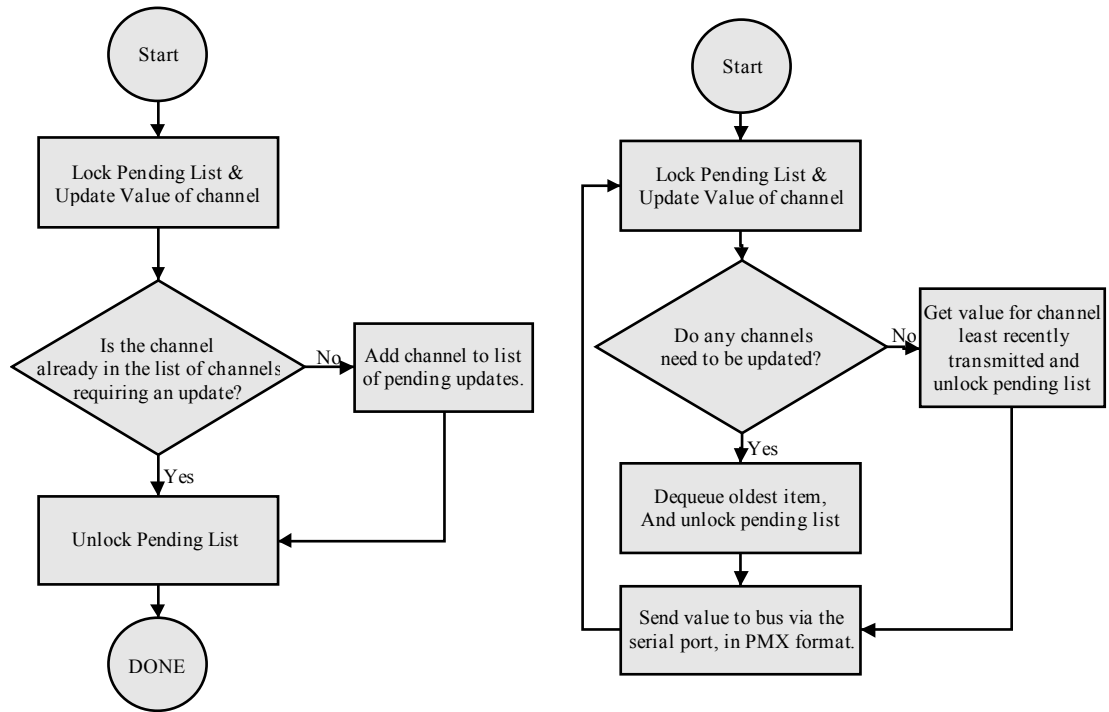


Figure 3.14 PMX Output Flowchart

Input Channels:

OL	Name + Description	Type Name	Format	Rate	Range	Def	Enumeration
1	CH(000-119)	Intensity	Float	Cons	0.0-1.0	0.0	
Set of 120 PMX output channels							

Output Channels:

(none)

Control Panel:

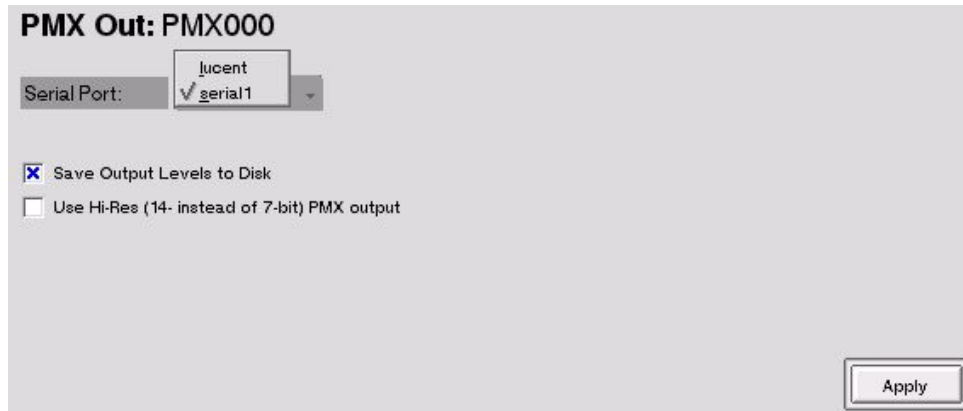


Figure 3.15 PMX Component Control Panel

The user may select the desired serial port for PMX output from two drop-down boxes. The user may also choose to have the state of all outputs saved (otherwise they are all 0 by default upon loading the component).

3.5.2 Output Simulator



Simulates a bank of 10 individually dimmed lamps in its own window - leftmost light is CH0, rightmost is CH9.

Operation:

It maintains a separate window for itself displaying a row of circles whose brightness varies depending upon the intensity for that channel. The window's BLooper for the



Figure 3.16 Lamp Simulator Window

interface handles updated brightness, not the thread that receives the new values. This separates the high priority back end from the lower priority display operations. The instance name of the component is used as the name of the window.

Input Channels:

OL	Name + Description	Type Name	Format	Rate	Range	Def	Enumeration
1	CH(0-9)	Intensity	Float	Cons	0.0-1.0	0.0	
	9 channels, labeled as indicated above						

Output Channels:

(none)

Control Panel:



Figure 3.17 Simulator Component Control Panel

User is given the option to adjust the color of each of the 10 channels by selecting the channel from a list, and then selecting its color using a standard BColorControl. The color control may be replaced with a more application specific color control at some point in the future (i.e. allowing the user to select from a list of Roscolux® gels).

4 Experimentation

Like any project doing things that have not been done before, much of the project is experimental in nature. Most ideas worked out fairly well. A few, however, were in need of refinement. So here are a few things that did not work, and the lessons learned through experiments on the path to getting it right.

4.1 Randomization

In an early revision of the Group of Lights component (Section 3.4.1), there was no enforcement of a required average output value. So when the component was set to operate in “random” mode, it would really do just that. When it received a pulse or timed out, the next set of outputs was assigned using a random number generator. It became apparent during testing that this operation would not be acceptable.

The results initially seemed acceptable when tested with 4 small lights in a dorm room. The next test was at an event held in “The Bronco” - SCU's on-campus bar. There the system was used to control the dimming of eight lights on four channels (two per channel) aimed on a stage used for a battle of the bands night. Three methods for controlling the analog lights were tested:

- Random – light intensities assigned by a random number generator
- Random Chase + Motion Blur – light intensities assigned by dimming all channels some amount, then setting one channel at random to 100%
- Light Organ – light intensities are assigned based on mapping different audio frequency response ranges to each light.

The “Light Organ” configuration actually managed to elicit a complaint. Most of the time it didn't light up the stage very well, and the amount of very rapid flickering was annoying, particularly to those on stage. While I had used this effect at a small dance party with success, it was not appreciated for a live concert environment.

The “Random” configuration also less than ideal results. This did not manage to generate complaints, but it was clearly sub-optimal. Comparing to the Chase + Blur configuration, I was able to generate some theories why this was the case. Due to randomization, the stage would be fairly dark sometimes for a period of a few pulses in a row, and at other times fairly bright with no significant change. The Chase + Blur configuration maintained a constant brightness level on the stage, while additionally providing a feel of greater activity by always changing which lights were lit.

The next major test of the Group of Lights component was in the Mayer theater on the SCU campus. The enforcement of an average intensity somewhat improved the “Random” mode operation. However, simply adding or subtracting some value from each of the channels frequently resulted in most of the channels having more or less the same intensity. Interestingly it seems that adjusting the rate at which things change is the most effective way to establish the energy level – more so than adjusting the overall brightness for example. So with most channels hovering around the same value, the amount of apparent activity is significantly reduced.

The solution to that problem was devised, implemented, and tested while in the theater. Once the required average intensity is achieved, another pass is done before the set of values is used. In the final pass, values are spread as far as they can be from the average value by some linear scaling factor. Assuming the average value A , is the average of a set of n channels holding values $V_0 \dots V_n$, if all channel are scaled by the same constant S , relative to A , the average value is maintained:

$$\frac{\sum_{i=1}^n (V_i - A)S + A}{n} = A \quad (4.1)$$

S is chosen such that the values are kept within the range of 0.0 to 1.0. The end result is increased contrast, helping keep the values as far from A as possible, providing the appearance of more activity since there is more variance in the output.

4.2 Color Matching

4.2.1 Background

One idea I had hoped to implement in this program was to allow a hierarchy where higher level functionality could be truly high level – rather than have something specify an exact sequence to run, abstract parameters would be specified. A perfect example of such a control mechanism that no current lighting technology supports would be to specify desired color.

A simple implementation of handling a color request would be to determine how bright simple dimmed lights should be based on their current gel color. Since the brightness of the lights is what would be determined, it would make most sense to pass the data in a way where brightness is not specified. Specifically, it was determined that passing Hue and Saturation information would be most appropriate. Such a structure lends itself well to a color wheel model, which is an easily understandable GUI element.

With the color model established, the next issue to address was how to determine how bright a light should be, given the requested color. After doing some experiments mentally, it seems the easiest approach would be to assign values based on their distance of some sort from a requested color, then apply the average brightness and contrast functions described in the previous section. If a user requests yellow, but only red and green are available, they will be mixed to create yellow. Red and green will be assigned similar brightnesses, as they are a similar distance from yellow, and scaled to the required brightness. This idea even works for colors like green if only yellow and cyan are available – if yellow = 50% red + 50% green, and cyan = 50% blue + 50% green, when mixed, there will be 100% green, 50% red, and 50% blue. The dominant frequency will be green, and the red and blue components will make the result a somewhat light shade of green.

4.2.2 Experiments

Given the distance function method, the next step would be determining what distance function to use. Since mathematics involving exponents (i.e. $\sqrt{a^2 + b^2}$) is costly in terms of CPU usage, my initial instinct was to devise a mathematically simple solution. The idea came about to manage hue and saturation in a way that would take their nature into account. If the saturation of a color is close to zero, it is close to white, and it can match almost any color. But if the saturation is high, it matters more that a value of the correct hue is selected. From that reasoning, I developed the following matching function (H_R = Requested Hue, S_R = Requested Saturation, H_A = Actual Hue (of light being tested against), S_A = Actual Saturation):

$$H_{\Delta} = H_R - H_A \quad (4.2)$$

$$Y = \begin{cases} H_{\Delta}, & H_{\Delta} \leq 0.5 \\ 1.0 - H_{\Delta}, & \text{otherwise} \end{cases} \quad (4.3)$$

$$Quality_A = 1.0 - 2.0 \cdot Y \cdot \sqrt{S_R \cdot S_A} \quad (4.4)$$

After initial implementation and testing it became clear this would not work. For example, white is always a perfect match. So after scaling and such, any white lights will always be the brightest. Also, the linear fall from testing the hue difference does not yield good results when a highly saturated color is desired, since other colors as much as 60° away on a color wheel will be as much as 50% lit, resulting in excessive saturation. (See Figure 4.1 below) Additionally, when requesting something highly saturated, for example a slightly blue-ish white, a deep blue light will be turned fully on, while a light yellow, which may be appropriate for lightening the overall color, will not be turned on at all because it is 180° away and the intensity mapping is a cardioid type pattern. (See Figure 4.2)

To get desirable results it seems involving multiple exponent operations would be necessary. A method would also be needed that understands that if saturation is zero,

regardless what the hue is, the color is still white, as established above. So the idea then arose to find the distance on a color wheel, where the radius indicates the saturation and angle is the hue. The most practical distance method would be to find a standard Euclidean distance in standard Cartesian coordinates, rather than polar coordinates. This yielded this matching function:

$$Quality_B = 1.0 - \frac{\sqrt{((\cos(H_R) \cdot S_R - \cos(H_A) \cdot S_A)^2 + (\sin(H_R) \cdot S_R - \sin(H_A) \cdot S_A)^2)}}{2.0} \quad (4.5)$$

It works somewhat better (see Figure 4.3), but still not ideally. For example, if the user wants white, it is unclear if they want only white lights or white as the average mixed color (in which case it may make sense to turn everything on). To allow this level of differentiation, another variable had to be added: Tolerance. In many ways the end result of the Tolerance parameter is similar to saturation – except it applies even to colors like white. The most practical way to implement this, was an exponential function, which means higher CPU expense, but results were finally satisfactory. The end user can finally accurately specify what they want in terms of color: white, deep blue, blue-ish, etc., and get results. Here is the final step, established mathematically:

$$Quality_Z = ((Quality_B)^2)^{(0.5 - Tolerance) \cdot 80.0^{-1}} \quad (4.6)$$

The constants are the result of further experimentation to determine what value ranges produced the best results without going to useless extremes.

This system works well in accurately mixing colors to attain the correct color. Its performance is limited by the lights available to it however. In tests using few inconvenient colors (cyan, magenta, yellow, as opposed to the more convenient red, green, and blue) color mixing did work. Requesting blue for instance did turn on cyan and magenta. Having a pink-ish hue light on did seem odd, but the mixed result was in fact a slightly washed out blue. Though perhaps if an operation wants “cool” (vs. “warm”)

colors, they may be better requesting a shade of blue closer to cyan. When given a wide selection of colors it performs rather admirably.

Below is a set of graphs demonstrating the operation of the two functions presented. The graphs are in polar coordinate space, where the radius represents saturation and angle represents hue. White is at the center, pure red is at the topmost point. The small black circle represents the desired color (hue-saturation tuple), the brightness levels on the graph represent the output intensity of a light whose color resides at those coordinates.

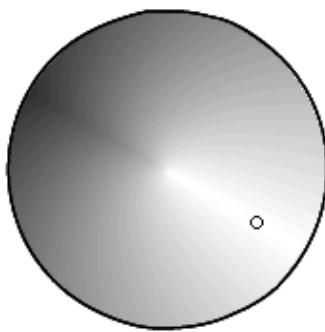


Figure 4.1
1st Method
Blue Requested

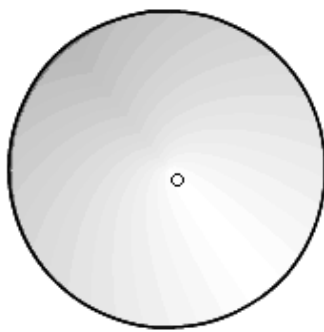


Figure 4.2
1st Method
Very Light Blue Requested

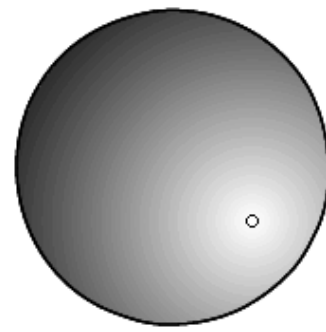


Figure 4.3
2nd Method
Blue Requested

4.3 Beat Detection

There are libraries available on the Internet for beat detection – specifically for beat per minute calculation. Unfortunately, these seem to be geared towards offline operation, and cannot process a stream of audio in real time. They typically yield excellent results. For this purpose, results are needed immediately, and accuracy is not as important.

As is fairly common knowledge, particularly among DJs, the beat is primarily concentrated in lower frequency (bass) parts of the audio spectrum, especially in dance

music. While this does not work for all music, it does work for nearly all dance music, which is what this will be typically used with. This is the general method most beat detection in DJ mixers use. Though a simple concept, implementation had its challenges. There is not any mathematically provable perfect algorithm given the complicated nature of music, and proper operation is somewhat subjective, as it does not necessarily work perfectly.

Initially the use of a search for RMS peaks after a z-transform based low pass filter seemed the best solution. Since there was also an interest in what is going on in other parts of the audio spectrum, it later seemed that the most practical solution would be using a Fourier Transform, and using its output for a variety of processing. Concern for the CPU demands for this method were quickly allayed after testing. On a 600MHz Pentium II, processing audio in real time using FFT required *less* than 5% of the CPU cycles.

The next challenge was detecting the beats as peaks. Observing output from the FFT, filtered into 8 frequency ranges arranged logarithmically (presented numbers are approximate) of <150Hz, 150-300Hz, ... , 11-22KHz, >22kHz. The output that seemed to correlate best to the beat (i.e. connecting that output to a light it would typically flash to the beat rather well) was in the 150-300Hz range. That particular range then needed to be searched for peaks.

To demonstrate the approaches to peak detection used, the following graph will be used as a reference. The audio data presented is not from a real source, but rather a simulation based on the nature of what the audio tends to be with special characteristics to help emphasize the problems encountered. The lines indicate the threshold point which if surpassed by the current reported audio value would be counted as a peak.

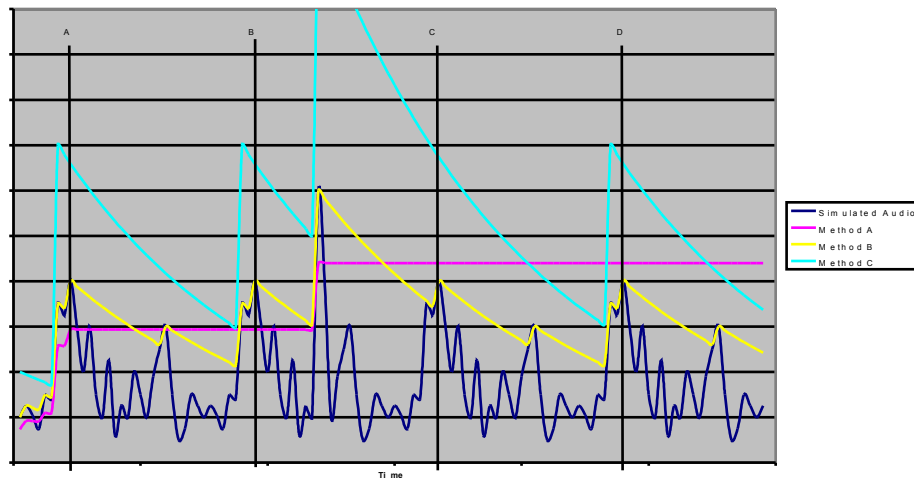


Figure 4.4 Beat Detection Simulations

Method A:

The initial approach was to look for the appearance of a value greater than some fraction (i.e. $> 2/3$) of the biggest peak found, then after some period of time that the current beat should have passed (to avoid marking parts of same beat as separate beats), start listening again. This doesn't work because it returns many false positives (i.e. between beats A&B), and after a very loud sound, it will miss beats or be unable to find anything (i.e. beats C&D).

Method B:

Some sort of adjustable mechanism was needed. To help resolve that, the peak point was set to slowly decay, and anything above the decayed peak was treated as a new beat and the peak value was replaced with the new peak. There was still a problem with false positives between the real beats.

Method C:

Slowing the rate of decay to avoid false positives was one solution tried without much success. Finally, the solution to the false positives problem was mostly resolved by

recording double the actual peak to use as the basis for the decay waiting for the next peak. It is not perfect, but overall works fairly well.

There was still one remaining problem: Testing was done with a small section of music. With other music, beat detection stopped working. After looking at the spectral output for the other music it became apparent that the software was looking at too small a range of frequencies for the beat. In some cases it would work perfectly, but in others it didn't work at all. So instead of using a narrow range, a weighted average of multiple ranges was used:

$$0.5 * [< 150\text{Hz}] + 2.0 * [150-300\text{Hz}] + 0.75 * [300-600\text{Hz}] + 0.33 * [600\text{Hz}-1.1\text{KHz}]$$

(4.7)

The weight coefficients are based purely on experimentation – these seemed to produce the best overall results with most music.

While results are still not perfect, they seem close enough. To a user watching a light show driven by this beat detection method, a clear association can be detected.

5 Other Issues

1. Social

The goal of this project to somewhat ease the work of lighting designers, and also provide a more enjoyable experience for the audience. Its contributions to society are only positive. It would hopefully lead to increased revenues for clubs, DJs, etc., due to a more satisfied audience, and less frustration for the designers. It does not replace the role of lighting designer, so will not be putting anyone out of work. Due to its narrow market, it is also capable of single handedly knocking any of the existing players in the lighting industry out of the market, as existing products have a generally larger intended market (i. e. most are suitable for theatrical work, this really is not).

2. Political

There is no export-restricted technology involved. No external sources of funds were involved. While it may attract curiosity, no corporate entities will be enraged or otherwise bothered by this technology. All-in-all it is rather devoid of political causes or effects.

3. Economic

For smaller setups, the program is not very resource (CPU, memory, etc) demanding, and as a software product may be distributed rather cheaply. For larger scale deployment, such as a large night club, system requirements may be much more demanding, but such customers have larger budgets. The project's deployment expenses are thus in many ways proportional to the budget of the installation site, as so is rather universally practical. Please see Section 6.1 on page 58 for more a more in depth analysis of the economic aspects of this project.

4. Heath and Safety

In some ways this project allows for greater levels of safety than existing products in the field. Specifically it can intelligently manage duty cycle requirements: Some lights may only be on for a certain amount of time before they require a proportional cooling period. The effects of violating of these requirements range from reduced product lifetime, to exploding bulbs (typically housed safely, but none-the-less undesirable), to fire. Other systems leave enforcement up to the user. However, like nearly all existing lighting systems, it is not suitable for pyrotechnic work. Generally it is assumed in the software that minor transitory output glitches are acceptable and will go unnoticed. However if the output is an explosive, a transitory glitch could mean discharge at an inappropriate time.

5. Manufacturability

As software, it may be easily distributed in CD-ROM form. There may be some licensing matters associated with the Fourier transform library used, but if need be, the library may be replaced with a free (though perhaps less efficient) equivalent. See Section 6.1 on page 58 for more information on the costs involved in practical deployment of a system. The system has been carefully designed to be sustainable for easy future expansion, further adding to it practicality as a releasable product. See Section 6.2 on page 60 for more information regarding the sustainability of the system.

6. Sustainability

During the design phase, many decisions were made to ensure the sustainability of the project. Specifically, it incorporates a highly modular design, involving rather minimal coupling between the various parts of the program (achieved using object oriented and messaging architectures). The design allows for expansion with plug-in modules by 3rd party developers to support any unique hardware imaginable, or any new ideas for managing the lights. For a more in depth look at the design decisions related to sustainability, see Section 6.2 on page 60.

7. Environmental Impact

Environmental side effects are near non-existent: As a software product the only direct environmental effect is the disposal of the packaging the product was delivered in. If a reseller wanted to be particularly environmentally conscious, it could even be distributed by way of the Internet, for no direct side effects. It is designed to not be extremely demanding on the computer it is executed on, and has been designed to work with most existing lighting equipment, so not much will need to be discarded or additionally when switching to this system.

8. Usability

It is a great challenge to make software that is both easy to use and powerful. As it stands, this particular program requires the user take some time to explore the features, perhaps through a manual and/or tutorials before they may be able to take full advantage of the program. Ideas for a wizard style automatic configuration script were discussed, but as a matter of practicality given time constraints was not implemented. Some of the foundation work for a wizard to get basic users up and running has been laid, and it is a practical possible add-on. Regarding the UI itself, it was difficult finding suitable testers for such a product, and what suitable testers were found, were unavailable to do actual testing. That being the case, the UI was designed by the developer, based on logical reasoning of what sort of layouts and such would make the most sense to achieve a particular goal. Though, since extensive user testing has not been done, the effectiveness of the UI design is unknown.

9. Lifelong learning

As is exhibited in the Experimental Analysis (Section 8, page 45), much testing and experimentation was done in the course of developing this software. Additionally new APIs (BeOS Multimedia, FFTW, among others) and development tools were explored

and utilized in the development of this project. So, like any real-world project, I engaged in research and experimentation with concepts and tools I did not know much about going into the project.

10. Compassion

The basis of the inspiration of this project lies in compassion. As established in the intro sections (Section 1), this project was begun to help fill a gap where current technology is not really sufficient. Hopefully it will serve to make the lives of some lighting designers more enjoyable as now they will have the technology to do that which was not possible before.

6 Analysis and Conclusions

The various modules and components discussed in the prior sections of this document can be traced to the fulfillment of various functional requirements set forth in the first chapter. The software does what it is required to do. However, perhaps the most important aspects of any product are not its success in meeting a particular functional requirement, but how practical it is as a marketable product and if the combined efforts of the various modules do in fact meet the board goals of the system. This section analyzes the project as a whole in these areas.

6.1 Economic Practicality Analysis

Part of the goals of this project was to make a product accessible to both small-scale mobile DJ operations and large nightclubs. To make a specialized product for the dance lighting market practically marketable, it must be able to accommodate as much of that narrow market as possible. The software has been designed to meet requirements ranging from simple to complex. The remaining factor in its practicality is what costs are involved. As a software product, physical media costs are negligible. However there is the matter of the cost of technology licensing and hardware requirements.

External technology requirements are the BeOS operating system and MIT's FFTW library. Be Inc. has made the Personal Edition of their Operating System freely distributable. The FFTW library was chosen because it is heavily optimized for minimal CPU use. It is not freely distributable, but may be licensed from MIT for a one-time fee of "several thousand dollars for unlimited use" or replaced with another freely distributed library (at a cost of slightly raised CPU requirements).

The hardware requirements are dependent largely on how much the system will need to control. With regard to RAM, a minimum of 64MB is required to guarantee no need for swapping any part of the program to disk while running, which could cause a performance hit. The software itself consumes less than 10MB of disk space on an existing BeOS installation. A full install of BeOS requires 512MB. Or, the program may be run from a CD, with no hard disk space requirements, though with significantly slower program load times. The GUI was optimized for display at 800x600 or 1024x768 pixel resolution, which any laptop in recent years can support. 15-bit or greater color depth would be recommended, but it will function with at least 8-bit color, again something just about any laptop is likely to support. CPU speed requirements are extremely variable - for a small setup, such as a mobile DJ, with less than 40 output channels, a 233MHz Pentium MMX would be sufficient (MMX helps for aspects such as the optimized FFT). CPU usage is approximately linear with regard to the number of channels in use, so a moderate dance club environment with about 500 channels would potentially demand a 2GHz CPU. If demands are even greater, due to the heavily multithreaded design, it will gracefully distribute the workload over additional CPUs in the same system.

These hardware demands are reasonable for their corresponding user-base. A laptop capable of the minimal requirements can be purchased even on a small budget for a few hundred dollars - about the cost of a far less capable single 24-channel hardware dimmer panel. (Though that does not include the cost of the software package, which could potentially be a few hundred dollars itself.) Larger operations have large hardware requirements, but correspondingly they also have larger budgets - a few-thousand dollar workstation is a relatively small investment as part of a \$100,000+ lighting system.

Based on these analysis, it is the author's feeling that the solution developed is practical for marketing to an end user.

6.2 Sustainability Analysis

As is important for any software product, this software has incorporated a variety of methods to ensure its sustainability. Its architecture has been designed to support both vertical (enhancements on existing portions of the software) and horizontal (addition of completely new features) expansion. This section explores the project's success in meeting this goal.

6.2.1 Vertical

6.2.1.1 Software Architecture

Software functionality has been broken into a variety of functions. Effort has been made to ensure that nearly all function calls from external components (plug-ins) are as abstract as possible. That is, most of their operations are well hid (within reason) behind a rather simple parameter list. That being the case, most functionality can be heavily re-written without needing to recompile plug-ins. This has been put into practice already, as the socket and plug item classes were significantly rewritten at one point during the development. While some of the other supporting code within the shell program needed adjustments, existing components were able to be used unmodified.

The object oriented nature of the program also aids the ease of expansion. All channel type objects for example are derived from the ChannelListItem class. If a radically new type of channel were to be implemented, it could be done as yet another subclass of ChannelListItem and be immediately usable by all code related to managing channels. Similarly, adjustments may be made enhancing the base functionality, yielding immediately enhanced derived classes.

6.2.1.2 File Format

Both for individual components and the file as a whole, information on the minimum version for correct processing is maintained. Ideally, these values should not need to

change much. Due to extensive use of flags to denote the existence of various pieces of information, files should be able to maintain some level of reverse computability. If reverse computability is not possible, the user can be informed of the minimum upgrade necessary to open a given file.

6.2.2 Horizontal

6.2.2.1 Additional Components

The program utilizes a modular design to allow for easy expansion through the development of external plug-in modules. This way, the program can expand to support whatever new innovations in lighting technology may be developed, both in terms of support for external hardware as well as new innovations for managing internal operations.

6.2.2.2 Internal Feature Additions

Much of the communication among various parts of the program is done through sending messages rather than traditional function calls. The sender is allowed little or no knowledge of the nature of the receiver. This is particularly useful for this sort of expansion. New portions of the program for example may subscribe to notification services from the AuroranWorld, and instantly receive updates about what is going on within the program (including, but not limited to the state of the channels), without any modification to the other portions of the program.

Advantages are not only limited to being on the receiving end of messages. In most cases the messages may be sent from anywhere as there is no need to know specific details about the sender. This gives the ability to also control or script most of the functionality of the program by sending messages. Control through these messages is the basis of the potential for creating a tool to automatically build a project through a wizard style GUI (discussed further in Section 2.2 on page 15).

6.2.2.3 File Format

Each component can be identified by a 16-bit vendor ID and 32-bit product ID. Its component-specific information is saved in a segment of a project file using those 6 bytes of information. So the project file format allows for easily accepting data from new, unimagined components.

Additionally, the format supports various different types of larger blocks, using a 16-bit identifier. So if the main program has a need to store some new type of information, it may do so using a new segment type. See Appendix C for more information on the file format.

6.2.3 Drawbacks

There is one design decision which has a potential to cause a snag in future developments. At the moment, channel values are relayed between threads using BeOS BMessages. Each update of a new value is sent as a separate event. This method was chosen as it was more practical (easy to implement + not re-inventing that which works well enough already). It is not perfect for the job – mechanisms had to be implemented to prevent the possibility of overfilling a receiving message queue, as could happen if one component generates information faster than that which it is connected to could process it. It has also been discovered in tests that a large portion of CPU time is spent unpacking & repacking messages in processing type components (i.e. the Mixer component). When the operations performed on the data are simple, more time is spent relaying the messages than processing them.

A fully custom message system could help resolve this, but would require most incoming data processing routines to be partially rewritten. At present, while it is inefficient, CPU usage is still low enough that it is not a serious problem. But if for some reason this needs to be optimized, i.e. Other new components are CPU intensive and there is no longer an excess of free CPU time, there would be a need to use a better fitted solution. The good

news is that the rewrites should not need to be all that extreme – the new system would still be a messaging system, just the code directly related to determining what the message is for, and what data it contains would need to be revised.

6.2.4 Conclusion

As a whole the present system architecture is satisfactory in terms of expandability. It is not perfect, and invariably there will be unforeseen challenges, but most alterations to the program will should be easily implementable without a need to alter code in other portions of the program.

6.3 Future Work

6.3.1 Expanded/Refined Features

While the program has met its goals, it is not a perfect tool, and there is room for further enhancement. Here are some ideas currently being experimented with for implementation in future revisions:

6.3.1.1 Audio Processing

The current audio processing does provide useful information, but there is room for further expansion. It does not yet provide an actual Beats Per Minute counter output, which could be potentially useful information. Perhaps even more useful may be a measurement of how close the next beat is to what was predicted, providing a measure of how accurately beat detection is working. If a song for some reason is becoming compatible with the beat detection, the use could make the show do something not as dependent upon the beat.

6.3.1.2 GUI

It was difficult finding suitable testers for this project, and what suitable testers were found, were unavailable to do actual testing. That being the case, the UI was designed by the developer, based on logical reasoning of what sort of layouts and such would make the most sense to achieve a particular goal. Though, since extensive user testing has not been done, the effectiveness of the UI design is unknown. Before any potential release as a commercial product, the UI needs to be reviewed and improved.

6.3.1.3 Project Wizard

It is a great challenge to make software that is both easy to use and powerful. As it stands, this particular program requires the user take some time to explore the features, perhaps through a manual and/or tutorials before they may be able to take full advantage of the program. Ideas for a wizard style automatic configuration script were discussed, but as a matter of practicality given time constraints was not implemented. Some of the foundation work for a wizard to get basic users up and running has been laid, and it is a practical possible add-on using BMessages to automate the creation and configuration of components. The Wizard's operation would be along the lines of the following: It would collect information about what sort of equipment the user has and how the hardware is configured. Then the user may select from a variety of show "types", each with a description. These show types would be linked to a script file, which when run will intelligently set up hardware (inputs, outputs) and supporting (state machines, mixers, etc.) components. The user will then be up and running with a decent light show in a matter of minutes rather than potentially hours of programming a uniquely new show. Advanced users may then tweak the generated show to their liking, or even tweak the scripts so that future generated shows will be closer to what they want. (Such script tweaking could prove invaluable to road shows.)

6.3.1.4 Internationalization

Recently, I have had some direct involvement in Open Source projects. Specifically, I added some features to a BeOS E-Mail, Scooby, client developed primarily by someone in Japan, added some better fault tolerance and management to the BeOS Open Tracker

project coordinated by someone in Germany, and brought up some problems encountered with a Go (board game) server whose developers are from as diverse places as China and Netherlands. The digital world is a small one. It has become clear that no software intended for a broad audience can meet its potential unless it can be fully accepted by its users in their own language. Since people everywhere dance, it is important if this program is to be widely released, that it be able to be localized.

6.3.2 Moving Forward

Development on this project will not be abandoned after the academic year is complete. While there are no specific plans of bringing this product out into the world, it is hoped that someday it or some future project based on this work will become accepted by the dance lighting community. Based on that fact, work is underway to secure rights to the technology with a patent. Additionally, at the moment plans are being made for various events that will use the project during the summer, and hopefully it will continue to get use throughout the following years.

Appendix A: Channel Properties

The components have sets of inputs and outputs, which for the most part correlate to a channel for carrying analog 0.0-1.0 values (as is the typical model for lighting equipment). However they may carry other data types, and therefore have flags for describing what type of data they carry, and what they are compatible with. For example an output may contain a value for the current hue of the lighting, so the type of data is “hue”, however it is being transported as a 0.0-1.0 value, and therefore is comparable with anything that accepts that format (i.e. the brightness of a lamp).

Data channels have the following primary attributes, which are used to establish what is presented to the user, and how the channels may be connected with each other:

- **Channel Name:** The name of the channel displayed in the GUI
- **Outline Level:** Where the channel exists in the hierarchy of channels. Channels are members of a group with a similar outline level whose parent has an outline level of the channel's level minus one & occurs earlier in the master list of sockets or plugs.
- **Owner:** Pointer to the component that owns the channel
- **Type Name:** A string, seen by the user, naming the type of data. Common names include: “Intensity”, “Hue”, “Saturation”, “Position”, “Speed”. Names are for the user, and do not affect compatibility, unless the data format is “Pointer”
- **Expected Rate:**
 - **Continuous:** Data is expected to be flowing in/out of this channel at a fairly regular rate – i.e. constantly adjusting brightness. Continuous inputs may accept continuous or periodic data.
 - **Periodic:** Data is expected to be infrequent – i.e. new data arrives to indicate the beat is occurring in the music. Periodic inputs only accept periodic data.
- **Data Format** (all are 32-bits), outputs may only connect to inputs of the same data format:
 - **Float:** The standard for carrying most data

- **Integer:** Useful for some enumeration data
- **Pointer:** Useful for carrying special data, to be avoided unless absolutely necessary. The type name must match to allow a connection.
- **Restricted Range:** The typical range restriction is 0.0-1.0, however channels may use whatever range they desire. Inputs only accept outputs whose range is within the range of the input.
- **Enumeration:** An input may have a set of default values for certain things. For example, a gobo wheel has a discrete set of patterns to choose from. A list of options pairing names with values may be passed upstream. An enumeration input can accept data from any source that meets the other input criteria. An enumeration output however can only connect to an input that sports enumeration. While a channel is connected, enumeration options may not be changed.

Components are provided as little information as possible about what they are connected to. Specifically, they can test if an input or output channel is connected to anything (thus being able to avoid unneeded work), if they are connected, is it to more than one other component, and are all destinations available for sending data to.

Appendix B: Interface Details & Style Standards

To establish a consistent user interface, various standards were established regarding color, layout, etc.

B.1 Component Standard Classes & Coloring

Class	Description	Color	R,G,B High	R,G,B Normal
Inputs	Sources of Channel Data, i.e. Audio Processor	Green	80,240,120	60,200,100
Show Control	Control Mechanisms, i.e. State Machine	Blue	80,140,255	45,100,200
Passive	Data Processing, i.e. mixer	Yellow	240,240,120	200,200,100
Device Abstraction	Abstraction of devices for easier control, i.e. Group of Lights	Purple	255,140,255	200,100,200
Outputs	Final destination for channel values, i.e. output to PMX bus	Red	250,110,120	210,90,100

B.2 GUI Enhanced Controls Styles

For the sake of consistency, it was decided that there would be two primary styles of slide bars:

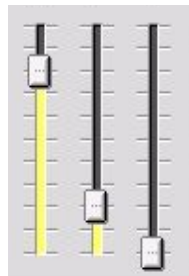


Figure B.1

Slide Style A



Figure B.2

Slide Style B

Style A is for use controlling intensity values, where the amount of yellow shown is analogous to the specified intensity level. Style B is intended for other cases, where it is not so much a matter of on/off or more/less, but rather some position, time offset, mix, etc.

B.3 Menus

Note: a — is used to indicate a separator bar within the menu

B.3.1 File Menu

- New: Creates a new, empty project
- Open...: Prompts user with a file open dialog, used to open a new project
- Save: Saves current project
- Save As...: User selects filename to save current project under, and project is saved.
- ---
- About: About box
- ---
- Quit: Closes program

B.3.2 Tools

- Add Component
 - <list major component groups>
 - <list of available component plug-ins in that group>: Clicking a component in the menu creates a new instance of the component in the project.
- Project Properties: Opens settings window that needs its own section. Basically user adjusts data that goes into the ‘AI’ block when saving the file.

B.3.3 Group Menu

The group menu allows the user to apply a filter limited which components are displayed in the component list.

- All: Show everything
- Inputs: Show only input class components
- Show Managers: Show only show manager class components
- Passive Processors: Show only passive processor components

- Device Grouping: Show only device grouping components
- Device Abstraction: Show only device abstraction components
- Output: Show only output class components
- User Defined: Show only user defined components
- ----
- <user's custom groups go here>
- ----
- Add Group... : Opens dialog asking for the name of a group
- Delete Group
 - <user;s custom groups repeated here>: If clicked, asks if you really want to delete it. Answering yet removes that custom group.

Appendix C: File Format

C.1 File Format Notation

I have not found any standardized means of expressing saved data, so I have developed one for use in this document. It is built on the general idea that a file's contents are typically a combination of some sort of hierarchy, and within levels of that hierarchy, there are typically sets of homogenous records.

The content of the file is expressed as a table with four columns: size (in bytes – either a fixed value, verbal explanation, or reference to another datum's value (indicated by italics)) of a datum, the type of the data (typically expressed as a standard C language data type), the content (either a fixed value, or a symbolic name that may be referenced elsewhere), and a verbal description of the content of the datum. If a symbolic name used for a datum is referenced elsewhere, later in the file, the name should be italicized to make the fact that it is a reference to something established within the file itself clear.

The sense of hierarchy is expressed by heavy bold vertical lines to the left of a table outlining a file's contents. The leftmost lines form the outer grouping of data, or the highest level in the hierarchy – everything established to the right of a line included as part of its “record”. A single record may therefore have sets of other records stored within it, indicated by increasing levels of more bold lines to the right of a line and within its span. At the bottom of the line, after the last element of the file at that level, there is the letter “x” followed by some value (either a constant, a reference to something established earlier in the file (indicated by italics), or a verbal description which could presumably be dealt with by the program by detecting an EOF or something similar). The value indicated how many times the collection of data outlined to the right of that bar is repeated within the file.

C.2 General File Format

This is the general wrapper format for storing information in a project file. The overall approach taken to the storage system is to provide several layers of encapsulation (i.e. the way networks pass data across at various layers): a generalized data processor handles one layer of data, then passes extra data on to a higher layer for further processing. The lowest storage layer is built around storing a blocks of information. Each block is identified by its type. Blocks that are not understood are skipped over to promote some level of easy backward compatibility if new features are added.

Size	Type	Content	Description
8	char x 8	'TSAurora'	Sanity check for the file type
4	uint32	1	Version of the file format - only to be increased if reverse compatibility is broken
4	uint32	BlockCount	Number of Entries in this file (1 + Number of Components + Number of Groups)
2	char x 2		Block type identifier
4	uint32	BlockSize	Size of this block (does not include 2 byte identifier, and this 4 byte value)
BlockSize	????	????	Content of this block

x BlockCount
x 1

There are currently 3 standard types of data blocks for use in the file, described in the following sections. They are typically expected to exist in the file in the order they are listed here.

C.2.1 'AI' Block: Project Information

The 'AI' block contains information some simple project information. Its primary use is for loading a project as a component within another project, and therefore contains some flags for how to interpret it as a component.

Size	Type	Content	Description
2	char x 2	'AI'	Block type identifier
4	uint32	578	Size of this block
32	String	Name	Name for the Project / Component
28	String	Prefix	Prefix for use with creating a component instance (new components are named as Prefix001, Prefix002, etc.)
4	uint32	Version	Revision of this file, automatically incremented each time it is saved.
2	uint16	Flags	Various options, currently only for which class of component this is: 0x0000 = Input 0x0001 = Output 0x0002 = Show Control 0x0003 = Passive Processor 0x0004 = Device Abstraction 0x0005 = Device Grouping 0x0006 = User Control / Other
512	String	Description	User's description of their project

x 1

C.2.2 'AC' Block: Component Information

'AC' blocks contain information specifying what components to load, how to connect them, and as well as information specific to the particular component in its own format.

Size	Type	Content	Description
2	char x 2	'AC'	Block type identifier
4	uint32	(varies)	Size of this block
4	char x 4	VendorID	Unique tag identifying the component's vendor: needed to determine what plug-in to load
2	char x 2	ProductID	Unique tag by the vendor to specify the component
4	uint32	Version	Version of the component's internal format
32	String	Name	Name for this instance of the component
4	uint32	Offset	Where in the file to jump to skip over the component specific data. (relative to the start of the file)
????	????	????	Block of data provided by the component plug-in, specifics listed in the section for this component.
1	char	0	Location that <i>Offset</i> points to.
4	uint32	ConnCount	Number of connections to be made
4	uint32	MyPlug	Which of my plugs I am connecting (1st in PlugList is 0, 2nd is 1, etc)
4	uint32	Destination	Which component this is connected to (1st component in file is 0, 2nd is 1, etc)
4	uint32	DestSocket	Which of the <i>Destination</i> 's sockets is being connected to by <i>MyPlug</i>

x ConnCount

x Number of Component Instances

C.2.3 ‘AG’ Block: Group Information

The ‘AG’ blocks contain user-defined groupings of components.

Size	Type	Content	Description
2	char x 2	‘AG’	Block type identifier
4	uint32	(varies)	Size of this block
32	String	Name	Name of this Group
4	uint32	MemberCount	Revision of this file, automatically incremented each time it is saved.
4	uint32	Component	A component in this group (1st component in file is 0, 2nd is 1, etc)

x *MemberCount*
x Number of user-defined groups

C.3 Utility File Formats

This section contain details pertaining to the storage of data not relevant to any other section of this appendix.

C.3.1 ChannelTools saved data

The ChannelTools GUI tool (discussed earlier in Section 2.4.2.1 on page 20) also provides utility functions for the channels it manages. One such function is the ability to dump and load information about the channels it manages to disk. This is used by a variety of components to store information about user defined channels. The components may specify a chunk of additional information to be stored with the standard channel information.

Size	Type	Content	Description
2	char x 2	'CL'	Magic number to identify section
2	uint16	ChanCount	Number of channels described
2	uint16	CDSIZE	Size of this channel's entry
2	uint16	OutlineLvl	OutlineLevel relative to parent channel
2	uint16	NStrLen	Length of the channel's name
<i>NStrLen</i>	String	Name	Name of the Channel
2	uint16	TStrLen	Length of the channel's type's name
<i>TStrLen</i>	String	TypeName	Name for the channel's data type
2	uint16	Flags	Channel flags, identifying its data type, etc.
4	CHData	Min	Minimum value the channel may carry
4	CHData	Max	Maximum value the channel may carry
2	uint16	EnumCount	Number of enumerated values
4	CHData	Value	Value to pass for this enumeration
2	uint16	EStrLen	Length of enumeration's name
<i>EStrLen</i>	Str	EnumName	Enumeration's name
x <i>EnumCount</i>			
????	?????	Extra Data	Block of additional information of a fixed specified size to include with information about the channel.
x <i>ChanCount</i>			
x 1			

C.4 Component Specific File Formats

This section contains details of what data gets stored for each particular component. Particularly, this outlines what fills the space labeled as “?????” in Appendix .2.2 above.

C.4.1 Simple Desk

(nothing saved)

C.4.2 Audio Processor

(nothing saved)

C.4.3 Signal Generator

Size	Type	Content	Description
4	uint32	CHCount	Number of User-Defined Channels
2	uint16	NameLen	Size of the Name of the channel
NameLen	String	Name	Name of the channel
4	Float	Period	in milliseconds
1	Char	Type	0x01 = Sine 0x02 = Cosine 0x03 = Triangle 0x04 = Saw 0x05 = Square 0x06 = Pulse
4	float	Span	(Maximum – Minimum)
4	float	Base	Minimum
4	bigtime_t	Offset	in milliseconds (not % like in the GUI)
28	float	reserved	

x CHCount

x 1

C.4.4 Light Bar Simulator

Size	Type	Content	Description
sizeof(BPoint)	BPoint	WindowPos	Position of the window
2	uint16	10	Number of columns of lamps, fixed at 10, stored if it is made variable later.
2	uint16	1	Number rows of lamps, fixed at 1, but could be expanded upon later.
sizeof(rgb_color)	rgb_color	Color	Color of a lamp, starting with CH0

x 10

x 1

C.4.5 PMX Output

Size	Type	Content	Description
4	uint32	Flags	0x0001 Indicates values for the 120 channels are stored on the disk, and should be read back on load. 0x0000 Data rate is 9600bps 0x0010 Data rate is 19200bps 0x0020 Data rate is 57600bps 0x0030 Data rate is 115200bps 0xFF00 Bits used to indicate serial port number (0-255) (port 0 = COM1, 255 = instead use port named below)
4	Float	Value	Saved values for output channels
x 120			
1	uint8	DevNameLen	Length of port's name
<i>CDevName</i>	String	DevName	Name of serial port to use
x 1			

C.4.6 State Machine

Size	Type	Content	Description
???????	CList*	Plugs	User-defined plugs
???????	CList*	Sockets	User-defined sockets
???????	CList*	Internal	User-defined internal variables
16	uint16	StateCount	Number of states
16	uint16	SNameLen	Length of the name of the state
<i>SNameLen</i>	String	SName	Name of the state
x <i>StateCount</i>			
2	uint16	ReqCount	Number of Requirements for this state
2	uint16	ReqSize	Size of this segment
2	uint16	DestState	Index of destination state
2	uint16	ParentReq	Index of this Requirement's parent Requirement
1	uint8	ReqType	Type of requirement
???????	???????	ReqData	requirement-specific information
x <i>ReqCount</i>			
2	uint16	EntryCount	Number of on state entry events
2	uint16	Target	Index on output channel to send data to
1	uint8	DoWhat	What to do on the event
4	???????	Data	event-dependant data
x <i>EntryCount</i>			
2	uint16	ExitCount	Number of on state entry events
2	uint16	Target	Index on output channel to send data to
1	uint8	DoWhat	What to do on the event
4	???????	Data	event-dependant data
x <i>ExitCount</i>			
x <i>StateCount</i>			
x 1			

* Records of type CList are defined in Section .3.1 on page 75.

C.4.7 Switch/Fork

	Size	Type	Content	Description
x <i>PlugCount</i>	4	uint32	PlugCount	Number of user defined output plugs
	32	String	PlugName	Name of the plug
x number of states	1	uchar	EndValue	The end value of this region. We have reached the last region when this is 255. The start value is the previous end value + 1, or 0 for the first entry
	32	String	StateName	Name of the state
x <i>ConnCount</i>	4	uint32	SocketCount	Number of user defined input plugs
	32	String	SocketName	Name of the socket
x number of states	4	uint32	ConnCount	Number of connections in the given state
	4	uint32	Destination	An element from the plug list that this socket is connected to in this state
x <i>SocketCount</i>				
x number of states				
x <i>SocketCount</i>				
x 1				

C.4.8 Rect to Polar / Polar to Rect

(nothing saved)

C.4.9 4CH Mixer

(nothing saved)

C.4.10 Group of Lights

Data for this component is saved through a call to save data in the format explained in appendix .3.1 on page 75. The additional payload saved in the Extra Data field is a data structure which includes the 12 bytes of unused information and the HSV color for the image.

Appendix D: Use of the Cortex Utility

For connecting to a source of audio, in the current implementation, the user must manually make a connection from the audio source to the program's listening BBufferConsumer. This may be done using a BeOS utility program called Cortex. It allows direct control over the BMediaRoster (all multimedia object) through a rather friendly user interface. Because of the ease of operation of using Cortex and the limited time to complete the project, this is the only way to attach to an audio source. The utility is a open-source program maintained primarily by Eric Moon (em@be.com) and managed though Sourceforge at <http://cortex.sourceforge.net/>.

To connect an Audio Processor component (Section 3.1.1) to an audio source, simply draw a line by dragging from the audio source in Cortex, to the “Aurora” item in Cortex:

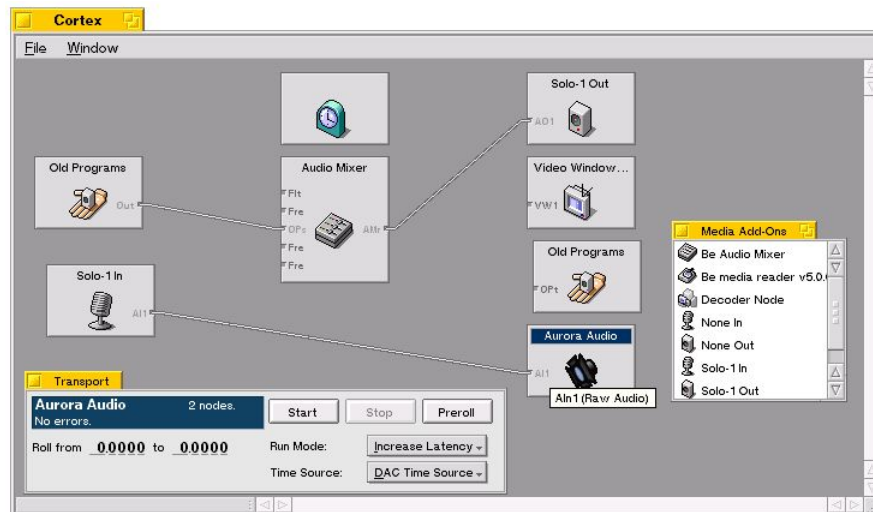


Figure D.1 Cortex GUI

After dragging to make the connection, the user can start or stop the flow of data using the corresponding options in the Transport dialogue.

Appendix E: Class Relationships

The diagram below shows the overall relationships between classes. Almost all objects have a pointer to their AuroranWorld, as that object contains pointers to any other significant object, and useful tools for managing the components, channels, etc. A channel's plugs and sockets are contained both in local lists and the master list. Content from the master lists in the AuroranWorld are used for the GUI displayed lists.

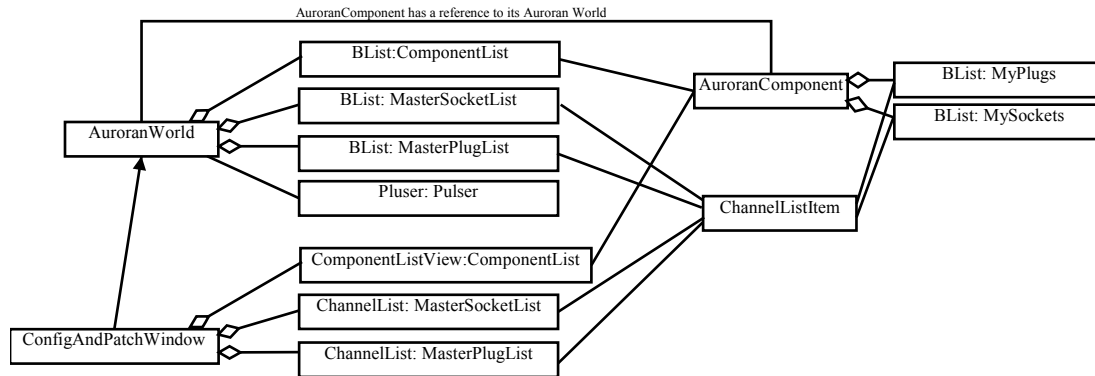


Figure E.1 Overall Object Relationships

Components are derived from a generic component class which takes care of most of the more complicated operations related to setting up a component & its channels and displaying it. Since it is also a list item, the component class itself contains the functions for rendering itself as an item in the component list in the GUI. All components are all also BLoopers – they have their own threads which operate at a priority of about 70 (GUI priority is 20, real time priority is 120), and receive messages from other components to indicate incoming channel data.

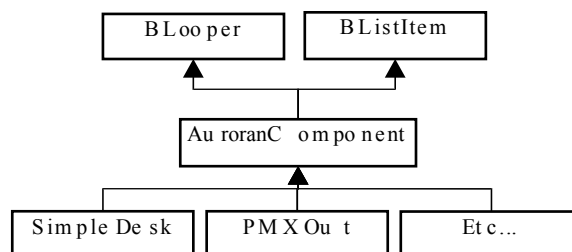


Figure E.2 Auroran Component Class Hierarchy

Most objects that are featured is lists in the GUI (components, channels) are inherited from at least a BListItem, as in the previous chart showing the inheritance relationships for a component. The charts below covers the inheritance relationships that are key to the containers (BLists in the GUI) and also the channels.

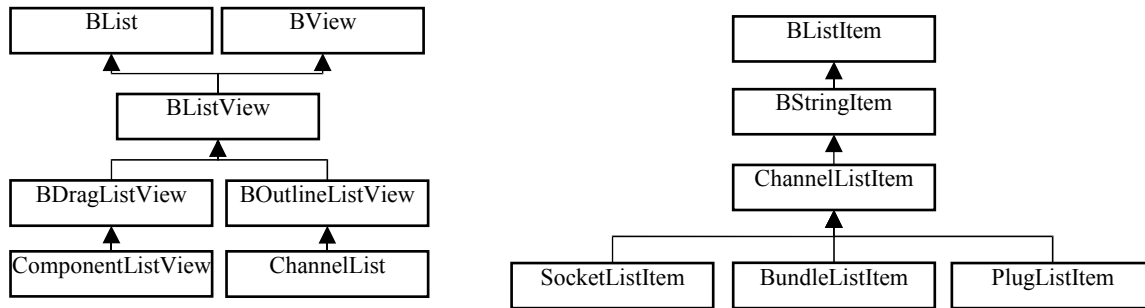


Figure E.3 List View and ChannelItem Inheritance Relationships

Appendix F: Source Code

Source Files

AColorControl.h.....	85
AColorControl.cpp.....	87
AboutBox.h.....	94
AboutBox.cpp.....	95
AuroraGeneral.h.....	97
AuroranWorld.h.....	98
AuroranWorld.cpp.....	100
BDragListView.h.....	103
BDragListView.cpp.....	104
ChannelList.h.....	108
ChannelList.cpp.....	109
ChannelListItem.h.....	110
ChannelListItem.cpp.....	113
ChannelTools.h.....	120
ChannelTools.cpp.....	122
Component.h.....	136
Component.cpp.....	140
ConfigAndPatchWindow.h.....	155
ConfigAndPatchWindow.cpp.....	157
ILooper.h.....	181
ILooper.cpp.....	182
Icons.h.....	183
Icons.cpp.....	184
MaxMinCalculator.h.....	185
MaxMinCalculator.cpp.....	186
PercentControl.h.....	191
PercentControl.cpp.....	192

PlugList.h.....	194
PlugList.cpp.....	195
PrettySlider.h.....	197
PrettySlider.cpp.....	198
Pulse.h.....	199
Pulse.cpp.....	201
ResourceSet.h.....	206
ResourceSet.cpp.....	210
SocketList.h.....	226
SocketList.cpp.....	227
SuperButton.h.....	230
SuperButton.cpp.....	231
ThreadPrimitive.h.....	232
Triangle.h.....	234
Triangle.cpp.....	235
app.h.....	237
app.cpp.....	238
colorspace.h.....	239
colorspace.cpp.....	240
main.cpp.....	243
Audio Processor Component	244
AudioComponent.h.....	244
AudioComponent.cpp.....	245
AudioListener.h.....	248
AudioListener.cpp.....	250
AudioProcessor.h.....	257
AudioProcessor.cpp.....	258
Simple Mixer Component.....	261
SimpleMixer.cpp.....	261
Signal Generator Component.....	265

Types.h.....	265
WaveView.h.....	266
WaveView.cpp.....	267
Component.cpp.....	270
State Machine Component	281
StateStuff.h.....	281
StateStuff.cpp.....	284
Requirement.cpp.....	288
State.cpp.....	293
StateMachine.cpp.....	296
4 Scene Mixer Component.....	315
Mixer.cpp.....	315
Switch Component	324
KrazyCtl.h.....	324
KrazyCtl.cpp.....	325
Component.cpp.....	332
Polar Conversion Component	344
Component.cpp.....	344
Group of Lights Component	347
Component.cpp.....	347
PMX Output Component.....	358
COMMThread.h.....	358
COMMThread.cpp.....	359
PMXOutput.cpp.....	362
Bar of 10 Lights Simulator Component.....	368
LightBar.h.....	368
Lightbar.cpp.....	369
Component.cpp.....	371

Filename: AColorControl.h

```
#include <Control.h>
#include "colorspace.h"

enum {
    ACC_VALUE=0x01,
    ACC_TOLERANCE=0x02,
    ACC_PREVIEW=0x04,
    ACC_PREVIEW_BOTTOM=0x04,
    ACC_PREVIEW_LEFT=0x0C
};

typedef struct hsvt {
    unsigned char Hue;
    unsigned char Saturation;
    unsigned char Value;
    unsigned char Tolerance;
};

typedef union hsvtcombo {
    hsvt h;
    uint32 x;
};

class AColorControl: public BControl {
private:
    BBitmap * MyBitmap;
    BControl * ValSlide;
    BControl * TolSlide;
    hsv_fcolor MyColor;
    bool MouseMoving;
    int Flags;
    void DrawPreview(BRect updateRect);

    void RenderWheel(BBitmap * dest, bool Tightness);
    int wheel_w;
    int wheel_h;
    float wheel_s;
    hsv_fcolor FindColorPoint(BPoint pt, bool quick);
    bool FirstDraw;
    bool NeedsRender;
    float MyValue;
    float MyTolerance;
    BLooper * DrawLooper;
public:
    AColorControl(int ColorFlags, BRect frame, const char * name,
    BMessage * message, uint32 resizingMode);
    ~AColorControl();
    void AllAttached();
    void Halt();
    void MessageReceived(BMessage * msg);
    void Draw(BRect uR);
    void MouseDown(BPoint point);
    void MouseUp(BPoint point);
    void MouseMoved(BPoint pt, uint32 code, const BMessage *msg);
    hsv_fcolor Color();
```

```
void SetColor(hsv_fcolor color);  
void DrawWheel();  
float Tolerance() {return MyTolerance;}  
void SetTolerance(float Tol);  
  
};
```

Filename: AColorControl.cpp

```
#include <InterfaceKit.h>
#include <Locker.h>
#include "AColorControl.h"
#include <math.h>
#include "colorspace.h"
#include "PrettySlider.h"
#include <stdio.h>

class CommandLooper : public BLooper {
private:
    AColorControl * Target;
    BMessenger * MyMsgr;
    bool UpdateNeeded;
public:
    bool isBusy;
    BLocker BusyLock;
    CommandLooper(AColorControl * Target) : BLooper("Color Wheel
Drawer", (B_NORMAL_PRIORITY+B_LOW_PRIORITY)/2), Target(Target)
{isBusy=false;Run();MyMsgr = new BMessenger(this);}
    void Redraw() {
        //return;
        //printf("Request to draw...\n");
        BusyLock.Lock();
        if (!isBusy) {
            MyMsgr->SendMessage('Draw');
            isBusy=true;
        }
        UpdateNeeded=true;
        BusyLock.Unlock();
    }
private:
    void MessageReceived(BMessage * msg) {
        while (UpdateNeeded) {
            UpdateNeeded=false;
            Target->DrawWheel();
        }
        //printf("Drawn...\n");
        BusyLock.Lock();
        isBusy=false;
        BusyLock.Unlock();
        //BLooper::MessageRecieved(msg);
    }
};

const rgb_color kViewGray = { 216, 216, 216, 255};

void AColorControl::SetTolerance(float tol) {
    MyTolerance=tol;
    if (Window()->LockWithTimeout(5000)==B_OK) {
        if (Flags&ACC_TOLERANCE) {
            TolSlide->SetTarget(NULL);
            TolSlide->SetValue(MyTolerance*511);
            TolSlide->SetTarget(this);
        }

        Window()->Unlock();
    }
}
```

```

    }
    if (DrawLooper) ((CommandLooper*)DrawLooper)->Redraw();
}

void AColorControl::SetColor(hsv_fcolor color) {

    //if (color!=Color()) {

        MyColor=color;
        MyValue=MyColor.v;

        if (Window()->LockWithTimeout(5000)==B_OK) {
            Draw(BRect(0,0,Frame().Width(),Frame().Height()));
            if (Flags&ACC_VALUE) {
                ValSlide->SetTarget(NULL);
                ValSlide->SetValue(MyValue*511);
                ValSlide->Invalidate();
                ValSlide->SetTarget(this);
            }

            Window()->Unlock();
        }
        if (DrawLooper) ((CommandLooper*)DrawLooper)->Redraw();

    //}

}

hsv_fcolor AColorControl::Color() {

    while (MyColor.h<0.0) MyColor.h+=M_PI*2;
    while (MyColor.h>M_PI*2) MyColor.h-=M_PI*2;
    MyColor.v=MyValue;
    return MyColor;

}

hsv_fcolor AColorControl::FindColorPoint(BPoint pt, bool quick) {

    float d,theta,xx,yy;
    hsv_fcolor Color;

    xx=wheel_w/2-pt.x;
    yy=wheel_h/2-pt.y;
    Color.s=sqrt(xx*xx+yy*yy);

    if (Color.s<=wheel_s || quick==false) {
        if (yy==0.0) yy=0.000000000001;
        Color.h=atan(xx/yy);
        if (yy < 0) {
            Color.h = M_PI+Color.h;
        }
    }

    return Color;

}

void AColorControl::RenderWheel(BBitmap * dest, bool Tightness) {

```



```

    rgb_color * Data = (rgb_color *)dest->Bits();
    float xx,yy;
    hsv_fcolor tcolor;
    int pos=0;
    float tolerance =0;
    hsv_fcolor TheColor = MyColor;
    if (Flags & ACC_TOLERANCE) tolerance = TolSlide->Value();
    for (int y = 0;y <= wheel_h; y++) {
        pos = y * (dest->BytesPerRow()>>2);
        for (int x = 0;x<=wheel_w;x++) {
            tcolor=FindColorPoint(BPoint(x,y),true);
            if (tcolor.s>(wheel_s-2) && tcolor.s <=wheel_s-0.5) {
                xx=wheel_w/2-x;
                yy=wheel_h/2-y;
                if ((-xx)<yy) {
                    Data[pos++]=(rgb_color){128,128,128,0};
                } else {
                    Data[pos++]=(rgb_color){255,255,255,0};
                }
            } else if (tcolor.s>wheel_s-0.5) { //outside of circle...
                Data[pos++]=kViewGray;
            } else {
                tcolor=FindColorPoint(BPoint(x,y),false);
                tcolor.s/=(wheel_s-2);
                rgb_color a;
                if ((Flags & ACC_TOLERANCE) && tolerance < 0.95*511) {
                    float m = 1.0-ColorMatch(tcolor.h,tcolor.s,TheColor.h,
TheColor.s);
                    m=pow(m,pow(2,(511.0/2-tolerance)/80.0));

                    a = rgb_fcolor(hsv_fcolor(tcolor.h,tcolor.s,m)).rgb();
                    //a = rgb_fcolor(m,m,m).rgb();
                } else {
                    a = rgb_fcolor(hsv_fcolor(tcolor.h,tcolor.s,1.0)).rgb();
                }

                Data[pos++]=(rgb_color){a.blue,a.green,a.red,0.0};
            }
        }
    }
}

AColorControl::AColorControl(int ColorFlags,BRect frame, const char *
name, BMessage * message,uint32 resizingMode) : BControl(frame,name,
NULL,message,resizingMode,B_WILL_DRAW), Flags(ColorFlags), MyColor
(0.0,0.0,1.0), MyValue(1.0), MyTolerance(1.0) {
    DrawLooper=NULL;
    MyBitmap = new BBitmap(BRect(0,0,frame.Width(),frame.Height()),
B_RGB32);

    FirstDraw=true;
    // int Words = MyBitmap->BytesPerRow()/4;
    // HeightPixels--;
    wheel_w = Frame().Width()-((Flags&ACC_VALUE)?30:0)-
((Flags&ACC_TOLERANCE)?40:0)-((Flags&ACC_PREVIEW_LEFT)
==ACC_PREVIEW_LEFT)?30:0;
    wheel_h = Frame().Height()-((Flags&ACC_PREVIEW && !
((Flags&ACC_PREVIEW_LEFT)==ACC_PREVIEW_LEFT))?30:0);
    wheel_s =wheel_h/2;

```

```

        if (wheel_w < wheel_h) wheel_s = wheel_w / 2;

        int w = Frame().Width();
        int h = Frame().Height();

        if (Flags & ACC_VALUE) {
            PrettySlider * Slider = new PrettySlider(BRect(w -
                ((Flags & ACC_TOLERANCE) ? 70 : 30), 0, w - ((Flags & ACC_TOLERANCE) ? 40 : 0), h),
                "Slide_Value", "Value", NULL, 0, 511, B_VERTICAL, B_BLOCK_THUMB, B_FOLLOW_TOP |
                B_FOLLOW_LEFT, B_FRAME_EVENTS | B_WILL_DRAW | B_NAVIGABLE);
            Slider->SetBarThickness(5);
            Slider->SetHashMarks(B_HASH_MARKS_BOTH);
            Slider->SetHashMarkCount(11);
            Slider->SetValue(511);
            BMessage * msg = new BMessage('ChgV');
            Slider->SetModificationMessage(msg);
            Slider->SetTarget(this);
            this->AddChild(ValSlide = Slider);
        } else {
            ValSlide = NULL;
        }

        if (Flags & ACC_TOLERANCE) {
            PrettySlider * Slider = new PrettySlider(BRect(w - 40, 0, w, h),
                "Slide_Value", "Tolerance", NULL, 0, 511, B_VERTICAL, B_BLOCK_THUMB,
                B_FOLLOW_TOP | B_FOLLOW_LEFT, B_FRAME_EVENTS | B_WILL_DRAW | B_NAVIGABLE);
            Slider->SetBarThickness(5);
            Slider->SetTarget(this);
            Slider->SetHashMarks(B_HASH_MARKS_BOTH);
            Slider->SetHashMarkCount(11);
            Slider->SetValue(511 / 2);
            BMessage * msg = new BMessage('ChgT');
            Slider->SetModificationMessage(msg);
            this->AddChild(TolSlide = Slider);
            DrawLooper = new CommandLooper(this);
        } else {
            TolSlide = NULL;
        }
        RenderWheel(MyBitmap, true);
    }

void AColorControl::Halt() {
    if (DrawLooper) {
        BLooper * x = DrawLooper;
        DrawLooper = NULL; // make sure it is not called after deletion
        ((CommandLooper*)x)->LockLooper();
        ((CommandLooper*)x)->Quit();
        // delete x; doh! this isn't needed when using Quit
        DrawLooper = NULL;
    }
    // BView::DetachedFromWindow();
}

AColorControl::~AColorControl() {
    delete MyBitmap;
}

void AColorControl::DrawPreview(BRect updateRect) {

```

```

        if ((Flags&ACC_PREVIEW_LEFT)==ACC_PREVIEW_LEFT) {
            SetHighColor(rgb_fColor(MyColor).rgb());
            int w = Frame().Width();
            if (updateRect.right<w) w = updateRect.right;
            int x = Frame().Width()-30;
            if (updateRect.left>x) x = updateRect.left;

            FillRect(BRect(x,updateRect.top,w,updateRect.bottom));

        } else if (Flags&ACC_PREVIEW_BOTTOM) {
            int w = Frame().Width()-(Flags&ACC_VALUE)?30:0)-
((Flags&ACC_TOLERANCE)?40:0);
            if (updateRect.right<w) w = updateRect.right;
            int y = Frame().Height()-29;
            if (updateRect.top>y) y = updateRect.top;

            FillRect(BRect(updateRect.left,y,w,updateRect.bottom));
        }
    }

void AColorControl::Draw(BRect updateRect) {
    BRect r2 = updateRect;
    if (r2.right>wheel_w) r2.right=wheel_w;
    if (r2.bottom>wheel_h) r2.bottom=wheel_h;
    DrawBitmapAsync(MyBitmap,r2,r2);
    DrawPreview(updateRect);

    if (ValSlide) ValSlide->SetValue(MyValue*511);
    if (TolSlide) TolSlide->SetValue(MyTolerance*511);

    SetDrawingMode(B_OP_INVERT);
    StrokeEllipse(BPoint(-sin(MyColor.h)*MyColor.s*(wheel_s-2)
+wheel_w/2,wheel_h/2-cos(MyColor.h)*MyColor.s*(wheel_s-2)),3.0,3.0);
    SetDrawingMode(B_OP_COPY);
    Flush();
}

void AColorControl::MessageReceived(BMessage * msg) {
    //cout << "msg recieved" << endl;
    msg->PrintToStream();
    switch (msg->what) {
        case 'ChgV':
            MyValue=MyColor.v=ValSlide->Value()/511.0;
            DrawPreview(BRect(0,0,Frame().Width(),Frame().Height()));
            Invoke();
            break;
        case 'ChgT':
            if (DrawLooper) ((CommandLooper*)DrawLooper)->Redraw();
            MyTolerance=TolSlide->Value()/511.0;
            Invoke();
            break;
        default:
            BView::MessageReceived(msg);
    }
}

void AColorControl::AllAttached() {

```

```

        if (Flags & ACC_VALUE) ValSlide->SetTarget(this,Window());
        if (Flags & ACC_TOLERANCE) TolSlide->SetTarget(this,Window());

        BView::AllAttached();
    }

void AColorControl::MouseDown(BPoint pt) {

    if (FindColorPoint(pt,true).s<=wheel_s) {
        SetDrawingMode(B_OP_INVERT);
        StrokeEllipse(BPoint(-sin(MyColor.h)*MyColor.s*(wheel_s-2)
+wheel_w/2,wheel_h/2-cos(MyColor.h)*MyColor.s*(wheel_s-2)),3.0,3.0);
        MyColor=FindColorPoint(BPoint(pt.x,pt.y),false);
        MyColor.s/=(wheel_s-2);
        if (MyColor.s>1.0) MyColor.s=1.0;
        MyColor.v=MyValue;
        MouseMoving=true;
        StrokeEllipse(BPoint(-sin(MyColor.h)*MyColor.s*(wheel_s-2)
+wheel_w/2,wheel_h/2-cos(MyColor.h)*MyColor.s*(wheel_s-2)),3.0,3.0);

        SetDrawingMode(B_OP_COPY);
        DrawPreview(BRect(0,0,Frame().Width(),Frame().Height()));
        if (DrawLooper) ((CommandLooper*)DrawLooper)->Redraw();

        Invoke();
    }

    //Position=1;
    //Draw(BRect(0,0,Frame().Width(),Frame().Height()));
}

void AColorControl::DrawWheel() {
//LockLooper();
    BWindow * hWin;
    if ((hWin=Window())==NULL) return; //apparently we were too late
    RenderWheel(MyBitmap,true);
    if (hWin->LockWithTimeout(800)==B_OK) {
        Draw(BRect(0,0,wheel_w,wheel_h));
        hWin->Unlock();
    }
//UnlockLooper();
}

void AColorControl::MouseUp(BPoint pt) {
    MouseMoving=false;
    if (Flags & ACC_TOLERANCE) {
        RenderWheel(MyBitmap,true);
        Draw(BRect(0,0,wheel_w,wheel_h));
    }
}

void AColorControl::MouseMoved(BPoint pt, uint32 code, const BMessage
*msg) {
    int32 buttons=0;
    Window()->CurrentMessage()->FindInt32("buttons", &buttons);
}

```

```

        if (BRect(0,0,Frame().Width(),Frame().Height()).Contains(pt)
        && (buttons & 1) && MouseMoving) {

            SetDrawingMode(B_OP_INVERT);

            //float Hue = MyColor.h;
            //float Saturation = MyColor.s;
            StrokeEllipse(BPoint(-sin(MyColor.h)*MyColor.s*(wheel_s-2)
+wheel_w/2,wheel_h/2-cos(MyColor.h)*MyColor.s*(wheel_s-2)),3.0,3.0);

            MyColor=FindColorPoint(BPoint(pt.x,pt.y),false);
            MyColor.s/=(wheel_s-2);
            if (MyColor.s>1.0) MyColor.s=1.0;
            MyColor.v=MyValue;
            StrokeEllipse(BPoint(-sin(MyColor.h)*MyColor.s*(wheel_s-2)
+wheel_w/2,wheel_h/2-cos(MyColor.h)*MyColor.s*(wheel_s-2)),3.0,3.0);
            //printf("Hue: %f\n",MyColor.h);

            SetDrawingMode(B_OP_COPY);
            DrawPreview(BRect(0,0,Frame().Width(),Frame().Height()));
            if (DrawLooper) ((CommandLooper*)DrawLooper)->Redraw();

            Invoke();

        }
}

```

Filename: AboutBox.h

```
void ShowAboutBox();
```

Filename: AboutBox.cpp

```
#include "AboutBox.h"
#include <InterfaceKit.h>
#include <TranslationKit.h>
#include <iostream.h>

class PictureView: public BView {
private:
    BBitmap * MyPicture;
    BFont fancy;
    static float offsetstring;
public:
    PictureView() : BView(BRect(0,0,512,256),"About Picture",
B_FOLLOW_NONE,B_WILL_DRAW|B_NAVIGABLE) {
        MyPicture= BTranslationUtils::GetBitmap(B_JPEG_FORMAT,
1001);

        fancy=be_plain_font;
        fancy.SetFamilyAndStyle("CommercialScript","Regular");
        fancy.SetSize(64);
    }
    ~PictureView() {delete MyPicture;}
    void Draw(BRect updateRect) {
        SetDrawingMode(B_OP_COPY);
        DrawBitmap(MyPicture,BRect(0,0,512,256));
        SetFont(&fancy,B_FONT_ALL);
        MovePenTo(20,60);
        SetHighColor((rgb_color){255,240,144,250});
        SetDrawingMode(B_OP_ALPHA);
        DrawString("Aurora");
        SetFont(be_plain_font,B_FONT_ALL);
        SetFontSize(12);

        MovePenTo(20,74);
        DrawString("Intelligent Lighting Control System");

        SetHighColor((rgb_color){255,255,255,255});
        MovePenTo(20,120);
        DrawString("Version 0.0.1");
        MovePenTo(20,133);
        DrawString("2003 Theodore K. Schundler");

        if (offsetstring==0.0) {
            offsetstring=(Frame().Width()-StringWidth("Click window
or press any key to close.))/2;
        }
        MovePenTo(offsetstring,Frame().Height()-3);
        DrawString("Click window or press any key to close.");
    }
    void MouseDown(BPoint point) {
        Window()->Quit();
    }
    void KeyDown(const char *bytes,int32 numBytes) {
        Window()->Quit();
    }
};

float PictureView::offsetstring=0.0;
```

```

class AboutWindow: public BWindow {
    public:
        AboutWindow() : BWindow(BRect(100,100,612,356),"About Aurora",
B_MODAL_WINDOW,B_NOT_ZOOMABLE|B_NOT_RESIZABLE|B_ASYNCHRONOUS_CONTROLS) {
            _Picture_View * View = new _Picture_View();
            AddChild(View);
            View->MakeFocus();
        }
};

void ShowAboutBox() {
    AboutWindow * About=new AboutWindow();
    About->Show();
}

```


Filename: AuroraGeneral.h

```
#ifndef H_AuroraGeneral
#define H_AuroraGeneral

#include <List.h>
const long int A_FILE_VERSION = 0;
unsigned short int enum Save_Block_Type {
    A_COMPONENT = 'AC'
};

extern BList Components; //actually in main.cpp
struct ComponentDescriptor;
extern ComponentDescriptor DefaultComponent;

#define DEFAULT_NAME_SIZE 32

#endif
```

Filename: AuroranWorld.h

```
#ifndef H_AWorld
#define H_AWorld

#ifndef H_MaxMinCalc
class MaxMinCalculator;
#endif

#include <InterfaceKit.h>
#include "ILooper.h"
#include "Pulse.h"
#include "ChannelList.h"
#include "ChannelListItem.h"
#include <iostream.h>

enum {
    A_NEW_SOCKET = 'AWNS',
    A_NEW_PLUG = 'AWNP',
    A_ALTER_SOCKET = 'AWAS',
    A_ALTER_PLUG = 'AWAP',
    A_RM_SOCKET = 'AWRS',
    A_RM_PLUG = 'AWRP',
    A_COMP_MOV = 'MOVs'
};

//class AuroranWorld;

struct Args {
//public:
//  Args(AuroranWorld * a,BMessage * b) : Me(a), msg(b) {}
    Pulse * Pls;
    BMessage * msg;
};

class AuroranWorld {
public:
    BListView * CList;
    BList SList;
    BList PList;
    MaxMinCalculator * MaxMinCalc;
    Pulse * Pulser;
    ILooper * ILoop;

    void ReOrder(int OldIdx, int AddAfter);

    void RegisterSocket(ChannelListItem * Socket);
    void RegisterSocketAfter(ChannelListItem * Socket,ChannelListItem *
ExistingSocket);

    void RegisterPlug(ChannelListItem * Plug);
    void RegisterPlugAfter(ChannelListItem * Plug,ChannelListItem *
ExistingPlug);

    void AlterItem(ChannelListItem * Item);
    bool RemoveItem(ChannelListItem * Item);

    void RequestUpdates(BLooper * who);
};
```

```

void StopUpdates(BLooper * who);

void MoveComponent(AuroranComponent * Who,int Where);
private:
void MoveChannels(BList * From,AuroranComponent * Who, int Where,
bool Sockets);
BList NotifyList;
static bool LoopFunc(void * a, void * b);// {cout << "Msg To Pulser:
"<<((Args*)a)->Me->Pulser << endl;((Args*)a)->Me->Pulser->OneShot
((BLooper*)a,((Args*)a)->msg);return false;}
inline void Broadcast(BMessage * msg) {Args a;    a.Pls=this-
>Pulser;a.msg=msg;  NotifyList.DoForEach(&LoopFunc,&a);delete msg;}
};

#endif

```

Filename: AuroranWorld.cpp

```
#include "AuroranWorld.h"
#include "Component.h"
#include <stdio.h>
#include <iostream.h>

void AuroranWorld::RegisterPlugAfter(ChannelListItem * Plug,
ChannelListItem * ExistingPlug) {
    PList.AddItem(Plug,PList.IndexOf(ExistingPlug)+1);
    BMessage * msg = new BMessage(A_NEW_PLUG);
    msg->AddPointer("Plug",Plug);
    //cout << "Pulser: " << Pulser << endl;
    Broadcast(msg);
    // printf("World recognizes a new Plug: %s\n",Socket->Text());
}

void AuroranWorld::RegisterPlug(ChannelListItem * Plug) {
    PList.AddItem(Plug);
    BMessage * msg = new BMessage(A_NEW_PLUG);
    msg->AddPointer("Plug",Plug);
    Broadcast(msg);
    // printf("World recognizes a new Socket: %s\n",Socket->Text());
}

void AuroranWorld::RegisterSocketAfter(ChannelListItem * Socket,
ChannelListItem * ExistingSocket) {
    long int addwhere = SList.IndexOf(ExistingSocket)+1;
    SList.AddItem(Socket,addwhere);
    BMessage * msg = new BMessage(A_NEW_SOCKET);
    msg->AddPointer("Socket",Socket);
    //cout << "Pulser: " << Pulser << endl;
    Broadcast(msg);
    //printf("World recognizes a new Socket: %s Added at location %d,
really %d\n",Socket->Text(),(int)addwhere, (int)SList.IndexOf(Socket));
}

void AuroranWorld::RegisterSocket(ChannelListItem * Socket) {
    SList.AddItem(Socket);
    BMessage * msg = new BMessage(A_NEW_SOCKET);
    msg->AddPointer("Socket",Socket);
    Broadcast(msg);
    //printf("World recognizes a new Socket: %s\n",Socket->Text());
}

void AuroranWorld::AlterItem(ChannelListItem * Item) {
    BMessage * msg;
    if (SList.HasItem(Item)) {
        msg = new BMessage(A_ALTER_SOCKET);
    } else {
        msg = new BMessage(A_ALTER_PLUG);
    }
    msg->AddPointer("Item",Item);
    Broadcast(msg);
}
```

```

void AuroranWorld::RequestUpdates(BLooper * who) {
    NotifyList.AddItem(who);
    cout << "Added " << who << " as a world event listener." << endl;
}

void AuroranWorld::StopUpdates(BLooper * who) {
    NotifyList.RemoveItem(who);
    cout << "Removed " << who << " as a world event listener." << endl;
}

bool AuroranWorld::LoopFunc(void * b, void * a) {
    //cout << "Msg To Pulser: "<<((Args*)a)->Pls << " and a is " << a <<
endl;
    ((Args*)a)->Pls->OneShot((BLooper*)b,new BMessage(*(Args*)a)-
>msg));
    return false;
}

void AuroranWorld::MoveChannels(BList * From,AuroranComponent * Who,
int Where,bool Sockets) {
    BList Temp;
    for (int i = From->CountItems()-1;i>=0;i--) {
        if (((ChannelListItem*)From->ItemAt(i))->GetOwner()==Who) {
            Temp.AddItem((ChannelListItem*)From->RemoveItem(i),0);
        }
    }

    if (Where==0) {
        From->AddList(&Temp,0);
    } else if (CList->CountItems() > Where+1) {
        From->AddList(&Temp,From->IndexOf(Sockets?((AuroranComponent *)
CList->ItemAt(Where+1))->ParentSocket():((AuroranComponent *)CList-
>ItemAt(Where+1))->ParentPlug()));
    } else {
        From->AddList(&Temp);
    }
}

void AuroranWorld::MoveComponent(AuroranComponent * Who,int Where) {
    CList->MoveItem(CList->IndexOf(Who),Where);

    //move sockets...
    MoveChannels(&SList,Who,Where,true);
    MoveChannels(&PList,Who,Where,false);

    BMessage * msg = new BMessage(A_COMP_MOV);
    msg->AddPointer("Component",Who);
    Broadcast(msg);
}

bool DirectLoopFunc(void * b, void * a) {
    ((BLooper*)b)->MessageReceived(new BMessage(*(BMessage*)a));
    return false;
}

bool AuroranWorld::RemoveItem(ChannelListItem * Item) {
    BMessage * msg;
    if (SList.HasItem(Item)) {

```

```

        msg = new BMessage(A_RM_SOCKET);
        SList.RemoveItem(Item);
    } else {
        msg = new BMessage(A_RM_PLUG);
        PList.RemoveItem(Item);
    }
    msg->AddPointer("Item", Item);

    NotifyList.DoForEach(&DirectLoopFunc, msg);
    delete msg;
}

//void AuroranWorld::Broadcast(BMessage * msg) {cout << "Pulser is" <<
this->Pulser<<endl; Args a; cout << "a is " << &a << endl;a.Pls=this-
>Pulser;a.msg=msg;  NotifyList.DoForEach(&LoopFunc, &a);}

```

Filename: BDragListView.h

```
#ifndef H_ComponentList
#define H_ComponentList

#include <Window.h>
#include <ListView.h>
#include <iostream.h>
#include "AuroranWorld.h"

//Make the list re-order-able
class BDragListView : public BListView {
public:
    BDragListView(BRect frame, const char *name, AuroranWorld * AW,
list_view_type type = B_SINGLE_SELECTION_LIST, uint32 resizeMode =
B_FOLLOW_LEFT | B_FOLLOW_TOP, uint32 flags = B_WILL_DRAW |
B_FRAME_EVENTS | B_NAVIGABLE) : BListView(frame,name,type,
resizeMask,flags) {TheWorld = AW;}

    virtual bool InitiateDrag(BPoint point, int32 index, bool
wasSelected);
    virtual void MouseDown(BPoint where);
    void MessageReceived(BMessage *msg);
    virtual void MouseMoved(BPoint pt, uint32 code, const BMessage
*msg);
private:
    BMessage * currentdrag;
    AuroranWorld * TheWorld;
};

#endif
```

Filename: BDragListView.cpp

```
#include "BDragListView.h"
#include "Component.h"
#include <iostream.h>

bool BDragListView::InitiateDrag(BPoint point, int32 index, bool
wasSelected) {
    //if it wasn't selected then we return false
    if(!wasSelected)
        return false;

    //Also make sure they are using the left mouse button
    BMessage* msg = Window()->CurrentMessage();
    int32 button = msg->FindInt32("buttons");
    if (button!=1)
        return false;

    //Get the frame of the item that is selected
    BRect itemFrame = ItemFrame( CurrentSelection() );
    //make a new message
    BMessage *drag = new BMessage('drag');
    currentdrag=drag;
    //Add the frame that we got above to the message
    //drag->AddRect("frame", itemFrame);
    //Find at what spot we were
    drag->AddInt32("index", index);

    AuroranComponent * MyItem = (AuroranComponent *)ItemAt
(index);
    MyItem->LockWait(); //wait until data is correct
    drag->AddInt32("Min",MyItem->MinMoveTo);
    drag->AddInt32("Max",MyItem->MaxMoveTo);
    //
    MyItem->MaxMinLock.Unlock(); //release the lock;

    //send off the drag message and return true to start the
drag
    //DragMessage(drag, itemFrame);
    DragMessage(drag, ((AuroranComponent *) (ItemAt
(CurrentSelection()))->GetBitmap(),B_OP_ALPHA,point-itemFrame.LeftTop
())); //BPoint(73,12));
    return true;
}

void BDragListView::MessageReceived(BMessage *msg) {
    if (msg->what == 'drag') {
        int32 buttons=0;
        BPoint point;
        //BRect frame;
        int32 index;
        int32 ipoint;
        //find the Item's frame that we stored in the message
        earlier
        //msg->FindRect("frame", &frame);
        //Get the index of the item we are dragging that we
        stored earlier
        msg->FindInt32("index", &index);
```



```

        /*Window()->CurrentMessage()->FindInt32("buttons",
&buttons);
        while ( buttons )
        {
            snooze(20 * 1000);
        }*/
        //after all the mouse buttons are released get the
mouse location
        GetMouse(&point, (uint32 *)&buttons, true);
        ipoint=IndexOf(point);

        if(ipoint != index && ItemAt(ipoint) != NULL) {
            AuroranComponent * MyItem;
            MyItem= ((AuroranComponent *) (ItemAt(index)));
            MyItem->LockWait(); //wait until data is correct
            if (ipoint <= MyItem->MaxMoveTo && ipoint >= MyItem-
>MinMoveTo) { //only if it's within range

                //clean up view
                if (ipoint<index && ipoint > 0) {
                    InvalidateItem(ipoint-1);
                }
                if (ipoint > index) {
                    Invalidate();
                }
                //cout << "Moveeee idx (this: "<<this<<"): " <<
index << "   IA:" << MyItem << endl;
                //((AuroranComponent *) (ItemAt(index)))-
>MoveSocketTo((AuroranComponent *)ItemAt(ipoint),ipoint-index>0);
                //MoveItem(index,ipoint);
                TheWorld->MoveComponent((AuroranComponent *)ItemAt
(index),ipoint);

                //BMessage * x;
                //Window()->PostMessage(x = new BMessage('MOVs'));
                //delete x;
            }
        }
        else {
            //msg->PrintToStream();
            BListView::MessageReceived(msg);
        }
    }

void BDragListView::MouseDown(BPoint where) {

    BMessage* msg = Window()->CurrentMessage();
    int32 clicks = msg->FindInt32("clicks");
    int32 button = msg->FindInt32("buttons");
    //cout << "clicked: " << clicks << " btn"<<button<<endl;

    if (button==2) { //got a right-click
        if ((AuroranComponent *) (ItemAt(CurrentSelection()))!=NULL &&
CurrentSelection()==IndexOf(where)) {
            //      BMessage * rclick = new BMessage(A_RIGHT_CLICKED);
            //      rclick->AddPoint("where",ConvertToScreen(where));
            //      ((AuroranComponent *) (ItemAt(CurrentSelection())))-

```

```

>ShowMenu(ConvertToScreen(where));
    return;
}
} else
if (clicks==2 && button==1) { //got a double click
    BMessage * x;
    ((AuroranComponent *) (ItemAt(CurrentSelection())))->PostMessage
(x = new BMessage(A_DBL_CLICKED));
    delete x;
}
BListView::MouseDown(where);
}

void BDragListView::MouseMoved(BPoint pt, uint32 code, const BMessage
*msg) {
    if (msg!=NULL && msg->what == 'drag') {
        int32 index,max,min;
        static int lastidx; //can only happen by one item at a time so
this should be safe

        //what is under the mouse?
        int32 ipoint=IndexOf(pt);
        if (pt.x < 0 || pt.x>176) ipoint=-1; //mouse isn't over
listview

        msg->FindInt32("index", &index);
        msg->FindInt32("Min", &min);
        msg->FindInt32("Max", &max);
        if (max < 0) max = 0x7FFFFFFF;

        if (lastidx!=ipoint) { //position changed
            if (lastidx!=index && lastidx>=min && lastidx<=max) { //
over a different item...
                //InvalidateItem(lastidx);
                //cout << "ReDraw!" << endl;
                if (lastidx < index) {
                    if (lastidx>0)
                        ItemAt(lastidx-1)->DrawItem(this,ItemFrame
(lastidx-1));
                } else {
                    if (lastidx<CountItems()-1) {
                        ItemAt(lastidx+1)->DrawItem(this,ItemFrame
(lastidx+1));
                    } else { //make sure it looks pretty
                        SetDrawingMode(B_OP_COPY);
                        BRect Frame = ItemFrame(lastidx);
                        SetHighColor(ViewColor());
                        BPoint Left = Frame.LeftBottom()+BPoint(0,1);
                        BPoint Right = Frame.RightBottom()+BPoint(0,1);
                        StrokeLine(Left,Right);
                        StrokeLine(Left+BPoint(0,1),Right+BPoint(0,1));
                        StrokeLine(Left+BPoint(0,2),Right+BPoint(0,2));
                        StrokeLine(Left+BPoint(0,3),Right+BPoint(0,3));

                    }
                }

                ItemAt(lastidx)->DrawItem(this,ItemFrame(lastidx));
            }
        }
    }
}

```

```

        bool Draw=false;
        BPoint Left,Right;
        if (ipoint < index && ipoint >= min ) { //we care about the
minnum...
            BRect Frame = ItemFrame(ipoint);
            Left = Frame.LeftTop();
            Right = Frame.RightTop();
            Draw=true;
        } else if (ipoint > index && ipoint <= max) {
            BRect Frame = ItemFrame(ipoint);
            Left = Frame.LeftBottom()+BPoint(0,1);
            Right = Frame.RightBottom()+BPoint(0,1);
            Draw=true;
        }

        if (Draw) {
            SetDrawingMode(B_OP_ALPHA);

            SetHighColor((rgb_color){153,153,153,192});
            StrokeLine(Left-BPoint(0,4),Right-BPoint(0,4));
            SetHighColor((rgb_color){204,204,204,255});
            StrokeLine(Left-BPoint(0,3),Right-BPoint(0,3));
            SetHighColor((rgb_color){255,255,255,255});
            StrokeLine(Left-BPoint(0,2),Right-BPoint(0,2));
            SetHighColor((rgb_color){230,230,230,255});
            StrokeLine(Left-BPoint(0,1),Right-BPoint(0,1));

            SetHighColor((rgb_color){204,204,204,255});
            StrokeLine(Left,Right);
            SetHighColor((rgb_color){166,166,166,255});
            StrokeLine(Left+BPoint(0,1),Right+BPoint(0,1));
            SetHighColor((rgb_color){128,128,128,255});
            StrokeLine(Left+BPoint(0,2),Right+BPoint(0,2));
            SetHighColor((rgb_color){84,84,84,192});
            StrokeLine(Left+BPoint(0,3),Right+BPoint(0,3));

        }

    }

    lastidx=ipoint;
}
BListView::MouseMoved(pt,code,msg);
}

```

Filename: ChannelList.h

```
#ifndef H_CHAN_LIST_VIEW
#define H_CHAN_LIST_VIEW

#include <UIKit.h>

#ifndef H_CH_LI
class ChannelListItem;
#endif

class ChannelList : public BOutlineListView {
public:
    ChannelList(BRect frame, const char * name, uint32 resizeMode =
B_FOLLOW_LEFT | B_FOLLOW_TOP, uint32 flags = B_WILL_DRAW |
B_FRAME_EVENTS | B_NAVIGABLE);
    unsigned int GreyBefore;
    unsigned int GreyAfter;
    ChannelListItem * TestCompatability;
};

#endif
```

Filename: ChannelList.cpp

```
#include "ChannelList.h"

ChannelList::ChannelList(BRect frame, const char * name, uint32
resizeMask = B_FOLLOW_LEFT | B_FOLLOW_TOP, uint32 flags = B_WILL_DRAW |
B_FRAME_EVENTS | B_NAVIGABLE) : BOutlineListView(frame, name,
B_SINGLE_SELECTION_LIST, resizeMask, flags | B_FULL_UPDATE_ON_RESIZE) {

    GreyBefore=0;
    GreyAfter=(unsigned int) (-1);
    TestCompatability=NULL;
}
```

Filename: ChannelListItem.h

```
#ifndef H_CH_LI
#define H_CH_LI

#include <InterfaceKit.h>
#include <Locker.h>
#include "Icons.h"
#include "AuroraGeneral.h"

#ifndef AuroranComponent
class AuroranComponent; //need to declare the existence of this...
#endif

//Data type for passing data between channels
typedef union {
    float f;
    int i;
    void * p;
} CHData;

//for managing enumerations
typedef struct {
    char Name[28];
    CHData Value;
} EnumData;

enum ConnectResults {
    COK = 0x00,
    CINCOMPAT = 0x01,
    CBACKWARDS = 0x02,
    CLOCKED = 0x03
};

short int enum ChannelFlags { //information regarding channel's data type
    Chan_Float = 0x0000,
    Chan_Int = 0x0001,
    Chan_Pointer = 0x0002,
    Chan_DontCare = 0x0003,

    Chan_Stream = 0x0000,
    Chan_Pulse = 0x0010
};

short int enum SocketFlags {

    Recv_Now = 0x0100,
    Recv_Later = 0x0200,
    Recv_Any = 0x0300,

    Recv_Dirty = 0x1000
};

typedef enum ChanStandardTypes {
    StdChan_Intensity = 0x0001,
    StdChan_Hue = 0x0002,
    StdChan_Sat = 0x0003,
```

```

        StdChan_Position = 0x0004,
        StdChan_Speed = 0x0005,
        StdChan_Pulse = 0x0006,
        StdChan_Online = 0x0007
};

struct Enum {
    CHData Value;
    char Name[DEFAULT_NAME_SIZE];
};

class ChannelListItem : public BStringItem {
public:
    ChannelListItem(const char *name, AuroranComponent * owner,
uint32 level=0);
    ~ChannelListItem();
    BString FullName();
    void SetTypeName(const char * Name);
    const char * GetTypeName() {return TypeName;};
    void SetAttributes(CHData Min, CHData Max, unsigned short
flags);
    CHData GetMin() {return CHMin;};
    CHData GetMax() {return CHMax;};
    void SetFlags(unsigned short int NewFlags) {SetAttributes
(CHMin,CHMax,NewFlags);}
    unsigned short GetFlags() {return MyFlags;};
    AuroranComponent * GetOwner() {return owner;}

    bool isSelectable(BView * owner);

    //Hook Stuff - value passing
    virtual int SetValue(CHData Value);
    void SetText(const char * text);
    int SetValue(int Value) {return SetValue((CHData){Value});}
    int SetValue(float Value) {return SetValue((CHData){Value});}
    int SetValue(void * Value) {CHData t;t.p=Value; return SetValue
(t);}

    virtual CHData GetValue();
    //Hook Stuff - attributes
    virtual bool isCompatable(ChannelListItem * who);
    virtual bool isPlug();
    virtual bool isSocket();
    virtual bool isBundle();
    virtual bool isAvailable();

    void DrawItem(BView * owner, BRect itemRect, bool
drawEverything = false);

    //Connecting...
    virtual int Connect(ChannelListItem* who, bool
NoCallBack=false);
    void Disconnect(ChannelListItem * who, bool NoCallBack=false);
    ChannelListItem * PopDisconnect();
    bool isConnected(ChannelListItem* who);
    bool isConnected(AuroranComponent * who);
    bool isConnected();
    ChannelListItem * NextConnection(ChannelListItem * Last); //get
the next connection in the list of connections - end at NULL

```

```

        virtual bool inheritProperties(ChannelListItem * who, bool
ShowReason=false);

        //enumeration stuff
        virtual BList * GetEnums();
        virtual Enum * GetEnum(CHData value);
protected:
    unsigned short int MyFlags;
    //static void SetUpIcons();
    IconSet Icons;
    unsigned int UseageCount;
    unsigned int MaxUseage;
    BLocker ConnList;
    ChannelListItem ** Connections;
    AuroranComponent * owner;
    CHData LatestValue;
private:
    char TypeName[DEFAULT_NAME_SIZE];
    CHData CHMin;
    CHData CHMax;
    bool NeedsRedraw;
};

class BundleListItem:public ChannelListItem {
public:
    BundleListItem(const char *name, AuroranComponent * owner,
uint32 level=0):ChannelListItem(name, owner,level) {};
    bool isBundle() {return true;};
    bool isCompatable(ChannelListItem * who);
    int Connect(ChannelListItem* who, bool NoCallBack=false);
};

#endif

```


Filename: ChannelListItem.cpp

```
#include <String.h>
#include <List.h>
#include <stdlib.h>
#include <stdio.h>
#include "ChannelListItem.h"
#include "ChannelList.h"
#include "Component.h"

ChannelListItem::ChannelListItem(const char * name, AuroranComponent *
owner,uint32 level) : BStringItem(name,level,false) {
    MyFlags=0;
    ChannelListItem::owner=owner;
    UsageCount=0;
    memset(TypeName,0,DEFAULT_NAME_SIZE); //ensure the string is null
    CHMin.f = 0.0;
    CHMax.f = 1.0;
    LatestValue.f=0.0;
    MaxUseage=0;
    Connections=NULL;
}

ChannelListItem::~ChannelListItem() {
    //clear any lingering connections
    while (PopDisconnect()) {}
    //free the memory block holding connections
    if (Connections) free(Connections);
}

void ChannelListItem::SetTypeName(const char * Name) {
    //int size = strlen(Name);
    strncpy(TypeName,Name,DEFAULT_NAME_SIZE);
    TypeName[DEFAULT_NAME_SIZE-1]=0; //just in case Name >= 32 bytes

    if (NeedsRedraw==false) {
        NeedsRedraw=true;
        owner->GetWorld()->AlterItem(this);
    }
}

void ChannelListItem::SetAttributes(CHData Min, CHData Max, unsigned
short flags) {
    CHMin=Min;
    CHMax=Max;
    MyFlags=flags;

    if (NeedsRedraw==false) {
        NeedsRedraw=true;
        owner->GetWorld()->AlterItem(this);
    }
}

bool ChannelListItem::isSelectable(BView * owner) {
    if (((ChannelList *)owner)->FullListIndexOf(this) <= ((ChannelList
*)owner)->GreyBefore ) || (((ChannelList *)owner)->FullListIndexOf
(this) >= ((ChannelList *)owner)->GreyAfter )) {
```

```

        return false;
    } else {
        return true;
    }
}

bool ChannelListItem::isCompatable(ChannelListItem * who) {return
false;}

int ChannelListItem::SetValue(CHData Value) {LatestValue=Value;}

CHData ChannelListItem::GetValue() {return LatestValue;}

bool ChannelListItem::isPlug() {return false;}

bool ChannelListItem::isSocket() {return false;}

bool ChannelListItem::isBundle() {return false;}

bool ChannelListItem::isAvailable() {return false;}

bool ChannelListItem::isConnected(ChannelListItem* who) {
    ConnList.Lock();

    for (unsigned int i = 0; i < UseageCount;i++) {
        if (Connections[i]==who) {
            ConnList.Unlock();
            return true;
        }
    }

    ConnList.Unlock();
    return false;
}

bool ChannelListItem::isConnected(AuroranComponent* who) {
    ConnList.Lock();

    for (unsigned int i = 0; i < UseageCount;i++) {
        if (Connections[i]->GetOwner()==who) {
            ConnList.Unlock();
            return true;
        }
    }

    ConnList.Unlock();
    return false;
}

bool ChannelListItem::isConnected() {
    return UseageCount > 0;
}

void ChannelListItem::DrawItem(BView * owner, BRect itemRect, bool
drawEverything) {

```

```

NeedsRedraw=false;
if (isSelectable(owner)) {
    owner->SetHighColor(0,0,0,255);
} else {
    owner->SetHighColor(212,212,212,255);
}
if (((ChannelList*)owner)->TestCompatability!=NULL) {
    if (isConnected(((ChannelList*)owner)->TestCompatability)) {
        owner->SetHighColor(255,0,0,255);
    } else if (isCompatable(((ChannelList*)owner)-
>TestCompatability)) {
        owner->SetHighColor(0,0,255,255);
    }
}
if (OutlineLevel()==0) {
    owner->SetFont(be_bold_font);
} else {
    owner->SetFont(be_plain_font);
}
BStringItem::DrawItem(owner,itemRect,drawEverything);
}

int ChannelListItem::Connect(ChannelListItem* who, bool NoCallBack) {

    //Don't connect if we are not compatable
    if (isCompatable(who)) {
        //1st try to get remote side to connect
        if (NoCallBack || who->Connect(this,true)==COK) {

            //Make sure nothing bad happens
            if (ConnList.LockWithTimeout(1000)!=B_OK) return CLOCKED;

            //if we are already connected, just return.
            if (isConnected(who)) {
                ConnList.Unlock();
                return COK;
            }
            if (MaxUseage < UseageCount+1) {
                ChannelListItem ** Temp = (ChannelListItem**)malloc
((MaxUseage+2)*2*sizeof(void*));
                if (Connections) {
                    memcpy(Temp,Connections,sizeof(ChannelListItem*)
*UseageCount);
                    free(Connections);
                }
                Connections=Temp;
                MaxUseage=(MaxUseage+2)*2;
            }
            //cout << "A: %d B: %d\n" << (int)Connections << endl;
            Connections[UseageCount]=who;
            UseageCount++;

            ConnList.Unlock();
            return COK;
        } else {
            return CBACKWARDS;
        }
    } else {

```

```

        return CINCOMPAT;
    }
}

void ChannelListItem::Disconnect(ChannelListItem * who, bool
NoCallBack=false) {
    if (!isConnected(who)) return;
    if (ConnList.LockWithTimeout(5000)==B_OK) {
        if (NoCallBack==false) who->Disconnect(this,true);
        bool found=false;
        //Compress array
        for (int i = 0;i<UseageCount;i++) {
            if (found) Connections[i-1]=Connections[i];
            if (Connections[i]==who) {
                found=true;
            }
        }
        UseageCount--;
        ConnList.Unlock();
    }
}

ChannelListItem * ChannelListItem::PopDisconnect() {
    ChannelListItem * who = NextConnection(NULL);
    Disconnect(who);
    return who;
}

ChannelListItem * ChannelListItem::NextConnection(ChannelListItem *
Last) {
    if (UseageCount>0) {
        ConnList.Lock();
        ChannelListItem * rval;
        if (Last==NULL) {rval=Connections[0];ConnList.Unlock();return
rval;}
        for (int i = 1; i<UseageCount;i++) {
            if (Connections[i-1]==Last) {rval=Connections[i];ConnList.
Unlock();return rval;}
        }
        ConnList.Unlock();
    }
    return NULL;
}

bool BundleListItem::isCompatable(ChannelListItem * who) {

    if (who==this) return false;
    ChannelListItem * MySide=this;
    ChannelListItem * TheirSide=who;
    bool SkipOnce=true;
    BList * MyList;
    BList * TheirList;
    if (owner->GetPlugList()->HasItem(this) || owner->ParentPlug()
==this) { //I am a plug
        MyList=owner->GetPlugList();
        TheirList=who->GetOwner()->GetSockets();
    }
}

```

```

    } else {
        MyList=owner->GetSockets();
        TheirList=who->GetOwner()->GetPlugList();
    }
    for (;;) {
        do {
            MySide=(ChannelListItem*)MyList->ItemAt(MyList->IndexOf
(MySide)+1);
            if (MySide==NULL || MySide->OutlineLevel() < this-
>OutlineLevel()) return true;
            printf("Trying From %s\n",MySide->Text());
            } while (MySide->isBundle());

            do {
                if (!SkipOnce) {

                    TheirSide=(ChannelListItem*)TheirList->ItemAt(TheirList-
>IndexOf(TheirSide)+1);
                    if (TheirSide==NULL || TheirSide->OutlineLevel() < who-
>OutlineLevel()) return true;
                }
                SkipOnce=false;
                printf("Trying To %s\n",TheirSide->Text());
                } while (TheirSide->isBundle());

            printf("Testing %s vs %s for Bundle match.\n",MySide->Text(),
TheirSide->Text());

            if (!(MySide->isCompatable(TheirSide))) {
                return false;
            }

        }

    }

}

int BundleListItem::Connect(ChannelListItem * who,bool
NoCallBack=false) {

    if (who==this) return false;
    ChannelListItem * MySide=this;
    ChannelListItem * TheirSide=who;
    bool SkipOnce=true;
    BList * MyList;
    BList * TheirList;
    if (owner->GetPlugList()->HasItem(this) || owner->ParentPlug()
==this) { //I am a plug
        MyList=owner->GetPlugList();
        TheirList=who->GetOwner()->GetSockets();
    } else {
        MyList=owner->GetSockets();
        TheirList=who->GetOwner()->GetPlugList();
    }
    for (;;) {
        do {
            MySide=(ChannelListItem*)MyList->ItemAt(MyList->IndexOf
(MySide)+1);
            if (MySide==NULL || MySide->OutlineLevel() < this-
>OutlineLevel()) return true;

```

```

        printf("Trying From %s\n",MySide->Text());
    } while (MySide->isBundle());

    do {
        if (!SkipOnce) {
            TheirSide=(ChannelListItem*)TheirList->ItemAt(TheirList-
>IndexOf(TheirSide)+1);
            if (TheirSide==NULL || TheirSide->OutlineLevel() < who-
>OutlineLevel()) return true;
        }
        SkipOnce=false;
        printf("Trying To %s\n",TheirSide->Text());
    } while (TheirSide->isBundle());

    printf("Testing %s vs %s for Bundle match.\n",MySide->Text(),
TheirSide->Text());

    if (!(MySide->isCompatable(TheirSide))) {
        return true;
    }
    MySide->Connect(TheirSide);
}
}

void ChannelListItem::SetText(const char * text) {
    BStringItem::SetText(text);

    //Handle updating the item it the main display
    if (NeedsRedraw==false) {
        NeedsRedraw=true;
        owner->GetWorld()->AlterItem(this);
    }
}

bool ChannelListItem::inheritProperties(ChannelListItem * who, bool
ShowReason) {

    cout << "Inheriting Properties...." << endl;

    ChannelListItem* NConn=NULL;
    while ((NConn=who->NextConnection(NConn))) {
        if (NConn->isCompatable(who)==false) {
            if (ShowReason) {
                BAlert("Update Channel",BString("Attempt to update this
channel failed, because it would break compatability with ").Append
(NConn->FullName()).Append(". To make the desired change, you must
first disconnect from that channel.")->String(),"Bummer",NULL,NULL,
B_WIDTH_AS_USUAL,B_STOP_ALERT).Go();
            }
            return false;
        }
    }

    cout << "    Type Name = " << who->GetTypeName() << endl;

    SetTypeName(who->GetTypeName());
}

```

```

//Handle updating the item it the main display
if (NeedsRedraw==false) {
    NeedsRedraw=true;
    owner->GetWorld()->AlterItem(this);
}
}

BString ChannelListItem::FullName() {
    return BString(owner->CustomName).Append("::").Append(Text());
}

BList * ChannelListItem::GetEnums() {return NULL;}
Enum * ChannelListItem::GetEnum(CHData value) {return NULL;}

```

Filename: ChannelTools.h

```
#ifndef CHTOOL_H
#define CHTOOL_H

#include <View.h>
#include <List.h>
#include <OutlineListView.h>
#include "ChannelListItem.h"
#include "Component.h"

enum {
    CHT_SOCKETS = 1,
    CHT_PLUGS = 2,
    CHT_BUNDLES = 8,
    CHT_ADVANCED = 16,
    CHT_ENUMS = 32,
    CHT_RATE = 64
};

void SaveChannelList(BPositionIO * object, AuroranComponent * MyOwner,
BList * CHList, ChannelListItem * Parent, BList * SupList, int SupSize);
void LoadChannelList(BPositionIO * object, AuroranComponent * MyOwner ,
BList * CHList, ChannelListItem * Parent, BList * SupList, int SupSize);

class ChannelTools : public BView {
public:
    ChannelTools(AuroranComponent * MyOwner, int CHTFlags, BList *
CHList, ChannelListItem * Parent, BMessage * AddMessage, BMessage *
AlterMessage, BMessage * DelMessage, BMessage * SelMessage, BRect frame,
char * name, uint32 resize, uint32 flags);
    ~ChannelTools();
    void InterfaceMessage(BMessage * message);
    ChannelListItem * CurrentSelection() {return CurrentItem;}
    void SetChanType(ChanStandardTypes ctype) {LoadDefaultType
(ctype);}
private:
    void LoadToUI();
    void LoadFromUI();
    void LoadDefaultType( ChanStandardTypes ctype);
    ChannelListItem * CurrentItem;
    ChannelListItem * TestItem;
    AuroranComponent * owner;
    int MyFlags;
    ChannelListItem * parent;
    unsigned int OutlineOffset;
    BList * OwnerList;
    BOutlineListView * MyList;

    BTextControl * txtName;
    BButton * btnUpdate;
    BButton * btnDel;
    BButton * btnUnder;

    BMessage * AddMessage;
    BMessage * AlterMessage;
    BMessage * DelMessage;
    BMessage * SelMessage;
```



```
BMessenger MyMessenger;

BTextControl * typ_Name;
BRadioButton * typ_Data;
BRadioButton * typ_Data_Alt;
BRadioButton * typ_Rate;
BTextControl * typ_Min;
BTextControl * typ_Max;
BListView * typ_Enums;

};

#endif
```

Filename: ChannelTools.cpp

```
#include "ChannelTools.h"
#include <InterfaceKit.h>
#include <String.h>
#include <iostream.h>
#include <DataIO.h>
#include <malloc.h>

/*
It is OK to work on two sublists within an outline,
but if there is any potential overlap, bad things will happen.
*/

ChannelTools::ChannelTools(AuroranComponent * MyOwner, int CHTFlags,
BList * CHList, ChannelListItem * Parent, BMessage * AddMessage,
BMessage * AlterMessage, BMessage * DelMessage, BMessage * SelMessage,
BRect frame, char * name, uint32 resize, uint32 flags) :
    BView(frame, name, resize, flags | B_WILL_DRAW), AddMessage
    (AddMessage), AlterMessage(AlterMessage), DelMessage(DelMessage),
    SelMessage(SelMessage), MyMessenger(BMessenger(MyOwner)) {

    owner=MyOwner;
    MyFlags=CHTFlags;
    parent=Parent;
    OwnerList=CHList;
    CurrentItem=NULL;
    if (MyFlags & CHT_PLUGS) { //initialize base item for testing
        TestItem = new PlugListItem("", owner, 0);
    } else {
        TestItem = new SocketListItem("", owner, owner, new BMessage
('SNew'), 0, 0);
    }
    LoadDefaultType(StdChan_Intensity);

    BMessage * MyMsg = new BMessage('CHTM');
    MyMsg->AddPointer("Owner", this);
    BMessage * ThisMsg;

    //cout << "Borm with flags: "<<flags<<endl;

    //Set up local list of channels
    MyList = new BOutlineListView(BRect(3, 3, 120, frame.Height()-6),
BString(name).Append(".List").String(), B_SINGLE_SELECTION_LIST,
B_FOLLOW_TOP_BOTTOM);
    ThisMsg=new BMessage(*MyMsg);
    ThisMsg->AddInt32("Action", 'LSel');
    MyList->SetSelectionMessage(ThisMsg);
    MyList->SetTarget(owner->GetWorld()->ILoop);

    //add all relevant items to the list...
    int i;
    if (parent==NULL) {
        OutlineOffset=0;
        i=0;
    } else {
```

```

        OutlineOffset=parent->OutlineLevel();
        if (parent->isBundle()) {
            OutlineOffset++;
        }
        i = OwnerList->IndexOf(parent)+1;
        if (i<0) i=0;
    }
    for (; i < OwnerList->CountItems() && ((BListItem*)OwnerList->ItemAt
(i))->OutlineLevel()>=OutlineOffset; i++) {
        MyList->AddItem(new BStringItem(((BStringItem*)OwnerList-
>ItemAt(i))->Text(), ((BStringItem*)OwnerList->ItemAt(i))->OutlineLevel
()-OutlineOffset));
    }

    cout << name << "'s Outline Offset: " << OutlineOffset << endl;

    this->AddChild(new BScrollView(BString(name).Append(".ListScroll").
String(),MyList,B_FOLLOW_TOP_BOTTOM,0,false,true));

    BControl * x;

    //input box for channel name & label
    this->AddChild(new BStringView(BRect(144,3,266,16),"lbl_InName",
BString(name).Append(" Name:").String()));
    this->AddChild(txtName = new BTextControl(BRect(144,18,266,33),
"txt_In", NULL,"",NULL));

    int top=0;

    if (MyFlags & CHT_BUNDLES) { //if this set of channels supports
bundles, we need to give them extra options

        ThisMsg=new BMessage(*MyMsg);
        ThisMsg->AddInt32("Action",'AddA');
        this->AddChild(x=new BButton(BRect(144,70,266,90),"btn_AddIn",
BString("Add ").Append(name).Append(" After").String(),ThisMsg));
        x->SetTarget(owner->GetWorld()->ILoop);

        ThisMsg=new BMessage(*MyMsg);
        ThisMsg->AddInt32("Action",'AddU');
        this->AddChild(x=btnUnder=new BButton(BRect(144,100,266,120),
"btn_AddUn",BString("Add ").Append(name).Append(" Under").String(),
ThisMsg));
        x->SetTarget(owner->GetWorld()->ILoop);
        x->SetEnabled(false);
        top=30;

    } else {

        ThisMsg=new BMessage(*MyMsg);
        ThisMsg->AddInt32("Action",'AddA');
        this->AddChild(x=new BButton(BRect(144,70,266,90),"btn_AddIn",
BString("Add ").Append(name).String(),ThisMsg));
        x->SetTarget(owner->GetWorld()->ILoop);
    }

    ThisMsg=new BMessage(*MyMsg);
    ThisMsg->AddInt32("Action",'Upda');
    this->AddChild(btnUpdate=new BButton(BRect(144,40,266,60),

```

```

"btn_UpdIn",BString("Update ").Append(name).String(),ThisMsg));
    btnUpdate->SetEnabled(false);
    btnUpdate->SetTarget(owner->GetWorld()->ILoop);

    ThisMsg=new BMessage(*MyMsg);
    ThisMsg->AddInt32("Action",'Dele');
    this->AddChild(btnDel=new BButton(BRect(144,160,266,180),
"btn_DelIn",BString("Remove ").Append(name).String(),ThisMsg));
    btnDel->SetTarget(owner->GetWorld()->ILoop);
    btnDel->SetEnabled(false);

    //Set up advanced channel properties if requested.
    if (MyFlags & CHT_ADVANCED) {
        BBox * Box=new BBox(BRect(266+8,5,Frame().Width()-6,Frame().
Height()-6),"RequireProps");

        BPopUpMenu * BPM = new BPopUpMenu("Data Type");
        BMenuItem * BMI;
        //BPM->SetFontSize(12);

        ThisMsg=new BMessage(*MyMsg);
        ThisMsg->AddInt32("Custom",'TypS');
        ThisMsg->AddInt8("Type",1);
        BPM->AddItem(BMI = new BMenuItem("Intensity (Default)",
ThisMsg,0,0));
        BMI->SetTarget(owner->GetWorld()->ILoop);
        BMI->SetMarked(true);

        ThisMsg=new BMessage(*MyMsg);
        ThisMsg->AddInt32("Custom",'TypS');
        ThisMsg->AddInt8("Type",2);
        BPM->AddItem(BMI = new BMenuItem("Pulse",ThisMsg,0,0));
        BMI->SetTarget(owner->GetWorld()->ILoop);

        ThisMsg=new BMessage(*MyMsg);
        ThisMsg->AddInt32("Custom",'TypS');
        ThisMsg->AddInt8("Type",3);
        BPM->AddItem(BMI = new BMenuItem("Hue",ThisMsg,0,0));
        BMI->SetTarget(owner->GetWorld()->ILoop);

        ThisMsg=new BMessage(*MyMsg);
        ThisMsg->AddInt32("Custom",'TypS');
        ThisMsg->AddInt8("Type",4);
        BPM->AddItem(BMI = new BMenuItem("Saturation",ThisMsg,0,0));
        BMI->SetTarget(owner->GetWorld()->ILoop);

        ThisMsg=new BMessage(*MyMsg);
        ThisMsg->AddInt32("Custom",'TypS');
        ThisMsg->AddInt8("Type",5);
        BPM->AddItem(BMI = new BMenuItem("Position",ThisMsg,0,0));
        BMI->SetTarget(owner->GetWorld()->ILoop);

        ThisMsg=new BMessage(*MyMsg);
        ThisMsg->AddInt32("Custom",'TypS');
        ThisMsg->AddInt8("Type",6);
        BPM->AddItem(BMI = new BMenuItem("Custom...",ThisMsg,0,0));
        BMI->SetTarget(owner->GetWorld()->ILoop);

```

```

        BMenuField * MenuOpt;
        Box->SetLabel(MenuOpt = new BMenuField(BRect(10,0,150,20),
"ChanTypes", NULL, BPM,true));

        //input box for channel name & label
        int x1,x2,h;
        x1 = 6;x2 = ((int)Box->Frame().Width())/2-12;
        h = ((int)Box->Frame().Height()-30)/4;
        Box->AddChild(new BStringView(BRect(x1,25,x2,39),
"lbl_TypName", "Name of Type: (optional)"));
        Box->AddChild(typ_Name=new BTextControl(BRect(x1-2,40,x2,55),
"txt_TypName", NULL,"",NULL));

        Box->AddChild(new BStringView(BRect(x1,25+h*1,x2,39+h*1),
"lbl_Format", "Numeric Values:"));
        BView * Group = new BView(BRect(x1-2,40+h*1,x2,55+h*1),
"vw_Format",B_FOLLOW_ALL,0);
        Group->SetViewColor(kViewGray);
        Group->AddChild(x=typ_Data = new BRadioButton(BRect(0,0,(x2-x1)
*4/7+3,15),"rad_Deci", "Analog",NULL));
        x->SetValue(B_CONTROL_ON);
        Group->AddChild(typ_Data_Alt = new BRadioButton(BRect((x2-x1)
*4/7+3,0,(x2-x1),15),"rad_Int", "Digital",NULL));
        Box->AddChild(Group);

        if (MyFlags & CHT_RATE) {
            Box->AddChild(new BStringView(BRect(x1,25+h*2,x2,39+h*2),
"lbl_TypName", "Data Rate:"));
            Group = new BView(BRect(x1-2,40+h*2,x2,55+h*2),"vw_Rate",
B_FOLLOW_ALL,0);
            Group->SetViewColor(kViewGray);
            Group->AddChild(x=typ_Rate = new BRadioButton(BRect(0,0,
(x2-x1)*4/7+3,15),"rad_Cont", "Continuous",NULL));
            if (MyFlags & CHT_SOCKETS) x->SetValue(B_CONTROL_ON);
            Group->AddChild(x=new BRadioButton(BRect((x2-x1)*4/7+3,0,
(x2-x1),15),"rad_Puls", "Pulses",NULL));
            if (!MyFlags & CHT_SOCKETS) x->SetValue(B_CONTROL_ON);
            Box->AddChild(Group);
        } else {
            typ_Rate=NULL;
        }

        Box->AddChild(new BStringView(BRect(x1,25+h*3,x2,39+h*3),
"lbl_TypName", "Range: (min->max)"));
        Box->AddChild(typ_Min = new BTextControl(BRect(x1-2,40+h*3,(x2-
x1)/2,55+h*3),"txt_TypName", NULL,"",NULL));
        Box->AddChild(typ_Max = new BTextControl(BRect((x2-x1)/
2+8,40+h*3,x2,55+h*3),"txt_TypName", NULL,"",NULL));

        if (MyFlags & CHT_ENUMS) {
            typ_Enums = new BListView(BRect(x2+7,27,((int)Box->Frame().
Width())-22,55+h*3-20),BString(name).Append(".EnumList").String(),
B_SINGLE_SELECTION_LIST,B_FOLLOW_TOP_BOTTOM);
            Box->AddChild(new BScrollView(BString(name).Append(".
EnumListScroll").String(),typ_Enums,B_FOLLOW_TOP_BOTTOM,0,false,true));
            Box->AddChild(x=new BButton(BRect(x2+7,55+h*3+5-20,(((int)
Box->Frame().Width())-7)-(x2+7))/2-2+x2+7,55+h*3+2),"btn_AdEn", "Add
Enum...",NULL));

```

```

        Box->AddChild(x=new BButton(BRect((((int)Box->Frame().
Width()-7)-(x2+7))/2+2+x2+7,55+h*3+5-20,((int)Box->Frame().Width()-
7,55+h*3+2),"btn_RmEn","Remove",NULL));

    }

    LoadToUI();

    //if not a plug or socket, tool to monitor current value

    AddChild(Box);

}

delete MyMsg;
}

ChannelTools::~ChannelTools() {

    //empty out our list, freeing all resources
    void * Item;
    while ((Item=MyList->RemoveItem((int32)0))) {
        delete Item;
    }
    if (AddMessage) delete AddMessage;
    if (AlterMessage) delete AlterMessage;
    if (DelMessage) delete DelMessage;
    if (SelMessage) delete SelMessage;
    if (TestItem) delete TestItem;
}

void ChannelTools::LoadDefaultType( ChanStandardTypes ctype) {

    switch (ctype) {
        case StdChan_Intensity:
            TestItem->SetTypeName("Intensity");
            TestItem->SetAttributes((CHData){0.0},(CHData){1.0},
Chan_Float|Chan_Stream);
            break;
    }

}

void ChannelTools::LoadFromUI() {
    TestItem->SetText(txtName->Text());
    if (MyFlags & CHT_ADVANCED) {
        TestItem->SetTypeName(typ_Name->Text());
    }
    // TestItem->SetAttributes((CHData){0.0},(CHData){1.0},Chan_Float|
Chan_Stream);
}

void ChannelTools::LoadToUI() {

    //txtName->SetText(TestItem->Text());
    if (MyFlags & CHT_ADVANCED) {

```

```

        typ_Name->SetText (TestItem->GetTypeName());
    }
}

void ChannelTools::InterfaceMessage(BMessage * message) {
    int itemp, itemp2;
    char stemp[33];

    cout << "Interface Message recieved!" << endl;
    int32 LocalMsg;
    if (message->FindInt32("Action",&LocalMsg)!=B_OK) return;
    switch (LocalMsg) {
        case 'Upda':
            if (CurrentItem) { //just incase...make sure there IS a
current item...
                LoadFromUI();
                if (AlterMessage!=NULL) {
                    BMessage reply;
                    BMessage * MyMsg = new BMessage(*AlterMessage);
                    MyMsg->AddPointer("Item",CurrentItem);
                    MyMsg->AddPointer("TestItem",TestItem);
                    MyMessenger.SendMessage(M
yMsg,&reply,5000,1000);
                    delete MyMsg;
                    if (reply.what!=B_NO_REPLY && reply.what!='OpOK') {
                        cout << "Update failed."<<endl;

                        return;
                    }
                }

                CurrentItem->SetText(txtName->Text());
                (((ChannelListItem*)MyList->ItemAt(MyList-
>CurrentSelection()))->inheritProperties(TestItem,true);
                CurrentItem->inheritProperties(TestItem,true);
                if (Window()->LockWithTimeout(5000)==B_OK) {
                    MyList->InvalidateItem(MyList->CurrentSelection());
                    Window()->Unlock();
                }

            }
            break;
        case 'LSel':
            //cout << "Channel Selected"<<endl;
            CurrentItem=NULL;
            if (MyList->CurrentSelection()>=0) {
                if (OwnerList->HasItem(parent)) {
                    CurrentItem = (ChannelListItem*)OwnerList->ItemAt
(MyList->CurrentSelection()+OwnerList->IndexOf(parent)+1);
                } else {
                    CurrentItem = (ChannelListItem*)OwnerList->ItemAt
(MyList->CurrentSelection());
                }
                TestItem->inheritProperties(CurrentItem);
                if (Window()->LockWithTimeout(5000)==B_OK) {
                    btnUpdate->SetEnabled(true);
                    btnDel->SetEnabled(true);
                }
            }
    }
}

```

```

        txtName->SetText(CurrentItem->Text());
        LoadToUI();
        Window()->Unlock();
    }
} else {
    if (Window()->LockWithTimeout(5000)==B_OK) {
        btnUpdate->SetEnabled(false);
        btnDel->SetEnabled(false);
        Window()->Unlock();
    }
}
if (SelMessage!=NULL) {
    //cout <<"Sending message...CurrentItem:
"<<CurrentItem<<endl;
    BMessage * MyMsg = new BMessage(*SelMessage);
    MyMsg->AddPointer("Item",CurrentItem);
    MyMessenger.SendMessage(MyMsg);
    delete MyMsg;
}
break;

case 'AddA':
    cout << "Adding After with flags:" <<MyFlags<<endl;
    itemp=MyList->CurrentSelection()+1;
    itemp2=0;

    //To make life easier for the user, we auto-increment
numbers at the end of channel names.
    strncpy(stemp,txtName->Text(),32); //transform the name if
need be
    stemp[32]=0;
    //cout << "Channel Name: "<< stemp << "("<<strlen(stemp)<<"
bytes)." << endl;
    if (stemp[0]==0) {stemp[0]='0';stemp[1]='0';stemp[2]='0';
stemp[3]=0;}
    //cout << "Channel Name2: "<< stemp << "("<<strlen(stemp)<<"
bytes)." << endl;
    if (stemp[strlen(stemp)-1]>='0' && stemp[strlen(stemp)-1]
<='9') {
        //channel ends in a number, so we want to increase that
number to give it a unique name.
        int state = 1; //state 1 = need to check if number needs
to be increased
        //      2 = number needs to be increased
        //      0 = number is OK
        while (state) {
            if (state==2) { //increase counter if need be.
                int digit=strlen(stemp)-1; //start w/ last
digit.
                while (state==2) {
                    if (stemp[digit]=='9') { //digit is a nine,
so roll over & work on the previous digit
                        stemp[digit]='0';
                        digit--;
                    } else if (stemp[digit]>=0 && stemp[digit]
<'9') {
                        stemp[digit]++;
                        state=1; //we think things are OK now
                    } else { //this "digit" must be a letter

```



```

        int l;
        if ((l=strlen(stemp))>30) { //if the
string is long, give up
            state=0;
        } else { //add a digit on the end.
            stemp[digit+1]='1';
            stemp[l]='0';
            stemp[l+1]=0;
            state = 1;
        }
    }
}
int i;
for (i = MyList->CountItems()-1;i>=0 && state==1;
i--) {
    //cout << stemp<<" vs " << ((BStringItem*)
MyList->ItemAt(i))->Text() << " == " << strcmp(stemp,((BStringItem*)
MyList->ItemAt(i))->Text(),32) << endl;
    if (strcmp(stemp,((BStringItem*)MyList->ItemAt
(i))->Text(),32)==0) { //this name already is in use
        state=2;
    }
}
if (state==1) state=0; //no match was found, so we
may proceed.
}

}

//cout << "Channel Name3: "<< stemp << "("<<strlen(stemp)<<"
bytes)." << endl;

if (itemp>0) {
    itemp2=((BListItem*)MyList->ItemAt(MyList-
>CurrentSelection()))->OutlineLevel();
}
ChannelListItem *NewItem=NULL;
if (MyFlags & CHT_PLUGS) {
    NewItem = new PlugListItem(stemp,owner,
itemp2+OutlineOffset);
} else {
    NewItem = new SocketListItem(stemp,owner,owner,new
BMessage('SNew'),0,itemp2+OutlineOffset);
}
LoadFromUI();
NewItem->inheritProperties(TestItem);

//Add the new item the owner's master list first
if (OwnerList->HasItem(parent)) {
    OwnerList->AddItem(NewItem,itemp+OwnerList->IndexOf
(parent)+1);
} else {
    OwnerList->AddItem(NewItem,itemp);
}

if (AddMessage!=NULL) {
    BMessage reply;
    BMessage * MyMsg = new BMessage(*AddMessage);

```

```

        MyMsg->AddPointer("Item",NewItem);
        MyMessenger.SendMessage(MyMsg,&reply,5000,1000);
        delete MyMsg;
        if (reply.what!=B_NO_REPLY && reply.what!='OpOK') {
            cout << "Add after failed."<<endl;
            OwnerList->RemoveItem(NewItem);
            delete NewItem;
            return;
        }
    }

    MyList->Window()->Lock();
    MyList->AddItem(new BStringItem(stemp,itemp2),itemp);
    MyList->Window()->Unlock();

    if (MyFlags & CHT_PLUGS) {
        ChannelListItem * Prev;
        if (parent==NULL) {
            Prev = owner->ParentPlug();
        } else {
            Prev=parent;
        }
        owner->GetWorld()->RegisterPlugAfter(NewItem,
(ChannelListItem*)owner->GetWorld()->PList.ItemAt(owner->GetWorld()-
>PList.IndexOf(Prev)+itemp));

        } else if (MyFlags & CHT_SOCKETS) {
            ChannelListItem * Prev;
            if (parent==NULL) {
                Prev = owner->ParentSocket();
            } else {
                Prev=parent;
            }
            owner->GetWorld()->RegisterSocketAfter(NewItem,
(ChannelListItem*)owner->GetWorld()->SList.ItemAt(owner->GetWorld()-
>SList.IndexOf(Prev)+itemp));
        }

        if (Window()->LockWithTimeout(5000)==B_OK) {
            MyList->Select(itemp);
            MyList->ScrollToSelection();
            Window()->Unlock();
        }

        break;
    }
}

```

```

void SaveChannelList(BPositionIO * object, AuroranComponent * MyOwner,
BList * CHList, ChannelListItem * parent, BList * SupList, int SupSize)
{
    unsigned short int usitemp;
    int itemp;

    int CounterPtr,NextCompPtr;
    int PtrDiff;

```

```

unsigned short int CHItemCount=0;

//magic number
usitemp='CL';
object->Write(&usitemp, 2);

//fill space
CounterPtr=object->Position();
object->Write(&usitemp, 2);

int OutlineOffset;
//add all relevant items to the list...
int i;
if (parent==NULL) {
    OutlineOffset=0;
    i=0;
} else {
    OutlineOffset=parent->OutlineLevel();
    if (parent->isBundle()) {
        OutlineOffset++;
    }
    i = CHList->IndexOf(parent)+1;
    if (i<0) i=0;
}
for (; i < CHList->CountItems() && ((BListItem*)CHList->ItemAt(i))-
>OutlineLevel()>=OutlineOffset; i++) {

    ChannelListItem * TheItem;
    TheItem = (ChannelListItem*)CHList->ItemAt(i);
    NextCompPtr=object->Position();
    object->Write(&PtrDiff,4);

    //1st store outline level
    usitemp=(unsigned int) (TheItem->OutlineLevel()-OutlineOffset);
    object->Write(&usitemp,2);

    //Now the name of the object
    usitemp=strlen(TheItem->Text());
    cout << "Writting Channel: "<< TheItem->Text()<<endl;
    object->Write(&usitemp,2);
    object->Write(TheItem->Text(),usitemp);

    //Now the name of the object's type
    usitemp=strlen(TheItem->GetTypeName());
    object->Write(&usitemp,2);
    object->Write(TheItem->GetTypeName(),usitemp);

    //flags
    usitemp=TheItem->GetFlags();
    object->Write(&usitemp,2);

    //CHT specific flags
    usitemp=(TheItem->isPlug()?CHT_PLUGS:0) | (MyOwner->GetSockets
()->HasItem(TheItem)?CHT_SOCKETS:0) | (TheItem->isBundle()?
CHT_BUNDLES:0);
    object->Write(&usitemp,2);

    //min/max
    itemp=TheItem->GetMin().i;

```

```

        object->Write(&itemp,4);
        itemp=TheItem->GetMax().i;
        object->Write(&itemp,4);

        //Enumeration data
        usitemp=0;
        object->Write(&usitemp,2); //number of enumerations

        //extra data
        if (SupSize>0 && SupList!=NULL) {
            cout << "Writing "<<SupSize<<" bytes Extra Data for: " <<
TheItem->Text() << endl;
            object->Write(SupList->ItemAt(CHItemCount),SupSize);
        }

        CHItemCount++;

        //go back and write size of this data block
        PtrDiff=object->Position()-(NextCompPtr+4);
        object->Seek(NextCompPtr,SEEK_SET);
        object->Write(&PtrDiff,4);
        object->Seek(object->Position()+PtrDiff,SEEK_SET);
    }

    //go back and announce number of items
    NextCompPtr=object->Position();
    object->Seek(CounterPtr,SEEK_SET);
    object->Write(&CHItemCount,2);
    object->Seek(NextCompPtr,SEEK_SET);
}

void LoadChannelList(BPositionIO * object, AuroranComponent * MyOwner ,
BList * CHList, ChannelListItem * Parent, BList * SupList, int SupSize)
{
    unsigned short int usitemp;
    unsigned short int usitemp2;
    int itemp,itemp2;

    char StringBufferA[DEFAULT_NAME_SIZE*2];
    char StringBufferB[DEFAULT_NAME_SIZE*2];

    int NextCompPtr;
    int PtrDiff;
    unsigned short int CHItemCount;

    //magic number test
    object->Read(&usitemp, 2);
    if (usitemp!='CL') return;

    //fill space
    object->Read(&CHItemCount, 2);

    cout << "ItemCount: "<< CHItemCount;

```

```

int OutlineOffset;
//add all relevant items to the list...
int i;
if (Parent==NULL) {
    OutlineOffset=0;
    i=0;
} else {
    OutlineOffset=Parent->OutlineLevel();
    if (Parent->isBundle()) {
        OutlineOffset++;
    }
    i = CHList->IndexOf(Parent)+1;
    if (i<0) i=0;
}
NextCompPtr=object->Position();
for (int j=0; j < CHItemCount ; j++) {

    ChannelListItem * TheItem;

    object->Seek(NextCompPtr, SEEK_SET);

    object->Read(&PtrDiff,4);
    NextCompPtr+=(PtrDiff+4);

    //1st store outline level
    object->Read(&usitemp,2);
    unsigned int OutlineLevel=usitemp+OutlineOffset;

    //Now the name of the object
    object->Read(&usitemp,2);
    if (usitemp>128)
return;
    object->Read(StringBufferA,usitemp);
    StringBufferA[usitemp]=0;
    //cout << "Reading Channel("<<j<<"): "<< StringBufferA<<endl;

    //Now the name of the object's type
    object->Read(&usitemp,2);
    if (usitemp>128) return;
    object->Read(StringBufferB,usitemp);
    StringBufferB[usitemp]=0;

    //flags
    object->Read(&usitemp,2);

    //CHT specific flags
    object->Read(&usitemp2,2);
    if ((usitemp2 & CHT_PLUGS) && !(usitemp2 & CHT_BUNDLES)) { //
item is a plug
        //cout << " --Loading as Plug."<<endl;
        ChannelListItem * Prev;
        if (Parent==NULL) {
            Prev = MyOwner->ParentPlug();
        } else {
            Prev=Parent;
        }
        TheItem = new PlugListItem(StringBufferA,MyOwner,

```

```

OutlineLevel);
        CHList->AddItem(TheItem);
        MyOwner->GetWorld()->RegisterPlugAfter(TheItem,
(ChannelListItem*)MyOwner->GetWorld()->PList.ItemAt(MyOwner->GetWorld
()->PList.IndexOf(Prev)+j));

        } else if (!(usitemp2 & CHT_BUNDLES)) { //item is a socket
        //cout << " --Loading as Socket."<<endl;
        ChannelListItem * Prev;
        if (Parent==NULL) {
            Prev = MyOwner->ParentSocket();
        } else {
            Prev=Parent;
        }
        TheItem = new SocketListItem(StringBufferA,MyOwner,
MyOwner,new BMessage('SNew'),usitemp,OutlineLevel);
        CHList->AddItem(TheItem);
        if (usitemp2 & CHT_SOCKETS) MyOwner->GetWorld()-
>RegisterSocketAfter(TheItem, (ChannelListItem*)MyOwner->GetWorld()-
>SList.ItemAt(MyOwner->GetWorld()->SList.IndexOf(Prev)+j));
        } else { //item is a bundle
            TheItem=NULL;
        }

        if (TheItem) {

            //min/max
            object->Read(&itemp,4);
            object->Read(&itemp2,4);

            //set flags with data we got
            TheItem->SetAttributes((CHData){itemp},(CHData){itemp2},
usitemp);

            //set the name of the datatype
            TheItem->SetTypeName(StringBufferB);

            //Enumeration data
            object->Read(&usitemp,2); //number of enumerations
            if (usitemp!=0) break; //can't handle this yet

            //extra data
            if (SupSize>0 && SupList!=NULL) {
                cout << "Loading "<<SupSize<<" bytes Extra Data for: "
<< TheItem->Text() << endl;
                void * xptra = malloc(SupSize);
                object->Read(xptra,SupSize);
                SupList->AddItem(xptra);
            }

        }

    }

}

```


Filename: Component.h

```
#ifndef H_Component
#define H_Component

#include <UIKit.h>
#include <View.h>
#include <TranslationDefs.h>
#include <String.h>
#include "SocketList.h"
#include "PlugList.h"
#include <OS.h>
#include "Pulse.h"

#include "AuroranWorld.h"

const rgb_color kViewGray = { 216, 216, 216, 255};

enum {
    A_ADD_COMP = 'AddC',
    A_PULSE = 'Plse',
    A_APPLY_CHANGES = 'AplC', //Associated with the BArchive returned
    from a GetView BView...
    A_DBL_CLICKED = 'Db1C', //When item is double clicked in list...
    A_RIGHT_CLICKED = 'RtCl', //Item has been right-clicked
    A_DELETE_COMP = 'DelC',
    A_DELETE_TO = 'DelT',
    A_DELETE_FROM = 'DelF',
    A_DUPLICATE_COMP = 'DupC',
    A_INVALIDATE_PLUGS = 'InvP'
};

enum {
    ACC_INPUT=1,
    ACC_OUTPUT=2,
    ACC_CONTROL=4,
    ACC_PASSIVE=8,
    ACC_GROUP=16,
    ACC_ABSTRACT=32,
    ACC_UNDEF=128
};

enum {
    AC_APPLY_TOP=16,
    AC_APPLY_BOTTOM=0,
    AC_APPLY_HIDDEN=128
};

//Hack to facilitate upgrade
#define AuroranParameters AuroranWorld

class AuroranComponent;

uint16 enum DescType {
    A_NORMAL_DESC = 0
};
```



```

uint16 const A_DESC_VER=0;
struct ComponentDescriptor {
    uint16 DescriptorVersion;
    uint16 DescriptorType;
    ComponentDescriptor * NextDescriptor;
    const char * Name;
    uint32 Creator;
    uint16 What;
    uint32 Version;
    uint16 CompClass;
    AuroranComponent* (*MakeComponent)(ComponentDescriptor * Who,
    AuroranParameters * Params);
    char Reserved[64];
};

struct NameTester;

class AuroranComponent : public BLooper, public BListItem {
public:
    AuroranComponent(AuroranWorld * Params, const char *
DefaultName=NULL);
    virtual void DrawItem(BView * owner, BRect frame, bool
complete);
    virtual void Update(BView * lv, const BFont * font);
    virtual BView * GetView(BRect frame);
    virtual void MessageReceived(BMessage * message);
    virtual ComponentDescriptor * GetDesc();
    BBitmap * GetBitmap();
    ~AuroranComponent();
    void PreDelete();
    char CustomName[33];
    //bool MoveSocketTo(AuroranComponent * owner, bool AddAfter);
    void FillPlugList() {PlugList->AddList(&MyPlugs);}
    BList * GetPlugList() {return &MyPlugs;}
    BList * GetSockets() {return &MySockets;}
    status_t Flatten(BPositionIO * object);
    status_t UnFlatten(BPositionIO * object, uint32 version);
    bool MaxMinLocked;
    BLocker MaxMinLock;
    void LockWait() {while (MaxMinLocked && TheWorld->Pulser-
>isBlacklisted(this)==false) snooze(1000);}
    uint32 LockOpBits;
    unsigned long MinMoveTo;
    unsigned long MaxMoveTo;
    void ShowMenu(BPoint where);
    virtual status_t Save(BPositionIO * object);
    virtual status_t Load(BPositionIO * object, uint32 version);
    virtual void InterfaceMessage(BMessage * message);
    virtual void OutOfSight();
protected:
    void AutoSetColors();
    virtual void DrawIt(BView * owner, BRect frame, bool
Complete=true, bool Selected=true);
    virtual void DrawBevelBox(BView * owner, BRect frame, bool
Inset=false);
    virtual void ProcessView(BString & ViewName, BMessage * msg);
    BBitmap * MyIcon;
    rgb_color HighlightBG;

```

```

        rgb_color NormalBG;
        virtual const char * MyName(); // {return "Auroran Component";}
        virtual const char * DefaultCustom() {return "Component";}
        BListView * MyList;
        bool ReName(BString * NewName); //to a safe change of the
CustomName
        BList * MasterSocketList;
        ChannellListItem * MySublist;
        BList * PlugList;
        SocketListItem * RegisterSocket(const char * name, char *
TypeName, CHData min, CHData max, short int flags=0, BMessage *
message=NULL, BHandler * handler=NULL);
        SocketListItem * RegisterSocket(const char * name, BMessage *
message, short int flags=0, BHandler * handler=NULL) {
            return RegisterSocket(name, "", (CHData){0.0}, (CHData){1.0},
flags, message, handler);
        }
        void RemoveSocket(int SocketNum);
        bool RenameSocket(int SocketNum, const char * newname);
        PlugListItem * RegisterPlug(const char * name, const char *
TypeName, CHData min, CHData max, unsigned short int flags);
        PlugListItem * RegisterPlug(const char * name,
ChanStandardTypes ctype);
        SocketListItem * RegisterSocket(const char * name,
ChanStandardTypes ctype, BMessage * message = NULL, BHandler *
handler=NULL);
        void RenamePlug(int32 index, const char * name);
        void DeletePlug(int32 index);
        BList MyPlugs;
        BPopupMenu MyMenu;
        AuroranWorld * TheWorld;
        BList MySockets;
    private:
        BBitmap * DefaultIcon;
        static bool ApproveName(BListItem * TestItem,void *
TheNameTester);
        BMenuItem * MenuDeleteIncoming;
        BMenuItem * MenuDeleteOutgoing;
        BMenuItem * MenuDelete;
    public:
        ChannellListItem * ParentSocket() {return MySublist;}
        ChannellListItem * LastSocket() {return ((NULL==MySockets.
LastItem())?MySublist:(SocketListItem *)MySockets.LastItem());}
        ChannellListItem * ParentPlug() {return MyParentPlug;}
    private:
        ChannellListItem * MyParentPlug;
        int SBundleLevel;
        int PBundleLevel;
    public:
        void BeginSBundle(const char * BundleName);
        void EndSBundle();
        void BeginPBundle(const char * BundleName);
        void EndPBundle();
    protected:
        unsigned char ApplyLoc;
    public:
        AuroranWorld * GetWorld(){return TheWorld;}
};

```

```
struct NameTester {  
    AuroranComponent * Exclude;  
    BString * Name;  
    bool IsOK;  
};  
  
#endif
```

Filename: Component.cpp

```
#include "Component.h"
#include <TranslationKit.h>
#include <TranslatorFormats.h>
#include <stdio.h>
#include <iostream.h>
#include <StringView.h>
#include <String.h>
#include <InterfaceDefs.h>
#include <InterfaceKit.h>
#include "AuroraGeneral.h"
#include "MaxMinCalculator.h"

#define World (*TheWorld)

const char * AuroranComponent::MyName() {return "Auroran Component";}

AuroranComponent::AuroranComponent(AuroranParameters * MyWorld, const
char * DefaultName) : BLooper(DefaultName,
(B_REAL_TIME_DISPLAY_PRIORITY*2+B_URGENT_DISPLAY_PRIORITY)/3), MyMenu
("Component Menu",false,false) {
    AuroranComponent::TheWorld=MyWorld;
    MasterSocketList=&TheWorld->SList;
    DefaultIcon=MyIcon=BTranslationUtils::GetBitmap(B_PNG_FORMAT, 2001);
    MyList=TheWorld->CList;
    PlugList=&TheWorld->PList;
    TheWorld->Pulser->Revive(this); //make sure we can be spoken to

    ApplyLoc=AC_APPLY_BOTTOM; //default apply button location;

    //set full range of dragging motion
    MinMoveTo=0;
    MaxMoveTo=-1;
    MaxMinLocked=false;
    LockOpBits=0; //no pending operations...

    //set bundle base
    SBundleLevel=1;
    PBundleLevel=1;

    //cout << "this: "<<this<< " MSL: " << MasterSocketList << "IA: "<<
((AuroranComponent *) (MyList->ItemAt(0))) << endl;

    //RegisterSocket("Hello",new BMessage('HELO'));
    //MyPlugs.AddItem(new PlugListItem(CustomName,Send_Trigger|
Recv_Later));

    //set up menu
    BMenuItem * MenuItem;
    BMenu * ParentMenu;
    BMessage * ItemMsg;

    MyMenu.AddItem(MenuItem = new BMenuItem("Stealth Mode",new BMessage
('hide'),0,0));
    MenuItem->SetTarget(this);
```

```

//duplication
MyMenu.AddItem(ParentMenu = new BMenu("Duplicate"));
    ItemMsg = new BMessage(A_ADD_COMP);
    ItemMsg->AddPointer("owner",this);
    ParentMenu->AddItem(MenuItem = new BMenuItem(BString("Create
new ").Append(DefaultName).Append(" here").String(),ItemMsg,0,0));
    MenuItem->SetTarget(TheWorld->CList->Window());
    ItemMsg = new BMessage(A_DUPLICATE_COMP);
    ItemMsg->AddPointer("owner",this);
    ParentMenu->AddItem(MenuItem = new BMenuItem("Duplicate",
ItemMsg,0,0));
    MenuItem->SetTarget(TheWorld->CList->Window());
    ItemMsg = new BMessage(A_DUPLICATE_COMP);
    ItemMsg->AddPointer("owner",this);
    ItemMsg->AddBool("Connect",true);
    ParentMenu->AddItem(MenuItem = new BMenuItem("Duplicate +
Connect",ItemMsg,0,0));
    MenuItem->SetTarget(TheWorld->CList->Window());

//deletion
MyMenu.AddSeparatorItem();
MyMenu.AddItem(ParentMenu = new BMenu("Remove"));
    ItemMsg = new BMessage(A_DELETE_COMP);
    ItemMsg->AddPointer("owner",this);
    ParentMenu->AddItem(MenuDelete = new BMenuItem("Delete",
ItemMsg,0,0));
    MenuDelete->SetTarget(TheWorld->CList->Window());
    ItemMsg = new BMessage(A_DELETE_TO);
    ItemMsg->AddPointer("owner",this);
    ParentMenu->AddItem(MenuDeleteIncoming = new BMenuItem("Clear
Incoming Connections",ItemMsg,0,0));
    MenuDeleteIncoming->SetTarget(TheWorld->CList->Window());
    ItemMsg = new BMessage(A_DELETE_FROM);
    ItemMsg->AddPointer("owner",this);
    ParentMenu->AddItem(MenuDeleteOutgoing = new BMenuItem("Clear
Outgoing Connections",ItemMsg,0,0));
    MenuDeleteOutgoing->SetTarget(TheWorld->CList->Window());

//Start up...
if (DefaultName==NULL) {
    rename_thread(Run(),DefaultCustom());
    (BString(DefaultCustom()).Append("000")).CopyInto
(CustomName,0,32); //No custom name set...
} else {
    Run();
    (BString(DefaultName).Append("000")).CopyInto(CustomName,0,32);
//No custom name set...
}

MySublist=new BundleListItem(CustomName,this,0);
MyParentPlug=new BundleListItem(CustomName,this,0);
//MySockets.AddItem(MySublist);
TheWorld->RegisterSocket(MySublist);
TheWorld->RegisterPlug(MyParentPlug);

```

```

BString NewName=CustomName;
while (ReName(&NewName)==false) {
    int idx=NewName.CountChars();
    idx--;
    while (NewName[idx]=='9') {
        NewName[idx]='0';
        idx--;
    }
    NewName[idx]++;
}

    HighlightBG= (rgb_color){180,180,180,128};
    NormalBG= (rgb_color){128,128,128,128};

}

void AuroranComponent::AutoSetColors() {

    //define my color...
    printf("My Class: %d\n",GetDesc()->CompClass);
    switch (this->GetDesc()->CompClass) {
        case (ACC_INPUT):
            HighlightBG=(rgb_color){80,240,120,128};
            NormalBG= (rgb_color){60,200,100,128};
            break;
        case (ACC_OUTPUT):
            HighlightBG=(rgb_color){250,110,120,128};
            NormalBG= (rgb_color){210,90,100,128};
            break;
        case (ACC_PASSIVE):
            HighlightBG=(rgb_color){240,240,120,128};
            NormalBG= (rgb_color){200,200,100,128};
            break;
        case (ACC_CONTROL):
            HighlightBG= (rgb_color){80,140,255,128};
            NormalBG= (rgb_color){45,100,200,128};
            break;
        case (ACC_ABSTRACT):
            HighlightBG= (rgb_color){255,140,255,128};
            NormalBG= (rgb_color){200,100,200,128};
            break;
        default:
            HighlightBG= (rgb_color){180,180,180,128};
            NormalBG= (rgb_color){128,128,128,128};

    }

}

void AuroranComponent::PreDelete() {

    //tell the world to forget about us

    TheWorld->RemoveItem(MySublist);

    for ( int32 i = 0; (MySockets.ItemAt(i)); i++ ) {
        TheWorld->RemoveItem((ChannelListItem *)MySockets.ItemAt(i));
    }
}

```

```

TheWorld->RemoveItem(MyParentPlug);

for ( int32 i = 0; (MyPlugs.ItemAt(i)); i++ ) {
    TheWorld->RemoveItem((ChannelListItem *)MyPlugs.ItemAt(i));
}

}

AuroranComponent::~AuroranComponent() {

    printf("bye bye\n");
    World.Pulser->Blacklist(this); //tell other threads we are dead
    World.Pulser->DeleteClient(this);
    World.Pulser->DeleteClient(World.ILoop);
    World.MaxMinCalc->StopCalc(this); //stop trying to calculate max min
data for this
    delete DefaultIcon;

    void *anItem;
    for ( int32 i = MyPlugs.CountItems(); i>=0; i-- ) {
        delete MyPlugs.ItemAt(i);
    }
    MyPlugs.MakeEmpty();

    for ( int32 i = 0; (anItem = MySockets.ItemAt(i)); i++ ) {
        delete anItem;
    }

    MySockets.MakeEmpty();
    delete MySublist;
    delete MyParentPlug;

}

bool AuroranComponent::ReName(BString * NewName) {
    NameTester TestName;
    TestName.IsOK=true;
    TestName.Exclude=this;
    TestName.Name=NewName;
    MyList->DoForEach(&ApproveName, (void *) (&TestName));
    if (TestName.IsOK==false) {
        return false;
    } else {
        NewName->CopyInto(CustomName,0,33);
        CustomName[33]=0; //Enforce 32 character limit on object's
name.
        rename_thread(this->Thread(),CustomName);
        MySublist->SetText(CustomName);
        MyParentPlug->SetText(CustomName);
        MenuDelete->SetLabel(BString("Delete ").Append(CustomName).
String());
        return true;
    }
}

BBitmap * AuroranComponent::GetBitmap() { //Make bitmap on the fly for
use in dragging

```

```

    BBitmap * m_bitmap;           //      bitmap to drag
    BView * m_bitmapView;         //      for drawing into bitmap

    //set up object for drawing...
    BRect r(0,0,176,38);
    m_bitmap = new BBitmap(r, B_RGB32, true);
    m_bitmapView = new BView(r, "_bmv", B_FOLLOW_NONE, B_WILL_DRAW);
    m_bitmap->Lock();
    m_bitmap->AddChild(m_bitmapView);

    DrawIt(m_bitmapView,r);
    DrawBevelBox(m_bitmapView,r);
    m_bitmapView->Sync();
    m_bitmap->Unlock();

    return m_bitmap;
}

//Subclassing the BListItem "DrawItem"
void AuroranComponent::DrawItem(BView * owner, BRect frame, bool
complete) {
    if (Height() <= 3) {
        owner->SetLowColor(NormalBG);
        owner->FillRect(frame,B_SOLID_LOW);
    } else {
        DrawIt(owner,frame,complete,IsSelected());
        DrawBevelBox(owner,frame,IsSelected());
    }
}

//Draw a bevel box (frame) in View (owner), Inset or not (default)
void AuroranComponent::DrawBevelBox(BView * owner, BRect frame, bool
Inset) {

    owner->SetDrawingMode(B_OP_BLEND);
    if (Inset) {
        owner->SetHighColor(0,0,0);
    } else {
        owner->SetHighColor(255,255,255);
    }
    owner->StrokeLine(BPoint(frame.left,frame.top),BPoint(frame.
left,frame.bottom));
    owner->StrokeLine(BPoint(frame.left,frame.top),BPoint(frame.
right,frame.top));
    if (Inset) {
        owner->SetHighColor(255,255,255);
    } else {
        owner->SetHighColor(0,0,0);
    }
    owner->StrokeLine(BPoint(frame.left,frame.bottom),BPoint(frame.
right,frame.bottom));
    owner->StrokeLine(BPoint(frame.right,frame.top),BPoint(frame.
right,frame.bottom));
}

void AuroranComponent::DrawIt(BView * owner, BRect frame, bool
Complete, bool Selected) {

```



```

//Color background
owner->SetDrawingMode(B_OP_COPY);
if (Selected) {
    owner->SetLowColor(HighlightBG);
} else {
    owner->SetLowColor(NormalBG);
}
owner->FillRect(frame,B_SOLID_LOW);

//Add Text
owner->SetDrawingMode(B_OP_ALPHA);
owner->SetHighColor((rgb_color){0,0,0,255});

font_height TextHeight;
owner->SetFont(be_bold_font);
owner->SetFontSize(13.5);
owner->GetFontHeight(&TextHeight);
owner->MovePenTo(frame.left+38, frame.top+3+TextHeight.
leading+TextHeight.ascent);
owner->DrawString(MyName());
owner->DrawString(":");
owner->MovePenTo(frame.left+38, frame.top+3+2*(TextHeight.
leading+TextHeight.ascent)+TextHeight.descent);
owner->SetFont(be_plain_font);
owner->SetFontSize(13.5);
owner->DrawString(CustomName);

//Add Image
//owner->SetDrawingMode(B_OP_ALPHA);
owner->DrawBitmap(MyIcon,frame.LeftTop()+BPoint(3,3));
}

void AuroranComponent::Update(BView * lv, const BFont * font) {
    BListItem::Update(lv, font);
    SetHeight(37.0);
}

BView * AuroranComponent::GetView(BRect frame) {
    BView * ThisView = new BView(frame,"ComponentView",B_FOLLOW_LEFT|
B_FOLLOW_TOP,B_WILL_DRAW);

    //Componet class
    ThisView->SetFont(be_bold_font);
    ThisView->SetFontSize(18);
    float NameWidth;
    NameWidth=ThisView->StringWidth((BString(MyName()).Append(": ")).
String());

    BStringView * lbl =new BStringView(BRect(0,2,NameWidth,20),
"lbl_ThisComp", (BString(MyName()).Append(": ")).String());
    lbl->SetFont(be_bold_font);
    lbl->SetFontSize(18);
    ThisView->AddChild(lbl);

    //Component's Unique name
    BTextView * NewName = new BTextView(BRect(NameWidth,-3,frame.Width
(),20),"txt_CustomName",BRect(0,0,frame.Width()-NameWidth,24),
B_FOLLOW_RIGHT|B_FOLLOW_TOP,B_WILL_DRAW);

```

```

        NewName->SetMaxBytes(31);
        BFont
* BigFont = new BFont(be_plain_font);
        BigFont->SetSize(18);
        NewName->SetFontAndColor((int32)0,(int32)0,BigFont);
        NewName->SetViewColor(kViewGray);
        NewName->SetWordWrap(false);
        NewName->MakeResizable(false);
        NewName->SetText(CustomName);
        NewName->DisallowChar(B_ENTER);
        NewName->DisallowChar(B_TAB);
        delete BigFont;
        ThisView->AddChild(NewName);

//Apply button
        BButton * NewButton;
        if (ApplyLoc==AC_APPLY_TOP) {
            NewButton = new BButton(BRect(frame.Width()-70,8,frame.Width()-
3,28),"btn_Apply","Apply", new BMessage(A_APPLY_CHANGES));
            NewButton->MakeDefault(true);
            ThisView->AddChild(NewButton);
        } else if (ApplyLoc==AC_APPLY_BOTTOM) {
            NewButton = new BButton(BRect(frame.Width()-70,frame.Height()-
28,frame.Width()-3,frame.Height()-8),"btn_Apply","Apply", new BMessage
(A_APPLY_CHANGES));
            NewButton->MakeDefault(true);
            ThisView->AddChild(NewButton);
        }

        return ThisView;
    }

void AuroranComponent::InterfaceMessage(BMessage *message) {
    switch (message->what) {
        case A_APPLY_CHANGES: {
            printf("Apply Changes!\n");
            BMessage * Views = new BMessage();
            int32 idx=0;
            while (message->FindMessage("_views",idx++,Views)==B_OK) {
                //Views->PrintToStream();
                BString ViewName;
                if (Views->FindString("_name",&ViewName)==B_OK) {
                    ProcessView(ViewName,Views);
                }
                //delete Views;
                //Views=new BMessage();
            }
            delete Views;
            break;
        }/*
        case A_INVALIDATE_PLUGS: {
            if (World.PList->Window()->LockWithTimeout(500)==B_OK) {
                World.PList->Invalidate();
                World.PList->Window()->Unlock();
            } else {
                cout << "LOCKED!!! While tring to update Plug list" <<
endl;

```

```

        World.Pulser->OneShot(World.ILoop, new BMessage
(A_INVALIDATE_PLUGS));
    }
    break;
}*/
}
BLooper::MessageReceived(message);
}

void AuroranComponent::MessageReceived(BMessage *message) {
    switch (message->what) {
        case 'hide':
            cout <<"Going into stealth mode..."<<endl;
            if (Height() > 3) {
                SetHeight(1.0);
            } else {
                SetHeight(24);
            }
            //World.CList->Invalidate();
            break;
        case A_DBL_CLICKED: {
            cout << "Double Clicked " << CustomName << endl;
            break;
        }
        default:
            BLooper::MessageReceived(message);
    }
}

void AuroranComponent::ShowMenu(BPoint where) {
    cout << "Right Clicked" << endl;
    //compute what needs to be greyed...
    MenuDelete->SetEnabled(false);
    if (MinMoveTo==0) { //no incoming connections
        MenuDeleteIncoming->SetEnabled(false);
    } else {
        MenuDeleteIncoming->SetEnabled(true);
    }
    if ((int32)MaxMoveTo<0) { //no outgoing connections
        MenuDeleteOutgoing->SetEnabled(false);
        if ((int32)MaxMoveTo<0) { //safe to delete
            MenuDelete->SetEnabled(true);
        }
    } else {
        MenuDeleteOutgoing->SetEnabled(true);
    }
    MyMenu.Go(where,true,false,false);
    //break;
}

void AuroranComponent::ProcessView(BString & ViewName, BMessage * msg) {
    printf("Processing %s\n",ViewName.String());
    if (ViewName=="txt_CustomName") {
        BString TheName;

        if (msg->FindString("_text",&TheName)==B_OK) {
            printf("Found Name: %s\n",TheName.String());
            if (ReName(&TheName)==false) {

```

```

        (new BAlert("Invalid Name",BString("The name you have
chosen, ").Append(TheName).Append(", is already in use elsewhere. All
component names must be unique.")).String(),"Oops",NULL,NULL,
B_WIDTH_AS_USUAL,B_OFFSET_SPACING,B_WARNING_ALERT))->Go();
    }
}

bool AuroranComponent::ApproveName(BListItem * TestItem,void *
TheNameTester) {
    if (((NameTester *)TheNameTester)->Exclude!=TestItem) {
        if (BString(((AuroranComponent *)TestItem)->CustomName)==*
((NameTester *)TheNameTester)->Name) {
            ((NameTester *)TheNameTester)->IsOK=false;
            return true;
        }
    }
    return false;
}

void AuroranComponent::BeginSBundle(const char * name) {
    ChannelListItem * NewSocket = new BundleListItem(name,this,
SBundleLevel++);
    World.RegisterSocketAfter(NewSocket,MySockets.IsEmpty()?MySublist:
LastSocket());
    MySockets.AddItem(NewSocket);
}

void AuroranComponent::EndSBundle() {
    SBundleLevel--;
    if (SBundleLevel < 1) SBundleLevel=1; //protect against bad stuff...
}

void AuroranComponent::BeginPBundle(const char * name) {
    ChannelListItem * NewPlug = new BundleListItem(name,this,
PBundleLevel++);
    World.RegisterPlugAfter(NewPlug,MyPlugs.IsEmpty()?MyParentPlug:
(ChannelListItem*)MyPlugs.LastItem());
    MyPlugs.AddItem(NewPlug);
}

void AuroranComponent::EndPBundle() {
    PBundleLevel--;
    if (PBundleLevel < 1) PBundleLevel=1; //protect against bad stuff...
}

//SocketListItem * AuroranComponent::RegisterSocket(const char * name,
BMessage * message, short int flags, BHandler * handler) {
SocketListItem * AuroranComponent::RegisterSocket(const char * name,
char * TypeName, CHData min, CHData max, short int flags, BMessage *
message, BHandler * handler=NULL) {

    if (handler==NULL) {handler=this;}
    SocketListItem * NewSocket = new SocketListItem(name,this,handler,
message,flags,SBundleLevel);
    NewSocket->SetTypeName(TypeName);
}

```

```

        NewSocket->SetAttributes(min,max,flags);

        World.RegisterSocketAfter(NewSocket,MySockets.IsEmpty()?MySublist:
LastSocket());

        MySockets.AddItem(NewSocket);
        return NewSocket;
    }

void AuroranComponent::RemoveSocket(int SocketNum) {
    ChannelListItem* Who=(ChannelListItem*)MySockets.ItemAt(SocketNum);
    if (!Who) return;

    World.MaxMinCalc->Lock();

    ChannelListItem * Dest = NULL;
    while ((Dest=Who->PopDisconnect())) {
        TheWorld->MaxMinCalc->DoCalc(Dest->GetOwner(),AC_CALC_MAX); //
this needs its maximum recalculated
    }
    TheWorld->MaxMinCalc->CalcMin(this);
    TheWorld->RemoveItem(Who);
    delete MySockets.RemoveItem(SocketNum);

    World.MaxMinCalc->Unlock();

    // MasterSocketList->RemoveItem(Who);
    // MasterSocketList->Invalidate();
}

bool AuroranComponent::RenameSocket(int SocketNum, const char *
newname) {

    SocketListItem * ThisSocket;
    if ((ThisSocket=(SocketListItem*)MySockets.ItemAt(SocketNum))) {
        ThisSocket->SetText(newname);
/*
        if (World.PList->Window()->LockWithTimeout(200)==B_OK) { //we
need to lock the view
            MasterSocketList->Invalidate();
            World.PList->Window()->Unlock();
        }*/
        return true;
    } else {
        return false;
    }
}

//
*****
****
struct AddParms {
    ChannelList * TheList;
    ChannelListItem * TheParent;
};

bool rmFromSList(void * ListItem, void * List) {
    ((ChannelList *)List)->RemoveItem((ChannelListItem *)ListItem);
    return false;
}

```

```

}

bool addToSList(void * ListItem, void * Parms) {
    ((AddParms *)Parms)->TheList->AddUnder((ChannelListItem *)ListItem,
    ((AddParms *)Parms)->TheParent);
    return false;
}

//
*****

PlugListItem * AuroranComponent::RegisterPlug(const char * name, const
char * TypeName, CHData min, CHData max, unsigned short int flags) {
    PlugListItem * NewPlug = new PlugListItem(name, this, PBundleLevel);
    NewPlug->SetAttributes(min,max,flags);
    NewPlug->SetTypeName(TypeName);
    //if ((AuroranComponent *)World.CList->ItemAt(World.CList-
>CurrentSelection())==this) {
        TheWorld->RegisterPlugAfter(NewPlug, (ChannelListItem*) (MyPlugs.
IsEmpty())?MyParentPlug:MyPlugs.LastItem());
        MyPlugs.AddItem(NewPlug);
    //}
    return NewPlug;
}

PlugListItem * AuroranComponent::RegisterPlug(const char * name,
ChanStandardTypes ctype) {

    switch (ctype) {
        case StdChan_Intensity:
            return RegisterPlug(name,"Intensity", (CHData){0.0}, (CHData)
{1.0},Chan_Float|Chan_Stream);
        case StdChan_Hue:
            return RegisterPlug(name,"Hue", (CHData){0.0}, (CHData){1.0},
Chan_Float|Chan_Stream);
        case StdChan_Sat:
            return RegisterPlug(name,"Saturation", (CHData){0.0},
(CHData){1.0},Chan_Float|Chan_Stream);
        case
StdChan_Position:
            return RegisterPlug(name,"Position", (CHData){0.0}, (CHData)
{1.0},Chan_Float|Chan_Stream);
        case StdChan_Speed:
            return RegisterPlug(name,"Speed", (CHData){0.0}, (CHData)
{1.0},Chan_Float|Chan_Stream);
        case StdChan_Pulse:
            return RegisterPlug(name,"Pulse", (CHData){0.0}, (CHData)
{0.0},Chan_Float|Chan_Pulse);
        case StdChan_Online:
            return RegisterPlug(name,"Online", (CHData){0.0}, (CHData)
{1.0},Chan_Float|Chan_Pulse);
    }
}

SocketListItem * AuroranComponent::RegisterSocket(const char * name,
ChanStandardTypes ctype, BMessage * msg, BHandler * hand) {

```

```

        switch (ctype) {
            case StdChan_Intensity:
                return RegisterSocket(name, "Intensity", (CHData){0.0},
(CHData){1.0}, Chan_Float|Chan_Stream, msg, hand);
            case StdChan_Hue:
                return RegisterSocket(name, "Hue", (CHData){0.0}, (CHData)
{1.0}, Chan_Float|Chan_Stream, msg, hand);
            case StdChan_Sat:
                return RegisterSocket(name, "Saturation", (CHData){0.0},
(CHData){1.0}, Chan_Float|Chan_Stream, msg, hand);
            case StdChan_Position:
                return RegisterSocket(name, "Position", (CHData){0.0},
(CHData){1.0}, Chan_Float|Chan_Stream, msg, hand);
            case StdChan_Speed:
                return RegisterSocket(name, "Speed", (CHData){0.0}, (CHData)
{1.0}, Chan_Float|Chan_Stream, msg, hand);
            case StdChan_Pulse:
                return RegisterSocket(name, "Pulse", (CHData){-10.0}, (CHData)
{10.0}, Chan_DontCare|Chan_Pulse, msg, hand);
            case StdChan_Online: {
                SocketListItem * x = RegisterSocket(name, "Online", (CHData)
{0.0}, (CHData){1.0}, Chan_Float|Chan_Stream, msg, hand);
                x->AddEnum((CHData){0.0}, "Off");
                x->AddEnum((CHData){0.50}, "On");
                x->AddEnum((CHData){1.0}, "Reset");
                return x;
            }
        }
    }
}

```

```

void AuroranComponent::RenamePlug(int32 index, const char * name) {

    ((PlugListItem *)MyPlugs.ItemAt(index))->SetText(name);
    /* //((PlugListItem *)MyPlugs.ItemAt(index))->SetFlags(flags);
    if ((AuroranComponent *)World.CList->ItemAt(World.CList-
>CurrentSelection())==this) {
        World.ILoop->PostMessage(A_INVALIDATE_PLUGS);
        if (World.PList->Window()->LockWithTimeout(1000)==B_OK) {
            World.PList->InvalidateItem(index);
            World.PList->Window()->Unlock();
        } else {
        }
    }
    */
}

```

```

void AuroranComponent::DeletePlug(int32 index) {
    ChannelListItem* Who=(ChannelListItem*)MyPlugs.ItemAt(index);
    if (!Who) return;

    World.MaxMinCalc->Lock();

    //clear connections
    ChannelListItem * Dest;// = ((PlugListItem *)MyPlugs.ItemAt(index))-
>Destination;

```

```

        while ((Dest=((PlugListItem *)MyPlugs.ItemAt(index))->PopDisconnect
        ())) {
            World.MaxMinCalc->CalcMin(Dest->GetOwner());
        }

        //and delete
        TheWorld->RemoveItem(Who);
        delete MyPlugs.RemoveItem(index);
        World.MaxMinCalc->CalcMax(this);

        World.MaxMinCalc->Unlock();
    }

    //Get the default file descriptor - this really should be overwritten
    by subclasses
    ComponentDescriptor * AuroranComponent::GetDesc() {return
    &DefaultComponent;}

    //Internal flatten
    status_t AuroranComponent::Flatten(BPositionIO * object) {

        if (object->Write(CustomName,32)!=32) return B_ERROR;

        long ConnectionsLocation;
        long PointerLocation;
        PointerLocation=object->Position();
        object->Write(&ConnectionsLocation,4);

        if (Save(object)!=B_OK) return B_ERROR;

        //Store where the connection will be, and jump to that location + 4
        (to give space to store the number of connections)
        ConnectionsLocation = object->Position();
        if (object->Seek(PointerLocation,SEEK_SET)!=PointerLocation) return
        B_ERROR;
        PointerLocation=ConnectionsLocation-(PointerLocation+4);
        object->Write(&PointerLocation,4);
        object->Seek(ConnectionsLocation+5,SEEK_SET);

        //store connections
        uint32 ConnectionCount=0;
        for (long int i = MyPlugs.CountItems()-1;i>=0;i--) {
            if (MyPlugs.ItemAt(i)!=NULL && ((PlugListItem *)MyPlugs.ItemAt
            (i))->isConnected()) { //this plug has a destination...
                ChannelListItem * Destination = NULL;
                while ((Destination = ((PlugListItem *)MyPlugs.ItemAt(i))-
                >NextConnection(Destination))) {
                    ConnectionCount++;
                    uint32 DestLocation = MyList->CountItems()-MyList-
                    >IndexOf(Destination->GetOwner());
                    /*
                        for (int j = MyList->CountItems()-1;i>=0;i--) {
                            DestLocation++;
                            if ((AuroranComponent *)MyList->Index(j)==Destination-
                            >GetOwner()) break;
                        }*/
                    uint32 SocketInset = Destination->GetOwner()->GetSockets
                    ()->IndexOf(Destination);
                    object->Write(&i,4); //store origin

```



```

        object->Write(&DestLocation,4); //store dest's owner
        object->Write(&SocketInset,4); //store dest
    }

}

PointerLocation=object->Position();
object->Seek(ConnectionsLocation,SEEK_SET);
ConnectionsLocation=0;
object->Write(&ConnectionsLocation,1);//start with a NULL (for
expansion purposes)
object->Write(&ConnectionCount,4);//write out the number of
connections
object->Seek(PointerLocation,SEEK_SET);

cout << "Plattened OK!" << endl;
return B_OK;
}

//Internal unflatten
status_t AuroranComponent::UnFlatten(BPositionIO * object, uint32
version) {
    char DataCache[32];

    cout << "Unflattening..." << endl;
    //get the object's name
    if (object->Read(DataCache,32)!=32) return B_ERROR;
    BString newname(DataCache,32);
    ReName(&newname);

    cout << "Rename to " << newname.String() << endl;

    //store connections location
    if (object->Read(DataCache,4)!=4) return B_ERROR;

    *((uint32 *)DataCache)+=object->Position()+1;

    //load object specspecific data
    if (Load(object,version)!=B_OK) return B_ERROR;

    //Jump to connections location
    if (object->Seek(*((uint32 *)DataCache),SEEK_SET)!=(* (uint32 *)
DataCache)) return B_ERROR;

    uint32 ConnectionCount;
    if (object->Read(&ConnectionCount,4)!=4) return B_ERROR;
    cout << "Found " << ConnectionCount << " connections." << endl;

    while ((ConnectionCount-->0) {
        if (object->Read(DataCache,12)!=12) return B_ERROR;
        AuroranComponent * DestOwner = (AuroranComponent *)MyList-
>ItemAt(MyList->CountItems()-((uint32 *)DataCache)[1]);
        if (DestOwner==NULL) {
            //return B_ERROR;
        } else {
            PlugListItem * ThePlug = (PlugListItem *)MyPlugs.ItemAt
(((uint32 *)DataCache)[0]);
            if (ThePlug==NULL) {return B_ERROR;}
            ThePlug->Connect((SocketListItem *)DestOwner->GetSockets()-

```

```

>ItemAt(((uint32 *)DataCache)[2]));
    }

    //restore connections
    cout << "UnPlattened OK!" << endl;
    return B_OK;
}

void AuroranComponent::OutOfSight() {}

//External flatten to be subclassed
status_t AuroranComponent::Save(BPositionIO * object) {return B_OK;}

//External unflattten to be subclassed
status_t AuroranComponent::Load(BPositionIO * object, uint32 version)
{return B_OK;}

```

Filename: ConfigAndPatchWindow.h

```
#include <Bitmap.h>
#include <View.h>
#include <StringView.h>
#include <Window.h>
#include <Application.h>
#include <Menu.h>
#include <MenuItem.h>
#include <MenuBar.h>
#include <ListView.h>
#include <iostream.h>
#include "Component.h"
#include "BDragListView.h"
#include "SocketList.h"
#include <FilePanel.h>
#include "AuroraGeneral.h"

const int32 ADD_COMPONENT = 'AddC';

#define TheParms TheWorld

class ConfigAndPatchWindow : public BWindow {
public:
    ConfigAndPatchWindow();
    ~ConfigAndPatchWindow();
    bool QuitRequested(); //silly user wants to close the program
    void MessageReceived(BMessage *message);

    ChannelList * DisplaySList;
    ChannelList * DisplayPList;
    void RefreshChanLists(bool MinorOnly = false, bool Sockets =
true, bool Plugs = true); //Make sure what is masked is correct...
    BCheckBox * SDisplayChk;
    BCheckBox * PDisplayChk;

private:
    void ApplyGroupMask();

    BView * MainView; //main background of this window
    BMenu * ComponentChoices;
    BMenu * ComponentChoices_Input;
    BMenu * ComponentChoices_Output;
    BMenu * ComponentChoices_Control;
    BMenu * ComponentChoices_Passive;
    BMenu * ComponentChoices_Grouping;
    BMenu * ComponentChoices_Abstract;
    BMenu * ComponentChoices_Undef;

    BMenuItem * G_In;
    BMenuItem * G_Ou;
    BMenuItem * G_SC;
    BMenuItem * G_Pa;
    BMenuItem * G_Gr;
    BMenuItem * G_Ab;
    BMenuItem * G_Ot;
```

```

    uint16 DevGroupMask;

    BList ComponentAddOns; //add on modules...
    BDragListView * ComponentList; //list of objects for light
control
    BView * ComponentConfig;
    AuroranComponent * CurComponent;
    int32 selected;
    BMenuItem * AutoApplyChanges;
    void ApplyComponentChanges(bool direct=false);
    //ChannelList * MasterSocketList;

    BButton * btn_ClearConnection;
    BButton * btn_MakeConnection;

    void SelectNext(BMessage * msg, bool PlugOnly=false,
SocketListItem * PrevDest=NULL); //select next items when making
connections

    BStringView * Connection_From1;
    BStringView * Connection_From2;
    BStringView * Connection_To1;
    BStringView * Connection_To2;

    ChannelListItem * CurDestination;

    bool StartNewProject();
    BFilePanel * FileOpen;
    BFilePanel * FileSave;
    BEntry CurrentFile;
    bool DataChanged; //i.e. we have to re-save the file upon close
    void SaveFile(BEntry * WhatFile);
    void OpenFile(BEntry * WhatFile);

    void PlugSelected(bool NoSocket=true);
    void SocketSelected();
    void ClearSelected();
    void ReCalcAll();

    AuroranWorld TheWorld;

    void SomethinSelected();

};

class WindowBkg : public BView {
public:
    WindowBkg(BRect frame, const char * name, uint32 resize, uint32
flags) : BView(frame,name,resize,flags) {}
    virtual void Draw(BRect updateRect);
};

```

Filename: ConfigAndPatchWindow.cpp

```
#include "ConfigAndPatchWindow.h"
#include <ScrollView.h>
#include <String.h>
#include <stdio.h>
#include <AppKit.h>
#include <StorageKit.h>
#include "Triangle.h"
#include <Message.h>
#include <InterfaceKit.h>
#include "SuperButton.h"
#include "MaxMinCalculator.h"

//Draw window background
void WindowBkg::Draw(BRect updateRect) {
    BView::Draw(updateRect);
    BeginLineArray(3);
    AddLine(BPoint(213,248),BPoint(770,248),(rgb_color)
{128,128,128,255});
    AddLine(BPoint(213,249),BPoint(770,249),(rgb_color)
{250,250,250,255});
    EndLineArray();
};

//Set up to make a default component...

AuroranComponent * MakeComponent(ComponentDescriptor * Who,
AuroranParameters * Params) {return new AuroranComponent(Params);}

ComponentDescriptor DefaultComponent = {
    A_DESC_VER,
    A_NORMAL_DESC,
    NULL,
    "Auroran Component",
    'TKS_',
    'AC',
    0,
    ACC_UNDEF,
    &MakeComponent
};

ConfigAndPatchWindow::ConfigAndPatchWindow() : BWindow(BRect
(20,30,230,560),"Aurora Main",B_TITLED_WINDOW,B_ASYNCHRONOUS_CONTROLS) {

    //Make MaxMinChecker
    TheParms.MaxMinCalc = new MaxMinCalculator(&TheParms);
    TheParms.MaxMinCalc->Run();

    //Interface looper
    TheParms.ILoop = new ILooper(&CurComponent);
    TheParms.ILoop->Run();

    //Pulser
    TheParms.Pulser = new Pulse();
    cout << "Pulser is: " << TheParms.Pulser << endl;
```

```

//Register for updates
TheParms.RequestUpdates(this);

//Restrict sizing
SetSizeLimits(210,210,320,8192);

//Make sure this variable is null!
ComponentConfig=NULL;
CurDestination=NULL;
selected=-1;
CurrentFile=BEntry();
CurrentFile.Unset();

//Set up menu...
BMenuBar* menuBar = new BMenuBar(BRect(0,0,0,0), "menu bar");
AddChild(menuBar);

//File Menu
BMenuItem* menuItem;
BMenu* menu = new BMenu("File");

menu->AddItem(new BMenuItem("New", new BMessage('FMNu'), 'N'));
menu->AddItem(new BMenuItem("Open...", new BMessage('FMOp'), 'O'));
menu->AddItem(new BMenuItem("Save", new BMessage('FMSs'), 'S'));
menu->AddItem(new BMenuItem("Save As...", new BMessage('FMSa')));

menu->AddSeparatorItem();

menuItem = new BMenuItem("About Aurora...", new BMessage
(B_ABOUT_REQUESTED), 'B');
menuItem->SetTarget(be_app);
menu->AddItem(menuItem);

menu->AddSeparatorItem();

menuItem = new BMenuItem("Quit", new BMessage(B_QUIT_REQUESTED),
'Q');
menuItem->SetTarget(be_app);
menu->AddItem(menuItem);

menuBar->AddItem(menu);

//Tools Menu
menu = new BMenu("Tools");

ComponentChoices = new BMenu("Add Component");

menu->AddItem(ComponentChoices);

ComponentChoices->AddItem(ComponentChoices_Input = new BMenu
("Input"));
ComponentChoices->AddItem(ComponentChoices_Output = new BMenu
("Output"));
ComponentChoices->AddItem(ComponentChoices_Control = new BMenu("Show
Control"));
ComponentChoices->AddItem(ComponentChoices_Passive = new BMenu
("Passive"));

```

```

        ComponentChoices->AddItem(ComponentChoices_Grouping = new BMenu
("Device Grouping"));
        ComponentChoices->AddItem(ComponentChoices_Abstract = new BMenu
("Device Abstraction"));
        ComponentChoices->AddItem(ComponentChoices_Undef = new BMenu
("Other"));

        AutoApplyChanges = new BMenuItem("Auto Apply Changes", new BMessage
('ApCh'));
        AutoApplyChanges->SetMarked(true);
        menu->AddItem(AutoApplyChanges);

        menuBar->AddItem(menu);

        //Groups Menu
        menu = new BMenu("Groups");
        menu->AddItem(new BMenuItem("All",new BMessage('G-Al')));
        menu->AddItem(new BMenuItem("None",new BMessage('G-No')));
        menu->AddSeparatorItem();
        menu->AddItem(G_In = new BMenuItem("Inputs",new BMessage('G-In')));
        menu->AddItem(G_Ou = new BMenuItem("Outputs",new BMessage('G-Ou')));
        menu->AddItem(G_SC = new BMenuItem("Show Control",new BMessage('G-
SC')));
        menu->AddItem(G_Pa = new BMenuItem("Passive",new BMessage('G-Pa')));
        menu->AddItem(G_Gr = new BMenuItem("Device Grouping",new BMessage
('G-Gr')));
        menu->AddItem(G_Ab = new BMenuItem("Device Abstraction",new BMessage
('G-Ab')));
        menu->AddItem(G_Ot = new BMenuItem("Other",new BMessage('G-Ot')));
        menu->AddSeparatorItem();
        menu->AddSeparatorItem();

        menuBar->AddItem(menu);

        //Create Open/Save windows - the user is going to need them anyway,
        might as well make them now
        FileOpen = new BFilePanel(B_OPEN_PANEL,NULL,NULL,B_FILE_NODE,false);
        FileOpen->SetTarget(this);
        FileSave = new BFilePanel(B_SAVE_PANEL,NULL,NULL,B_FILE_NODE,false);
        FileSave->SetTarget(this);

        /* give it a gray background view */
        BRect aRect;
        aRect = Frame();
        aRect.OffsetTo(B_ORIGIN);
        aRect.top += menuBar->Frame().Height() + 2;
        //aRect.bottom -= 100;
        MainView= new WindowBkg(aRect, "Background View", B_FOLLOW_ALL,
B_WILL_DRAW);
        MainView->SetViewColor(kViewGray);
        AddChild(MainView);

```

```

//Add controls
//Components Label
BStringView * Label = new BStringView(BRect(10,10,180,24),
"lbl_components","System Components:");
Label->SetFontSize(12.5);
MainView->AddChild(Label);

//Expander for Component Options
BTriangle * MyTriangle = new BTriangle(BRect(186,6,200,20),
"tri_expand",new BMessage('XPND'));
MainView->AddChild(MyTriangle);

//Component List
aRect = MainView->Frame();
aRect.OffsetTo(B_ORIGIN);
aRect.top += 26;
aRect.right = 200 - B_V_SCROLL_BAR_WIDTH;
aRect.left += 10;
aRect.bottom -= 10;
TheParms.CList=ComponentList = new BDragListView(aRect,
"lst_components",&TheWorld,B_SINGLE_SELECTION_LIST,B_FOLLOW_TOP_BOTTOM);
ComponentList->SetSelectionMessage(new BMessage('nSel')); //what to
send when item selected
// for (char i ='A';i<'z';i++) ComponentList->AddItem(new BStringItem
(&i,1));
MainView->AddChild(new BScrollView("scroll_components",
ComponentList,B_FOLLOW_TOP_BOTTOM,0,false,true));

//Master Socket List
aRect = MainView->Frame();
aRect.OffsetTo(B_ORIGIN);
DisplaySList = new ChannelList(BRect(500,258,755,aRect.bottom-90),
"lst_Sockets",B_FOLLOW_TOP_BOTTOM);
DisplaySList->SetSelectionMessage(new BMessage('sSel'));
MainView->AddChild(new BScrollView("scroll_Sockets",DisplaySList,
B_FOLLOW_TOP_BOTTOM,0,false,true));

//Plug list
DisplayPList=new ChannelList(BRect(215,258,470,aRect.bottom-90),
"lst_Plugs",B_FOLLOW_TOP_BOTTOM);
DisplayPList->SetSelectionMessage(new BMessage('pSel'));
MainView->AddChild(new BScrollView("scroll_Plugs",DisplayPList,
B_FOLLOW_TOP_BOTTOM,0,false,true));

//List masking selection check boxes
MainView->AddChild(PDisplayChk = new BCheckBox(BRect(215,aRect.
bottom-85, 470, aRect.bottom-70),"chk_Plugs","Show only for selected
component",new BMessage('ChkP'),B_FOLLOW_BOTTOM));
MainView->AddChild(SDisplayChk = new BCheckBox(BRect(500,aRect.
bottom-85, 755, aRect.bottom-70),"chk_Sockets","Show only for selected
component",new BMessage('ChkS'),B_FOLLOW_BOTTOM));

//Connection buttons
MainView->AddChild(
    btn_ClearConnection = new SuperButton(BRect(660,aRect.bottom-
60,772,aRect.bottom-40),"btn_ClearConnection","Clear Connection",new
BMessage('cCon'),B_FOLLOW_BOTTOM)
);
btn_ClearConnection->SetEnabled(false);

```



```

        MainView->AddChild(
            btn_MakeConnection = new SuperButton(BRect(660,aRect.bottom-
30,772,aRect.bottom-10),"btn_MakeConnection","Make Connection",new
BMessage('mCon'),B_FOLLOW_BOTTOM)
        );
        btn_MakeConnection->SetEnabled(false);

        //Connection descriptions
        Label = new BStringView(BRect(213,aRect.bottom-62,250,aRect.bottom-
45),"lbl_From","From:",B_FOLLOW_BOTTOM);
        Label->SetFont(be_bold_font);
        MainView->AddChild(Label);
        MainView->AddChild(
            Connection_From1 = new BStringView(BRect(250,aRect.bottom-
62,650,aRect.bottom-45),"lbl_Fro1","",B_FOLLOW_BOTTOM)
        );
        Connection_From1->SetFontSize(12);
        MainView->AddChild(
            Connection_From2 = new BStringView(BRect(250,aRect.bottom-
46,650,aRect.bottom-33),"lbl_Fro2","",B_FOLLOW_BOTTOM)
        );
        Connection_From2->SetFontSize(12);

        Label = new BStringView(BRect(213,aRect.bottom-32,250,aRect.bottom-
15),"lbl_To","To:",B_FOLLOW_BOTTOM);
        Label->SetFont(be_bold_font);
        MainView->AddChild(Label);
        MainView->AddChild(
            Connection_To1 = new BStringView(BRect(250,aRect.bottom-32,650,
aRect.bottom-15),"lbl_To1","",B_FOLLOW_BOTTOM)
        );
        Connection_To1->SetFontSize(12);
        MainView->AddChild(
            Connection_To2 = new BStringView(BRect(250,aRect.bottom-16,650,
aRect.bottom-3),"lbl_To2","",B_FOLLOW_BOTTOM)
        );
        Connection_To2->SetFontSize(12);

        //Populate Add Component Menu
        BMessage * MenuMessage;
        Components.AddItem(&DefaultComponent);
        MenuMessage = new BMessage('AddC');
        MenuMessage->AddPointer("Descriptor",&DefaultComponent);
        ComponentChoices->AddItem(new BMenuItem((DefaultComponent.Name),
MenuMessage,0,0));

        image_idaddonId;
        status_t err = B_NO_ERROR;
        //BMenuItem* (*GetMenuItem)(AuroranParameters * Params,
image_id); //addon function prototype
        ComponentDescriptor* (*GetDesc)(image_id);
        app_info info;
        BPath path;

        //look in app directory for effects
        be_app->GetAppInfo(&info);
        BEntry entry(&info.ref);
        entry.GetPath(&path);

```

```

path.GetParent(&path);
path.Append("Components");

BDirectory dir( path.Path() );

//load all effects
while( err == B_NO_ERROR ){
    err = dir.GetNextEntry( (BEntry*)&entry, TRUE );

    if( entry.InitCheck() != B_NO_ERROR ){
        break;
    }
    if( entry.GetPath(&path) != B_NO_ERROR ){
        printf( "entry.GetPath failed\n" );
    }else{
        addonId = load_add_on( path.Path() );
        if( addonId < 0 ){
            printf( "load_add_on( %s ) failed\n", path.Path() );
        }else{
            printf( "load_add_on( %s ) ... ", path.Path() );
            if( get_image_symbol( addonId,
                                "GetDesc",
                                B_SYMBOL_TYPE_TEXT,
                                (void *)&GetDesc ) ){
                printf( "get_image_symbol( GetDesc ) failed\n" );
                unload_add_on( addonId );
            }else{
                printf("successful!\n");

                //Load items in LinkedList of Descriptors
                ComponentDescriptor * CurDesc=(*GetDesc)(addonId);
                BMessage * MenuMessage;
                while (CurDesc!=NULL) {
                    if (CurDesc->DescriptorType==A_NORMAL_DESC)
                {
                    Components.AddItem(CurDesc);
                    MenuMessage = new BMessage('AddC');
                    MenuMessage->AddPointer("Descriptor",
CurDesc);

                    switch (CurDesc->CompClass) { //add item to
create component to applicable menu
                        case (ACC_INPUT):
                            ComponentChoices_Input->AddItem(new
BMenuItem((CurDesc->Name), MenuMessage, 0, 0));
                            break;

                        case (ACC_OUTPUT):
                            ComponentChoices_Output->AddItem(new
BMenuItem((CurDesc->Name), MenuMessage, 0, 0));
                            break;

                        case (ACC_CONTROL):
                            ComponentChoices_Control->AddItem(new
BMenuItem((CurDesc->Name), MenuMessage, 0, 0));
                            break;

                        case (ACC_PASSIVE):
                            ComponentChoices_Passive->AddItem(new
BMenuItem((CurDesc->Name), MenuMessage, 0, 0));

```

```

        break;

        case (ACC_GROUP):

            ComponentChoices_Grouping->AddItem(new
BMenuItem((CurDesc->Name), MenuMessage, 0, 0));
            break;

        case (ACC_ABSTRACT):

            ComponentChoices_Abstract->AddItem(new
BMenuItem((CurDesc->Name), MenuMessage, 0, 0));
            break;

        default:

            ComponentChoices_Undef->AddItem(new
BMenuItem((CurDesc->Name), MenuMessage, 0, 0));
            }
        };
        CurDesc=CurDesc->NextDescriptor;
    }
    ComponentAddOns.AddItem((void*)addonId); //add to
add-on list
    }
    }
    }

    SetSizeLimits(780,780,440,8192);
    ResizeBy(520,0);
    MyTriangle->MouseUp(BPoint(0,0));

    CurComponent=NULL;
}

//clean up
ConfigAndPatchWindow::~ConfigAndPatchWindow() {
    printf("Closing up...\n");

    //clean out menu
    BMenuItem * MyItem;
    while ((MyItem=ComponentChoices->RemoveItem((int32)0))) {
        delete MyItem;
    }

    DataChanged=false; //make sure the new project routine doesn't
harass the user...
    StartNewProject(); //easy way to unload everything...

    TheParms.Pulser->Blacklist(TheParms.MaxMinCalc);
    TheParms.Pulser->Blacklist(TheParms.ILoop);

    TheParms.MaxMinCalc->Lock(); //stop calculating max & mins
    TheParms.ILoop->Lock();

    //unload add-ons

```

```

    for (int32 i=0; i<ComponentAddOns.CountItems(); i++) {
        image_id addon = (image_id)ComponentAddOns.ItemAt(i);
        if (addon >= 0) {
            unload_add_on(addon);
        }
    }

    //Clean up Open/Close dialogue boxes
    delete FileOpen;
    delete FileSave;

    TheParms.ILoop->Quit();
    TheParms.MaxMinCalc->Quit(); //stop calculating max & mins

    delete TheParms.Pulser;

    printf("Closed\n\n");
}

bool ConfigAndPatchWindow::QuitRequested() {

    be_app->PostMessage(B_QUIT_REQUESTED);
    return(true);
}

void ConfigAndPatchWindow::SelectNext(BMessage * msg, bool PlugOnly,
SocketListItem * PrevDest) {

    //first clean up connection info...
    if (PrevDest!=NULL) { //socket's plug changed...
        cout << "clean old socket" << endl;
        TheWorld.MaxMinCalc->CalcMin(PrevDest->GetOwner());
    }
    if (CurComponent!=NULL) { //plug's socket changed
        cout << "clean myself" << endl;
        TheWorld.MaxMinCalc->CalcMax(CurComponent);
    }
    if (((PlugListItem *)DisplayPList->ItemAt(DisplayPList-
>CurrentSelection()))->isConnected()) {
        cout << "clean new socket" << endl;
        if (CurDestination) TheWorld.MaxMinCalc->CalcMin
(CurDestination->GetOwner());
    }

    int32 mods;
    msg->FindInt32("modifiers",&mods);
    if (mods & 1) {
        if (!PlugOnly) DisplayPList->SetSelectionMessage(NULL); //don't
do duplicate selection messages...
        int32 selection;
        selection = DisplayPList->CurrentSelection();
        DisplayPList->Select(++selection);
        PlugSelected(true);
        DisplayPList->ScrollToSelection();
        if (!PlugOnly) {
            selection = DisplaySList->CurrentSelection();

```

```

        DisplaySList->Select(++selection);
        SocketSelected();
        DisplayPList->SetSelectionMessage(new BMessage('pSel')); //
normal messages again
        DisplaySList->ScrollToSelection();
        DisplaySList->Invalidate();
    }
}

DisplayPList->Invalidate();
}

void ConfigAndPatchWindow::ApplyComponentChanges(bool direct) {
    if (ComponentConfig) {
        //Archive current setup...
        BMessage * Arch = new BMessage(A_APPLY_CHANGES);
        TheParms.ILoop->Lock();
        CurComponent->Lock();
        ComponentConfig->Archive(Arch,true);
        CurComponent->Unlock();
        TheParms.ILoop->Unlock();
        //Tell the Component the new information
        if (direct) {
            if (CurComponent) CurComponent->InterfaceMessage(Arch);
        } else {
            TheParms.ILoop->PostMessage(Arch);
        }
        delete Arch;
        //Update view
        ComponentList->InvalidateItem(selected);
        DisplaySList->InvalidateItem(DisplaySList->IndexOf
(CurComponent->ParentSocket()));
        DisplayPList->InvalidateItem(DisplayPList->IndexOf
(CurComponent->ParentPlug()));
    }
}

void ConfigAndPatchWindow::ClearSelected() {
    selected=-1;
    CurDestination=NULL;

    if ((ComponentConfig) && (CurComponent)) {
        CurComponent->OutOfSight(); //let it know to close up
    }

    CurComponent=NULL;

    //Ensure looper knows there is no more CurComponent...
    snooze(1000);
    if (TheParms.ILoop->LockWithTimeout(500000)==B_OK) {
        TheParms.ILoop->Unlock();
    } else {
        //give up - don't want to deadlock
    }

    if (ComponentConfig) {
        MainView->RemoveChild(ComponentConfig);
        delete ComponentConfig;
    }
}

```

```

        ComponentConfig=NULL;
    }
    ComponentList->DeselectAll();

    CurDestination=NULL;
    CurComponent=NULL;

    DisplayPList->DeselectAll();
    DisplayPList->MakeEmpty();

    btn_ClearConnection->SetEnabled(false);
    btn_MakeConnection->SetEnabled(false);

    Connection_From1->SetText("");
    Connection_To1->SetText("");
    Connection_From2->SetText("");
    Connection_To2->SetText("");
}

bool ConfigAndPatchWindow::StartNewProject() {
    if (DataChanged) { //user has changed something...
    }

    TheParms.Pulser->ClearAll();

    ClearSelected();

    DisplaySList->DeselectAll();

    CurDestination=NULL;

    //Properly Quit() all comopnents...
    AuroranComponent * TheItem = NULL;
    while ((TheItem=(AuroranComponent *)ComponentList->RemoveItem
((int32)0))!=NULL) {
        //TheParms.MaxMinCalc->StopCalc(TheItem);
        TheItem->PreDelete();
        TheItem->Lock();
        TheItem->Quit();
    }
    ComponentList->MakeEmpty();

    DisplayPList->DeselectAll();
    DisplayPList->MakeEmpty();

    DisplaySList->MakeEmpty();

    TheWorld.PList.MakeEmpty();
    TheWorld.SList.MakeEmpty();

    return true;
}

//someone selected a plug or a socket
void ConfigAndPatchWindow::SomethinSelected() {
    // printf("Processing Selection...\n");
    //are two things selected?
    if (DisplayPList->CurrentSelection()>=0 && DisplaySList-
>CurrentSelection()>=0) {

```

```

//      printf("It has potential...\n");
      if (DisplaySList->TestCompatability->isConnected(DisplayPList-
>TestCompatability)) {
          btn_ClearConnection->SetEnabled(true);
          btn_MakeConnection->SetEnabled(false);
      } else if (DisplaySList->TestCompatability->isCompatable
(DisplayPList->TestCompatability)) {
          btn_MakeConnection->SetEnabled(true);
          btn_ClearConnection->SetEnabled(false);
      }
  } else { //if not we can't do anything exiting...
      btn_ClearConnection->SetEnabled(false);
      btn_MakeConnection->SetEnabled(false);
  }

  //Display text info about the plug
  if (DisplayPList->CurrentSelection()>=0) { //only if there is also a
Plug selected...
      ChannelListItem * SelPlug =(ChannelListItem*)DisplayPList-
>ItemAt(DisplayPList->CurrentSelection());
      char stemp[64];
      BString Title = BString(SelPlug->FullName());
      if (!SelPlug->isBundle()) {
          Title.Append(" (");
          if (strlen(SelPlug->GetTypeName())>0) {
              Title.Append(SelPlug->GetTypeName());
              Title.Append(": ");
          }

          if ((SelPlug->GetFlags() & Chan_DontCare)==Chan_DontCare) {
              Title.Append("no value used");
          } else if (SelPlug->GetFlags() & Chan_Pointer) {
              Title.Append("unknown binary data");
          } else if (SelPlug->GetFlags() & Chan_Int) {
              snprintf(stemp,63,"Digital Range [%d, %d]",SelPlug-
>GetMin().i,SelPlug->GetMax().i);
              stemp[63]=0;
              Title.Append(stemp);
          } else {
              snprintf(stemp,63,"Analog Range [%f, %f]",SelPlug-
>GetMin().f,SelPlug->GetMax().f);
              stemp[63]=0;
              Title.Append(stemp);
          }
          Title.Append(")");
      }
      Connection_From1->SetText(Title.String());
      ChannelListItem* NConn=NULL;
      BString Dests = BString("");
      while ((NConn=SelPlug->NextConnection(NConn))) {
          Dests.Append((NConn->FullName()).Append(", "));
      }
      Connection_From2->SetText(Dests.String());
      //TheParms.PList->TestCompatability=CurDestination;
  } else {
      //CurDestination=NULL;
      Connection_From1->SetText("");
      Connection_From2->SetText("");
  }
}

```

```

    }

    //Display text information about the socket
    if (DisplaySList->CurrentSelection()>=0) { //only if there is also a
Plug selected...
        CurDestination=(ChannelListItem*)DisplaySList->ItemAt
(DisplaySList->CurrentSelection());
        char stemp[64];
        BString Title = BString(CurDestination->FullName());
        if (!CurDestination->isBundle()) {
            Title.Append(" (");
            if (strlen(CurDestination->GetTypeName())>0) {
                Title.Append(CurDestination->GetTypeName());
                Title.Append(": ");
            }

            if ((CurDestination->GetFlags() & Chan_DontCare)
==Chan_DontCare) {
                Title.Append("no value used");
            } else if (CurDestination->GetFlags() & Chan_Pointer) {
                Title.Append("unknown binary data");
            } else if (CurDestination->GetFlags() & Chan_Int) {
                snprintf(stemp,63,"Digital Range [%d, %d]",
CurDestination->GetMin().i,CurDestination->GetMax().i);
                stemp[63]=0;
                Title.Append(stemp);
            } else {
                snprintf(stemp,63,"Analog Range [%f, %f]",
CurDestination->GetMin().f,CurDestination->GetMax().f);
                stemp[63]=0;
                Title.Append(stemp);
            }
            Title.Append(")");
        }
        Connection_To1->SetText(Title.String());
        ChannelListItem* NConn=NULL;
        BString Dests = BString("");
        while ((NConn=CurDestination->NextConnection(NConn))) {
            Dests.Append((NConn->FullName()).Append(", "));
        }
        Connection_To2->SetText(Dests.String());
    } else {
        CurDestination=NULL;
        Connection_To1->SetText("");
        Connection_To2->SetText("");
    }
};

void ConfigAndPatchWindow::PlugSelected(bool NoSocket) {
    DisplaySList->TestCompatability=(ChannelListItem *)DisplayPList-
>ItemAt(DisplayPList->CurrentSelection());
    DisplaySList->Invalidate();

    SomethinSelected();
}

void ConfigAndPatchWindow::SocketSelected() {

```



```

        DisplayPList->TestCompatability=(SocketListItem *)DisplaySList-
>ItemAt(DisplaySList->CurrentSelection());
        DisplayPList->Invalidate();

        CurDestination=(SocketListItem *)DisplaySList->ItemAt(DisplaySList-
>CurrentSelection());

        SomethinSelected();
    }

    //Check to make sure what is displayed in the socket and plug lists is
    correct
    void ConfigAndPatchWindow::RefreshChanLists(bool MinorOnly, bool
    Sockets, bool Plugs) {
        BListItem * PtrData;

        if (Sockets && SDisplayChk->Value()==B_CONTROL_ON ||
        MinorOnly==false) {
            PtrData = DisplaySList->ItemAt(DisplaySList-
            >CurrentSelection());
            DisplaySList->MakeEmpty();
            if (SDisplayChk->Value()==B_CONTROL_ON) {
                if (CurComponent!=NULL) {
                    DisplaySList->AddItem(CurComponent->ParentSocket
                    ());
                    DisplaySList->AddList(CurComponent->GetSockets());
                }
            } else if (MinorOnly==false) {
                DisplaySList->AddList(&TheWorld.SList);
            }
            if (DisplaySList->IndexOf(PtrData)>=0) DisplaySList->Select
            (DisplaySList->IndexOf(PtrData));
            SocketSelected();
            DisplaySList->Invalidate();
        }

        if (Plugs && PDisplayChk->Value()==B_CONTROL_ON || MinorOnly==false)
        {
            PtrData = DisplayPList->ItemAt(DisplayPList-
            >CurrentSelection());
            DisplayPList->MakeEmpty();
            if (PDisplayChk->Value()==B_CONTROL_ON) {
                if (CurComponent!=NULL) {
                    DisplayPList->AddItem(CurComponent->ParentPlug());
                    DisplayPList->AddList(CurComponent->GetPlugList());
                }
            } else if (MinorOnly==false) {
                DisplayPList->AddList(&TheWorld.PList);
            }
            if (DisplayPList->IndexOf(PtrData)>=0) DisplayPList->Select
            (DisplayPList->IndexOf(PtrData));
            PlugSelected();
            DisplayPList->Invalidate();
        }
    }

    void ConfigAndPatchWindow::SaveFile(BEntry * WhatFile) {
        BFile WhereToWrite(WhatFile,B_WRITE_ONLY|B_CREATE_FILE|

```

```

B_ERASE_FILE);
    if (WhereToWrite.InitCheck()==B_OK) { //opened OK
        CurrentFile=*WhatFile; //ok, we're committed to this file now.
        WhereToWrite.Seek(0,SEEK_SET); //Make sure we're at the
beginning of the file...
        WhereToWrite.Write("TSAurora",8); //Write basic header
        long int LongData=A_FILE_VERSION;
        WhereToWrite.Write(&LongData,4); //Write file version number
        LongData=ComponentList->CountItems();
        WhereToWrite.Write(&LongData,4); //Write the number of entries
in the file.
        int StartPosition, EndPosition;
        while (--LongData >= 0) { //go through the components bottom-up
            AuroranComponent * Current;
            unsigned short int SaveType=A_COMPONENT;
            WhereToWrite.Write(&SaveType,2); //Data type..

            Current = (AuroranComponent *)ComponentList->ItemAt
(LongData);
            WhereToWrite.Write(&LongData,4); //Store 4 bytes - we will
go back later and write the size of the data here
            StartPosition = WhereToWrite.Position();

            //Write Descriptor
            WhereToWrite.Write(&(Current->GetDesc()->Creator),10); //
Who Made it,

            Current->Flatten(&WhereToWrite);

            EndPosition = WhereToWrite.Position();
            WhereToWrite.Seek(StartPosition-4,SEEK_SET); //go back to
where we need to write the data size...
            StartPosition=EndPosition-StartPosition;
            WhereToWrite.Write(&StartPosition,4); //Store the size of
the data
            WhereToWrite.Seek(EndPosition,SEEK_SET); //And now jump
back to where we left off
        }

    } else {
        (new BAlert("File Error","Unable access requested file. Perhaps
it is locked, or its directory has been deleted","OK",NULL,NULL,
B_WIDTH_AS_USUAL,B_STOP_ALERT))->Go();
    }
}

void BadFormatAlert() {
    (new BAlert("File Error","The File has been opened, but does
not appear to be an Auroran data file","OK",NULL,NULL,B_WIDTH_AS_USUAL,
B_STOP_ALERT))->Go();
}

void ConfigAndPatchWindow::OpenFile(BEntry * WhatFile) {
    TheWorld.StopUpdates(this);
    BFile WhereToRead(WhatFile,B_READ_ONLY);
    char DataCache[16]; //Some space to work with...
    if (WhereToRead.InitCheck()==B_OK) { //opened OK
        WhereToRead.Seek(0,SEEK_SET); //shouldn't be necessary...
        WhereToRead.Read(DataCache,8); //get header...
    }
}

```

```

        if (BString(DataCache,8)=="TSAurora") {
            WhereToRead.Read(DataCache,4); //get version...
            if (*(long *)DataCache)<=A_FILE_VERSION) {
                long Entries;
                unsigned short int SaveType;
                long Size;
                long Start;
                WhereToRead.Read(&Entries,4); //get the number of
entries...
                cout << "Beginning read of " << Entries << " entries."
<< endl;
                while (Entries > 0) {
                    if (WhereToRead.Read(&SaveType,2)) {
                        if (WhereToRead.Read(&Size,4)) {
                            Start=WhereToRead.Position();
                            cout << "Entry Start: " << Start << " Size:
" << Size << " Save Type: 0x" << hex << SaveType << dec << endl;
                            switch (SaveType) {
                                case A_COMPONENT: { //we are loading a
component
                                    cout << "A_COMPONENT found!" <<
endl;
                                    WhereToRead.Read(DataCache,10);
                                    cout << "Read Descriptor!" << endl;
                                    ComponentDescriptor * Comp;
                                    for (int i=Components.CountItems()-
1;i>=0;i--) {
                                        Comp=(ComponentDescriptor *)
Components.ItemAt(i);
                                        cout << "Is it " << Comp-
>Name << "-" << Comp->What << ":"<<Comp->What << " vs " << *((uint32 *)
DataCache) << ":" << *((uint16 *)DataCache+2) << endl;
                                        if (Comp->Creator == *
((uint32 *)DataCache)) { //Creator matches...
                                            if (Comp->What == *
((uint16 *)DataCache+2)) { //Item matches
                                                AuroranComponent
* ThisComponent;
                                                ComponentList-
>AddItem(ThisComponent=(*Comp->MakeComponent) (Comp, &TheParms));
                                                TheWorld.
MoveComponent(ThisComponent,0); //move the component to where we relly
want it
                                                //Make sure the
socket list is in the right place
                                                /*if
((SocketListItem *)TheParms.SList->FirstItem())->GetOwner()!
=ThisComponent) {
ThisComponent->MoveSocketTo(((SocketListItem *)TheParms.SList-
>FirstItem())->GetOwner(),false);
                                                }*/
                                                ThisComponent-
>UnFlatten(&WhereToRead,* (uint32 *) ((char *)DataCache+6));
                                                break; //OK to
break from For-Loop
                                            }
                                        }
                                    }
                                }
                            }
                        }
                    }
                }
            }
        }
    }
}

```

```

                                break;
                                }
                                }
                                WhereToRead.Seek(Start+Size,SEEK_SET);
                                } else {
                                    break; //EOF!
                                }
                                } else {
                                    break; //EOF!!
                                }
                                Entries--;
                            }
                            CurrentFile=*WhatFile; //ok, we're committed to this
file now.
                                //Calculate max & mins
                                ReCalcAll();
                                } else {
                                    BadFormatAlert();
                                }
                                } else {
                                    BadFormatAlert();
                                }
                                } else {
                                    (new BAlert("File Error","Unable access requested file. Perhaps
it is locked, or its diretcory has been deleted","OK",NULL,NULL,
B_WIDTH_AS_USUAL,B_STOP_ALERT))->Go();
                                }

                                RefreshChanLists(false,true,true);
                                TheWorld.RequestUpdates(this);
                            }

void ConfigAndPatchWindow::ReCalcAll() {
    //re-calc max and mins
    TheParms.MaxMinCalc->ReCalcAll();
}

void ConfigAndPatchWindow::MessageReceived(BMessage *message) {
    void * PtrData;
    int itemp;

    switch (message->what)
    {
        //this (AddP) is for debugging only
        case 'AddP':
            cout << "PLUG ADDED INWINDOW LOOPER" << endl;
            break;

        case 'XPND': { //BTriangle's event
            bool hello;
            message->FindBool("TriPosition",0,&hello);
            if (hello) {
                SetSizeLimits(780,780,440,8192);
                ResizeBy(520,0);
            } else {
                SetSizeLimits(210,210,200,8192);
                ResizeBy(-530,0);
            }
        }
    }
}

```

```

        break;
    }
    case A_APPLY_CHANGES: { //save any change in selection....kinda
        ApplyComponentChanges();
        break;
    }
    case 'nSel': { //Component List selection change
        TheParms.Pulser->DeleteClient(TheParms.ILoop);
        if (ComponentConfig!=NULL) {
            if (CurComponent!=NULL) {
                ApplyComponentChanges(true);
                CurComponent->OutOfSight(); //let component know it
is about to be hidden
            }
            snooze(1000); //sleep so the ILooper can handle pending
events if there are any...;
            CurComponent=NULL;

            if (TheParms.ILoop->LockWithTimeout(100000)!=B_OK) { //
if we are deadlocked, give up and try again
                TheParms.Pulser->OneShot(this,new BMessage
(*message));
                break;
            }

            //Destroy the BView
            MainView->RemoveChild(ComponentConfig);
            delete ComponentConfig;
            ComponentConfig=NULL;

            TheParms.ILoop->Unlock();

            DisplaySList->GreyBefore=0;

        }
        if ((selected = ComponentList->CurrentSelection())>=0)
        { // item is actually selected
            if (TheParms.ILoop->LockWithTimeout(100000)!=B_OK) { //
if we are deadlocked, give up and try again
                TheParms.Pulser->OneShot(this,new BMessage
(*message));
                break;
            }
            ComponentConfig=(CurComponent=((AuroranComponent *)
(ComponentList->ItemAt(selected)))->GetView(BRect(211,5,770,240)));
            //cout << "Current Component: " << CurComponent << " in
" << ComponentList << endl;
            ComponentConfig->SetViewColor(MainView->ViewColor());
            MainView->AddChild(ComponentConfig);
            TheParms.ILoop->Unlock();

        }

        RefreshChanLists(true); //Make sure what is masked is
correct...

        DisplaySList->Invalidate();
        break;
    }
}

```

```

        case 'pSel': { //Plug Selected
            PlugSelected(false);
            break;
        }
        case 'sSel': { //Socket Selected
            SocketSelected();
            break;
        }
        case 'cCon': { //Clear connection
            SocketListItem * PrevDest = NULL; //((PlugListItem *)
TheParms.PList->ItemAt(TheParms.PList->CurrentSelection()))->
>Destination;
            ChannelListItem * ThisPlug = (PlugListItem *)DisplayPList->
>ItemAt(DisplayPList->CurrentSelection());
            if (ThisPlug->isConnected(CurDestination)) {
                PrevDest=(SocketListItem*)CurDestination;
            } else {
                break;
            }

            if (CurComponent->LockWithTimeout(50000)==B_OK) {

                ((PlugListItem *)DisplayPList->ItemAt(DisplayPList->
>CurrentSelection()))->Disconnect(PrevDest);

                CurComponent->Unlock();

            } else { //if we timed out, we are deadlocked, and doing
this is OK.
                ((PlugListItem *)DisplayPList->ItemAt(DisplayPList->
>CurrentSelection()))->Disconnect(PrevDest);
            }

            //CurDestination=NULL;
            DisplaySList->InvalidateItem(DisplaySList->CurrentSelection
());
            DisplayPList->InvalidateItem(DisplayPList->CurrentSelection
());

            SelectNext(message,true,PrevDest);
            break;
        }
        case 'mCon': { //make connection
            //SocketListItem * PrevDest = ((PlugListItem *)TheParms.
PList->ItemAt(TheParms.PList->CurrentSelection()))->Destination;

            if (CurComponent->LockWithTimeout(50000)==B_OK) {

                int o = ((PlugListItem *)DisplayPList->ItemAt
(DisplayPList->CurrentSelection()))->Connect(CurDestination);
                printf("Did a connect - result: %d.\n",o);

                CurComponent->Unlock();

            } else { //if we timed out, we are deadlocked, and doing
this is OK.
                ((PlugListItem *)DisplayPList->ItemAt(DisplayPList->
>CurrentSelection()))->Connect(CurDestination);
            }

```

```

        //update...
        DisplaySList->InvalidateItem(DisplaySList->CurrentSelection
    ());
        DisplayPList->InvalidateItem(DisplayPList->CurrentSelection
    ());

        SelectNext(message,false,NULL);
        //TheParms.PList->Invalidate();
        break;
    }
    case 'MOVs': {
        selected=ComponentList->CurrentSelection();
        //Calculate max & mins
        for (int i = ComponentList->CountItems()-1;i>=0;i--) {
            TheParms.MaxMinCalc->CalcMaxMin((AuroranComponent *)
ComponentList->ItemAt(i));
        }
        RefreshChanLists(false,true,true);
        //DisplaySList->GreyBefore=DisplaySList->FullListIndexOf
((AuroranComponent *) (ComponentList->ItemAt(selected)))->LastSocket());
        break;
    }
    case 'ApCh': { //toggle auto apply changes
        AutoApplyChanges->SetMarked(!AutoApplyChanges->IsMarked());
        break;
    }
    case 'FMNu': { ///File/New
        StartNewProject();//start up a new project...
        CurrentFile.Unset(); //We no longer have an associated file
to save to...
        break;
    }
    case 'FMOp': { //File/Open...
        FileOpen->Show();
        break;
    }
    case B_SAVE_REQUESTED: { //dialog wants us to open a file.
        entry_ref Dir;
        BEntry TheFileEntry;
        BString Name;
        message->FindRef("directory",&Dir);
        message->FindString("name",&Name);
        BDirectory ThePath(&Dir);
        if (ThePath.InitCheck()==B_OK) {
            if (TheFileEntry.SetTo(&ThePath,Name.String(),true)
==B_OK) {
                SaveFile(&TheFileEntry); //save file
            } else {
                (new BAlert("Bad File","Unable access requested
file. Perhaps it is locked?","OK",NULL,NULL,B_WIDTH_AS_USUAL,
B_STOP_ALERT))->Go();
            }
        } else {
            (new BAlert("Bad Path","Unable to switch to the
requested directory. Perhaps it has been deleted?","OK",NULL,NULL,
B_WIDTH_AS_USUAL,B_STOP_ALERT))->Go();
        }
        message->PrintToStream();
    }

```

```

        break;
    }
    case 'FMSs': {
        if (CurrentFile.InitCheck()==B_OK) { //Only save if we have
a file to save to...otherwise we are doing a save as
            SaveFile(&CurrentFile); //save file
            break;
        }
    }
    case 'FMSa': { //File/Save As...
        FileSave->Show();
        break;
    }
    case B_REFS_RECEIVED: { //dialog asked to open a file
        message->PrintToStream();
        entry_ref File;
        BEntry TheFileEntry;
        message->FindRef("refs",&File);
        if (TheFileEntry.SetTo(&File,true)==B_OK) {
            if (StartNewProject()) { //only proceede if the user is
ok with wiping out the current project
                OpenFile(&TheFileEntry);
            }
        } else {
            (new BAlert("Bad File","Unable access requested
file. Perhaps it is locked or has been deleted?","OK",NULL,NULL,
B_WIDTH_AS_USUAL,B_STOP_ALERT))->Go();
        }
        break;
    }
    case 'AddC': { //add a component
        ComponentDescriptor * Comp;
        AuroranComponent * owner;
        if(message->FindPointer("Descriptor",(void **)&Comp)==B_OK)
{ //adding from the menu
            ComponentList->AddItem((*Comp->MakeComponent)(Comp,
&TheParms));
        } else if (message->FindPointer("owner",(void **)&owner)
==B_OK) { //adding from an item
            AuroranComponent * ThisComponent;
            ComponentList->AddItem(ThisComponent=(*owner->GetDesc()-
>MakeComponent)(owner->GetDesc(),&TheParms),ComponentList->IndexOf
(owner)+1);
            //ThisComponent->MoveSocketTo(owner,true);
            ReCalcAll();
        }
        break;
    }
    case A_DUPLICATE_COMP: { //duplicate component
        cout << "Duplicate comp" << endl;
        AuroranComponent * owner;
        if (message->FindPointer("owner",(void **)&owner)==B_OK) {
            AuroranComponent * ThisComponent;
            ComponentList->AddItem(ThisComponent=(*owner->GetDesc()-
>MakeComponent)(owner->GetDesc(),&TheParms),ComponentList->IndexOf
(owner)+1);
            //ThisComponent->MoveSocketTo(owner,true);

            { //copy settings

```



```

        BMallocIO VirtualFile;
        owner->Save(&VirtualFile);
        VirtualFile.Seek(0, SEEK_SET);
        ThisComponent->Load(&VirtualFile, owner->GetDesc() -
>Version);
    }

    if (message->FindBool("Connect")==true) { //restore
connections...
        BList * PlugList = owner->GetPlugList();
        BList * PlugList2 = ThisComponent->GetPlugList();
        if (!PlugList->IsEmpty()) {
            ChannelListItem * Dest;
            for (int i = PlugList->CountItems()-1; i>=0; i--)
            {
                Dest=NULL;
                while ((Dest=((PlugListItem *)PlugList-
>ItemAt(i))->NextConnection(Dest))) {
                    //((PlugListItem *)PlugList2->ItemAt
(i))->Destination=((PlugListItem *)PlugList->ItemAt(i))->Destination;
                    ((PlugListItem *)PlugList2->ItemAt(i))-
>Connect(Dest);
                }
            }
        }
        ReCalcAll();
    }
    break;
}
case A_DELETE_TO: {
    AuroranComponent * owner;
    if (message->FindPointer("owner", (void **)&owner)==B_OK) {
        //pester the user first
        if ((new BAlert("Delete", BString("You are about to
DELETE ALL\n\n        INCOMING\n\nconnections to ").Append(owner-
>CustomName).String(), "OK", "Cancel", NULL, B_WIDTH_AS_USUAL,
B_STOP_ALERT))->Go()==0) {
            TheParms.MaxMinCalc->DoCalc(owner, AC_CLEAR_IN);
        }
    }
    break;
}
case A_DELETE_FROM: {
    AuroranComponent * owner;
    if (message->FindPointer("owner", (void **)&owner)==B_OK) {
        //pester the user first
        if ((new BAlert("Delete", BString("You are about to
DELETE ALL\n\n        OUTGOING\n\nconnections from ").Append(owner-
>CustomName).String(), "OK", "Cancel", NULL, B_WIDTH_AS_USUAL,
B_STOP_ALERT))->Go()==0) {
            TheParms.MaxMinCalc->DoCalc(owner, AC_CLEAR_OUT);
        }
    }
    break;
}
case A_DELETE_COMP: { //delete component
    AuroranComponent * owner;

```

```

        if (message->FindPointer("owner", (void **)&owner)==B_OK) {
            //pester the user first
            if ((new BAlert("Delete", BString("You are about to
DELETE ").Append(owner->CustomName).Append(".\nThere is no un-do
feature. Are you sure you want to do this?").String(), "OK", "Cancel",
NULL, B_WIDTH_AS_USUAL, B_STOP_ALERT))->Go()==0) {
                ClearSelected();
                TheParms.CList->RemoveItem(owner);
                owner->PreDelete();
                owner->Lock();
                owner->Quit();
                ReCalcAll();
            }
        }
        break;
    }

    //*****
    //Processing channel operations
    case A_NEW_SOCKET:
        if (message->FindPointer("Socket", &PtrData)==B_OK) {
            //printf("New Socket Registered...\n");
            if (SDisplayChk->Value()==B_CONTROL_ON) {
                if (((ChannelListItem*)PtrData)->GetOwner()
==CurComponent) {
                    DisplaySList->AddItem((BListItem*)PtrData,
TheWorld.SList.IndexOf(PtrData)-TheWorld.SList.IndexOf(CurComponent-
>ParentSocket()));
                    DisplaySList->Invalidate();
                }
            } else {
                DisplaySList->AddItem((BListItem*)PtrData, TheWorld.
SList.IndexOf(PtrData));
                DisplaySList->Invalidate();
            }
        }
        break;
    case A_NEW_PLUG:
        if (message->FindPointer("Plug", &PtrData)==B_OK) {
            //printf("New Plug Registered...\n");
            if (PDisplayChk->Value()==B_CONTROL_ON) {
                if (((ChannelListItem*)PtrData)->GetOwner()
==CurComponent) {
                    DisplayPList->AddItem((BListItem*)PtrData,
TheWorld.PList.IndexOf(PtrData)-TheWorld.PList.IndexOf(CurComponent-
>ParentPlug()));
                    DisplayPList->Invalidate()
;
                }
            } else {
                DisplayPList->AddItem((BListItem*)PtrData, TheWorld.
PList.IndexOf(PtrData));
                DisplayPList->Invalidate();
            }
        }
        break;
    case A_ALTER_PLUG:
        if (message->FindPointer("Item", &PtrData)==B_OK) {
            if ((itemp=DisplayPList->IndexOf((BListItem*)PtrData))

```

```

    >=0) {
        DisplayPList->InvalidateItem(itemp);
    }
}
case A_ALTER_SOCKET:
    if (message->FindPointer("Item",&PtrData)==B_OK) {
        if ((itemp=DisplaySList->IndexOf((BListItem*)PtrData))
    >=0) {
        DisplaySList->InvalidateItem(itemp);
    }
}
case A_RM_PLUG:
    if (message->FindPointer("Item",&PtrData)==B_OK) {
        DisplayPList->RemoveItem((BListItem*)PtrData);
        TheWorld.Pulser->OneShot(this, new BMessage('InvP'));
    }
case A_RM_SOCKET:
    if (message->FindPointer("Item",&PtrData)==B_OK) {
        DisplayPList->RemoveItem((BListItem*)PtrData);
        TheWorld.Pulser->OneShot(this, new BMessage('InvS'));
    }
case 'InvP':
    DisplayPList->Invalidate();
case 'InvS':
    DisplaySList->Invalidate();
case 'ChkP':
    RefreshChanLists(false,false,true);
    break;
case 'ChkS':
    RefreshChanLists(false,true,false);
    break;
case 'CMsk':
    //CheckChanListMasks();
    break;

//*****
//Processing group mask stuff
case 'G-Al':
    DevGroupMask = (uint16) (-1);
    ApplyGroupMask();
    break;
case 'G-No':
    DevGroupMask = 0;
    ApplyGroupMask();
    break;
case 'G-In':
    DevGroupMask^=ACC_INPUT;
    ApplyGroupMask();
    break;
case 'G-Ou':
    DevGroupMask^=ACC_OUTPUT;
    ApplyGroupMask();
    break;
case 'G-SC':
    DevGroupMask^=ACC_CONTROL;
    ApplyGroupMask();
    break;
case 'G-Pa':

```

```

        DevGroupMask^=ACC_PASSIVE;
        ApplyGroupMask();
        break;
    case 'G-Gr':
        DevGroupMask^=ACC_GROUP;
        ApplyGroupMask();
        break;
    case 'G-Ab':
        DevGroupMask^=ACC_ABSTRACT;
        ApplyGroupMask();
        break;
    case 'G-Ot':
        DevGroupMask^=ACC_UNDEF;
        ApplyGroupMask();
        break;

    default:
        BWindow::MessageReceived(message);
}

}

void ConfigAndPatchWindow::ApplyGroupMask() {
    G_In->SetMarked(DevGroupMask & ACC_INPUT);
    G_Ou->SetMarked(DevGroupMask & ACC_OUTPUT);
    G_SC->SetMarked(DevGroupMask & ACC_CONTROL);
    G_Pa->SetMarked(DevGroupMask & ACC_PASSIVE);
    G_Gr->SetMarked(DevGroupMask & ACC_GROUP);
    G_Ab->SetMarked(DevGroupMask & ACC_ABSTRACT);
    G_Ot->SetMarked(DevGroupMask & ACC_UNDEF);

    int LongData=ComponentList->CountItems();
    while (--LongData >= 0) { //go through the components bottom-up
        AuroranComponent * Current;
        Current = (AuroranComponent *)ComponentList->ItemAt(LongData);

        if (Current->GetDesc()->CompClass & DevGroupMask) {
            //printf("keepa...\n");
            Current->SetHeight(37.0);
        } else {
            //printf("hide...\n");
            Current->SetHeight(2.0);
        }
    }
    ComponentList->Invalidate();
}

```

Filename: ILooper.h

```
#ifndef ILooper_h
#define ILooper_h

#include <Looper.h>

class AuroranComponent;

class ILooper : public BLooper {
    AuroranComponent ** CurComponent;
public:
    ILooper(AuroranComponent ** CurComponent): BLooper("Interface
Looper",B_URGENT_DISPLAY_PRIORITY), CurComponent(CurComponent) {}
    void MessageReceived(BMessage * message);
};

#endif
```

Filename: ILooper.cpp

```
#include "ILooper.h"
#include "Component.h"
#include <iostream.h>

void ILooper::MessageReceived(BMessage * message) {
//  cout << "Message Received!!!!!!!!!!!!!!" << endl;
//  message->PrintToStream();
    if (*CurComponent!=NULL) { //only proceed if the destination exists
        (**CurComponent).InterfaceMessage(message);
    }
}
```

Filename: Icons.h

```
#ifndef _ICON_MANAGEMENT
#define _ICON_MANAGEMENT

class BBitmap;

struct IconCollection {
    BBitmap * Pulse;
    BBitmap * Wave;
    BBitmap * Bolt;
    BBitmap * Clock;
    BBitmap * Any;
    BBitmap * Flat;
};

class IconSet {
public:
    IconSet();
    static IconCollection Icon;
    void Free();
private:
    static long initialized;
};

#endif
```

Filename: Icons.cpp

```
#include "Icons.h"
#include <TranslationKit.h>
#include <InterfaceKit.h>

IconSet::IconSet() {
    if (atomic_or(&initialized,1) == 0) {

        Icon.Pulse = BTranslationUtils::GetBitmap(B_PNG_FORMAT, 3001);
        Icon.Wave = BTranslationUtils::GetBitmap(B_PNG_FORMAT, 3002);
        Icon.Bolt = BTranslationUtils::GetBitmap(B_PNG_FORMAT, 3003);
        Icon.Clock = BTranslationUtils::GetBitmap(B_PNG_FORMAT, 3004);
        Icon.Any = BTranslationUtils::GetBitmap(B_PNG_FORMAT, 3005);
        Icon.Flat = BTranslationUtils::GetBitmap(B_PNG_FORMAT, 3006);

        atomic_or(&initialized,2);
    } else {
        while ((initialized & 2) == 0) snooze(1000);
    }
}

IconCollection IconSet::Icon={NULL,NULL,NULL,NULL,NULL,NULL};

int32 IconSet::initialized=0;

void IconSet::Free() {
}
```


Filename: MaxMinCalculator.h

```
#ifndef H_MaxMinCalc
#define H_MaxMinCalc

#include <Looper.h>
#include <Message.h>
#include <iostream.h>

struct AuroranWorld;
class AuroranComponent;
enum {
    AC_CALC_MIN=1,           //Calculate Minimum
    AC_CALC_MAX=2,           //Calculate Maximum

    AC_CLEAR_IN=256,         //Clear Incoming
    AC_CLEAR_OUT=512,        //Clear Outgoing

    AC_PEND_DELETE=1024, //Accept no new calculations
    AC_DELETE_OK=2048       //AC_PEND_DELETE processed
};

class MaxMinCalculator : public BLooper {
public:
    //no fancy initializtion - Just set up looper for un-intrusive
    background operation.
    MaxMinCalculator(AuroranWorld * TheWorld) : BLooper
    ("MaxMinCalculator",B_LOW_PRIORITY), World(TheWorld) {}
    void DoCalc(AuroranComponent * Who, uint32 WhatOp);
    void CalcMax(AuroranComponent * Who) {DoCalc(Who,
AC_CALC_MAX);}
    void CalcMaxMin(AuroranComponent * Who) {DoCalc(Who,
AC_CALC_MAX | AC_CALC_MIN);}
    void CalcMin(AuroranComponent * Who) {DoCalc(Who,
AC_CALC_MIN);}
    void StopCalc(AuroranComponent * Who);
public:
    AuroranWorld * World;
    //BLocker OkToCalc;
    void MessageReceived(BMessage * message);
    void ReCalcAll();
    void ReCalcMaxBefore(AuroranComponent * Who);
    void ReCalcMinAfter(AuroranComponent * Who);
private:
    BMessage * CalcMsg(AuroranComponent * Who, uint32 WhatOp);
};

#endif
```

Filename: MaxMinCalculator.cpp

```
#include "MaxMinCalculator.h"
#include <MessageQueue.h>
#include "Component.h"
#include <OS.h>
#include <iostream.h>

void MaxMinCalculator::DoCalc(AuroranComponent * Who,uint32 WhatOp) {

    if (World->Pulser->isBlacklisted(Who)) return; //don't calculate for
    blacklisted items...

    BMessage * SayWhat;
    if ((SayWhat = CalcMsg(Who,WhatOp))) {
        if (Thread()==find_thread(NULL)) { //be careful not to call
ourselves
            World->Pulser->OneShot(this,SayWhat);
        } else {
            PostMessage(SayWhat);
            delete SayWhat;
        }
    }
}

BMessage * MaxMinCalculator::CalcMsg(AuroranComponent * Who,uint32
WhatOp) {
    WhatOp&=(~AC_DELETE_OK);
    if (World->Pulser->isBlacklisted(Who)) return NULL; //don't
calculate for blacklisted items...
    Who->MaxMinLock.Lock();
    if (Who->LockOpBits) { //operations are pending
        if ((Who->LockOpBits & AC_PEND_DELETE == 0)) { //only if no
delete is pending
            Who->LockOpBits |= WhatOp;
            Who->MaxMinLocked=true; //don't let user move object
until we're done
        }
    } else {
        Who->LockOpBits |= WhatOp;
        Who->MaxMinLocked=true; //don't let user move object until
we're done
        Who->MaxMinLock.Unlock();
        return new BMessage((uint32)Who);
    }
    Who->MaxMinLock.Unlock();
    return NULL;
}

void MaxMinCalculator::StopCalc(AuroranComponent * Who) {
    //cout << "halting calculations" << endl;
    //Lock();
    Who->MaxMinLock.Lock();
    if (Who->LockOpBits) { //operations are pending
        Who->LockOpBits = AC_PEND_DELETE;
    } else {
```

```

        Who->LockOpBits = AC_PEND_DELETE;
        BMessage * x;
        PostMessage(x = new BMessage((uint32)Who));
        delete x;
    }
    Who->MaxMinLock.Unlock();
    //Unlock();

    while ((Who->LockOpBits & AC_DELETE_OK)==0) {snooze(100);}
    cout << "halted" << endl;
}

void MaxMinCalculator::MessageReceived(BMessage * msg) {
    //OkToCalc.Lock();

    BMessageQueue MyQueue;

    MyQueue.AddMessage(new BMessage(*msg));

    BMessage * message = NULL;
    while ((message=MyQueue.NextMessage())) {

        uint32 OpBits;
        OpBits=0;

        AuroranComponent * Who;
        Who = (AuroranComponent *)message->what;

        cout << "Processing operations for " << Who->CustomName << endl;

        Who->MaxMinLock.Lock();

        OpBits=Who->LockOpBits;
        Who->LockOpBits = (OpBits & AC_PEND_DELETE); //clear all bits
        but this one.

        Who->MaxMinLock.Unlock(); //user is free to move the object
        again

        if (OpBits & AC_CLEAR_OUT) {
            BList * PlugList = Who->GetPlugList();
            if (!PlugList->IsEmpty()) {
                if (Who->LockWithTimeout(100000)!=B_OK) { //don't want
                to screw anything up
                    break; //just in case the object is being deleted
                }
                ChannelListItem * PrevDest=NULL;
                for (int i = PlugList->CountItems()-1;i>=0;i--) {
                    //PrevDest=((PlugListItem *)PlugList->ItemAt(i))-
                >Destination;

                    while ((PrevDest = ((PlugListItem *)PlugList-
                >ItemAt(i))>PopDisconnect())) { //this was connected to something
                        BMessage * NewMsg=CalcMsg(PrevDest->GetOwner(),
                AC_CALC_MIN); //this needs its maximum recalculated
                        if ((NewMsg)) MyQueue.AddMessage(NewMsg);

                        (((PlugListItem *)PlugList->ItemAt(i))-
                >Destination=NULL;

```

```

        }
    }
    Who->Unlock();
    World->ILoop->PostMessage(A_INVALIDATE_PLUGS); //update
the view the safe way.
    }
    Who->MaxMoveTo=-2;
    cout << "Outgoing cleared for " << Who->CustomName << "."
<< endl;

    } else if (OpBits & AC_CALC_MAX) { //calculate maximum
    cout << "Calculating MAX for " << Who->CustomName << endl;
        uint32 max=-1;

        for (unsigned int i = World->CList->IndexOf(Who); i <
World->CList->CountItems() ; i++) {
            BList * SktList = ((AuroranComponent *)World->CList-
>ItemAt(i))->GetSockets();
            if (!SktList->IsEmpty()) {
                for (int j = SktList->CountItems()-1;j>=0;j--) {
                    //new, slightly less confusing way
                    if (((ChannelListItem *)SktList->ItemAt(j))-
>isConnected(Who)) {
                        if (i<max) max=i; //new max value...
                    }
                }
            }
        }

        Who->MaxMoveTo=(--max);
        cout << "Maximum Calculated for " << Who->CustomName << ":
" << max << endl;
    }

    //Clear incomming connections
    if (OpBits & AC_CLEAR_IN) {

        BList * SktList = Who->GetSockets();
        if (!SktList->IsEmpty()) {
            if (Who->LockWithTimeout(100000)!=B_OK) { //don't want
to screw anything up
                break; //just in case the object is being deleted
            }
            ChannelListItem * PrevDest=NULL;
            for (int i = SktList->CountItems()-1;i>=0;i--) {
                //PrevDest=((PlugListItem *)PlugList->ItemAt(i))-
>Destination;
                while ((PrevDest = ((ChannelListItem *)SktList-
>ItemAt(i))->PopDisconnect())) { //this was connected to something
                    BMessage * NewMsg=CalcMsg(PrevDest->GetOwner(),
AC_CALC_MAX); //this needs its maximum recalculated
                    if ((NewMsg)) MyQueue.AddMessage(NewMsg);

                    (((PlugListItem *)PlugList->ItemAt(i))-
>Destination=NULL;
                }
            }
        }
    }
}

```

```

        }
        Who->Unlock();
        //World->ILoop->PostMessage(A_INVALIDATE_PLUGS); //
update the view the safe way.
    }

    Who->MinMoveTo=0;
    cout << "Incoming Cleared for " << Who->CustomName << "."
<< endl;
    } else if (OpBits & AC_CALC_MIN) { //calculate minimum
        uint32 min=0;

        for (unsigned int i = World->CList->IndexOf(Who); i > 0 ;
i--) {
            BList * PlugList = ((AuroranComponent *)World->CList-
>ItemAt(i-1))->GetPlugList();
            bool DoesConnect;
            DoesConnect=false;
            if (!PlugList->IsEmpty()) {
                //SocketListItem * Dest;
                for (int j = PlugList->CountItems()-1;j>=0;j--) {
                    /* Old way
                    Dest = ((PlugListItem *)PlugList->ItemAt(j))-
>Destination;

                    if (Dest!=NULL && Dest->GetOwner()==Who) {
                        if (i>min) min=i; //new minimum value...
                    }
                    */
                    //new, slightly less confusing way
                    if (((ChannelListItem *)PlugList->ItemAt(j))-
>isConnected(Who)) {
                        if (i>min) min=i; //new minimum value...
                    }
                }
            }
        }

        Who->MinMoveTo=min;
        cout << "Minimum Calculated for " << Who->CustomName << ":
" << min << endl;
    }

    Who->MaxMinLock.Lock(); //don't let anyone else post message
while we are processing

    //OpBits=Who->LockOpBits;
    if ((Who->LockOpBits & AC_PEND_DELETE) == Who->LockOpBits) {
        Who->MaxMinLocked=false; //calculations done, so it's
safe to move
    }
    if (OpBits & AC_PEND_DELETE) { //we got the delete pending
notice, so this won't be processed again...
        Who->LockOpBits |= AC_DELETE_OK;
    }

    Who->MaxMinLock.Unlock(); //user is free to move the object
again

```

```

        cout << "Calcs done for " << Who->CustomName << endl;
        delete message;
        message=NULL; //just to be clean

    }
    if (message!=NULL) { //something bad happened and we bailed out early
- probably deadlocked
        if (World->Pulser->isBlacklisted(((AuroranComponent *)message->what))) {
            delete message;
        } else {
            World->Pulser->OneShot(this, message);
        }
        while ((message=MyQueue.NextMessage()))
        {
            if (World->Pulser->isBlacklisted(((AuroranComponent *)message->what))) {
                delete message;
            } else {
                World->Pulser->OneShot(this, message);
            }
        }
    }
}

void MaxMinCalculator::ReCalcAll() {
    for (int i = World->CList->CountItems()-1;i>=0;i--) {
        CalcMaxMin((AuroranComponent *)World->CList->ItemAt(i));
    }
}

void MaxMinCalculator::ReCalcMaxBefore(AuroranComponent * Who) {
    for (int i = World->CList->IndexOf(Who)-1;i>=0;i--) {
        CalcMax((AuroranComponent *)World->CList->ItemAt(i));
    }
}

void MaxMinCalculator::ReCalcMinAfter(AuroranComponent * Who) {
    for (int i = World->CList->CountItems()-1;i>World->CList->IndexOf(Who);i--) {
        CalcMin((AuroranComponent *)World->CList->ItemAt(i));
    }
}

```

Filename: PercentControl.h

```
#include <Control.h>
class BSlider;

class PercentView : public BControl {
public:
    PercentView(BRect frame, char * name, char * label, BMessage *
message, uint32 resize, uint32 flags);
    void Draw(BRect updateRect);
private:
    float Value;
};

class PercentControl : public BView {
public:
    PercentControl(BRect frame, char * name, const char * label,
BMessage * message, uint32 resize = B_FOLLOW_TOP|B_FOLLOW_LEFT, uint32
flags = 0);
    ~PercentControl();
    void Draw(BRect updateRect);
    void SetValue(float value);
    float Value();
    status_t SetTarget(BHandler * Handler);
private:
    BSlider * MySlider;
    const char * lbl;
};
```

Filename: PercentControl.cpp

```
#include "PercentControl.h"
#include <Slider.h>
#include <InterfaceKit.h>
#include <stdio.h>

PercentView::PercentView(BRect frame, char * name, char * label,
BMessage * message, uint32 resize, uint32 flags) :
    BControl(frame, name, label, message, resize, flags | B_WILL_DRAW) {
    Value = 50.25;
}

void PercentView::Draw(BRect updateRect) {
    static char Buffer[32];
    SetLowColor(ui_color(B_PANEL_BACKGROUND_COLOR));
    FillRect(updateRect, B_SOLID_LOW);
    sprintf(Buffer, "%.2f", Value);
    MovePenTo((Frame().Width()-StringWidth(Buffer))/2.0, Frame().Height
());
    DrawString(Buffer);
}

PercentControl::PercentControl(BRect frame, char * name, const char *
label, BMessage * message, uint32 resize, uint32 flags) :
    BView(frame, name, resize, flags | B_WILL_DRAW) {

    SetViewColor(ui_color(B_PANEL_BACKGROUND_COLOR));

    lbl = label;
    AddChild(MySlider = new BSlider(BRect(frame.Width()/2,0,frame.Width
()),frame.Height()),NULL,NULL,message,0,36,B_TRIANGLE_THUMB,
B_FOLLOW_TOP|B_FOLLOW_H_CENTER|B_FOLLOW_RIGHT));
    MySlider->SetViewColor(ui_color(B_PANEL_BACKGROUND_COLOR));
    MySlider->SetHashMarkCount(7);
    MySlider->SetHashMarks(B_HASH_MARKS_TOP);
    MySlider->SetBarColor((rgb_color){152,152,255,255});
}

PercentControl::~PercentControl() {delete MySlider;}

void PercentControl::Draw(BRect updateRect) {
    SetLowColor(ui_color(B_PANEL_BACKGROUND_COLOR));
    FillRect(updateRect, B_SOLID_LOW);
    MovePenTo(3,Frame().Height()-3);
    DrawString(lbl);
}

void PercentControl::SetValue(float value) {
    MySlider->SetValue(value*36.0);
}

float PercentControl::Value() {
    return (MySlider->Value())/36.0+0.0001;
}
```



```
status_t PercentControl::SetTarget(BHandler * Handler) {  
    return MySlider->SetTarget(Handler);  
}
```

Filename: PlugList.h

```
#ifndef PL_H
#define PL_H

#include <InterfaceKit.h>
#include "Icons.h"
#include "ChannelListItem.h"
#include "ChannelList.h"

class PlugListItem : public ChannelListItem {
public:
    PlugListItem(const char * name, AuroranComponent * owner,
uint32 level = 1);
    virtual void DrawItem(BView * owner, BRect frame, bool
complete=false);
    //ChannelListItem * Destination;

    //hooking these functions
    int SetValue(CHData value);
    CHData GetValue();
    bool isPlug();
    bool isCompatable(ChannelListItem * who);

    //kept for tradition
    int SendMessage(float value) {return SetValue((CHData)
{value});};

    ~PlugListItem();

    BList * GetEnums();
    Enum * GetEnum(CHData value);

private:
    //IconSet Icons;
    //BMessenger * MyMessenger;
    //BHandler * LastTarget;
};
#endif
```

Filename: PlugList.cpp

```
#include "Component.h"
#include "PlugList.h"
#include "String.h"
#include "SocketList.h"
#include <Debug.h>

PlugListItem::PlugListItem(const char * name, AuroranComponent * owner,
uint32 level) : ChannelListItem(name,owner,level) {

}

void PlugListItem::DrawItem(BView * owner, BRect itemRect, bool
complete=false) {
    ChannelListItem::DrawItem(owner,itemRect,complete);

    owner->SetDrawingMode(B_OP_ALPHA);

    if (isConnected()) {
        owner->MovePenTo(itemRect.Width()/3,owner->PenLocation().y);

        owner->DrawString(((SocketListItem *)NextConnection(NULL))-
>FullName().String());
    }

    owner->SetDrawingMode(B_OP_COPY);
}

PlugListItem::~PlugListItem() {
// if (MyMessenger!=NULL) delete MyMessenger;
}

CHData PlugListItem::GetValue() {
    if (isConnected()) {
        ConnList.Lock();
        CHData x = Connections[0]->GetValue();
        ConnList.Unlock();
        return x;
    } else {
        return LatestValue;
    }
}

int PlugListItem::SetValue(CHData value) {
    LatestValue=value;

    PRINT(("s:%s sending %f ",owner->CustomName,Text(),value.f));
    if (ConnList.LockWithTimeout(500)==B_OK) {
        for (unsigned int i =0; i<UseageCount;i++) {
            Connections[i]->SetValue(value);
            PRINT(("to s:%s ",Connections[i]->GetOwner()->CustomName,
Connections[i]->Text()));
        }
        ConnList.Unlock();
    }
    PRINT(("n"));
}
```

```

        return 0;
    }

    bool PlugListItem::isCompatable(ChannelListItem * who) {
        return who->isSocket();
    }

    bool PlugListItem::isPlug() {
        return true;
    }

    BList * PlugListItem::GetEnums() {
        if (isConnected()) {
            return NextConnection(NULL)->GetEnums();
        } else return NULL;
    }

    Enum * PlugListItem::GetEnum(CHData value) {
        if (isConnected()) {
            return NextConnection(NULL)->GetEnum(value);
        } else return NULL;
    }

```

Filename: PrettySlider.h

```
#include <Slider.h>

class PrettySlider: public BSlider {
    public:
        PrettySlider(BRect frame, const char *name, const char *label,
            BMessage *message, int32 minValue, int32 maxValue, orientation posture
            = B_HORIZONTAL, thumb_style thumbType = B_BLOCK_THUMB, uint32
            resizingMode = B_FOLLOW_LEFT | B_FOLLOW_TOP, uint32 flags = B_NAVIGABLE
            | B_WILL_DRAW | B_FRAME_EVENTS) :
            BSlider(frame, name, label, message, minValue, maxValue, posture,
            thumbType, resizingMode, flags) {}
        virtual void DrawBar();
};
```

Filename: PrettySlider.cpp

```
#include "PrettySlider.h"

void PrettySlider::DrawBar() {
    BSlider::DrawBar();
    float value=Position();
    // float Red,Green,Blue;
    rgb_color Start={250,240,112,255};
    OffscreenView()->SetHighColor(Start);
    BRect MyRect = BarFrame();
    MyRect.bottom--;
    MyRect.left++;
    MyRect.right--;
    MyRect.top=MyRect.bottom-MyRect.Height()*value;
    OffscreenView()->FillRect(MyRect);
}
```

Filename: Pulse.h

```
#ifndef Pulse_h
#define Pulse_h

#include "ThreadPrimitive.h"
#include <List.h>
#include <Locker.h>

class AuroranComponent;

struct OneShotClient {
    BLooper * Client;
    BMessage * msg;
    OneShotClient * next;
};

struct ClientData {
    BLooper * Client;
    float Delay;
    float Elapsed;
    bool OK;
};

struct BlackListItem {
    void * object;
    bigtime_t SinceWhen;
};

class Pulse : public ThreadPrimitive {
public:
    //setup
    Pulse();
    ~Pulse();
    void RunThread();

    //pulses
    void AddClient(BLooper * Client, float Delay);
    void UpdateClient(BLooper * Client, float Delay);
    void GotIt(BLooper * Client);

    //single shot message
    void OneShot(BLooper * Client, BMessage * message);

    //general
    void DeleteClient(BLooper * Client);
    void ClearAll();

    //blacklist functions
    void Blacklist(void * Who);
    bool isBlacklisted(void * Who);
    void Revive(void * Who);
private:
    BLocker ListLock;
    BLocker OneShotLock;
    BLocker BlackListLock;

    BList Clients;
```

```
        OneShotClient * OneShots;  
        BList TheBlackList;  
};  
#endif
```


Filename: Pulse.cpp

```
#include "Pulse.h"
#include "Component.h"
#include <iostream.h>
#include <OS.h>

Pulse::Pulse() : ThreadPrimitive(THREAD_HIGH,"Pulser") {
    Go();
    OneShots=NULL;
}

void Pulse::OneShot(BLooper * Client, BMessage * message) {
    //cout << "Got request for OneShot to be sent to " << Client<<
    " ("<<Client->Name()<<")"<<endl;

    if (isBlacklisted(Client)) {
        cout << "Request for oneshot failed - blacklisted!" << endl;
        return; //don't operate on dead objects
    }

    if (kill==false && OneShotLock.LockWithTimeout(200000)==B_OK) { //
if this times out, it must have died, or there is a deadlock
        OneShotClient * OSC = new OneShotClient();
        OSC->next=NULL;
        OSC->Client = Client;
        OSC->msg = message;
        if (OneShots==NULL) { //list is empty
            OneShots=OSC;
        } else { //list has other pending items
            OneShotClient * COS = OneShots;
            while ((COS->next)) {COS=COS->next;} //find last item in
linked list
            COS->next = OSC;
        }
        //cout << "Request for oneshot complete" << endl;
        OneShotLock.Unlock();
    } else {
        cout << "Request for oneshot failed - killed or deadlock" <<
endl;
    }
}

void Pulse::AddClient(BLooper * Client, float Delay) {

    if (isBlacklisted(Client)) return; //don't operate on dead objects

    ClientData * CD = new ClientData;
    CD->Client = Client;
    CD->Delay = Delay;
    CD->Elapsed = 0.0;
    CD->OK=true;
    if (kill==false && ListLock.LockWithTimeout(300000)==B_OK) { //if
this times out, it must have died, so give up
        Clients.AddItem(CD);
        ListLock.Unlock();
    } else {
        delete CD; //clean up
    }
}
```

```

    }
}

void Pulse::UpdateClient(BLooper * Client, float Delay) {
    ClientData * CD;
    if (kill==false && ListLock.LockWithTimeout(150000)==B_OK) { //if
this times out, it must have died, so give up
        for (int i = Clients.CountItems()-1; i >= 0; i--) {
            CD = (ClientData *)Clients.ItemAt(i);
            if (CD->Client == Client) {
                CD->Delay=Delay;
                CD->Elapsed = 0.0;
                break;
            }
        }
        ListLock.Unlock();
    }
}

void Pulse::GotIt(BLooper * Client) {
    ClientData * CD;
    if (kill==false && ListLock.LockWithTimeout(500000)==B_OK) { //if
this times out, it must have died, so give up
        for (int i = Clients.CountItems()-1; i >= 0; i--) {
            CD = (ClientData *)Clients.ItemAt(i);
            if (CD->Client == Client) {
                CD->OK=true;
            }
        }
        ListLock.Unlock();
    }
}

void Pulse::DeleteClient(BLooper * Client) {
    ClientData * CD;
    if (ListLock.LockWithTimeout(1000000)==B_OK) { //better be careful
        for (int i = Clients.CountItems()-1; i >= 0; i--) {
            CD = (ClientData *)Clients.ItemAt(i);
            if (CD->Client == Client) {
                Clients.RemoveItem(i);
                delete CD;
                //break;
            }
        }
        ListLock.Unlock();
    }

    if (OneShotLock.LockWithTimeout(1000000)==B_OK) { //if this times
out, it must have died, or there is a deadlock
        //if we're at the start of the list, fix that
        while (OneShots!=NULL && OneShots->Client!=Client) {
            OneShots=OneShots->next;
        }

        //if we're elsewhere
        if ((OneShots)) {
            OneShotClient * OSC;
            OSC=OneShots;
            while ((OSC->next)) {

```

```

        if (OSC->next->Client==Client) {
            OneShotClient * garbage = OSC->next;
            OSC->next = OSC->next->next;
            delete garbage;
        } else {
            OSC=OSC->next;
        }
    }
}

OneShotLock.Unlock();
}
}

void Pulse::ClearAll() {
    //clear pulse requests
    if (ListLock.LockWithTimeout(1000000)==B_OK) { //better be careful
        while ((Clients.ItemAt(0))) {
            delete Clients.RemoveItem((int32)0);
        }
        ListLock.Unlock();
    } else {
        snooze(500);
        if (kill==false) { //proceed unless we're dead
            cout << "List Locked! " << ListLock.IsLocked() << "
                " by thread: " << ListLock.LockingThread() << " with "
                << ListLock.CountLocks() << " locks and has " <<
                ListLock.CountLockRequests() << " other threads waiting. I
am " << find_thread(NULL) << endl;
            ClearAll();
        }
    }

    //clear one-shot's
    if (OneShotLock.LockWithTimeout(1000000)==B_OK) { //better be
careful
        while (OneShots!=NULL) {
            OneShotClient * OS;
            OS = OneShots;
            delete OneShots->msg;
            OneShots = OneShots->next;
            delete OS;
        }
        OneShotLock.Unlock();
    } else {
        snooze(500);
        if (kill==false) { //proceed unless we're dead
            cout <<"OneShot Locked! "<<OneShotLock.IsLocked() << " by
thread: " << OneShotLock.LockingThread() << " with " << OneShotLock.
CountLocks() << " locks and has " << OneShotLock.CountLockRequests() <<
" other threads waiting" << endl;
            ClearAll();
        }
    }
}

void Pulse::Blacklist(void * Who) {
    BlackListLock.Lock();
}

```

```

        for (int i = TheBlackList.CountItems()-1; i >=0 ; i--) {
            if (((BlackListItem *)TheBlackList.ItemAt(i))->object==Who) {
                ((BlackListItem *)TheBlackList.ItemAt(i))-
>SinceWhen=real_time_clock_usecs();
                return;
            }
        }

        BlackListLock.Unlock();
    }

bool Pulse::isBlacklisted(void * Who) {
    BlackListLock.Lock();

    bigtime_t DeleteOlder = real_time_clock_usecs() - 30000000; //oldest
items to allow

    BlackListItem * BLI;

    for (int i = 0; (BLI=(BlackListItem *)TheBlackList.ItemAt(i)) ; ) {
        if (BLI->object==Who) {
            return true;
        } else if (BLI->SinceWhen < DeleteOlder) {
            TheBlackList.RemoveItem(i);
        } else {
            i++;
        }
    }

    BlackListLock.Unlock();
    return false;
}

void Pulse::Revive(void * Who) {
    BlackListLock.Lock();

    bigtime_t DeleteOlder = real_time_clock_usecs() - 30000000; //oldest
items to allow

    BlackListItem * BLI;

    for (int i = 0; (BLI=(BlackListItem *)TheBlackList.ItemAt(i)) ; ) {
        if (BLI->object==Who || BLI->SinceWhen < DeleteOlder) {
            TheBlackList.RemoveItem(i);
        } else {
            i++;
        }
    }

    BlackListLock.Unlock();
}

Pulse::~~Pulse() {
    kill=true;
    ClearAll();
    //lock up
    ListLock.Lock();
    OneShotLock.Lock();
}

```

```

}

void Pulse::RunThread() {
    bigtime_t LastTime;
    bigtime_t ThisTime;
    float Elapsed;
    ClientData * CD;
    ThisTime=real_time_clock_usecs();

    while (kill==false) {
        LastTime=ThisTime;

        snooze(1000); //delay 1ms

        //handle items that need a regular pulse
        ThisTime=real_time_clock_usecs();
        if (ListLock.LockWithTimeout(500)==B_OK) {
            if (!Clients.IsEmpty()) {
                Elapsed = (ThisTime-LastTime) / 1000.0;
                for (int i = Clients.CountItems()-1; i >= 0; i--) {
                    CD = (ClientData *)Clients.ItemAt(i);
                    CD->Elapsed+=Elapsed;
                    if (CD->Elapsed > CD->Delay) { //time to bug them
                        CD->Elapsed -= CD->Delay;
                        if (CD->OK==true) { //only if they replied to
our last one!
                            CD->OK=false;
                            CD->Client->PostMessage(A_PULSE);
                        }
                    }
                }
                ListLock.Unlock();
            } else {
                cout << "Pulser list is locked" << endl;
            }

            //process delayed message requests
            if (OneShotLock.LockWithTimeout(500)==B_OK) {
                while (OneShots!=NULL) {
                    OneShotClient * OS;
                    OS = OneShots;
                    OneShots->Client->PostMessage(OneShots->msg);
                    delete OneShots->msg;
                    OneShots = OneShots->next;
                    //cout << "Sent OneShot to thread: " << OS->Client-
>Thread() << endl;
                    delete OS;
                }
                OneShotLock.Unlock();
            } else {
                cout << "One-shot list is locked!!" << endl;
            }
        }
    }
}

```

Filename: ResourceSet.h

```
/
*****
*****
/
/   File:           ResourceSet.h
/
/   Description:    Experimental class for managing resources.
/
*****
*****/

/*
   Copyright 2000, Be Incorporated.   All Rights Reserved.
   This file may be used under the terms of the Be Sample Code License.
*/

#ifndef RESOURCE_SET_H
#define RESOURCE_SET_H

#ifndef _BITMAP_H
#include <Bitmap.h>
#endif

#ifndef _CURSOR_H
#include <Cursor.h>
#endif

#ifndef _LIST_H
#include <List.h>
#endif

#ifndef _LOCKER_H
#include <Locker.h>
#endif

#ifndef _MESSAGE_H
#include <Message.h>
#endif

#ifndef _SUPPORT_DEFS_H
#include <SupportDefs.h>
#endif

class BResources;
class BDataIO;

namespace BExperimental {

namespace BResourcePrivate {
    class TypeItem;
    class TypeList;
}

using namespace BResourcePrivate;

// Some new types we handle.
```

```

enum {
    B_CURSOR_TYPE          = 'CURS',
    B_BITMAP_TYPE          = 'BBMP'
};

// String conversion. The default implementation does a
// getenv() on the requested name.
class BStringMap
{
public:
    BStringMap();
    virtual ~BStringMap();

    virtual const char* FindString(const char* name);
};

class BResourceSet
{
public:
    BResourceSet();
    virtual ~BResourceSet();

    // Add resource files that are searched for data. Resources are
    // checked from first to last; the first one that has a match for
    // a particular request is used for that data.

    status_t AddResources(BResources* resources, bool at_front=true);
    status_t AddResources(const void* image_addr, bool at_front=true);

    // Add directories that are searched for data. Directories are
    // always searched before resource files; they are only searched
    // when looking for a resource by name.

    status_t AddDirectory(const char* full_path, bool at_front=true);
    status_t AddEnvDirectory(const char* env_path,
                             const char* default_value = 0,
                             bool at_front=true,
                             BStringMap* variables=0);

    // Functions to look up generic resource data.

    const void* FindResource(type_code type, int32 id,
                             size_t* out_size);
    const void* FindResource(type_code type, const char* name,
                             size_t* out_size);

    // Functions to look up specific types of data.

    const BBitmap* FindBitmap(type_code type, int32 id);
    const BBitmap* FindBitmap(type_code type, const char* name);

    const BCursor* FindCursor(type_code type, int32 id);
    const BCursor* FindCursor(type_code type, const char* name);

    const BMessage* FindMessage(type_code type, int32 id);
    const BMessage* FindMessage(type_code type, const char* name);

    // Functions to look up specific types of data with standard type
    codes.

```

```

const BBitmap* FindBitmap(int32 id);
const BBitmap* FindBitmap(const char* name);

const BCursor* FindCursor(int32 id);
const BCursor* FindCursor(const char* name);

const BMessage* FindMessage(int32 id);
const BMessage* FindMessage(const char* name);

protected:
class TypeObject {
public:
    TypeObject();
    virtual ~TypeObject();
    void Delete();

private:
    TypeObject(const TypeObject& o);
    TypeObject& operator=(const TypeObject& o);
    bool operator==(const TypeObject& o);
    bool operator!=(const TypeObject& o);

    bool fDeleteOK;
};

class BitmapObject : public BBitmap, public TypeObject
{
public:
    BitmapObject(BRect bounds,
                 uint32 flags,
                 color_space depth,
                 int32 bytesPerRow=B_ANY_BYTES_PER_ROW,
                 screen_id screenID=B_MAIN_SCREEN_ID);
    BitmapObject(const BBitmap* source,
                 bool accepts_views = false,
                 bool need_contiguous = false);
    BitmapObject(BMessage *data);

    virtual ~BitmapObject();
};

class CursorObject : public BCursor, public TypeObject
{
public:
    CursorObject(const void* data);
    CursorObject(BMessage *data);
    virtual ~CursorObject();
};

class MessageObject : public BMessage, public TypeObject
{
public:
    MessageObject();
    virtual ~MessageObject();
};

virtual BitmapObject* GenerateBitmap(const void* data, size_t size);
virtual CursorObject* GenerateCursor(const void* data, size_t size);

```



```

    virtual MessageObject* GenerateMessage(const void* data, size_t
size);

private:
    friend class BResourcePrivate::TypeItem;

    status_t expand_string(BString* out, const char* in,
                          BStringMap* variables);
    TypeList* find_type_list(type_code type);

    TypeItem* find_item_id(type_code type, int32 id);
    TypeItem* find_item_name(type_code type, const char* name);

    TypeItem* load_resource(type_code type, int32 id, const char* name,
                          TypeList** inout_list = 0);

    BBitmap* return_bitmap_item(type_code type, TypeItem* from);
    BCursor* return_cursor_item(TypeItem* from);
    BMessage* return_message_item(TypeItem* from);

    static BitmapObject* read_png_image(BDataIO* stream);

    BLocker fLock;                // access control.
    BList fResources;             // containing BResources objects.
    BList fDirectories;           // containing BPath objects.
    BList fTypes;                 // containing TypeList objects.
};

} // namespace BExperimental

using namespace BExperimental;

#endif

```

Filename: ResourceSet.cpp

```
/*
    Copyright 2000, Be Incorporated. All Rights Reserved.
    This file may be used under the terms of the Be Sample Code License.
*/

#include <BeBuild.h>

namespace BExperimental {
class _EXPORT BResourceSet;
}

#define USE_RESOURCES 1
#define HAVE_CURSORS 1

#include "ResourceSet.h"

#ifdef HAVE_CURSORS
#include <Cursor.h>
#endif

#include <Bitmap.h>

#include <Entry.h>
#include <File.h>
#include <Path.h>

#ifdef USE_RESOURCES
#include <Resources.h>
#endif

#include <Autolock.h>
#include <DataIO.h>
#include <Debug.h>
#include <String.h>

#include <stdlib.h>
#include <unistd.h>
#include <ctype.h>

#include <png.h>

namespace BExperimental {
namespace BResourcePrivate {

    class TypeItem {
    public:
        TypeItem(int32 id, const char* name, const void* data, size_t
size)
            : fID(id), fName(name),
              fData(const_cast<void*>(data)), fSize(size), fObject(0),
              fOwnData(false), fSourceIsFile(false)
        {
        }
        TypeItem(int32 id, const char* name, BFile* file)
            : fID(id), fName(name), fData(0), fSize(0), fObject(0),
              fOwnData(true), fSourceIsFile(false)
        {
        }
    };
};
};
```

```

    {
        off_t size;
        if( file->GetSize(&size) == B_OK ) {
            fSize = (size_t)size;
            fData = malloc(fSize);
            if( file->ReadAt(0, fData, fSize) < B_OK ) {
                free(fData);
                fData = 0;
                fSize = 0;
            }
        }
    }
    virtual ~TypeItem()
    {
        if( fOwnData ) {
            free(fData);
            fData = 0;
            fSize = 0;
        }
        SetObject(0);
    }

    int32 ID() const { return fID; }
    const char* Name() const { return fName.String
()); }
    const void* Data() const { return fData; }
    size_t Size() const { return fSize; }

    void SetObject(BResourceSet::TypeObject* object)
    {
        if( fObject ) fObject->Delete();
        fObject = object;
    }
    BResourceSet::TypeObject* Object() const { return fObject;
}

    void SetSourceIsFile(bool state) { fSourceIsFile = state; }
    bool SourceIsFile() const { return fSourceIsFile;
}

private:
    int32 fID;
    BString fName;
    void* fData;
    size_t fSize;
    BResourceSet::TypeObject* fObject;
    bool fOwnData;
    bool fSourceIsFile;
};

static bool free_type_item_func(void* item)
{
    delete reinterpret_cast<TypeItem*>(item);
    return false;
}

class TypeList {
public:
    TypeList(type_code type)

```

```

        : fType(type)
    {
    }

    virtual ~TypeList()
    {
        fItems.DoForEach(free_type_item_func);
        fItems.MakeEmpty();
    }

    type_code Type() const      { return fType; }

    TypeItem* FindItemByID(int32 id)
    {
        for( int32 i=0; i<fItems.CountItems(); i++ ) {
            TypeItem* it = (TypeItem*)fItems.ItemAt(i);
            if( it->ID() == id ) return it;
        }
        return 0;
    }

    TypeItem* FindItemByName(const char* name)
    {
        for( int32 i=0; i<fItems.CountItems(); i++ ) {
            TypeItem* it = (TypeItem*)fItems.ItemAt(i);
            if( strcmp(it->Name(), name) == 0 ) return it;
        }
        return 0;
    }

    void AddItem(TypeItem* item)
    {
        fItems.AddItem(item);
    }

private:
    type_code fType;
    BList fItems;
};

} // namespace BResourcePrivate
} // namespace BExperimental

using namespace BExperimental::BResourcePrivate;

// ----- BStringMap
-----

BStringMap::BStringMap()
{
}

BStringMap::~BStringMap()
{
}

const char* BStringMap::FindString(const char* name)
{
    return getenv(name);
}

```

```

}

// ----- TypeObject
-----

BResourceSet::TypeObject::TypeObject()
    : fDeleteOK(false)
{
}
BResourceSet::TypeObject::~~TypeObject()
{
    if( !fDeleteOK ) debugger("deleting object owned by BResourceSet");
}
void BResourceSet::TypeObject::Delete()
{
    fDeleteOK = true;
    delete this;
}

BResourceSet::BitmapObject::BitmapObject(BRect bounds,
                                           uint32 flags,
                                           color_space depth,
                                           int32 bytesPerRow,
                                           screen_id screenID)
    : BBitmap(bounds, flags, depth, bytesPerRow, screenID)
{
}

BResourceSet::BitmapObject::BitmapObject(const BBitmap* source,
                                           bool accepts_views,
                                           bool need_contiguous)
    : BBitmap(source, accepts_views, need_contiguous)
{
}

BResourceSet::BitmapObject::BitmapObject(BMessage *data)
    : BBitmap(data)
{
}

BResourceSet::BitmapObject::~~BitmapObject()
{
}

BResourceSet::CursorObject::CursorObject(const void* data)
    : BCursor(data)
{
}

BResourceSet::CursorObject::CursorObject(BMessage *data)
    : BCursor(data)
{
}

BResourceSet::CursorObject::~~CursorObject()
{
}

BResourceSet::MessageObject::MessageObject()

```

```

{
}

BResourceSet::MessageObject::~~MessageObject()
{
}

// ----- BResourceSet
-----

BResourceSet::BResourceSet()
{
}

#ifdef USE_RESOURCES
static bool free_resources_func(void* item)
{
    delete reinterpret_cast<BResources*>(item);
    return false;
}
#endif

static bool free_path_func(void* item)
{
    delete reinterpret_cast<BPath*>(item);
    return false;
}

static bool free_type_func(void* item)
{
    delete reinterpret_cast<TypeList*>(item);
    return false;
}

BResourceSet::~~BResourceSet()
{
    BAutolock l(fLock);
#ifdef USE_RESOURCES
    fResources.DoForEach(free_resources_func);
    fResources.MakeEmpty();
#endif
    fDirectories.DoForEach(free_path_func);
    fDirectories.MakeEmpty();
    fTypes.DoForEach(free_type_func);
    fTypes.MakeEmpty();
}

status_t BResourceSet::AddResources(BResources* resources, bool
at_front)
{
#ifdef USE_RESOURCES
    if(!resources) return B_BAD_VALUE;

    BAutolock l(fLock);
    status_t err = (at_front ? fResources.AddItem(resources, 0)
                            : fResources.AddItem(resources))
                  ? B_OK : B_ERROR;
    if( err != B_OK ) delete resources;
    return err;

```

```

#else
    (void)resources;
    (void)at_front;
    return B_ERROR;
#endif
}

// Return the image_id corresponding to the given address, in either
// its text or data section.
static image_id find_image(const void *image_addr)
{
    image_info info;
    int32 cookie = 0;
    while (get_next_image_info(0, &cookie, &info) == B_OK)
        if ((info.text <= image_addr && (((uint8 *)info.text)+info.
text_size) > image_addr)
            || (info.data <= image_addr && (((uint8 *)info.data)+info.
data_size) > image_addr))
            // Found the image.
            return info.id;

    return -1;
}

status_t BResourceSet::AddResources(const void* image_addr, bool
at_front)
{
    #if USE_RESOURCES
        // Lookup the image for this address.
        image_id image = find_image(image_addr);
        if( image < B_OK ) return image;

        // Get information about the returned image.
        image_info info;
        status_t err = get_image_info(image, &info);
        if( err != B_OK ) return err;

        // Open the image's file.
        BFile file;
        err = file.SetTo(&info.name[0], B_READ_ONLY);
        if( err != B_OK ) return err;

        // Create a new resource object on that file.
        BResources* res = new BResources;
        if( !res ) return B_NO_MEMORY;

        err = res->SetTo(&file);
        if( err != B_OK ) {
            delete res;
            return err;
        }

        // Add it to the set.
        err = AddResources(res, at_front);
        if( err != B_OK ) delete res;

        return err;
    #else
        (void)image_addr;
    #endif
}

```

```

        return B_ERROR;
#endif
}

status_t BResourceSet::AddDirectory(const char* full_path, bool
at_front)
{
    if( !full_path ) return B_BAD_VALUE;
    BPath* p = new BPath(full_path);
    status_t err = p->InitCheck();
    if( err != B_OK ) {
        delete p;
        return err;
    }

    BAutolock l(fLock);
    err = (at_front ? fDirectories.AddItem(p, 0)
                  : fDirectories.AddItem(p))
        ? B_OK : B_ERROR;
    if( err != B_OK ) delete p;
    return err;
}

status_t BResourceSet::AddEnvDirectory(const char* in,
                                       const char* default_value,
                                       bool at_front,
                                       BStringMap* variables)
{
    BStringMap def_variables;
    if( !variables ) variables = &def_variables;

    BString buf;
    status_t err = expand_string(&buf, in, variables);

    if( err != B_OK ) {
        if( default_value ) return AddDirectory(default_value,
at_front);
        return err;
    }

    return AddDirectory(buf.String(), at_front);
}

status_t BResourceSet::expand_string(BString* out, const char* in,
                                     BStringMap* variables)
{
    PRINT(("Expanding string: %s\n", in));

    const char* start = in;
    while( *in ) {
        if( *in == '$' ) {
            if( start < in ) out->Append(start, (int32)(in-start));

            in++;
            char variableName[1024];
            size_t i = 0;
            if( *in == '{' ) {
                in++;
                while( *in && *in != '}' && i<sizeof(variableName)-1 )

```



```

        variableName[i++] = *in++;
        if( *in ) in++;
    } else {
        while( isalnum(*in) || *in == '_' && i<sizeof
(variableName)-1 )
            variableName[i++] = *in++;
    }

    start = in;

    variableName[i] = '\\0';

    const char *val = variables
        ? variables->FindString(variableName)
        : 0;

    if( !val ) {
        PRINT(("Error: env var %s not found.\\n", &variableName
[0]));
        return B_NAME_NOT_FOUND;
    }

    status_t err = expand_string(out, val, variables);
    if( err != B_OK ) return err;

    } else if( *in == '\\\\' ) {
        if( start < in ) out->Append(start, (int32)(in-start));
        in++;
        start = in;
        in++;
    } else
        in++;
}

if( start < in ) out->Append(start, (int32)(in-start));

PRINT(("-> Expanded %s to %s\\n", in, out->String()));

return B_OK;
}

const void* BResourceSet::FindResource(type_code type, int32 id,
size_t* out_size)
{
    TypeItem* item = find_item_id(type, id);

    if( out_size ) *out_size = item ? item->Size() : 0;
    return item ? item->Data() : 0;
}

const void* BResourceSet::FindResource(type_code type, const char* name,
size_t* out_size)
{
    TypeItem* item = find_item_name(type, name);

    if( out_size ) *out_size = item ? item->Size() : 0;
    return item ? item->Data() : 0;
}

```

```

// -----

const BBitmap* BResourceSet::FindBitmap(type_code type, int32 id)
{
    return return_bitmap_item(type, find_item_id(type, id));
}

const BBitmap* BResourceSet::FindBitmap(type_code type, const char*
name)
{
    return return_bitmap_item(type, find_item_name(type, name));
}

const BBitmap* BResourceSet::FindBitmap(int32 id)
{
    return return_bitmap_item(B_BITMAP_TYPE, find_item_id(B_BITMAP_TYPE,
id));
}

const BBitmap* BResourceSet::FindBitmap(const
char* name)
{
    return return_bitmap_item(B_BITMAP_TYPE, find_item_name
(B_BITMAP_TYPE, name));
}

// -----

#ifdef HAVE_CURSORS
const BCursor* BResourceSet::FindCursor(type_code type, int32 id)
{
    return return_cursor_item(find_item_id(type, id));
}

const BCursor* BResourceSet::FindCursor(type_code type, const char*
name)
{
    return return_cursor_item(find_item_name(type, name));
}

const BCursor* BResourceSet::FindCursor(int32 id)
{
    return return_cursor_item(find_item_id(B_CURSOR_TYPE, id));
}

const BCursor* BResourceSet::FindCursor(const char* name)
{
    return return_cursor_item(find_item_name(B_CURSOR_TYPE, name));
}
#endif

// -----

const BMessage* BResourceSet::FindMessage(type_code type, int32 id)
{
    return return_message_item(find_item_id(type, id));
}

```

```

const BMessage* BResourceSet::FindMessage(type_code type, const char*
name)
{
    return return_message_item(find_item_name(type, name));
}

const BMessage* BResourceSet::FindMessage(int32 id)
{
    return return_message_item(find_item_id(B_MESSAGE_TYPE, id));
}

const BMessage* BResourceSet::FindMessage(const char* name)
{
    return return_message_item(find_item_name(B_MESSAGE_TYPE, name));
}

// -----

TypeList* BResourceSet::find_type_list(type_code type)
{
    BAutolock l(fLock);

    for( int32 i=0; i<fTypes.CountItems(); i++ ) {
        TypeList* list = (TypeList*)fTypes.ItemAt(i);
        if( list && list->Type() == type ) return list;
    }

    return 0;
}

TypeItem* BResourceSet::find_item_id(type_code type, int32 id)
{
    TypeList* list = find_type_list(type);
    TypeItem* item = 0;

    if( list ) {
        BAutolock _l(fLock);
        item = list->FindItemByID(id);
    }

    if( !item ) {
        item = load_resource(type, id, 0, &list);
    }

    return item;
}

TypeItem* BResourceSet::find_item_name(type_code type, const char* name)
{
    TypeList* list = find_type_list(type);
    TypeItem* item = 0;

    if( list ) {
        BAutolock _l(fLock);
        item = list->FindItemByName(name);
    }

    if( !item ) {
        item = load_resource(type, -1, name, &list);
    }
}

```

```

    }

    return item;
}

TypeItem* BResourceSet::load_resource(type_code type, int32 id,
                                      const char* name,
                                      TypeList** inout_list)
{
    TypeItem* item = 0;

    if( name ) {
        BEntry entry;

        // If a named resource, first look in directories. Keep
        // the object locked while looking, so nobody tramples us.
        fLock.Lock();
        for( int32 i=0; item == 0 && i<fDirectories.CountItems(); i++ )
        {
            BPath* dir = (BPath*)fDirectories.ItemAt(i);
            if( dir ) {
                fLock.Unlock();
                // For each directory, try to open the named file.
                // We unlock the resource set at this point so that
                // other threads are not blocked while we do a
                // possibly length operation.
                BPath path(dir->Path(), name);
                if( entry.SetTo(path.Path(), true) == B_OK ) {
                    BFile file(&entry, B_READ_ONLY);
                    if( file.InitCheck() == B_OK ) {
                        item = new TypeItem(id, name, &file);
                        item->SetSourceIsFile(true);
                    }
                }
                fLock.Lock();
            }
        }
        fLock.Unlock();
    }
}

#if USE_RESOURCES
if( !item ) {
    // Look through resource objects for data. Keep
    // the object locked while looking, so nobody tramples us.
    fLock.Lock();
    for( int32 i=0; item == 0 && i<fResources.CountItems(); i++ ) {
        BResources* r = (BResources*)fResources.ItemAt(i);
        if( r ) {
            // Note we DON'T unlock here, because BResources is
            // not thread safe (though it -really- should be).
            const void* data = 0;
            size_t size = 0;
            if( id >= 0 ) {
                data = r->LoadResource(type, id, &size);
            } else if( name != 0 ) {
                data = r->LoadResource(type, name, &size);
            }
            if( data && size ) {
                item = new TypeItem(id, name, data, size);
            }
        }
    }
}
#endif

```

```

        item->SetSourceIsFile(false);
    }
}
fLock.Unlock();
}
#endif

if( item ) {
    BAutolock l(fLock);

    TypeList* list = inout_list ? *inout_list : 0;
    if( !list ) {
        // Don't currently have a list for this type -- check if
there is
        // already one.
        list = find_type_list(type);
    }

    if( !list ) {
        // Need to make a new list for this type.
        list = new TypeList(type);
        fTypes.AddItem(list);
        list->AddItem(item);
    } else {
        // Check to make sure someone else hasn't already added this
        // item.
        TypeItem* existing = name ? list->FindItemByName(name)
                                : list->FindItemByID(id);

        if( existing ) {
            // This one has already been added. Delete what we went
            // to all the work to make, and return the other.
*sniff*
            delete item;
            item = existing;
        } else {
            list->AddItem(item);
        }
    }

    if( inout_list ) *inout_list = list;
}

return item;
}

BBitmap* BResourceSet::return_bitmap_item(type_code type, TypeItem*
from)
{
    (void)type;
    if( !from ) return 0;

    TypeObject* obj = from->Object();
    BitmapObject* bm = dynamic_cast<BitmapObject*>(obj);
    if( bm ) return bm;

    // Can't change an existing object.
    if( obj ) return 0;
}

```

```

// Note that we are creating the bitmap without the resource set
// being locked, since it is a potentially expensive operations.
// We thus need to deal with the situation where someone else
// created this at the same time, and got it added before us.
bm = GenerateBitmap(from->Data(), from->Size());

if( bm ) {
    BAutolock l(fLock);
    if( from->Object() != 0 ) {
        // Whoops! Someone snuck in under us.
        bm->Delete();
        bm = dynamic_cast<BitmapObject*>(from->Object());
    } else {
        from->SetObject(bm);
    }
}

return bm;
}

enum {
    PNG_CHECK_BYTES = 8
};

static void read_png_data(png_structp png_ptr,
                          png_bytep data,
                          png_uint_32 length)
{
    BDataIO* io = (BDataIO*)png_get_io_ptr(png_ptr);
    ssize_t amount = io->Read(data, length);
    if( amount < B_OK ) png_error(png_ptr, "Read Error");
}

BResourceSet::BitmapObject* BResourceSet::read_png_image(BDataIO*
stream)
{
    png_byte buf[PNG_CHECK_BYTES];

    // Make sure there is a header
    if( stream->Read(buf, PNG_CHECK_BYTES) != PNG_CHECK_BYTES ) return
0;

    // Check the header
    if( png_sig_cmp(buf, 0, PNG_CHECK_BYTES) != 0 ) return 0;

    png_structp png_ptr;
    png_infop info_ptr;

    png_ptr = png_create_read_struct(PNG_LIBPNG_VER_STRING, NULL, NULL,
NULL);
    if (!png_ptr)
        return 0;

    info_ptr = png_create_info_struct(png_ptr);
    if (!info_ptr) {
        png_destroy_read_struct(&png_ptr, (png_infopp)NULL,
(png_infopp) NULL);
        return 0;
    }
}

```

```

    }

    if (setjmp(png_ptr->jmpbuf)) {
        png_destroy_read_struct(&png_ptr, &info_ptr, (png_infopp)
NULL);
        return 0;
    }

    png_set_read_fn(png_ptr, stream, read_png_data);
    png_set_sig_bytes(png_ptr, PNG_CHECK_BYTES);

    png_read_info(png_ptr, info_ptr);

    // Set up color space conversion
    png_byte color_type(png_get_color_type(png_ptr, info_ptr));
    png_byte bit_depth(png_get_bit_depth(png_ptr, info_ptr));

    if (bit_depth <= 8) {
        png_set_expand(png_ptr);
        png_set_packing(png_ptr);
    }

    if (png_get_valid(png_ptr, info_ptr, PNG_INFO_tRNS))
        png_set_expand(png_ptr);

    if (bit_depth > 8)
        png_set_strip_16(png_ptr);

    png_set_bgr(png_ptr);

    if (!(color_type & PNG_COLOR_MASK_COLOR))
        png_set_gray_to_rgb(png_ptr);

    if (!(color_type & PNG_COLOR_MASK_ALPHA))
        png_set_filler(png_ptr, 0xFF, PNG_FILLER_AFTER);

    png_read_update_info(png_ptr, info_ptr);
    png_uint_32 width = png_get_image_width(png_ptr, info_ptr);
    png_uint_32 height = png_get_image_height(png_ptr, info_ptr);

    // Create bitmap
    BitmapObject* bitmap =
        new BitmapObject(BRect(0, 0, width-1, height-1), 0, B_RGBA32);

    png_bytep* rows = new png_bytep[height];
    for( size_t i=0; i<height; i++ ) {
        rows[i] = ((png_bytep)bitmap->Bits()) + i*bitmap->BytesPerRow
());
    }
    png_read_image(png_ptr, rows);

    png_destroy_read_struct(&png_ptr, (png_infopp)NULL, (png_infopp)
NULL);

    return bitmap;
}

BResourceSet::BitmapObject*
BResourceSet::GenerateBitmap(const void* data, size_t size)

```

```

{
    // Don't have a bitmap in the item -- we'll try to create one.
    BMemoryIO stream(data, size);

    // First, see if it can be read as a png image.
    BitmapObject* bm = read_png_image(&stream);

    if( !bm ) {
        // Well, maybe it's an archived bitmap.
        stream.Seek(0, SEEK_SET);
        BMessage archive;
        if( archive.Unflatten(&stream) == B_OK ) {
            bm = new BitmapObject(&archive);
            if( bm && bm->InitCheck() != B_OK ) {
                bm->Delete();
            }
        }
    }

    return bm;
}

#if HAVE_CURSORS
BCursor* BResourceSet::return_cursor_item(TypeItem* from)
{
    if( !from ) return 0;

    TypeObject* obj = from->Object();
    CursorObject* cr = dynamic_cast<CursorObject*>(obj);
    if( cr ) return cr;

    // Can't change an existing object.
    if( obj ) return 0;

    // Don't have a cursor in the item -- we'll create one.
    // Note that we are creating the cursor without the resource set
    // being locked, since it is a potentially expensive operations.
    // We thus need to deal with the situation where someone else
    // created this at the same time, and got it added before us.
    cr = GenerateCursor(from->Data(), from->Size());

    if( cr ) {
        BAutolock l(fLock);
        if( from->Object() != 0 ) {
            // Whoops! Someone snuck in under us.
            cr->Delete();
            cr = dynamic_cast<CursorObject*>(from->Object());
        } else {
            from->SetObject(cr);
        }
    }

    return cr;
}

BResourceSet::CursorObject*
BResourceSet::GenerateCursor(const void* data, size_t size)
{
    return new CursorObject(data);
}

```



```

}
#endif

BMessage* BResourceSet::return_message_item(TypeItem* from)
{
    if( !from ) return 0;

    TypeObject* obj = from->Object();
    MessageObject* ms = dynamic_cast<MessageObject*>(obj);
    if( ms ) return ms;

    // Can't change an existing object.
    if( obj ) return 0;

    // Note that we are creating the message without the resource set
    // being locked, since it is a potentially expensive operations.
    // We thus need to deal with the situation where someone else
    // created this at the same time, and got it added before us.
    ms = GenerateMessage(from->Data(), from->Size());

    if( ms ) {
        BAutolock l(fLock);
        if( from->Object() != 0 ) {
            // Whoops!
            Someone snuck in under us.
            ms->Delete();
            ms = dynamic_cast<MessageObject*>(from->Object());
        } else {
            from->SetObject(ms);
        }
    }

    return ms;
}

BResourceSet::MessageObject*
BResourceSet::GenerateMessage(const void* data, size_t size)
{
    MessageObject* ms = new MessageObject();
    if (ms->Unflatten((const char*)data) != B_OK) {
        ms->Delete();
        ms = NULL;
    }
    return ms;
}

```

Filename: SocketList.h

```
#ifndef _SOCKET_LIST_VIEW
#define _SOCKET_LIST_VIEW
#include <InterfaceKit.h>
#include <Locker.h>
#include "Icons.h"
#include "ChannelListItem.h"
#include "ChannelList.h"

class SocketListItem : public ChannelListItem {
public:
    //for items
    SocketListItem(const char *name, AuroranComponent * owner,
BHandler * handler, BMessage * message, short int flags, uint32 level);
    ~SocketListItem();
    bool isSocket();

    //hooks
    virtual void DrawItem(BView * owner, BRect itemRect, bool
drawEverything = false);
    bool isCompatable(ChannelListItem * who);
    int SetValue(CHData value);
    CHData GetValue();
    CHData GetValueAndClean();
    bool isAvailable() {return Available;}
    void setAvailable(bool a) {Available=a;}

    //new stuff
    bool isDirty() {return Dirty;}

    //this lets the program set up a plug to automatically forward
values to. Basically it enables a sort of swithing fabric
    void StartForwarding(ChannelListItem * NextHop);
    void StopForwarding(ChannelListItem * NextHop);
    void StopForwarding();

    //Enumeration functions
    BList * GetEnums();
    Enum * GetEnum(CHData value);
    void ResetEnums();
    void AddEnum(CHData value, char * name);
    void RemoveEnum(Enum * who);
    void RemoveEnum(CHData value) {RemoveEnum(GetEnum(value));}

private:
    BMessage * msg;
    BHandler * recv;
    bool Dirty;
    bool Available;
    BLocker DirtyLock;
    BMessenger * MyMessenger;
    ChannelListItem * ForwardingDest;
    BList Enums;
};

#endif
```

Filename: SocketList.cpp

```
#include "SocketList.h"
#include <TranslationKit.h>
#include "Component.h"
#include <iostream.h>
extern IconCollection Icons;

//IconSet SocketListItem::Icons={NULL,NULL,NULL,NULL,NULL,NULL};

SocketListItem::SocketListItem(const char *name, AuroranComponent *
owner, BHandler * handler, BMessage * message, short int flags,uint32
level) : ChannelListItem(name,owner,level),Dirty(false) {
    MyFlags=flags;
    ForwardingDest=NULL;
    msg=message;
    if (msg) msg->AddPointer("SocketPtr",this);
    //cout << "This: "<< this << endl;
    recv=handler;
    if (msg) MyMessenger = new BMessenger(handler);
}

SocketListItem::~SocketListItem() {
    if (msg) delete MyMessenger;
}

bool SocketListItem::isCompatable(ChannelListItem * who) {
    return who->isPlug();
}

bool SocketListItem::isSocket() {
    return true;
}

void SocketListItem::DrawItem(BView * owner, BRect itemRect, bool
drawEverything) {
    ChannelListItem::DrawItem(owner,itemRect,drawEverything);

    // if (isSocket) {
    /*
        owner->SetDrawingMode(B_OP_ALPHA);

        if ((MyFlags & Send_Any) == Send_Any) {
            owner->DrawBitmap(Icons.Icon_Any,itemRect.RightTop()+BPoint
(-30,(itemRect.Height()-10)/2));
        } else if (MyFlags & Send_Wave) {
            owner->DrawBitmap(Icons.Icon.Wave,itemRect.RightTop()
+BPoint(-30,(itemRect.Height()-10)/2));
        } else if (MyFlags & Send_Trigger) {
            owner->DrawBitmap(Icons.Icon.Pulse,itemRect.RightTop()
+BPoint(-30,(itemRect.Height()-10)/2));
        } else if (MyFlags & Send_Flat) {
            owner->DrawBitmap(Icons.Icon.Flat,itemRect.RightTop()
+BPoint(-30,(itemRect.Height()-10)/2));
        }
    }
}
```

```

        if ((MyFlags & Recv_Any)==Recv_Any) {
            owner->DrawBitmap(Icons.Icon.Any,itemRect.RightTop()+BPoint
(-13,(itemRect.Height()-10)/2));
        } else if (MyFlags & Recv_Now) {
            owner->DrawBitmap(Icons.Icon.Bolt,itemRect.RightTop()
+BPoint(-13,(itemRect.Height()-10)/2));
        } else if (MyFlags & Recv_Later) {
            owner->DrawBitmap(Icons.Icon.Clock,itemRect.RightTop()
+BPoint(-13,(itemRect.Height()-10)/2));
        }

        owner->SetDrawingMode(B_OP_COPY);
*/
// }

}

CHData SocketListItem::GetValue() {
    return LatestValue;
}

CHData SocketListItem::GetValueAndClean() {
    DirtyLock.Lock();
    Dirty=false;
    DirtyLock.Unlock();
    return LatestValue;
}

int SocketListItem::SetValue(CHData value) {

//    cout << "Trying to send new data:" << value.f << endl;
    LatestValue=value;

    if (ForwardingDest) { //forward packet immediately if requested
        ForwardingDest->SetValue(value);
    }

    if (msg) {

        DirtyLock.Lock();
        if (!Dirty || !(MyFlags & Recv_Dirty)) {
            //send BMessage to destination
            //cout << "Sending new data:" << value.f << endl;
            BMessage * MyMessage = new BMessage(*msg);
            MyMessage->AddFloat("value",value.f);
            MyMessenger->SendMessage(MyMessage);
            delete MyMessage;
        }
        Dirty=true;
        DirtyLock.Unlock();
    }
    return 0;
}

void SocketListItem::StartForwarding(ChannelListItem * NextHop) {
    ForwardingDest=NextHop;
}

void SocketListItem::StopForwarding(ChannelListItem * NextHop) {

```

```

        if (ForwardingDest==NextHop) ForwardingDest=NULL;
    }
void SocketListItem::StopForwarding() {
    ForwardingDest=NULL;
}

BList * SocketListItem::GetEnums() {return &Enums;}
Enum * SocketListItem::GetEnum(CHData value) {

    for (int i = Enums.CountItems()-1;i>=0;i--) {
        if ((MyFlags & Chan_Float) == Chan_Float) {
            if (((Enum*)Enums.ItemAt(i))->Value.f < value.f) return
(Enum*)Enums.ItemAt(i);
        } else if ((MyFlags & Chan_Int) == Chan_Int) {
            if (((Enum*)Enums.ItemAt(i))->Value.i < value.i) return
(Enum*)Enums.ItemAt(i);
        }
    }

}

void SocketListItem::ResetEnums() {
    while (Enums.RemoveItem(Enums.FirstItem())) {}
}

void SocketListItem::AddEnum(CHData value, char * name) {
    Enum * NewEnum = new Enum();
    NewEnum->Value=value;
    strncpy(NewEnum->Name,name,DEFAULT_NAME_SIZE);
    NewEnum->Name[DEFAULT_NAME_SIZE-1]=0; //just in case Name >= 32
bytes
    Enums.AddItem(NewEnum,Enums.IndexOf(GetEnum(value))+1);
//    Enums->
}

void SocketListItem::RemoveEnum(Enum * who) {
    Enums.RemoveItem(who);
}

```

Filename: SuperButton.h

```
#include <Button.h>

class SuperButton: public BButton {
public:
    SuperButton(BRect frame, const char *name, const char *label,
        BMessage *message, uint32 resizingMode = B_FOLLOW_LEFT | B_FOLLOW_TOP,
        uint32 flags = B_WILL_DRAW | B_NAVIGABLE): BButton(frame, name, label,
        message, resizingMode, flags) {}
    virtual status_t Invoke(BMessage *message = NULL);
};
```

Filename: SuperButton.cpp

```
#include "SuperButton.h"
#include "Window.h"
#include "iostream.h"

status_t SuperButton::Invoke(BMessage * message) {
    if (message==NULL) {
        message = Message();
    }
    BMessage* msg = Window()->CurrentMessage();
    int32 mods;
    msg->FindInt32("modifiers",&mods);
    message->RemoveName("modifiers");
    message->AddInt32("modifiers",mods);
    return BButton::Invoke(message);
}
```

Filename: ThreadPrimitive.h

```
#ifndef Threads_h //Prevent Duplicates
#define Threads_h

#include <scheduler.h>
//#include <stdio.h>

const int32 THREAD_LOW = 5;
const int32 THREAD_NORM = 20;
const int32 THREAD_HIGH = 50;

class ThreadPrimitive {
public:

    ThreadPrimitive(int32 priority = B_LOW_PRIORITY,
        const char *name = 0)
        :   scanThread(-1),
            priority(priority),
            name(name)
        {}

    virtual ~ThreadPrimitive()
    {
        //printf("Killing thread %s...\n");
        kill=true;
        status_t x;
        if (wait_for_thread(scanThread,&x)==B_INTERRUPTED) {
            kill_thread(scanThread);
        }
        //printf("...killed.\n");
    }

    void Go()
    {
        kill=false;
        scanThread = spawn_thread(&ThreadPrimitive::RunBinder,
            name ? name : "UntitledThread", priority, this);
        resume_thread(scanThread);
    }

    virtual void RunThread() = 0;

private:

    static status_t RunBinder(void *castToThis)
    {
        // In this call we do the dirty casting work, making
        // the rest of the interfaces fully typed and clean

        ThreadPrimitive *self = (ThreadPrimitive *)castToThis;
        self->RunThread();

        return B_OK;
    }
}
```



```
protected:

    bool kill;
    thread_id scanThread;
    int32 priority;

private:
    const char *name; // only valid in the constructor and in
                      // the Go call
};

#endif //Threads_h
```

Filename: Triangle.h

```
#include <Control.h>
#include <Shape.h>

class BTriangle : public BControl {
public:
    BTriangle(BRect frame, const char *name, BMessage *message,
uint32 resizingMode=B_FOLLOW_LEFT | B_FOLLOW_TOP, uint32 flags =
B_WILL_DRAW | B_NAVIGABLE);
    virtual void Draw(BRect updateRect);
    virtual void MouseDown(BPoint point);
    virtual void MouseUp(BPoint point);
    virtual void MouseMoved(BPoint pt, uint32 code, const BMessage
*msg);
private:
    BRect MyFrame;
    int Position;
    BShape Tri1;
    BShape Tri2;
    BShape Tri3;
};
```

Filename: Triangle.cpp

```
#include "Triangle.h"
#include <stdio.h>
#include <Window.h>
#include <iostream.h>

BTriangle::BTriangle(BRect frame, const char *name, BMessage *message,
uint32 resizingMode,uint32 flags) : BControl(frame,name,"",message,
resizingMode,flags) {
    Position=0;
    SetValue(0);
    Tri3.MoveTo(BPoint(frame.Width()/3,0));
    Tri3.LineTo(BPoint(frame.Width(),frame.Height()/2));
    Tri3.LineTo(BPoint(frame.Width()/3,frame.Height()));
    Tri3.LineTo(BPoint(frame.Width()/3,0));
    Tri3.Close();

    Tri2.MoveTo(BPoint(0,frame.Height()/6));
    Tri2.LineTo(BPoint(frame.Width()/2,5*frame.Height()/6));
    Tri2.LineTo(BPoint(frame.Width(),frame.Height()/6));
    Tri2.LineTo(BPoint(0,frame.Height()/6));
    Tri2.Close();

    Tri1.MoveTo(BPoint(2*frame.Width()/3,0));
    Tri1.LineTo(BPoint(0,frame.Height()/2));
    Tri1.LineTo(BPoint(2*frame.Width()/3,frame.Height()));
    Tri1.LineTo(BPoint(2*frame.Width()/3,0));
    Tri1.Close();

    Message()->AddBool("TriPosition",Value()!=0);
}

void BTriangle::Draw(BRect updateRect) {
/* if (Position==1) {
    int32 buttons=0;
    Window()->CurrentMessage()->FindInt32("buttons", &buttons);
    if ((buttons & 1)==0) {Position=(Value())?2:0;}
}*/
SetDrawingMode(B_OP_COPY);
SetLowColor(ViewColor());
FillRect(updateRect,B_SOLID_LOW);

//printf("Redraw: %d (%f,%f,%f,%f)\n",Position,updateRect.left,
updateRect.top,updateRect.right,updateRect.bottom);
SetLowColor((rgb_color){128,128,128,255});
SetHighColor((rgb_color){0,0,0,255});
if (Position==0) {
    FillShape(&Tri1,B_SOLID_LOW);
    StrokeShape(&Tri1);
} else if (Position==1) {
    SetLowColor((rgb_color){96,96,96,255});
    FillShape(&Tri2,B_SOLID_LOW);
    StrokeShape(&Tri2);
} else {
```

```

        FillShape(&Tri3,B_SOLID_LOW);
        StrokeShape(&Tri3);
    }
}

void BTriangle::MouseDown(BPoint point) {
    Position=1;
    Draw(BRect(0,0,Frame().Width(),Frame().Height()));
}

void BTriangle::MouseUp(BPoint point) {
    SetValue(1-Value()); //toggle current value
    Position=(Value())?2:0;
    Message()->RemoveName("TriPosition");
    Message()->AddBool("TriPosition",Value()!=0);
    Draw(BRect(0,0,Frame().Width(),Frame().Height()));
    Invoke();
}

void BTriangle::MouseMoved(BPoint pt, uint32 code, const BMessage *msg)
{
    int32 buttons=0;
    Window()->CurrentMessage()->FindInt32("buttons", &buttons);
    if (BRect(0,0,Frame().Width(),Frame().Height()).Contains(pt) &&
(buttons & 1)) {
        if (Position!=1) {
            Position=1;
            Draw(BRect(0,0,Frame().Width(),Frame().Height()));
        }
    } else {
        if (Position==1) {
            Position=(Value())?2:0;
            Draw(BRect(0,0,Frame().Width(),Frame().Height()));
        }
    }
    BControl::MouseMoved(pt,code,msg);
    //if (msg!=NULL) {msg->PrintToStream();}
}

```

Filename: app.h

```
/*
  Project: Aurora
  File:    app.h
  Purpose: Set up main application object, start things running
*/

#include <Application.h>

class AuroraApp : public BApplication {
public:
    AuroraApp(); //default constructor (duh)
    virtual void      MessageReceived(
                        BMessage *message);
    virtual void AboutRequested();
};
```

Filename: app.cpp

```
/*
    Project: Aurora
    File:    app.cpp
    Purpose: Set up main application object, start things running
*/

#include "app.h"
#include "ConfigAndPatchWindow.h"
#include "Alert.h"
#include "AboutBox.h"

AuroraApp::AuroraApp() : BApplication("application/x-vnd.
AuroranLights") {

    ConfigAndPatchWindow *MyWindow = new ConfigAndPatchWindow();
    MyWindow->Show();

}

void AuroraApp::MessageReceived(BMessage *message) {
    switch (message->what)
    {
        default:
            BApplication::MessageReceived(message);
    }
}

void AuroraApp::AboutRequested() {
    ShowAboutBox();
}
```

Filename: colorspace.h

```
#ifndef h_A_colorspace
#define h_A_colorspace

#include <GraphicsDefs.h>

class rgb_fcolor;

class hsv_fcolor {
public:
    float h;
    float s;
    float v;
    hsv_fcolor() : h(0), s(0), v(0) {};
    hsv_fcolor(float h, float s, float v) : h(h), s(s), v(v) {};
    hsv_fcolor(rgb_fcolor c);
};

class rgb_fcolor {
public:
    float r;
    float g;
    float b;
    rgb_fcolor(): r(0), g(0), b(0) {};
    rgb_fcolor(float r, float g, float b): r(r), g(g), b(b) {};
    rgb_fcolor(rgb_color c) : r(c.red/255.0), g(c.green/255.0), b(c.
blue/255.0) {};
    rgb_fcolor(hsv_fcolor c);
    rgb_color rgb() {return (rgb_color){floor(r*255),floor(g*255),floor
(b*255),0.0};}
};

float ColorMatch(float h1,float s1,float h2,float s2);

#endif
```

Filename: colorspace.cpp

```
#include <math.h>
#include "colorspace.h"
#include <stdio.h>

float maximum(float a, float b, float c) {
    if (a >= b && a >= c) return a;
    if (b >= a && b >= c) return b;
    return c;
}

float minimum(float a, float b, float c) {
    if (a <= b && a <= c) return a;
    if (b <= a && b <= c) return b;
    return c;
}

//rgb_color rgb_fcolor::rgb() {
//    return rgb_color{
//}

//http://www.cs.fit.edu/wds/classes/cse5255/cse5255/davis/rgbhsv.html
hsv_fcolor::hsv_fcolor(rgb_fcolor c) {

    printf("Converting from RGB: %f / %f / %f\n",c.r,c.g,c.b);

    float max = maximum(c.r,c.g,c.b);
    float min = minimum(c.r,c.g,c.b);
    float delta = max-min;

    v=max;

    if (max>0) s = delta/max;
    else s = 0;

    if (delta==0) h = 0;
    else {
        if (c.r==max) {
            h=(c.g-c.b)/delta;
        } else if (c.g==max) {
            h=2+(c.b-c.r)/delta;
        } else {
            h = 4+(c.r-c.g)/delta;
        }
        h*=60;
        if (h<0.0) h=h+360;
    }
    h=h*M_PI/180.0;

    printf(" --> to HSV: : %f / %f / %f\n",h,s,v);
}

//http://www.cs.fit.edu/wds/classes/cse5255/cse5255/davis/hsvrgb.html
rgb_fcolor::rgb_fcolor(hsv_fcolor c) {
```



```

c.h*=180.0/M_PI;
while (c.h>(120*2+60)) c.h=c.h-360;
while (c.h<(-60)) c.h=c.h+360;
//printf("Angle: %f\n",c.h);
if (c.h>(-60.0) && c.h <60) { //red is dominant
    r=1.0; //red is dominant
    if (c.h<0) {
        g=1.0-c.s; //green is darkest
        b=g-(c.h/60.0)*c.s;
    } else {
        b=1.0-c.s; //green is darkest
        g=(c.h/60.0)*c.s+b;
    }
} else if (c.h>=60 && c.h < 180) {
    g=1.0; //green is dominant
    c.h-=120.0; //shift things around
    if (c.h<0) {
        b=1.0-c.s; //blue is darkest
        r=b-(c.h/60.0)*c.s;
    } else {
        r=1.0-c.s; //red is darkest
        b=(c.h/60.0)*c.s+r;
    }
} else {
    b=1.0; //blue is dominant
    c.h-=240.0; //shift things around
    if (c.h<0) {
        r=1.0-c.s; //red is darkest
        g=r-(c.h/60.0)*c.s;
    } else {
        g=1.0-c.s; //green is darkest
        r=(c.h/60.0)*c.s+g;
    }
}

//scale brightness accordingly
r*=c.v;
g*=c.v;
b*=c.v;
}

float ColorMatch(float h1,float s1,float h2,float s2) {
    float x1,x2,y1,y2;

    x1=cos(h1)*(s1);
    x2=cos(h2)*(s2);

    y1=sin(h1)*(s1);
    y2=sin(h2)*(s2);

    // h1=h1/(2*M_PI);
    // h2=h2/(2*M_PI);

    // printf("H1: %4.1f H2: %4.1f\n",h1*360,h2*360);
    // printf("S1: %4.1f S2: %4.1f\n",s1,s2);
    // float HueDiff=fabs(h1-h2);
    // if (HueDiff > 0.5) HueDiff=1.0-HueDiff;
    return (sqrt((x1-x2)*(x1-x2)+(y1-y2)*(y1-y2)))/2.0; //HueDiff*sqrt

```

```
((1.0-s1)*(1.0-s2));  
}
```

Filename: main.cpp

```
/*
    Project: Aurora
    File:    main.cpp
    Purpose: Set up main application object, start things running
*/

#include <iostream.h>
#include "app.h"
#include "AuroraGeneral.h"
BList Components;

int main() {
    cerr << "Starting Aurora..." << endl;

    new AuroraApp();

    be_app->Run();

    delete be_app;

    return 0;
}
```

Filename: AudioComponent.h

```
#ifndef AC_Inc
#define AC_Inc

#include "Component.h"
#include "AudioListener.h"

class AudioComponent : public AuroranComponent {
private:
    AudioListener * MyListener;

public:
    AudioComponent(AuroranParameters * Parms) ;
    ~AudioComponent();

    //we have no interface...
    virtual void InterfaceMessage(BMessage * message);

    //maybe will register with a pulser...
    virtual void MessageReceived(BMessage * message);
    virtual ComponentDescriptor * GetDesc();

    void SendData(unsigned char CH, float value);
protected:
    virtual char * MyName();
    virtual char * DefaultCustom();
};

#endif
```

Filename: AudioComponent.cpp

```
#include "Component.h"
#include "ResourceSet.h"
#include <stdio.h>
#include <iostream.h>
#include "PrettySlider.h"
#include "AudioComponent.h"

//*****
//this resource management stuff is taken from one issue of the
BMessage newsletter...
//http://www.be.com/developers/bmessage/issue09.html
#include <Locker.h>
#include <TranslatorFormats.h>

static int32 gResourcesCreated = 0;
static BResourceSet gResources;

static BResourceSet& Resources()
{
    // Return the BResourceSet object for this image, initializing
    // if needed.
    if ((gResourcesCreated&2) != 0) {
        // Quick return if already initialized.
        return gResources;
    } else if (atomic_or(&gResourcesCreated, 1) == 0) {
        // Not yet initialized -- do so.
        gResources.AddResources(&gResourcesCreated);
        // Mark that we are done.
        atomic_or(&gResourcesCreated, 2);
    } else {
        // Wait for resources to be initialized.
        while ((gResourcesCreated&2) == 0) snooze(50000);
    }
    return gResources;
}

//*****

AuroranComponent * MakeComponent(ComponentDescriptor * Who,
AuroranParameters * Params);

static ComponentDescriptor MyDescriptor = {
    A_DESC_VER,
    A_NORMAL_DESC,
    NULL,
    "Audio Processor",
    'TKS_',
    'AI',
    0,
    ACC_INPUT,
    &MakeComponent
};

AudioComponent::AudioComponent(AuroranParameters * Params) :
AuroranComponent(Params, DefaultCustom()) {
```

```

        //Set up how the button for us will look
        HighlightBG=(rgb_color){80,240,120,128};
        NormalBG= (rgb_color){60,200,100,128};
        MyIcon=(BBitmap *)Resources().FindBitmap(B_PNG_FORMAT,
2001);

        //register our one plug - Pk-Pk
        RegisterPlug("RMS",StdChan_Intensity);
        RegisterPlug("Low7",StdChan_Intensity);
        RegisterPlug("Low6",StdChan_Intensity);
        RegisterPlug("Low5",StdChan_Intensity);
        RegisterPlug("Mid4",StdChan_Intensity);
        RegisterPlug("Mid3",StdChan_Intensity);
        RegisterPlug("Hi2",StdChan_Intensity);
        RegisterPlug("Hi",StdChan_Intensity);
        RegisterPlug("Hi0",StdChan_Intensity);
        RegisterPlug("Peak256",StdChan_Intensity);
        RegisterPlug("Peak10",StdChan_Intensity);
        RegisterPlug("WeightPeak256",StdChan_Intensity);
        RegisterPlug("WeightPeak10",StdChan_Intensity);
        RegisterPlug("Beat",StdChan_Intensity);
        RegisterPlug("Volume",StdChan_Intensity);
        BMediaRoster::Roster();
        MyListener=new AudioListener("Aurora Audio");
        MyListener->SetOwner(this);
    }
    AudioComponent::~~AudioComponent() {delete MyListener;printf
("time to sleep\n");}

    //we have no interface...
    void AudioComponent::InterfaceMessage(BMessage * message) {
        switch (message->what) {
            default:
                AuroranComponent::InterfaceMessage(message);
        }
    }

    //maybe will register with a pulser...
    void AudioComponent::MessageReceived(BMessage * message) {
        switch (message->what) {
        }
        AuroranComponent::MessageReceived(message);
    }

    void AudioComponent::SendData(unsigned char CH, float value) {
        ((PlugListItem *)MyPlugs.ItemAt(CH))->SendMessage(value);
    }

    ComponentDescriptor * AudioComponent::GetDesc() {return
&MyDescriptor;}
    char * AudioComponent::MyName() {return "Audio Processor";}
    char * AudioComponent::DefaultCustom() {return "AudioIn";}

    AuroranComponent * MakeComponent(ComponentDescriptor * Who,
    AuroranParameters * Params) {return new AudioComponent(Params);}

```

```
extern "C" _EXPORT ComponentDescriptor* GetDesc();  
ComponentDescriptor* GetDesc() {  
    return &MyDescriptor;  
}
```

Filename: AudioListener.h

```

#ifndef AL_Inc
#define AL_Inc

#include <MediaKit.h>

class AudioListener: public BBufferConsumer, public BMediaEventLooper {

private:
    media_input mInput;           // descriptor of our single input
    bigtime_t mSchedulingLatency; // our scheduling latency
    (estimated at run time)
    long mLateBuffers;           // track how many late buffers
    we've gotten
    int32 mProducerDataStatus;    // whether our producer has data
    available

    // controllable parameters and their change history
    bigtime_t mLatency;           // our
    internal latency
    float mSpinPercentage;        // how much
    of our latency time to spin the CPU
    int32 mPriority;               //
    our control thread's priority
    bigtime_t mLastLatencyChange; // when did we last
    change our latency?
    bigtime_t mLastSpinChange;    // when did we
    last change our CPU usage?
    bigtime_t mLastPrioChange;    // when did we
    last change thread priority?

    void resetInput();            //reset the input
protected:
    void * Owner;
    char BufferData[512];
    int BufferPosition;
public:
    AudioListener(const char * Name="AudioSpy");
    ~AudioListener();
    virtual void ProcessBuffer();
    virtual void InitializeStuff();
    void SetOwner(void * x) {Owner=x;}

    //BMediaNode Stuff
    BMediaAddOn* AddOn(int32 *) const;
    void SetRunMode(run_mode mode);
    void Preroll();
    void SetTimeSource(BTimeSource* time_source);
    status_t RequestCompleted(const media_request_info &info);

    //BBufferConsumer Stuff
    status_t AcceptFormat(const media_destination &dest,
        media_format* format);
    status_t GetNextInput(int32* cookie, media_input* out_input);
    void DisposeInputCookie(int32);
    void BufferReceived(BBuffer* buffer);

```



```

    void ProducerDataStatus(const media_destination & for_whom, int32
status, bigtime_t at_performance_time);
    status_t GetLatencyFor(const media_destination& for_whom, bigtime_t*
out_latency, media_node_id* out_timesource);
    status_t Connected(
        const media_source& producer,
        const media_destination& where,
        const media_format& with_format,
        media_input* out_input);
    void Disconnected(
        const media_source& producer,
        const media_destination& where);
    status_t FormatChanged(
        const media_source& producer,
        const media_destination& consumer,
        int32 change_tag,
        const media_format& format);

    //BMediaEventLooper Stuff
    void NodeRegistered();
    void HandleEvent(const media_timed_event *event, bigtime_t /*
lateness */, bool /* realTimeEvent */);

};

#endif

```

Filename: AudioListener.cpp

```
#include <iostream.h>
#include "AudioListener.h"
#include <be/media/MediaDefs.h>

//Based on Be's Logging Consumer
AudioListener::AudioListener(const char * Name)
    :   BMediaNode(Name),
        BBufferConsumer(B_MEDIA_RAW_AUDIO),
        BMediaEventLooper(),
        mLateBuffers(0),
        mLatency(50 * 1000),           // default to 50 milliseconds
        mSpinPercentage(0.10),       // default to spinning 10% of total
latency
        mPriority(B_REAL_TIME_PRIORITY),
        mLastLatencyChange(0),
        mLastSpinChange(0),
        mLastPrioChange(0),
        Owner(NULL)
    {
        InitializeStuff();

        resetInput();
        BMediaRoster * rstr = BMediaRoster::Roster();

        rstr->RegisterNode(this);
        //media_node clone;
        //cout << "getting node..." << rstr->GetNodeFor(Node().node, &clone)
<< endl;
        //cout << "setting run mode..." << rstr->SetRunModeNode(clone,
B_RECORDING) << endl;
    }

AudioListener::~AudioListener() {
    BMediaRoster::Roster()->UnregisterNode(this);
    BMediaEventLooper::Quit();
}

void AudioListener::resetInput() {
    media_input x;
    mInput=x;
}

//BMediaNode specific stuff
//*****
BMediaAddOn* AudioListener::AddOn(int32 *) const {
    return NULL;
}

void AudioListener::SetRunMode(run_mode mode) {
    // The BMediaEventLooper tracks our run mode for us
    BMediaEventLooper::SetRunMode(mode);
}

void AudioListener::Preroll() {
```

```

        BMediaEventLooper::Preroll();
    }

void AudioListener::SetTimeSource(BTimeSource* time_source) {
    BMediaNode::SetTimeSource(time_source);
}

status_t AudioListener::RequestCompleted(const media_request_info
&info) {
    return BMediaNode::RequestCompleted(info);
}

//BBufferConsumer Specific Stuff...
//*****
status_t AudioListener::AcceptFormat(const media_destination &dest,
    media_format* format) {

    cout << "Format check requested for type "<<format->type <<
endl;
    cout << "Raw Audio CH count:"<<format->u.raw_audio.
channel_count << endl;
    cout << "Raw Audio Frame Rate:"<<format->u.raw_audio.frame_rate
<< endl;
    cout << "Raw Audio buffer_size:"<<format->u.raw_audio.
buffer_size << endl;
    cout << "Raw Audio format:"<<format->u.raw_audio.format <<
endl;

    //dest must be valid
    if (dest != mInput.destination) return B_MEDIA_BAD_DESTINATION;

    //must bve audio or wildcard
    if( format->type != B_MEDIA_UNKNOWN_TYPE && format->type !=
B_MEDIA_RAW_AUDIO )
        return B_MEDIA_BAD_FORMAT;

    if (format->u.raw_audio.channel_count==0) format->u.raw_audio.
channel_count=1;
    if (format->u.raw_audio.buffer_size==0) format->u.raw_audio.
buffer_size=1024;

    return B_OK;
} ;

status_t AudioListener::GetNextInput(int32* cookie, media_input*
out_input) {
    cout << "Wants next input: "<< *cookie << endl;

    if (0 == *cookie) {
        mInput.format.type = B_MEDIA_RAW_AUDIO;
        *out_input = mInput;
        *cookie = 1;
        return B_OK;
    }
}

```

```

        else return B_BAD_INDEX;
    };

void AudioListener::DisposeInputCookie(int32) {};

void AudioListener::BufferReceived(BBuffer* buffer) {
    if (B_MEDIA_PARAMETERS == buffer->Header()->type) {
        //ApplyParameterData(buffer->Data(),buffer->SizeUsed());
        buffer->Recycle();
    } else {
        status_t err;
        media_timed_event event(buffer->Header()->start_time,
            BTimedEventQueue::B_HANDLE_BUFFER,
            buffer, BTimedEventQueue::B_RECYCLE_BUFFER);
        err = EventQueue()->AddEvent(event);
        if (err) buffer->Recycle();
    }
};

void AudioListener::ProducerDataStatus(const media_destination &
for_whom, int32 status, bigtime_t at_performance_time) {
    if (for_whom == mInput.destination) {
        media_timed_event event(at_performance_time,
            BTimedEventQueue::B_DATA_STATUS, &mInput,
            BTimedEventQueue::B_NO_CLEANUP, status, 0, NULL);
        EventQueue()->AddEvent(event);
    }
};

status_t AudioListener::GetLatencyFor(const media_destination&
for_whom, bigtime_t* out_latency, media_node_id* out_timesource)
{
    // make sure this is one of my valid inputs
    if (for_whom != mInput.destination) return
B_MEDIA_BAD_DESTINATION;

    // report internal latency + downstream latency here, NOT
including scheduling latency.
    // we're a final consumer (no outputs), so we have no
downstream latency.
    *out_latency = mLatency;
    *out_timesource = TimeSource()->ID();
    return B_OK;
}

status_t AudioListener::Connected(
    const media_source& producer,
    const media_destination& where,
    const media_format& with_format,
    media_input* out_input)
{
    cout << "Connected!" << endl;

    if (where != mInput.destination) return
B_MEDIA_BAD_DESTINATION;

    // calculate my latency here, because it may depend on buffer

```

```

sizes/durations, then
    // tell the BMediaEventLooper how early we need to get the
buffers
    SetEventLatency(mLatency);

    // record useful information about the connection, and return
success
    mInput.source = producer;

    mInput.format=with_format;
    *out_input = mInput;
    cout << "Raw Audio CH count:"<<with_format.u.raw_audio.
channel_count << endl;
    cout << "Raw Audio Frame Rate:"<<with_format.u.raw_audio.
frame_rate << endl;
    cout << "Raw Audio buffer_size:"<<with_format.u.raw_audio.
buffer_size << endl;
    cout << "Raw Audio format:"<<with_format.u.raw_audio.format <<
endl;

    return B_OK;
};

void AudioListener::Disconnected(
    const media_source& producer,
    const media_destination& where)
{
    cout << "Disconnected!" << endl;

    // wipe out our input record
    resetInput();

    mInput.destination.port = ControlPort();
    mInput.destination.id = 0;
    mInput.node = Node();
    ::strcpy(mInput.name, "w00t");
}

status_t AudioListener::FormatChanged(
    const media_source& producer,
    const media_destination& consumer,
    int32 change_tag,
    const media_format& format)
{
    //should probably return an error...
    return B_OK;

    cout << "wanted to change format..." << endl;
}

//BMediaEventLooper Specific Stuff...
//*****
void AudioListener::NodeRegistered() {
    // Set our priority and start the BMediaEventLooper's thread
    SetPriority(mPriority);
    Run();

    // Initialize as much of our input as we can, now that the

```

```

Media Kit really "knows" about us
    mInput.destination.port = ControlPort();
    mInput.destination.id = 0;
    mInput.node = Node();
    ::strcpy(mInput.name, "AIin1");
}

void AudioListener::HandleEvent(const media_timed_event *event,
bigtime_t /* lateness */, bool /* realTimeEvent */)
{
    //cout << "Got event..." << endl;

    switch (event->type) {

    case BTimedEventQueue::B_HANDLE_BUFFER:
    {
        BBuffer* buffer = const_cast<BBuffer*>((BBuffer*) event-
>pointer);
        if (buffer)
        {
            media_header* hdr = buffer->Header();
            if (hdr->destination == mInput.destination.id)
            {
                bigtime_t now = TimeSource()->Now();
                bigtime_t perf_time = hdr->start_time;

                // the how_early calculate here doesn't include
scheduling latency because
                // we've already been scheduled to handle the
buffer
                //bigtime_t how_early = perf_time - mLatency - now;

                // logMsg.now is already set
                // if the buffer is late, we ignore it and report
the fact to the producer
                // who sent it to us

                {
                    // burn some percentage of our stated latency
in CPU time (controlled by
                    // a BParameter).  this simulates a user-
configurable amount of CPU cost
                    // associated with the consumer.
                    //cout << "got data: " << hdr->size_used <<
"bytes. " << endl;

                    unsigned int framesize,bufferptr;
                    framesize = (mInput.format.u.raw_audio.format &
media_raw_audio_format::B_AUDIO_SIZE_MASK)* mInput.format.u.raw_audio.
channel_count;
                    //cout << "Frame size: " << framesize << ".
Type: " << mInput.format.u.raw_audio.format << " Size: " << mInput.format.
u.raw_audio.buffer_size << endl;
                    for (bufferptr=0;bufferptr<mInput.format.u.
raw_audio.buffer_size;bufferptr+=framesize) {
                        void * x = (void *) ((char *)buffer->Data()
+bufferptr);

                        BufferPosition++;
                        BufferPosition%=512;
                        if (BufferPosition==0) {

```

```

        ProcessBuffer(); //process newly filled
buffer
    }

    //assuming no need for endian translation!

    //translate data if needed
    switch (mInput.format.u.raw_audio.format) {
        case media_raw_audio_format::
B_AUDIO_INT:
            BufferData[BufferPosition]=(*((long
*)x)) >> 24;
            break;
        case media_raw_audio_format::
B_AUDIO_SHORT:
            BufferData[BufferPosition]=(*
((short *)x)) >> 8;
            break;
        case media_raw_audio_format::
B_AUDIO_CHAR:
            BufferData[BufferPosition]=(*((char
*)x));
            break;
        case media_raw_audio_format::
B_AUDIO_UCHAR:
            BufferData[BufferPosition]=(*
((unsigned char *)x))-128;
            break;
        case media_raw_audio_format::
B_AUDIO_FLOAT:
            BufferData[BufferPosition]=(int)((*
((float *)x))*127);
            break;
    }
}

    // Offline mode requires some special handling.
When running offline the producer
    // doesn't just spit buffers as fast as it can (and
block waiting for downstream buffers
    // to be recycled), it waits for the consumer to
explicitly request them. This means
    // that it can react to other control messages
while it's waiting to send new buffers.
    if ((B_OFFLINE == RunMode()) && (B_DATA_AVAILABLE
== mProducerDataStatus))
    {
        RequestAdditionalBuffer(mInput.source, buffer);
    }
    else
    {
        //fprintf(stderr, "* Woah! Got a buffer for a
different destination!\n");
    }

    // we're done "processing the buffer"; now recycle it
and return to the loop

```

```

        buffer->Recycle();
    }
}
break;

case BTimedEventQueue::B_START:
    // okay, let's go!
    break;

case BTimedEventQueue::B_STOP:
    // stopping implies not handling any more buffers. So, we
flush all pending
    // buffers out of the event queue before returning to the
event loop.
    EventQueue()->FlushEvents(0, BTimedEventQueue::B_ALWAYS,
true, BTimedEventQueue::B_HANDLE_BUFFER);
    break;

case BTimedEventQueue::B_SEEK:
    // seeking the log doesn't make any sense, so we just log
that we handled the seek
    // and return without doing anything else
    break;

case BTimedEventQueue::B_WARP:
    // similarly, time warps aren't meaningful to the logger,
so just record it and return
    break;

case BTimedEventQueue::B_DATA_STATUS:
    // we track our producer's data status for cases like
offline, where we need to
    // consult it in order to decide whether to request another
buffer of data
    mProducerDataStatus = event->data;
    break;

default:
    // hmm, someone enqueued a message that we don't
understand. log and ignore it.
    break;
}

}

```


Filename: AudioProcessor.h

```
#include "AudioComponent.h"  
#include "AudioListener.h"
```

Filename: AudioProcessor.cpp

```
#include "AudioProcessor.h"
#include "AudioComponent.h"
#include <math.h>
#include <rfftw.h>

rfftw_plan p;

float FindPeak(rfftw_real * Data,int samples) {
    int peakloc;
    float max=0.0;
    for (int i=0;i<samples;i++) {
        if (Data[i]>max) {
            peakloc=i;
            max=Data[i];
        }
    }
    return peakloc/(samples+0.0);
}

float FindPeak2(rfftw_real * Data,int samples) {
    float peakloc=0.0;
    float sum=0.0;
    for (int i=0;i<samples;i++) {
        peakloc+=Data[i]*i;
        sum+=Data[i];
    }
    //cout << " peak loc: "<<peakloc/sum << endl;;
    return (peakloc/sum)/(samples+0.0);
}

void AudioListener::InitializeStuff() {

    //setup fftw
    p = rfftw_create_plan(512, FFTW_REAL_TO_COMPLEX, FFTW_ESTIMATE);
}

void AudioListener::ProcessBuffer() {

    static float BassPeak=0.0;
    static float Volume=0.0;

    int BigSum=0;
    float Low,Mid;

    for (int i=0;i<512;i++) {
        BigSum+=(BufferData[i]*BufferData[i]);
    }

    rfftw_real in[512], out[512], power_spectrum[512/2+1];

    int k,N=512;
    static int rollover=0;
    for (int i=0;i<512;i++) {
        in[i]=BufferData[i]/128.0;
    }
}
```

```

    rfftw_one(p, in, out);
    power_spectrum[0] = out[0]*out[0]; /* DC component */
    for (k = 1; k < (N+1)/2; ++k) /* (k < N/2 rounded up) */
        power_spectrum[k] = out[k]*out[k] + out[N-k]*out[N-k];
    if (N % 2 == 0) /* N is even */
        power_spectrum[N/2] = out[N/2]*out[N/2]; /* Nyquist freq.
*/

fftw_real peaks[8];
fftw_real md[8];
int m=0;
int o2;
int o3;
for (int o=0;o<225;o=o2) {
    o2=o*2+1;
    double myamt=0.0;
    for (int i=o;i<o2 && i<255;i++) {
        myamt+=(power_spectrum[i]);
    }

    if (myamt>peaks[m]) peaks[m]=myamt;
    myamt/=(peaks[m]);

    unsigned char Data[5];
    if (m==5) {
        Data[0]=128;
        Data[1]=myamt/2.1;
        Data[2]=0;
        Mid=myamt;
    }
    if (m==0) {
        Data[0]=129;
        Data[1]=myamt/2.1;
        Low=myamt;
    }

    if (Owner) {
        ((AudioComponent *)Owner)->SendData(1+m,myamt);
    }
    md[m]=myamt;
    m++;
}

if (Owner) {
    BassPeak*=0.97;
    float BassAmt = md[0]*0.5+md[1]*2+md[2]*0.75+md[3]*0.33;
    if (BassAmt>BassPeak) {
        BassPeak=BassAmt*3;
        ((AudioComponent *)Owner)->SendData(13,0.5);
        cout << "BOOM!"<<rand() << endl;
    }

    ((AudioComponent *)Owner)->SendData(0,sqrt(BigSum/512.0)/
127.0); //RMS
    Volume=Volume*0.985+0.015*sqrt(BigSum/512.0)/127.0;

```

```

        ((AudioComponent *)Owner)->SendData(14,Volume); //Volume
        ((AudioComponent *)Owner)->SendData(11,FindPeak
(power_spectrum,256));
        ((AudioComponent *)Owner)->SendData(12,FindPeak(md,8));
        ((AudioComponent *)Owner)->SendData(9,FindPeak2
(power_spectrum,256));
        ((AudioComponent *)Owner)->SendData(10,FindPeak2(md,8));

    }

}

```

Filename: SimpleMixer.cpp

```
#include "Component.h"
#include "ResourceSet.h"
#include <stdio.h>
#include <iostream.h>
#include "PrettySlider.h"

//*****
//this resource management stuff is taken from one issue of the
BMessage newsletter...
//http://www.be.com/developers/bmessage/issue09.html
#include <Locker.h>
#include <TranslatorFormats.h>

static int32 gResourcesCreated = 0;
static BResourceSet gResources;

static BResourceSet& Resources()
{
    // Return the BResourceSet object for this image, initializing
    // if needed.
    if ((gResourcesCreated&2) != 0) {
        // Quick return if already initialized.
        return gResources;
    } else if (atomic_or(&gResourcesCreated, 1) == 0) {
        // Not yet initialized -- do so.
        gResources.AddResources(&gResourcesCreated);
        // Mark that we are done.
        atomic_or(&gResourcesCreated, 2);
    } else {
        // Wait for resources to be initialized.
        while ((gResourcesCreated&2) == 0) snooze(50000);
    }
    return gResources;
}

//*****

AuroranComponent * MakeComponent(ComponentDescriptor * Who,
AuroranParameters * Params);

static ComponentDescriptor MyDescriptor = {
    A_DESC_VER,
    A_NORMAL_DESC,
    NULL,
    "Simple Desk",
    'TKS_',
    'SM',
    1,
    ACC_INPUT,
    &MakeComponent
};

#define World (*TheWorld)

class SimpleMixer : public AuroranComponent {
```

```

private:
    float Values[20];
    BSlider * Slide[20];
public:
    SimpleMixer(AuroranParameters * Params) : AuroranComponent
(Parms, DefaultCustom()) {
        MyIcon=(BBitmap *)Resources().FindBitmap(B_PNG_FORMAT,
2001);

        AutoSetColors();

        char name[3] = {'0','0',0};
        for (int i=0;i<19;i++) {
            Values[i]=0.0;
            RegisterPlug(BString("CH ").Append(name).String(),
StdChan_Intensity);
            if (name[1]=='9') {
                name[0]++;
                name[1]='0';
            } else {
                name[1]++;
            }
        }
        //~SimpleMixer() {printf("time to sleep\n");}
    BView * GetView(BRect frame) {
        World.Pulser->DeleteClient(World.ILoop);
        World.Pulser->AddClient(World.ILoop,100.0);
        BView * TheView = AuroranComponent::GetView(frame);
        char name[4] = {'0','0',0,'C'};
        int i;
        BSlider * Slider;
        BMessage * msg;
        for (i=0;i<19;i++) {
            Slider = new PrettySlider(BRect(i*30,20,20+i*30,200),
BString("Slide_").Append(name).String(),name,NULL,0,511,B_VERTICAL,
B_BLOCK_THUMB,B_FOLLOW_TOP|B_FOLLOW_LEFT,B_FRAME_EVENTS|B_WILL_DRAW|
B_NAVIGABLE);
            Slide[i]=Slider;
            //Slider->SetOrientation(B_VERTICAL);
            Slider->SetBarThickness(5);
            Slider->SetTarget(this);
            Slider->SetHashMarks(B_HASH_MARKS_BOTH);
            Slider->SetHashMarkCount(11);
            Slider->SetValue(Values[i]*511);
            msg=new BMessage('ChgC');
            msg->AddPointer("Plug",MyPlugs.ItemAt(i));
            msg->AddPointer("Slide",Slider);
            Slider->SetModificationMessage(msg);
            //cout << "Snooze Amount: " << Slider->SnoozeAmount() <<

endl;
//
            Slider->SetBarColor((rgb_color){45,255,128,255});
            TheView->AddChild(Slider);
            if (name[1]=='9') {
                name[0]++;
                name[1]='0';
            } else {
                name[1]++;
            }
        }
    }

```

```

    }

    return TheView;
}

void OutOfSight() {
    World.Pulser->DeleteClient(World.ILoop);
}

virtual void InterfaceMessage(BMessage * message) {
    switch (message->what) {
        case A_PULSE:
            //case 'Poll':
            {
                double TempValue;
                for (int i=0;i<19;i++) {
                    if (((PlugListItem *)MyPlugs.ItemAt(i))-
>isConnected()) {
                        //TempValue = ((PlugListItem *)MyPlugs.
ItemAt(i))->SendMessage(-1.0);
                        TempValue = (((PlugListItem *)MyPlugs.ItemAt
(i))->GetValue()).f;
                        if (TempValue >= 0.0 && TempValue <= 1.0) {
                            if (Slide[i]->Window()->LockWithTimeout
(1000)==B_OK) { //make sure we don't deadlock!
                                Slide[i]->SetTarget(NULL); //
disable sending messages
                                Slide[i]->SetValue(TempValue*511);
                                Slide[i]->SetTarget(this); //
disable sending messages
                                Values[i]=TempValue;
                                Slide[i]->Window()->Unlock();
                            }
                        }
                    }
                }
            }
        World.Pulser->GotIt(World.ILoop);
    }
    default:
        AuroranComponent::InterfaceMessage(message);
}

}

virtual void MessageReceived(BMessage * message) {
    switch (message->what) {
        case ('ChgC'): {
            BSlider * Slide;
            PlugListItem * Plug;
            if (message->FindPointer("Slide", (void **)&Slide)
==B_OK) {
                //
                cout << "New Value: " << Slide->Position() <<
endl;
                if (message->FindPointer("Plug", (void **)&Plug)
==B_OK) {
                    Plug->SendMessage(Values[MyPlugs.IndexOf
(Plug)]=Slide->Position());
                }
            }
        }
    }
}

```

```

        }
        break;
    }
}
AuroranComponent::MessageReceived(message);
}

    virtual ComponentDescriptor * GetDesc() {return &MyDescriptor;}
protected:
    virtual char * MyName() {return "Simple Desk";}
    virtual char * DefaultCustom() {return "Desk";}
};

AuroranComponent * MakeComponent(ComponentDescriptor * Who,
AuroranParameters * Params) {return new SimpleMixer(Params);}

extern "C" _EXPORT ComponentDescriptor* GetDesc();

ComponentDescriptor* GetDesc() {
    return &MyDescriptor;
}

```


Filename: Types.h

```
#ifndef SigTypes_h
#define SigTypes_h

enum SigType {
    S_SINE=1,
    S_COSINE=2,
    S_TRIANGLE=3,
    S_SAW=4,
    S_SQUARE=5,
    S_PULSE=6
};

struct SignalData {
    float Period; //in milliseconds
    char Type;    //What sort of wave
    float Span;
    float Base;
    bigtime_t Offset;
    float LastValue;
    char Extra[24];
};

#endif
```

Filename: WaveView.h

```
#include <View.h>
#include <Bitmap.h>
#include "Types.h"

float DoMath(float Value, SignalData * SD, float Output=-1);

class WaveView : public BView {
    SignalData MyData;
    SignalData MyOldData;
    BBitmap * MyBitmap;
public:
    WaveView(BRect frame, const char * name, uint32 resize =
B_FOLLOW_LEFT|B_FOLLOW_TOP);
    ~WaveView() {delete MyBitmap;}
    void Draw(BRect updateRect);
    void NewData(SignalData &NewData);
};
```

Filename: WaveView.cpp

```
#include "WaveView.h"
#include <iostream.h>

WaveView::WaveView(BRect frame, const char * name, uint32 resize =
B_FOLLOW_LEFT|B_FOLLOW_TOP) : BView(frame,name,resize,B_WILL_DRAW) {
    SetViewColor((rgb_color){0,0x50,0,255});
    MyBitmap = new BBitmap(BRect(0,0,frame.Width(),frame.Height()),
B_RGB32);
    uint32 * Data = (uint32 *)MyBitmap->Bits();
    int RowPixels = frame.Width();
    int HeightPixels = frame.Height();
    int Words = MyBitmap->BytesPerRow()/4;
    int TempData = HeightPixels * Words;
    for (int i = 0;i<RowPixels;i++) {
        Data[i]=0x00505050;
        Data[i+TempData]=0x00F0F0F0;
    }
    TempData = (HeightPixels+1)*Words;
    for (int i = 0;i<TempData;i+=Words) {
        Data[i]=0x00505050;
        Data[i+RowPixels-1]=0x00F0F0F0;
    }
    SignalData A = (SignalData){1.0,1,1.0,0.0,0,0.0,0};
    MyData=A;
    NewData(A);
}

void WaveView::NewData(SignalData &NewData) {
    MyOldData=MyData;
    MyData=NewData;
    uint32 * Data = (uint32 *)MyBitmap->Bits();
    int Width = MyBitmap->BytesPerRow()/4;
    int Height = MyBitmap->Bounds().Height();
    int Offset;
    int Separators = (Width-3)/12;
    int SepC = 1;
    uint32 BaseColor;
    uint32 YColor;
    int YSep;
    int YSepC;

    float ValueDelta=2.0/(Width-2);

    MyData.LastValue=1.0;
    float AValue=MyData.Offset/(float)MyData.Period;
    float AOutput;
    float ALastOutput;
    int AYMin,AYMax;
    AOutput=DoMath(AValue,&MyData,0.0);

    MyOldData.LastValue=1.0;
    float BValue=MyOldData.Offset/(float)MyOldData.Period;
    float BOutput;
    float BLastOutput;
    int BYMin,BYMax;
```

```

BOutput=DoMath(BValue,&MyOldData,0.0);

for (int x = 1;x < (Width-2); x++) {
    YSep = (Height)/6;
    YSepC=1;
    Offset = Width + x;
    if (x==Separators) {
        BaseColor=0x00505050;
        SepC++;
        Separators=((Width-3)*SepC)/12;
    } else {
        BaseColor=0x00005000;
    }

    //calculate value @ this point
    AValue+=ValueDelta;
    if (AValue > 1.0) AValue-=1.0;
    AOutput=DoMath(AValue,&MyData,ALastOutput=AOutput);
    MyData.LastValue=AValue;
    if (fabs(AOutput-ALastOutput) <= 1.0/(Height-2)) {
        AYMin=AYMax=Height-(AOutput*(Height-2)+1);
    } else {
        if (AOutput < ALastOutput) {
            AYMax=Height-(AOutput*(Height-2)+1);
            AYMin=Height-(ALastOutput*(Height-2)+1);
        } else {
            AYMin=Height-(AOutput*(Height-2)+1);
            AYMax=Height-(ALastOutput*(Height-2)+1);
        }
    }

    BValue+=ValueDelta;
    if (BValue > 1.0) BValue-=1.0;
    BOutput=DoMath(BValue,&MyOldData,BLastOutput=BOutput);
    MyOldData.LastValue=BValue;
    if (fabs(BOutput-BLastOutput) <= 1.0/(Height-2)) {
        BYMin=BYMax=Height-(BOutput*(Height-2)+1);
    } else {
        if (BOutput < BLastOutput) {
            BYMax=Height-(BOutput*(Height-2)+1);
            BYMin=Height-(BLastOutput*(Height-2)+1);
        } else {
            BYMin=Height-(BOutput*(Height-2)+1);
            BYMax=Height-(BLastOutput*(Height-2)+1);
        }
    }

    for (int y = 1; y < Height; y++) {
        YColor=BaseColor;

        if (y==YSep) {
            YSepC++;
            YSep=((Height)*YSepC)/6;
            YColor=0x00505050;
        }
        if (y==Height/2) {
            YColor=0x00909090;
        }
    }
}

```

```

        if (y>=AYMin && y<=AYMax) {
            YColor=0x0090FF50;
        } else if (y>=BYMin && y<=BYMax) {
            YColor=0x00508000;
        }

        Data[Offset]=YColor;
        Offset+=Width;
    }
}

void WaveView::Draw(BRect updateRect) {
    DrawBitmap(MyBitmap,updateRect,updateRect);
}

float DoMath(float Value, SignalData * SD, float Output) {
    switch (SD->Type) {
        case S_SINE: {
            Output = (((sin((Value)*6.28)+1.0)/2.0) * SD->Span + SD->Base);
            break;
        }
        case S_COSINE: {
            Output = (((cos((Value)*6.28)+1.0)/2.0) * SD->Span + SD->Base);
            break;
        }
        case S_SAW: {
            Output = ((Value)*SD->Span + SD->Base);
            break;
        }
        case S_SQUARE: {
            if (Value > .5 && SD->LastValue <= .5) { //turn off
                Output = SD->Base;
            } else if (Value <= .5 && SD->LastValue > .5) { //turn on
                Output = SD->Base + SD->Span;
            }
            break;
        }
        case S_TRIANGLE: {
            if (Value < .5) { //slope up
                Output = (Value * 2.0)*SD->Span + SD->Base;
            } else {
                Output = (2.0-Value * 2.0)*SD->Span + SD->Base;
            }
            break;
        }
        case S_PULSE: {
            if (SD->LastValue > Value) {
                Output = SD->Base + SD->Span;
            }
            break;
        }
    }
    return Output;
}

```

Filename: Component.cpp

```
#include "Component.h"
#include "ResourceSet.h"
#include <stdio.h>
#include <iostream.h>
#include <SerialPort.h>
#include "Pulse.h"
#include "MaxMinCalculator.h"
#include "PercentControl.h"
#include <alloc.h>

//My own files
#include "Types.h"
#include "WaveView.h"

//this resource management stuff is taken from one issue of the
BMessage newsletter...
//http://www.be.com/developers/bmessage/issue09.html
#include <Locker.h>
#include <TranslatorFormats.h>

//*****

static int32 gResourcesCreated = 0;
static BResourceSet gResources;

static BResourceSet& Resources()
{
    // Return the BResourceSet object for this image, initializing
    // if needed.
    if ((gResourcesCreated&2) != 0) {
        // Quick return if already initialized.
        return gResources;
    } else if (atomic_or(&gResourcesCreated, 1) == 0) {
        // Not yet initialized -- do so.
        gResources.AddResources(&gResourcesCreated);
        // Mark that we are done.
        atomic_or(&gResourcesCreated, 2);
    } else {
        // Wait for resources to be initialized.
        while ((gResourcesCreated&2) == 0) snooze(50000);
    }
    return gResources;
}

AuroranComponent * MakeComponent(ComponentDescriptor * Who,
AuroranParameters * Params);

static ComponentDescriptor MyDescriptor = {
    A_DESC_VER,
    A_NORMAL_DESC,
    NULL,
    "Signal Generator", //Menu Name
    'TKS_', //Creator
    'SG', //ID
    0, //Version
}
```

```

        ACC_INPUT,          //Class
        &MakeComponent
    };
#define World (*TheWorld)

class SigGen : public AuroranComponent {
private:
    int SignalAccuracy;
    float SignalPeriod;
    BListView * WaveList;
    BSlider * AccuracySlide;
    BList Waves; //my list of waves

    //newly updated data
    char NewType;
    BTextControl * NewName;
    BTextControl * NewPeriod;
    BPopupMenu * SignalTypes;
    PercentControl * NewOffset;
    PercentControl * NewMin;
    PercentControl * NewMax;

    WaveView * MyWaveView;

    BButton * DeleteBtn;

    // PlugListItem * Sine;
    // PlugListItem * Cosine;
public:
    //Don't Touch
    virtual ComponentDescriptor * GetDesc() {return &MyDescriptor;}

    SigGen(AuroranParameters * Worl) : AuroranComponent(Worl,
DefaultCustom()) { //set up
        HighlightBG=(rgb_color){80,240,120,128};
        NormalBG= (rgb_color){60,200,100,128};
        MyIcon=(BBitmap *)Resources().FindBitmap(B_PNG_FORMAT,
2001);

        SignalAccuracy=60;
        SignalPeriod=1000.0/60.0;

        TheWorld->Pulser->AddClient(this,SignalPeriod);

        //Sine = RegisterPlug("0.1Hz Sine");
        //Cosine = RegisterPlug("0.1Hz Cosine");

    }
    ~SigGen() { //Clean up
        cout << " Signal Generator Closing up" << endl;
        World.Pulser->DeleteClient(this);
    }
    BView * GetView(BRect frame) {
        BView * TheView = AuroranComponent::GetView(frame);
        NewType=S_SINE;

        //List 'O Waves
        WaveList = new BListView(BRect(3,40,120,230),"1st_Waves",

```

```

B_SINGLE_SELECTION_LIST,B_FOLLOW_TOP_BOTTOM);
    WaveList->SetSelectionMessage(new BMessage('SlNS'));
    WaveList->SetTarget(World.ILoop); //send to interafce
messenger
    TheView->AddChild(new BScrollView("scroll_Waves",WaveList,
B_FOLLOW_TOP_BOTTOM,0,false,true));
        for (int i=0;i<MyPlugs.CountItems();i++) {
            WaveList->AddItem(new BStringItem(((PlugListItem *)
MyPlugs.ItemAt(i))>Text()));
        }

    TheView->AddChild(new BStringView(BRect(3,25,120,38),
"lbl_Waves","Signals:"));

    BButton * NewButton;
    BMessage * msg = new BMessage('NSig');
    TheView->AddChild(NewButton=new BButton(BRect(400,frame.
Height()-28,475,frame.Height()-8),"btn_New","New Signal",msg));
    NewButton->SetTarget(World.ILoop); //send to interafce
messenger

    msg = new BMessage('DSig');
    TheView->AddChild(DeleteBtn=new BButton(BRect(315,frame.
Height()-28,390,frame.Height()-8),"btn_New","Delete Signal",msg));
    DeleteBtn->SetTarget(World.ILoop); //send to interafce
messenger

    DeleteBtn->SetEnabled(false);

    msg = new BMessage('NuAc');
    char Buffer[5];
    sprintf(Buffer,"%d", SignalAccuracy);
    AccuracySlide = new BSlider(BRect(149,188,305,220),
"slide_Accuracy",BString("Signal Accuracy: ").Append(Buffer).Append
("Hz").String(),msg,1,240,B_TRIANGLE_THUMB,B_FOLLOW_LEFT|B_FOLLOW_TOP,
B_FRAME_EVENTS|B_WILL_DRAW|B_NAVIGABLE);
    AccuracySlide->SetBarThickness(6);
    AccuracySlide->SetHashMarkCount(13);
    AccuracySlide->SetHashMarks(B_HASH_MARKS_TOP);
    AccuracySlide->SetLimitLabels("1Hz","240Hz");
    AccuracySlide->SetValue(SignalAccuracy);
    AccuracySlide->SetTarget(this);
    AccuracySlide->SetBarColor((rgb_color){152,152,255,255});

    TheView->AddChild(AccuracySlide);

    //Signal's Name
    TheView->AddChild(NewName=new BTextControl(BRect
(149,30,350,45),"txt_SlideName", "Signal Name:", "", NULL));
    NewName->SetFontSize(12);

    //Signal's Type
    SignalTypes = new BPopupMenu("Signal Types");
    SignalTypes->SetFontSize(12);
    msg = new BMessage('ChTp');
    msg->AddInt8("Type",S_SINE);
    BMenuItem * Sig;
    SignalTypes->AddItem(Sig = new BMenuItem("Sine
",

```



```

msg,0,0));
    Sig->SetTarget(World.ILoop);

    msg = new BMessage('ChTp');
    msg->AddInt8("Type",S_COSINE);
    SignalTypes->AddItem(Sig = new BMenuItem("Cosine",
msg,0,0));
    Sig->SetTarget(World.ILoop);

    msg = new BMessage('ChTp');
    msg->AddInt8("Type",S_TRIANGLE);
    SignalTypes->AddItem(Sig = new BMenuItem("Triangle",
msg,0,0));
    Sig->SetTarget(World.ILoop);

    msg = new BMessage('ChTp');
    msg->AddInt8("Type",S_SAW);
    SignalTypes->AddItem(Sig = new BMenuItem("Saw",
msg,0,0));
    Sig->SetTarget(World.ILoop);

    msg = new BMessage('ChTp');
    msg->AddInt8("Type",S_SQUARE);
    SignalTypes->AddItem(Sig = new BMenuItem("Square",
msg,0,0));
    Sig->SetTarget(World.ILoop);

    msg = new BMessage('ChTp');
    msg->AddInt8("Type",S_PULSE);
    SignalTypes->AddItem(Sig = new BMenuItem("Pulse",
msg,0,0));
    Sig->SetTarget(World.ILoop);
    SignalTypes->ItemAt(0)->SetMarked(true);
    BMenuField * MenuOpt;
    TheView->AddChild(MenuOpt = new BMenuField(BRect
(149,50,350,65), "List_SigTypes", "Signal Type:", SignalTypes));
    MenuOpt->SetFontSize(12);

    //Signal Period
    TheView->AddChild(NewPeriod=new BTextControl(BRect
(149,75,350,90),"txt_Period", "Period:", "1.000",NULL));
    NewPeriod->SetFontSize(12);

    //Offset
    TheView->AddChild(NewOffset = new PercentControl(BRect
(149,100,350,115),"percent_offset", "Offset:",new BMessage('NDta')));
    NewOffset->SetTarget(World.ILoop);
    NewOffset->SetFontSize(12);

    TheView->AddChild(NewMin = new PercentControl(BRect
(149,125,350,140),"percent_min", "Minimum:",new BMessage('NDta')));
    NewMin->SetTarget(World.ILoop);
    NewMin->SetFontSize(12);

    TheView->AddChild(NewMax = new PercentControl(BRect
(149,150,350,165),"percent_max", "Maximum:",new BMessage('NDta')));
    NewMax->SetTarget(World.ILoop);
    NewMax->SetFontSize(12);
    NewMax->SetValue(1.0);

```

```

        //wave viewer
        TheView->AddChild(MyWaveView = new WaveView(BRect
(370,30,560,130), "wave_viewer"));

        //TheView->AddChild(SView = new BStringView(BRect
(150,50,250,65), "lbl_Name", "Signal Type:"));
        //SView->SetFontSize(12);

        TheView->AddChild(new BStringView(BRect(383,130,560,145),
"lbl_Osc", "Oscilloscope displays two cycles.));
        TheView->AddChild(new BStringView(BRect(378,145,560,158),
"lbl_Osc", "Time displayed is thus 2x the Period.));

        return TheView;
    }

    void DoApplyChanges() { //save stuff
        int32 idx;
        if ((idx=WaveList->CurrentSelection())>=0) {
            if (World.CList->Window()->LockWithTimeout(2000)==B_OK)
{
                if (LockWithTimeout(8000)==B_OK) {

                    SignalData * SD;
                    SD = (SignalData*)Waves.ItemAt(idx);
                    SD->Type=NewType;

                    float Temp;

                    sscanf(NewPeriod->Text(), "%f", &Temp);
                    if (Temp < 0.01) Temp=0.01;
                    SD->Period=Temp*1000000.0;

                    SD->Offset=NewOffset->Value()*SD->Period;
                    cout << "Offset = " << SD->Offset << " from "
<< NewOffset->Value() << endl;

                    SD->Base=NewMin->Value();
                    SD->Span=NewMax->Value()-SD->Base;
                    MyWaveView->NewData(*SD);
                    MyWaveView->Invalidate();

                    PlugListItem * Plg = (PlugListItem*)MyPlugs.
ItemAt(idx);
                    if (NewType==S_PULSE || NewType==S_SQUARE) {
                        Plg->SetFlags(Plg->GetFlags() & (~Chan_Stream)
|Chan_Pulse);
                        Plg->SetTypeName("Intensity/Pulse");
                    } else {
                        Plg->SetFlags(Plg->GetFlags() & (~Chan_Pulse)
|Chan_Stream);
                        Plg->SetTypeName("Intensity");
                    }

                    if (strcmp(((PlugListItem *)MyPlugs.ItemAt
(idx))->Text(), NewName->Text())) {
                        ((BStringItem*)MyPlugs.ItemAt(idx))->SetText

```

```

(NewName->Text());
((BStringItem *)WaveList->ItemAt(idx))-
>SetText(NewName->Text());
WaveList->InvalidateItem(idx);
}

Unlock();
World.CList->Window()->Unlock();
} else {
World.CList->Window()->Unlock();
World.Pulser->OneShot(World.ILoop, new BMessage
(A_APPLY_CHANGES)); //try again in 1ms
}
} else {
World.Pulser->OneShot(World.ILoop, new BMessage
(A_APPLY_CHANGES)); //try again in 1ms
}
}

}

virtual void InterfaceMessage(BMessage * message) { //interface
messages
switch (message->what) {
case 'ChTp': {
char NS;
if (message->FindInt8("Type", (int8*)&NS)==B_OK) {
NewType=NS;
}
//Roll over into "New Data"
}
case 'NDta': {
if (World.CList->Window()->LockWithTimeout(1000)
==B_OK) {

SignalData SD;
SD.Type=NewType;

SD.Period=1000.0;

SD.Offset=NewOffset->Value()*1000.0;

SD.Base=NewMin->Value();
SD.Span=NewMax->Value()-SD.Base;

MyWaveView->NewData(SD);
MyWaveView->Invalidate();

World.CList->Window()->Unlock();
}
else { //failed to aquire lock
cout << "Locked failed for " << message->what
<< endl;
World.Pulser->OneShot(World.ILoop, new BMessage
(*message)); //try again in 1ms
}
break;
}
case 'SlNS': { //selection changed

```

```

        int32 idx;
        if (World.CList->Window()->LockWithTimeout(1000)
==B_OK) {
            if ((idx=WaveList->CurrentSelection())>=0) {
                char Buffer[256];
                NewName->SetText(((BStringItem *)WaveList-
>ItemAt(idx))->Text());
                SignalData * SD;
                SD = (SignalData*)Waves.ItemAt(idx);
                for (int i = SignalTypes->CountItems() - 1;
i>=0 ; i--) {
                    if (SignalTypes->ItemAt(i)->Message()-
>FindInt8("Type")==SD->Type) {
                        SignalTypes->ItemAt(i)->SetMarked
(true);
                        NewType=SD->Type;
                    }
                }
                sprintf(Buffer,"%f",SD->Period/1000000.0);
                NewPeriod->SetText(Buffer);

                NewOffset->SetValue((SD->Offset*1.0)/(SD-
>Period*1.0));
                cout << "Got offset: " << SD->Offset << " so
sending " << (SD->Offset*1.0)/(SD->Period*1.0) << " read as: " <<
NewOffset->Value() << endl;

                NewMin->SetValue(SD->Base);
                NewMax->SetValue(SD->Base+SD->Span);

                MyWaveView->NewData(*SD);
                MyWaveView->Invalidate();

                DeleteBtn->SetEnabled(true);
            } else {
                DeleteBtn->SetEnabled(false);
            }
            World.CList->Window()->Unlock();
        } else { //failed to aquire lock...
            cout << "Locked failed for " << message->what
<< " by " << World.CList->Window()->LockingThread() << endl;
            World.Pulser->OneShot(World.ILoop, new BMessage
(*message)); //try again in 1ms
        }
        break;
    }
    case 'NSig': { //new signal
        if (World.CList->Window()->LockWithTimeout(1000)
==B_OK) {
            if (LockWithTimeout(1000)==B_OK) { //prevent us
from doing anything too...

                SignalData * SD=new SignalData;

                //Disallow nameless entries
                if (*(NewName->Text())==0) NewName->SetText
("-");

                if (NewType==S_PULSE || NewType==S_SQUARE) {

```

```

        PlugListItem * NP;
        NP = RegisterPlug(NewName->Text(),
StdChan_Intensity);
        NP->SetFlags(NP->GetFlags() &
(~Chan_Stream) | Chan_Pulse);
        NP->SetTypeName("Intensity/Pulse");
    } else {
        RegisterPlug(NewName->Text(),
StdChan_Intensity);
    }
    WaveList->AddItem(new BStringItem(NewName-
>Text()));
    WaveList->Select(WaveList->CountItems()-1);
    Waves.AddItem(SD);

    DoApplyChanges(); //save the rest through a
call to "DoApplyChanges"

    UnlockLooper();
    World.CList->Window()->Unlock();
    } else { //unable to lock self
        World.CList->Window()->Unlock(); //release
the lock we did get
        cout << "Locked failed level 2 for " <<
message->what << endl;
        World.Pulser->OneShot(World.ILoop, new
BMessage(*message)); //try again in 1ms
    }
    } else {
        cout << "Locked failed for " << message->what
<< endl;
        World.Pulser->OneShot(World.ILoop, new BMessage
(*message)); //try again in 1ms
    }

    break;
}
case 'DSig': { //delete signal
    int32 idx;
    if ((idx=WaveList->CurrentSelection())>=0) {

        if (World.CList->Window()->LockWithTimeout
(1000)==B_OK) {
            if (World.MaxMinCalc->LockWithTimeout(1000)
==B_OK) {
                if (LockWithTimeout(1000)==B_OK) {

                    delete WaveList->RemoveItem(idx);
                    DeletePlug(idx);
                    delete Waves.RemoveItem(idx);

                    Unlock();
                    World.MaxMinCalc->Unlock();
                    World.CList->Window()->Unlock();
                } else {
                    World.MaxMinCalc->Unlock();
                    World.CList->Window()->Unlock();

```

```

World.Pulser->OneShot(World.ILoop,
new BMessage(*message)); //try again in 1ms
    }
    } else {
        World.CList->Window()->Unlock();

        World.Pulser->OneShot(World.ILoop, new
BMessage(*message)); //try again in 1ms
    }
    } else {
        World.Pulser->OneShot(World.ILoop, new
BMessage(*message)); //try again in 1ms
    }
    }
    break;
}
case A_APPLY_CHANGES: { //Apply changes
    DoApplyChanges();
} //DO NOT BREAK - we want to flow into the standard
Apply Changes
default: {
    AuroranComponent::InterfaceMessage(message);
}
}
}
virtual void MessageReceived(BMessage * message) {
    switch (message->what) {
        case A_PULSE: {
            bigtime_t now = real_time_clock_usecs();
            float Value;

            for (int i=Waves.CountItems()-1;i>=0;i--) {
                SignalData * SD = (SignalData *)Waves.ItemAt(i);

                Value = ((now+SD->Offset) % ((int)(SD-
>Period)))/(SD->Period);

                float Output = DoMath(Value,SD,-1);
                if (Output>1.0) Output=1.0;
                //cout << "Output" << Output <<endl;

                if (Output >= 0) ((PlugListItem *)MyPlugs.
ItemAt(i))->SendMessage(Output);
                SD->LastValue = Value;
            }

            World.Pulser->GotIt(this);

            break;
        }
        case 'NuAc': { //new level of accuracy - processed here
as to not screw up timing *that* much
            SignalAccuracy=AccuracySlide->Value();
            char Buffer[5];
            *((long *)Buffer)=0;
            sprintf(Buffer,"%d", SignalAccuracy);

```

```

        if (AccuracySlide->Window()->LockWithTimeout(2000)
==B_OK) {
            AccuracySlide->SetLabel(BString("Signal
Accuracy: ").Append(Buffer).Append("Hz").String());
            AccuracySlide->Window()->Unlock();
        } else {
            World.Pulser->OneShot(this, new BMessage
(*message)); //try again in 1ms
        }
        SignalPeriod=1000.0/SignalAccuracy;
        World.Pulser->UpdateClient(this,SignalPeriod);
        break;
    }
    case 'NSig': { //new signal
        break;
    }
}
AuroranComponent::MessageReceived(message);
}
virtual status_t Save(BPositionIO * object) {
    uint16 SignalCount = Waves.CountItems();
    object->Write(&SignalCount, 2);

    for (int i=0; i<SignalCount; i++) {
        uint16 Int16Temp;

        //save name
        const char * Name = ((PlugListItem *)MyPlugs.ItemAt(i))-
>Text();

        Int16Temp=strlen(Name)+1;
        object->Write(&Int16Temp,2);
        object->Write(Name,Int16Temp);

        cout << "Name Length: " << Int16Temp << endl;

        //save data...
        object->Write((SignalData *)Waves.ItemAt(i),sizeof
(SignalData));
    }

    return B_OK;
}
virtual status_t Load(BPositionIO * object, uint32 version) {
    uint16 SignalCount;
    object->Read(&SignalCount,2);

    LockLooper(); //we know we are safe to lock
    World.ILoop->LockLooper(); //ditto to this

    for (int i=0; i<SignalCount; i++) {
        uint16 Int16Temp;

        //load name
        char * Name;
        object->Read(&Int16Temp,2);

        Name = (char *)malloc(Int16Temp);

```

```

        object->Read(Name,Int16Temp);

        cout << "Name Length: " << Int16Temp << endl;

        SignalData * SD = new SignalData;
        //read data...
        object->Read(SD,sizeof(SignalData));

        Waves.AddItem(SD);

        if (SD->Type==S_PULSE || SD->Type==S_SQUARE) {
            PlugListItem * Plg = RegisterPlug(Name,
StdChan_Intensity);
            Plg->SetFlags(Plg->GetFlags() & (~Chan_Stream) |
Chan_Pulse);
            Plg->SetName("Intensity/Pulse");
        } else {
            PlugListItem * Plg = RegisterPlug(Name,
StdChan_Intensity);
        }

        //free memory used to load name.
        free(Name);
    }

    World.ILoop->UnlockLooper();
    UnlockLooper();

    return B_OK;
}
protected:
    virtual char * MyName() {return "Signal Generator";}
    virtual char * DefaultCustom() {return "Signal";}
};

AuroranComponent * MakeComponent(ComponentDescriptor * Who,
AuroranParameters * Paras) {return new SigGen(Paras);}

extern "C" _EXPORT ComponentDescriptor* GetDesc();

ComponentDescriptor* GetDesc() {
    return &MyDescriptor;
}

```


Filename: StateStuff.h

```
#include <InterfaceKit.h>
#include "ChannelListItem.h"
#include <String.h>

enum {
    REQ_TIME = 0,
    REQ_VALUE = 1,
    REQ_PULSE = 2,
    REQ_CHANCE = 3,
    REQ_FAIL = 4,
    REQ_CONN = 5
};

class AState;

class EventManager : public BView {
public:
    EventManager(BLooper * ILoop, BList * External, ChannelListItem
* Parent, BList * Internal, BRect frame, const char * Nomen);
    ~EventManager();
    void AddItem(ChannelListItem * Item, int Location);
    void MessageReceived(BMessage * msg);
    void AllAttached();
    void SetActiveState(AState * State, BList * AList);
    //void AttachedToWindow();
private:
    BLooper * ILoop;
    BListView * MyList;
    BButton * btnApply;
    BButton * btnPull;
    BCheckBox * chkSum;
    AState * ActiveState;
    BList * ActiveList;
    BList * External;
    BTextControl * txtValue;
};

class DState : public BStringItem {
public:
    DState(BStringItem * real) : BStringItem(real->Text()) {
        Active = false;
        Counterpart=real;
    }
    bool Active;
    BStringItem * Counterpart;
    void DrawItem(BView * owner, BRect frame, bool complete) {
        if (Active) {
            owner->SetHighColor((rgb_color){0,0,0,0});
            owner->FillEllipse(BRect(frame.left+2, frame.top+2, frame.
left+frame.Height()-4, frame.top+frame.Height()-4));
            owner->SetFont(be_bold_font);
        } else {
            owner->SetFont(be_plain_font);
        }
    }
};
```

```

        BStringItem::DrawItem(owner, BRect(frame.Height()-2, frame.
top, frame.right, frame.bottom), complete);
    }
};

enum {

    VMODE_SOCKET=1,
    VMODE_LOCAL=2,

    VMODE_LESS=8,
    VMODE_GTR=16,
    VMODE_EQU=32

};

class Requirement : public BStringItem {
public:
    Requirement(AState* Dest, Requirement* Par) ;
    AState * Destination;
    Requirement * Parent;
    int SubItemCount;
    int SubReadyCount;

    uint8 reqType;

    //Readiness
    bool Ready;

    //Related to this being a REQ_CHANCE
    int32 Probability;

    //Related to this being a time out
    float TimeOut;
    bigtime_t StartCount;

    //Related to Pulse reception + value properties
    ChannelListItem * SourceChan;

    //Related to value preproperties
    CHData ActivateValue;
    uint8 ActivationMode;

    void SetLabel() ;

    bool PrevReady;
    void reset(bool full=false);
    AState * TestReadiness(void * GenPtr);

    void Save(BPositionIO * Object, AuroranComponent* owner, BList
* StateList, BList * ReqList, BList * Internals);
    static Requirement * Load(BPositionIO * Object,
AuroranComponent* owner, BList * StateList, BList * ReqList, BList *
Internals);
};

enum EventType {
    EVT_UNDEF=0x00,

```

```

        EVT_SETVAL=0x01,
        EVT_ADD=0x02,
};

struct Event {
    public:
        ChannelListItem * Target;
        EventType DoWhat;
        CHData Value;

        Event(ChannelListItem * Target, EventType DoWhat, CHData Value) :
            Target(Target), DoWhat(DoWhat), Value(Value) {
        }
};

class AState : public BStringItem {
    private:
    public:
        AState(const char * name);
        BString Notes;
        BList Requirements;
        BList EntryEvents;
        BList ExitEvents;
        bool Active;
        void DrawItem(BView * owner, BRect frame, bool complete);

        AState * CheckState(void * GenPtr);

        void EnterState();

        void LeaveState();

        void SaveState(BPositionIO * Object, AuroranComponent* owner,
BList * StateList, BList * Internals);
        void LoadState(BPositionIO * Object, AuroranComponent* owner,
BList * StateList, BList * Internals);
};

```

Filename: StateStuff.cpp

```
#include "StateStuff.h"
#include <iostream.h>
#include <String.h>
#include <Rect.h>
#include "Component.h"
#include <stdio.h>
#include <stdlib.h>

void EventManager::MessageReceived(BMessage * msg) {
    cout << "msg recieved" << endl;
    msg->PrintToStream();
    switch (msg->what) {
        case 'ESel':
            cout << "event selected" << endl;
            if (MyList->CurrentSelection() >= 0 && ActiveState!=NULL) {
                char Buffer[64];
                int idx=MyList->CurrentSelection();
                Event * MyEvent=NULL;
                for (int j = ActiveList->CountItems()-1;j>=0;j--) { //
clear stuff being active or not
                    if (((DState*)MyList->ItemAt(idx))->Counterpart ==
((Event*)ActiveList->ItemAt(j))->Target) {
                        MyEvent = (Event*)ActiveList->ItemAt(j);
                    }
                }
                if (MyEvent!=NULL) {
                    snprintf((char *)&Buffer,63,"%f",MyEvent->Value.f);
                    txtValue->SetText(Buffer);
                    chkSum->SetValue((MyEvent->DoWhat==EVT_ADD)?
B_CONTROL_ON:B_CONTROL_OFF);
                }
            }
            break;
        case 'bPul':
            cout << "pull pressed" << endl;
            if (MyList->CurrentSelection() >= 0 ) {
                char Buffer[64];
                int idx=MyList->CurrentSelection();
                //Event * MyEvent=NULL;
                //if (MyEvent!=NULL) {
                    snprintf((char *)&Buffer,63,"%f",
((ChannelListItem*)External->ItemAt(idx))->GetValue().f);
                    txtValue->SetText(Buffer);
                //}
            }
            break;
        case 'bApl':
            cout << "Apply buton pressed" << endl;
            if (ActiveState!=NULL && MyList->CurrentSelection()>=0) {
                int idx=MyList->CurrentSelection();
                float Value;

                //clear any existing event for this output
                for (int j = ActiveList->CountItems()-1;j>=0;j--) { //
clear stuff being active or not
```

```

        if (((DState*)MyList->ItemAt(idx))>Counterpart ==
((Event*)ActiveList->ItemAt(j))>Target) {
            ActiveList->RemoveItem(j);
        }
    }

    sscanf(txtValue->Text(), "%f", &Value);

    ActiveList->AddItem(new Event(((ChannelListItem*)
((DState*)MyList->ItemAt(idx))>Counterpart), (chkSum->Value()
==B_CONTROL_ON)?EVT_ADD:EVT_SETVAL, (CHData){Value}));
    ((DState*)MyList->ItemAt(idx))>Active=true;
    MyList->InvalidateItem(idx);
}
break;
case 'AddP':
{
    cout << "New Plug!" << endl;
    ChannelListItem * Item;
    int32 Loc;
    msg->FindPointer("Item", (void**) &Item);
    msg->FindInt32("Location", &Loc);
    MyList->AddItem(new DState(Item), Loc);
}
break;
default:
    BView::MessageReceived(msg);
}

}

void EventManager::AddItem(ChannelListItem * Item, int Location) {
    MyList->AddItem(new DState(Item), Location);
}

void EventManager::AllAttached() {

    //for (int i = CountChildren()-1; i>=0; i--) {
    MyList->SetTarget(this, Window());
    btnApply->SetTarget(this, Window());
    btnPull->SetTarget(this, Window());
    //}
    BView::AllAttached();
    //Window()->RemoveHandler(this);
    //ILoop->AddHandler(this);

}

EventManager::EventManager(BLooper * ILoop, BList * External,
ChannelListItem * Parent, BList * Internal, BRect frame, const char *
Nomen) : BView(frame, Nomen, B_FOLLOW_ALL, 0), ILoop(ILoop), External
(External) {
    ActiveState=NULL;

    MyList = new BListView(BRect(3, 3, 120, frame.Height()-6), ".List",
B_SINGLE_SELECTION_LIST, B_FOLLOW_TOP_BOTTOM);
    //ThisMsg=new BMessage(*MyMsg);
    //ThisMsg->AddInt32("Action", 'LSel');

```

```

MyList->SetSelectionMode(new BMessage('ESel'));

//add all relevant items to the list...
//DState * temp;
for (int i = 0; i < External->CountItems(); i++) {
    MyList->AddItem(new DState((BStringItem*)External->ItemAt(i)));
}

this->AddChild(new BScrollView(".ListScroll", MyList,
B_FOLLOW_TOP_BOTTOM, 0, false, true));
BRect a = BRect(frame.Width()-70, frame.Height()-30, frame.Width()-8,
frame.Height()-11);
this->AddChild(btnApply = new
    BButton(a, BString(Nomen).Append(".btnEventApply").String(),
"Apply", new BMessage('bApl'))
);

a = BRect(140, 60, 250, 85);
this->AddChild(btnPull = new
    BButton(a, BString(Nomen).Append(".btnEventPull").String(), "Pull
Current Value", new BMessage('bPul'))
);

//Sumation
a = BRect(140, 40, 350, 65);
AddChild(chkSum = new BCheckBox(a, "chk_Sum", "Add this value to the
current value", NULL));

this->AddChild(txtValue = new BTextControl(BRect(140, 18, 300, 33),
"txtValue", NULL, "0.5", NULL));
txtValue->SetAlignment(B_ALIGN_LEFT, B_ALIGN_RIGHT);
}

EventManager::~EventManager() {
    BListItem * x;

    //free memory from displayed list items
    while ((x = MyList->RemoveItem((long int)0))) {
        delete x;
    }
}

void EventManager::SetActiveState(AState* State, BList * AList) {
    if (Window()) {
        Window()->Lock();
    }
    ActiveState = State;
    ActiveList = AList;
    for (int i = MyList->CountItems()-1; i >= 0; i--) { //clear stuff
being active or not
        ((DState*)MyList->ItemAt(i))->Active = false;
    }
    if (State == NULL) {
        btnApply->SetEnabled(false);
    } else {
        btnApply->SetEnabled(true);
        for (int i = MyList->CountItems()-1; i >= 0; i--) { //clear
stuff being active or not

```

```

        for (int j = ActiveList->CountItems()-1;j>=0;j--) { //
clear stuff being active or not
        cout << "T"<<i<<": ";
        cout << ((void*)((DState*)MyList->ItemAt(i))-
>Counterpart);
        cout << " vs ";
        cout << (void*)((Event*)ActiveList->ItemAt(j))-
>Target;
        bool x = (((DState*)MyList->ItemAt(i))->Counterpart
== ((Event*)ActiveList->ItemAt(j))->Target);
        cout << " res: " << x << endl;
        if (x) ((DState*)MyList->ItemAt(i))->Active=x; //
only set active, don't clear
        }
    }

    if (Window()) {
        MyList->Invalidate();
        Window()->Unlock();
    }
}

```

Filename: Requirement.cpp

```
#include "StateStuff.h"
#include "Component.h"
#include <stdio.h>
#include <stdlib.h>

Requirement::Requirement(AState* Dest, Requirement* Par) :
BStringItem("", (Par==NULL)?0:(Par->OutlineLevel()+1)), Destination
(Dest), Parent(Par) {
    reqType=255;
    SubItemCount=0;
    SubReadyCount=0;
    if (Parent) {
        Parent->SubItemCount++;
    }
    SourceChan=NULL;
}

void Requirement::SetLabel() {
    char Buffer[64];
    Buffer[0]=0;
    switch (reqType) {
        case REQ_CHANCE:
            snprintf(Buffer, 63, "%0.1f%% Chance", Probability/
2.0);
            break;
        case REQ_TIME:
            snprintf(Buffer, 63, "%0.2fs Timeout", TimeOut);
            break;
        case REQ_PULSE:
            if (SourceChan) {
                snprintf(Buffer, 63, "Pulse on %s", SourceChan-
>Text());
            } else {
                snprintf(Buffer, 63, "Pulse on ...");
            }
            break;
        case REQ_VALUE:
            if (SourceChan) {
                char a = '=';
                if (ActivationMode & VMODE_LESS) a = '<';
                if (ActivationMode & VMODE_GTR) a = '>';

                snprintf(Buffer, 63, "%s %c %0.2f", SourceChan-
>Text(), a, ActivateValue.f);
            } else {
                snprintf(Buffer, 63, "Broken Value");
            }
            break;
    }
    SetText(Buffer);
}

void Requirement::reset(bool full=false) {
    Ready=false;
}
```



```

        if (full) {
            PrevReady=false;
            SubReadyCount=0;
        }
        switch (reqType) {
            case REQ_CHANCE:
                Ready=((rand()%201)<=Probability);
                break;
            case REQ_TIME:
                StartCount=real_time_clock_usecs();
                break;
        }
    };
    AState * Requirement::TestReadiness(void * GenPtr) {
        switch (reqType) {
            case REQ_TIME:
                if (!Ready) { //once ready, no need to check any
more
                    if (real_time_clock_usecs()-StartCount >
TimeOut*1000000) Ready=true;
                }
                break;
            case REQ_PULSE:
                if (!Ready) {
//cout << "Ptr: "<<GenPtr<<" Local:
"<<SourceChan<<endl;
                    if (GenPtr==SourceChan) Ready=true;
                }
                break;
            case REQ_VALUE:
                {
                    Ready=false;
                    if ((ActivationMode & VMODE_GTR) && SourceChan-
>GetValue().f>ActivateValue.f) Ready=true;
                    if ((ActivationMode & VMODE_EQU) && SourceChan-
>GetValue().f>ActivateValue.f-0.01 && SourceChan->GetValue().
f<ActivateValue.f+0.01) Ready=true;
                    if ((ActivationMode & VMODE_LESS) &&
SourceChan->GetValue().f<ActivateValue.f) Ready=true;
                }
                break;
        }
        if (Parent) { //this is part of a list of requirements
            if (PrevReady==false && Ready==true && (SubItemCount==0
|| SubReadyCount>0)) {
                Parent->SubReadyCount++;
                PrevReady=true;
            } else if (PrevReady==true && (Ready==false || !
(SubItemCount==0 || SubReadyCount>0))) {
                Parent->SubReadyCount--;
                PrevReady=false;
            }
        } else if (Ready && (SubItemCount==0 || SubReadyCount>0)) {
//no parent and we are fully ready
            return Destination;
        }
        return NULL;
    };
};

```

```

void Requirement::Save(BPositionIO * object, AuroranComponent* owner,
BList * StateList, BList * ReqList, BList * Internals) {

    //first save the transition requirements to other states...
    unsigned short int usitemp;
    int itemp;

    // usitemp=Requirements.CountItems();
    object->Write(&usitemp,2); //placeholder
    int spos;
    spos=object->Position();
    cout <<"Position: " << (int32)object->Position() << endl;

    usitemp=StateList->IndexOf(Destination);
    cout <<"Which State: " << usitemp << endl;
    object->Write(&usitemp,2);

    usitemp=ReqList->IndexOf(Parent);
    cout <<"Which Parent: " << usitemp << endl;
    object->Write(&usitemp,2);

    object->Write(&reqType,1);

    switch (reqType) {
        case REQ_TIME:
            object->Write(&TimeOut,sizeof(TimeOut));
            break;
        case REQ_PULSE:

            usitemp=owner->GetSockets()->IndexOf(SourceChan);
            object->Write(&usitemp,2);
            break;
        case REQ_CHANCE:
            object->Write(&Probability,sizeof(Probability));
            break;
        case REQ_FAIL:
            usitemp=owner->GetSockets()->IndexOf(SourceChan);
            object->Write(&usitemp,2);
            break;
        case REQ_VALUE:
            object->Write(&ActivationMode,1);
            object->Write(&ActivateValue,sizeof(CHData));
            if (ActivationMode & VMODE_SOCKET) {
                usitemp=owner->GetSockets()->IndexOf(SourceChan);
            } else {
                usitemp=Internals->IndexOf(SourceChan);
            }
            object->Write(&usitemp,2);
            break;
    }

    int epos;
    epos = object->Position();
    usitemp=epos-spos;
    object->Seek(spos-2, SEEK_SET);

```

```

        object->Write(&usitemp,2); //placeholder
        cout << "Data block size: " << usitemp << endl;
        object->Seek(epos,SEEK_SET);
    }
Requirement * Requirement::Load(BPositionIO * object, AuroranComponent*
owner, BList * StateList,BList * ReqList, BList * Internals) {

    Requirement * MyReq;
    //
    unsigned short int usitemp;
    unsigned int segsize;

    int epos;
    cout <<"Position: " << (int32)object->Position() << endl;
    object->Read(&usitemp,2);
    cout << "Data block size: " << usitemp << endl;

    epos=object->Position()+usitemp;

    object->Read(&usitemp,2);
    cout << "Dest IDX: "<<usitemp << endl;
    AState * Destination=(AState*)StateList->ItemAt(usitemp);
    cout << "Dest State: "<<Destination->Text()<<endl;

    object->Read(&usitemp,2);
    cout <<"Which Parent: " << usitemp << endl;
    Requirement * Parent;
    if (usitemp>=0 && usitemp < 2048) {
        Parent=(Requirement*)ReqList->ItemAt(usitemp);
    } else {
        Parent=NULL;
    }

    MyReq=new Requirement(Destination,Parent);

    object->Read(&MyReq->reqType,1);

    switch (MyReq->reqType) {
        case REQ_TIME:
            object->Read(&MyReq->TimeOut,sizeof(MyReq->TimeOut));
            break;
        case REQ_PULSE:
            object->Read(&usitemp,2);
            MyReq->SourceChan=(ChannelListItem*)owner->GetSockets()-
>ItemAt(usitemp);
            break;
        case REQ_CHANCE:
            object->Read(&MyReq->Probability,sizeof(MyReq-
>Probability));
            break;
        case REQ_FAIL:
            object->Read(&usitemp,2);
            MyReq->SourceChan=(ChannelListItem*)owner->GetSockets()-
>ItemAt(usitemp);
            break;
        case REQ_VALUE:
            object->Read(&MyReq->ActivationMode,1);

```

```

        object->Read(&MyReq->ActivateValue, sizeof(CHData));
        object->Read(&usitemp, 2);
        if (MyReq->ActivationMode & VMODE_SOCKET) {
            MyReq->SourceChan=(ChannelListItem*) owner->GetSockets() -
>ItemAt(usitemp);
        } else {
            MyReq->SourceChan=(ChannelListItem*) Internals->ItemAt
(usitemp);
        }
        break;
    }

    object->Seek(epos, SEEK_SET);
    MyReq->SetLabel();
    return MyReq;
}

```

Filename: State.cpp

```
#include "StateStuff.h"
#include "Component.h"
#include <iostream.h>
#include <String.h>

AState::AState(const char * name) : BStringItem(name) {
    Active=false;
}

void AState::DrawItem(BView * owner,BRect frame,bool complete) {
    //cout << owner->fname << endl;
    if (Active) {
        owner->SetHighColor((rgb_color){0,0,0,0});
        owner->FillEllipse(BRect(frame.left+2,frame.top+2,frame.
left+frame.Height()-4,frame.top+frame.Height()-4));
        owner->SetFont(be_bold_font);
    } else {
        owner->SetFont(be_plain_font);
    }
    BStringItem::DrawItem(owner,BRect(frame.Height()-2,frame.top,frame.
right,frame.bottom),complete);
}

AState* AState::CheckState(void * GenPtr) {
    AState * temp;
    for (int i = Requirements.CountItems()-1;i>=0;i--) {
        temp=((Requirement*)Requirements.ItemAt(i))->TestReadiness
(GenPtr);
        if (temp) return temp;
    }
    return NULL;
}

void DoEvents(BList * Events) {
    for (int i = Events->CountItems()-1;i>=0;i--) {
        Event * ThisEvent = (Event*)Events->ItemAt(i);
        switch (ThisEvent->DoWhat) {
            case EVT_SETVAL:
                ThisEvent->Target->SetValue(ThisEvent->Value);
                cout << "Applying event for: " << ThisEvent->Target-
>Text() << " w/ Value: "<<ThisEvent->Value.f<<endl;;

                break;
            case EVT_ADD:
                { float v = ThisEvent->Target->GetValue().f;
                  v+=ThisEvent->Value.f;
                  if (v>ThisEvent->Target->GetMax().f) v = ThisEvent-
>Target->GetMax().f;
                  if (v<ThisEvent->Target->GetMin().f) v = ThisEvent-
>Target->GetMin().f;
                  ThisEvent->Target->SetValue((CHData){v});
                }
                break;
        }
    }
}
```

```

}

void AState::EnterState() {
    Active=true;
    //reset all requirements
    for (int i = Requirements.CountItems()-1;i>=0;i--) {
        ((Requirement*)Requirements.ItemAt(i))->reset(true);
    }

    //Do applicable events
    DoEvents(&EntryEvents);
}

void AState::LeaveState() {
    Active=false;

    //Do applicable events
    DoEvents(&ExitEvents);
}

void AState::SaveState(BPositionIO * object, AuroranComponent* owner,
BList * StateList, BList * Internals) {

    //first save the transition requirements to other states...
    unsigned short int usitemp,usitemp2;
    usitemp=Requirements.CountItems();
    object->Write(&usitemp,2);
    cout << "Req Count: "<<usitemp<<" Pos:"<<object->Position()<<endl;
    for (int i =0;i<usitemp;i++) {
        ((Requirement*)Requirements.ItemAt(i))->Save(object,owner,
StateList,&Requirements,Internals);
    }

    usitemp=EntryEvents.CountItems();
    object->Write(&usitemp,2);
    cout << "Entry Count: "<<usitemp<<" Pos:"<<object->Position()<<endl;
    for (int i =0;i<usitemp;i++) {
        Event * MyEvent=(Event*)EntryEvents.ItemAt(i);
        usitemp2=owner->GetPlugList()->IndexOf(MyEvent->Target);
        object->Write(&usitemp2,2);
        object->Write(&MyEvent->DoWhat,sizeof(EventType));
        object->Write(&MyEvent->Value,4);
    }

    usitemp=ExitEvents.CountItems();
    object->Write(&usitemp,2);
    cout << "Exit Count: "<<usitemp<<" Pos:"<<object->Position()<<endl;
    for (int i =0;i<usitemp;i++) {
        Event * MyEvent=(Event*)ExitEvents.ItemAt(i);
        usitemp2=owner->GetPlugList()->IndexOf(MyEvent->Target);
        object->Write(&usitemp2,2);
        object->Write(&MyEvent->DoWhat,sizeof(EventType));
        object->Write(&MyEvent->Value,4);
    }
}

```

```

}
void AState::LoadState(BPositionIO * object, AuroranComponent* owner,
BList * StateList, BList * Internals) {

    //
    unsigned short int usitemp,usitemp2;
    object->Read(&usitemp,2);
    cout << "Req Count: "<<usitemp<<" Pos:"<<object->Position()<<endl;
    for (int i =0;i<usitemp;i++) {
        Requirements.AddItem(Requirement::Load(object,owner,StateList,
&Requirements,Internals));
    }

    object->Read(&usitemp,2);
    cout << "Entry Count: "<<usitemp<<" Pos:"<<object->Position()<<endl;
    for (int i =0;i<usitemp;i++) {
        Event * MyEvent=new Event(NULL,EVT_UNDEF,(CHData){0.0});
        object->Read(&usitemp2,2);
        MyEvent->Target = (ChannelListItem*)owner->GetPlugList()-
>ItemAt(usitemp2);
        object->Read(&MyEvent->DoWhat,sizeof(EventType));
        object->Read(&MyEvent->Value,4);
        cout << MyEvent->Target->FullName().String() << endl;
        EntryEvents.AddItem(MyEvent);
    }

    object->Read(&usitemp,2);
    cout << "Exit Count: "<<usitemp<<" Pos:"<<object->Position()<<endl;
    for (int i =0;i<usitemp;i++) {
        Event * MyEvent=new Event(NULL,EVT_UNDEF,(CHData){0.0});
        object->Read(&usitemp2,2);
        MyEvent->Target = (ChannelListItem*)owner->GetPlugList()-
>ItemAt(usitemp2);
        object->Read(&MyEvent->DoWhat,sizeof(EventType));
        object->Read(&MyEvent->Value,4);
        cout << MyEvent->Target->FullName().String() << endl;
        ExitEvents.AddItem(MyEvent);
    }

}

```

Filename: StateMachine.cpp

```
#include "Component.h"
#include "ResourceSet.h"
// #include <stdio.h>
#include <Debug.h>
#include <iostream.h>
#include "PrettySlider.h"
#include "ChannelTools.h"
#include <stdlib.h>

//*****
//this resource management stuff is taken from one issue of the
BMessage newsletter...
//http://www.be.com/developers/bmessage/issue09.html
#include <Locker.h>
#include <TranslatorFormats.h>

static int32 gResourcesCreated = 0;
static BResourceSet gResources;

static BResourceSet& Resources()
{
    // Return the BResourceSet object for this image, initializing
    // if needed.
    if ((gResourcesCreated & 2) != 0) {
        // Quick return if already initialized.
        return gResources;
    } else if (atomic_or(&gResourcesCreated, 1) == 0) {
        // Not yet initialized -- do so.
        gResources.AddResources(&gResourcesCreated);
        // Mark that we are done.
        atomic_or(&gResourcesCreated, 2);
    } else {
        // Wait for resources to be initialized.
        while ((gResourcesCreated & 2) == 0) snooze(50000);
    }
    return gResources;
}

//*****

AuroranComponent * MakeComponent(ComponentDescriptor * Who,
AuroranParameters * Params);

#include "StateStuff.h"

static ComponentDescriptor MyDescriptor = {
    A_DESC_VER,
    A_NORMAL_DESC,
    NULL,
    "State Machine",
    'TKS_',
    'ST',
    0,
    ACC_CONTROL,
    &MakeComponent
```



```
};
```

```
class StateMachine : public AuroranComponent {
private:
    BList InternalValues;
    BTextControl * txtStateName;
    BTextView * txtStateNotes;
    BListView * lstStates;
    BListView * lstDst;
    BList States;
    AState * CState;

    //Does the displayed state list need updating?
    bool RefreshNeeded;
    bool isVisible;

    int8 CurrentReqType;
    BBox * ReqBox;
    BView * ReqView;
    BListView * lstReq;

    EventManager * enterView;
    EventManager * exitView;

    int DropListIndex;

    bool isOnline;
public:
    StateMachine(AuroranParameters * Parms) :
        AuroranComponent(Parms, DefaultCustom()),
        InternalValues(BList()),
        States(BList())
        //ReqChance(BView(BRect(),"Random",))
        {
            AutoSetColors();
            MyIcon=(BBitmap *)Resources().FindBitmap(B_PNG_FORMAT,
2001);

            ApplyLoc=AC_APPLY_HIDDEN;
            TheWorld->Pulser->AddClient(this,20.00);

            States.AddItem(CState= new AState("Start"));
            CState->Active=true;
            isVisible=false;

            RegisterSocket("Online",new BMessage('MkOL'));
            isOnline=true;
        }
    ~StateMachine() {printf("State Machine gesterbt\n");}
    BView * GetView(BRect frame) {
        TheWorld->Pulser->DeleteClient(TheWorld->ILoop);
        //TheWorld->Pulser->AddClient(TheWorld->ILoop,100.0);
        BView * TheView = AuroranComponent::GetView(frame);

        BTabView * megaTabView = new BTabView(BRect(0,25,frame.
Width(),235),"megatab");

        BBox * Box;
        BSlider * Slider;
```

```

        BMessage * msg;
        BView * view;
        BButton * b;
        BTextControl * Text;

        BTab * aTab;

        //aTab = new BTab();
        BView * statesView = new BView(megaTabView->Bounds(),
"States",B_FOLLOW_LEFT|B_FOLLOW_TOP,B_WILL_DRAW);
        statesView->SetViewColor(kViewGray);

        lstStates = new BListView(BRect(3,3,115,megaTabView-
>Bounds().Height()-megaTabView->TabHeight()-6),"lst_States",
B_SINGLE_SELECTION_LIST,B_FOLLOW_TOP_BOTTOM);
        lstStates->AddList(&States);
        lstStates->SetSelectionMessage(new BMessage('SlNS'));
        lstStates->SetInvocationMessage(new BMessage('GoSt'));
        lstStates->SetTarget(TheWorld->ILoop); //send to
interafce messenger
        statesView->AddChild(new BScrollView("scroll_States",
lstStates,B_FOLLOW_TOP_BOTTOM,0,false,true));

        BTabView * StatePropTabs = new BTabView(BRect(135,3,
frame.Width()-5,megaTabView->Bounds().Height()-megaTabView->TabHeight
()-6),"stateprops",B_WIDTH_FROM_WIDEST);

        BView * propView = new BView(megaTabView->Bounds(),
"Properties",B_FOLLOW_LEFT|B_FOLLOW_TOP,B_WILL_DRAW);
        propView->SetViewColor(kViewGray);

        propView->AddChild(new BStringView(BRect
(3,5,110,18),"lbl_Name","Name:"));
        propView->AddChild(txtStateName = new
BTextControl(BRect(2,18,266,33),"txtStateName", NULL,"",NULL));
        propView->AddChild(new BStringView(BRect
(3,43,110,53),"lbl_Notes","Notes:"));
        propView->AddChild(view=new BScrollView
("scroll_Notes",txtStateNotes = new BTextView(BRect(7,55,252,150),
"txtStateNotes",BRect(0,0,263,100),B_FOLLOW_ALL_SIDES,B_FRAME_EVENTS|
B_PULSE_NEEDED|B_WILL_DRAW),B_FOLLOW_TOP_BOTTOM,0,false,true));
        //txtStateNotes->MakeResizable(true);
        propView->AddChild(b=new BButton(BRect
(300,18,390,38),"btn_AddSt","Add State",new BMessage('AdSt')));
        b->SetTarget(TheWorld->ILoop);

        StatePropTabs->AddTab(propView);

        BView * nextView = new BView(megaTabView->Bounds(),
"Transitions",B_FOLLOW_LEFT|B_FOLLOW_TOP,B_WILL_DRAW);
        nextView->SetViewColor(kViewGray);

        nextView->AddChild(new BStringView(BRect
(3,3,110,15),"lbl_Dsts","Destination:"));
        lstDst = new BListView(BRect(3,18,110,
StatePropTabs->Bounds().Height()-megaTabView->TabHeight()-6),
"lst_States",B_SINGLE_SELECTION_LIST,B_FOLLOW_TOP_BOTTOM);
        lstDst->SetSelectionMessage(new BMessage

```

```

('SlND'));
lstDst->SetTarget(TheWorld->ILoop); //send to
interafce messenger
for (int i =0;i<States.CountItems();i++) {
    lstDst->AddItem(new DState(((AState*)States.
ItemAt(i))));
}

nextView->AddChild(new BScrollView
("scroll_Dsts",lstDst,B_FOLLOW_TOP_BOTTOM,0,false,true));

nextView->AddChild(new BStringView(BRect(135-
2,3,110+132-2,15),"lbl_Reqs","Requirements:"));
lstReq = new BOutlineListView(BRect(135-
2,18,110+132,StatePropTabs->Bounds().Height()-megaTabView->TabHeight()-
6),"lst_Requirements",B_SINGLE_SELECTION_LIST,B_FOLLOW_TOP_BOTTOM);
lstReq->SetSelectionMode(new BMessage
('SlNR'));
lstReq->SetTarget(TheWorld->ILoop); //send to
interafce messenger
nextView->AddChild(new BScrollView
("scroll_Req",lstReq,B_FOLLOW_TOP_BOTTOM,0,false,true));

//Set up requirement definition box

Box=new BBox(BRect(132+135-4,5,415,
StatePropTabs->Bounds().Height()-megaTabView->TabHeight()-6-30),
"RequireProps");

BPopupMenu * BPM = new BPopupMenu("Requirement
Type");
BMenuItem * BMI;
//BPM->SetFont(12);

//Time Out
msg = new BMessage('ChTP');
msg->AddInt8("Type",REQ_TIME);
BPM->AddItem(BMI = new BMenuItem("Time Out",
msg,0,0));
BMI->SetTarget(TheWorld->ILoop);

//Value
msg = new BMessage('ChTP');
msg->AddInt8("Type",REQ_VALUE);
BPM->AddItem(BMI = new BMenuItem("Value
Is...",msg,0,0));
BMI->SetTarget(TheWorld->ILoop);

//Pulse
msg = new BMessage('ChTP');
msg->AddInt8("Type",REQ_PULSE);
BPM->AddItem(BMI = new BMenuItem("Pulse
Recieved",msg,0,0));
BMI->SetTarget(TheWorld->ILoop);

//Random
msg = new BMessage('ChTP');
msg->AddInt8("Type",REQ_CHANCE);
BPM->AddItem(BMI = new BMenuItem("Random",

```

```

msg,0,0));

BMI->SetTarget(TheWorld->ILoop);

//Failure
msg = new BMessage('ChTP');
msg->AddInt8("Type",REQ_FAIL);
BPM->AddItem(BMI = new BMenuItem("Failure",
msg,0,0));

BMI->SetTarget(TheWorld->ILoop);

//Connected
msg = new BMessage('ChTP');
msg->AddInt8("Type",REQ_CONN);
BPM->AddItem(BMI = new BMenuItem("Is
Connected",msg,0,0));

BMI->SetTarget(TheWorld->ILoop);

CurrentReqType=-1;
BMenuItem * MenuOpt;
Box->SetLabel(MenuOpt = new BMenuItem(BRect
(10,0,150,15), BString("List_nmm").String(), NULL, BPM));

ReqView=NULL;
ReqBox=Box;
nextView->AddChild(Box);

int w=46;
int x=263+1;
nextView->AddChild(b=new BButton(BRect(x,
StatePropTabs->Bounds().Height()-megaTabView->TabHeight()-6-25,x+w,
StatePropTabs->Bounds().Height()-megaTabView->TabHeight()-6-5),
"btn_AddRe","Add",new BMessage('+Req')));
b->SetTarget(TheWorld->ILoop);
nextView->AddChild(b=new BButton(BRect(x+w+5,
StatePropTabs->Bounds().Height()-megaTabView->TabHeight()-6-25,x+w+w+5,
StatePropTabs->Bounds().Height()-megaTabView->TabHeight()-6-5),
"btn_RmRe","Del",new BMessage('-Req')));
b->SetTarget(TheWorld->ILoop);
nextView->AddChild(b=new BButton(BRect
(x+w+w+10,StatePropTabs->Bounds().Height()-megaTabView->TabHeight()-6-
25,x+w+w+w+10,StatePropTabs->Bounds().Height()-megaTabView->TabHeight
()-6-5),"btn_AplRe","Apply",new BMessage('*Req')));
b->SetTarget(TheWorld->ILoop);

StatePropTabs->AddTab(nextView);

//TheWorld->ILoop->Lock();

enterView = new EventManager(TheWorld->ILoop,
GetPlugList(),NULL,&InternalValues,BRect(0,0,StatePropTabs->Bounds().
Width(),StatePropTabs->Bounds().Height()-StatePropTabs->TabHeight()),
"On Entry");//new BView(megaTabView->Bounds(),"On Entry",B_FOLLOW_LEFT|
B_FOLLOW_TOP,B_WILL_DRAW);
enterView->SetViewColor(kViewGray);
//TheWorld->ILoop->AddHandler(enterView);
StatePropTabs->AddTab(enterView);

exitView = new EventManager(TheWorld->ILoop,
GetPlugList(),NULL,&InternalValues,BRect(0,0,StatePropTabs->Bounds().

```

```

Width(),StatePropTabs->Bounds().Height()-StatePropTabs->TabHeight()),
"On Exit");

        StatePropTabs->AddTab(exitView);

        statesView->AddChild(StatePropTabs);

        megaTabView->AddTab(statesView);

        BView * inView = new ChannelTools(this, CHT_SOCKETS|
        CHT_BUNDLES|CHT_ADVANCED|CHT_ENUMS|CHT_RATE, GetSockets(),
        (ChannelListItem*)GetSockets()->ItemAt(0),new BMessage('AddS'),new
        BMessage('AltS'),new BMessage('DelS'), NULL, BRect(0,0,megaTabView-
        >Bounds().Width(),megaTabView->Bounds().Height()-megaTabView->TabHeight
        ()), "Input",B_FOLLOW_LEFT|B_FOLLOW_TOP,B_WILL_DRAW);
        inView->SetViewColor(kViewGray);
        //inView->AddChild(new
        ChannelTools(BRect(5,5,600,300), "InTools",B_FOLLOW_LEFT|
        B_FOLLOW_TOP,0));
        megaTabView->AddTab(inView);

        //aTab = new BTab();
        BView * outView = new ChannelTools(this, CHT_PLUGS|
        CHT_BUNDLES|CHT_ADVANCED|CHT_RATE, GetPlugList(), ParentPlug(),new
        BMessage('AddP'),new BMessage('AltP'),new BMessage('DelP'), NULL, BRect
        (0,0,megaTabView->Bounds().Width(),megaTabView->Bounds().Height()-
        megaTabView->TabHeight()), "Output",B_FOLLOW_LEFT|B_FOLLOW_TOP,
        B_WILL_DRAW);
        outView->SetViewColor(kViewGray);
        //aTab->SetView(outView);
        megaTabView->AddTab(outView);

        //aTab = new BTab();
        BView * intView = new ChannelTools(this, CHT_ADVANCED|
        CHT_ENUMS, &InternalValues, NULL, NULL,NULL,NULL, NULL,BRect(0,0,
        megaTabView->Bounds().Width(),megaTabView->Bounds().Height()-
        megaTabView->TabHeight()), "Internal",B_FOLLOW_LEFT|B_FOLLOW_TOP,
        B_WILL_DRAW);
        intView->SetViewColor(kViewGray);
        //aTab->SetView(intView);
        megaTabView->AddTab(intView);

        TheView->AddChild(megaTabView);
        isVisible=true;
        RefreshNeeded=false;
        return TheView;
    }

    void OutOfSight() {
        //TheWorld->ILoop->Lock();
        // TheWorld->ILoop->RemoveHandler(exitView);
        // TheWorld->ILoop->RemoveHandler(enterView);
        //TheWorld->ILoop->Unlock();
        Lock(); //make sure no events happen that think we are
visable
        isVisible=false;
        Unlock();
    }

```

```

lstStates->DeselectAll();

if (ReqView) {
    ReqView->RemoveSelf();
    delete ReqView;
}

//clear temporary objects used for destination list
while (lstDst->ItemAt(0)) {
    delete lstDst->RemoveItem((long int)0);
}
}

BString ViewText(BMessage * Arch) {
    int32 idx=0;
    BMessage * Views = new BMessage();
    while (Arch->FindMessage("_views",idx++,Views)==B_OK) {
        //Views->PrintToStream();
        BString ViewName;
        if (Views->FindString("_name",&ViewName)==B_OK) {
            if (ViewName.Compare("_input")==0) {
                Views->FindString("_text",&ViewName);
                delete Views;
                return ViewName;
            }
        }
    }
    delete Views;
    return BString("0.0");
}

BString ViewMenu(BMessage * Arch) {
    int32 idx=0;
    BMessage * Views = new BMessage();
    cout << "AAAAAAAAAAa" << endl;
    Arch->PrintToStream();
    Arch->FindMessage("_views",0,Arch);
    cout << "CCCCCCCCC" << endl;
    Arch->PrintToStream();
    Arch->FindMessage("_items",0,Arch);
    cout << "QQQQQQQQQQQ" << endl;
    Arch->PrintToStream();
    Arch->FindMessage("_submenu",0,Arch);
    cout << "PPPPPPPPPP" << endl;
    Arch->PrintToStream();
    while (Arch->FindMessage("_items",idx++,Views)==B_OK) {
        cout << "ZZZZZZZZZZz" << endl;
        Views->PrintToStream();
        BString ViewName;
        if (Views->FindString("_name",&ViewName)==B_OK) {
            if (ViewName.Compare("_input")==0) {
                Views->FindString("_text",&ViewName);
                delete Views;
                return ViewName;
            }
        }
    }
    delete Views;
}

```

```

        return BString("0.0");
    }

    virtual void InterfaceMessage(BMessage * message) {
        void * ptemp;
        int itemp;
        int8 ctemp;

        //cout << "Interface Message!" << endl;
        switch (message->what) {
            case 'AddP':
                cout << "Plug added in WRONG PLACE" << endl;
                break;
            case 'CHTM':
                if (message->FindPointer("Owner",&ptemp)==B_OK) {
                    ((ChannelTools *)ptemp)->InterfaceMessage
(message);
                }
                break;

            case 'Refr': //refresh the list of states
                if (this->LockWithTimeout(1000)==B_OK) {
                    if (isVisable && RefreshNeeded) {
                        if (lstStates->Window()->LockWithTimeout(1000)
==B_OK) {

                            RefreshNeeded=false;
                            this->Unlock();

                            lstStates->Invalidate();

                            lstStates->Window()->Unlock();
                        } else {
                            this->Unlock();
                            TheWorld->Pulser->OneShot(TheWorld->ILoop,
new BMessage('Refr'));
                        }
                    } else this->Unlock();
                } else {
                    TheWorld->Pulser->OneShot(TheWorld->ILoop,new
BMessage('Refr'));
                }
                break;
            case 'AdSt':
                lstStates->Window()->Lock();
                AState * x;
                lstStates->AddItem(x = new AState(txtStateName-
>Text()));

                lstDst->AddItem(new DState(x));
                States.AddItem(x);
                lstStates->Window()->Unlock();
                break;
            case 'SlNS': //new state selected
                //clear active destinations before rebuilding them
                for (int i =0;i<lstDst->CountItems();i++) {
                    ((DState*)lstDst->ItemAt(i))->Active=false;
                }
                if (lstStates->Window()->LockWithTimeout(1000)
==B_OK) {

                    if (lstStates->CurrentSelection()>=0) {

```



```

TheState->Requirements.ItemAt(i));
        }
    }

    }
    lstStates->Window()->Unlock();
} else {
    TheWorld->Pulser->OneShot(TheWorld->ILoop,new
BMessage('SlND'));
    break;
}

case 'SlNR': //new requirement selected
    if (lstStates->Window()->LockWithTimeout(1000)
==B_OK) {
        Requirement * ThisReq;
        if ((ThisReq=(Requirement*)lstReq->ItemAt
(lstReq->CurrentSelection()))==NULL) {
            lstStates->Window()->Unlock();
            break; //no need to continue
        }
        message->AddInt8("Type", (ThisReq)->reqType);
        lstStates->Window()->Unlock();
    } else {
        TheWorld->Pulser->OneShot(TheWorld->ILoop,new
BMessage('SlNR'));
    }
    //note the lack of a "break"
case 'ChTP':
    if (lstStates->Window()->LockWithTimeout(1000)
==B_OK) {
        if (ReqView) {
            ReqView->RemoveSelf();
            delete ReqView;
        }
        ReqView = new BView(BRect(5,21,146,115),
"ReqControls",B_FOLLOW_ALL, 0);
        ReqView->SetViewColor(kViewGray);

        BTextControl * Text;
        BSlider * Slide;

        Requirement * Req = NULL;

        if (lstReq->CurrentSelection()>=0) {
            Req=(Requirement*)lstReq->ItemAt(lstReq-
>CurrentSelection());
        }

        if (message->FindInt8("Type",&ctemp)==B_OK) {
            cout << "Type changed to " << ctemp*1.0 <<

endl;

            switch (ctemp) {
                case REQ_CHANCE:
                    //Build Requirement option children
                    Slide = new BSlider(BRect
(0,15,141,100),"slide_Chance","Chance:",NULL,1,199,B_TRIANGLE_THUMB,
B_FOLLOW_LEFT|B_FOLLOW_TOP,B_FRAME_EVENTS|B_WILL_DRAW|B_NAVIGABLE);

```

```

Slide->SetBarThickness(6);
Slide->SetHashMarkCount(11);
Slide->SetHashMarks

(B_HASH_MARKS_TOP);

Slide->SetLimitLabels("1%", "99%");
Slide->SetBarColor((rgb_color)

{152,152,255,255});

if (Req) {
    Slide->SetValue(Req-
>Probability);
}
ReqView->AddChild(Slide);
break;
case REQ_VALUE: {
    //Build Timeout
    ReqView->AddChild(new BStringView
(BRect(0,5+40,110,18+40),"lbl_Name","Value:"));
    if (Req) {
        char Buffer[64];
        snprintf(Buffer,63," %.4f",
Req->ActivateValue.f);
        if (Req-
>ActivationMode&VMODE_LESS) Buffer[0]='<';
        if (Req-
>ActivationMode&VMODE_GTR) Buffer[0]='>';
        if (Req-
>ActivationMode&VMODE_EQU) Buffer[0]='=';
        ReqView->AddChild(Text = new
BTextControl(BRect(0,18+40,140,33+40),"txtVL", NULL,Buffer,NULL));
    } else {
        ReqView->AddChild(Text = new
BTextControl(BRect(0,18+40,140,33+40),"txtVL", NULL,"1.0",NULL));
    }
    Text->SetAlignment(B_ALIGN_LEFT,
B_ALIGN_RIGHT);
    //ReqView->AddChild(new BStringView
(BRect(0,35,140,52),"lbl_Name","
Decimals OK"));

    DropDownListIndex=-1;
    ReqView->AddChild(new BStringView
(BRect(0,5,110,18),"lbl_Name","Source:"));
    BPopUpMenu * BPM = new BPopUpMenu
("Pulse Source");

    BMenuItem * BMI;

    //Time Out
    for (int i=1; i < GetSockets

()->CountItems();i++) {
        BMessage * amsg = new
BMessage('DDNS');
        amsg->AddInt32("Index",
i);
        BPM->AddItem(BMI = new
BMenuItem(((ChannelListItem*)GetSockets()->ItemAt(i))->Text(),
amsg,0,0));
        BMI->SetTarget
(TheWorld->ILoop);
        if (Req && Req-
>SourceChan==(ChannelListItem*)GetSockets()->ItemAt(i)) BMI->SetMarked

```

```

(true);

}

BMenuField * MenuOpt;
ReqView->AddChild(MenuOpt = new
BMenuField(BRect(0,18,140,33),
"List_Source", NULL, BPM));

}
break;
case REQ_TIME:
//Build Timeout
ReqView->AddChild(new BStringView
(BRect(0,5,110,18),"lbl_Name","Time Out (seconds:)");
if (Req) {
char Buffer[64];
snprintf(Buffer,63,"%f",
Req->TimeOut);
ReqView->AddChild(Text = new
BTextControl(BRect(0,18,140,33),"txtTO", NULL,Buffer,NULL));
} else {
ReqView->AddChild(Text = new
BTextControl(BRect(0,18,140,33),"txtTO", NULL,"1.0",NULL));
}
Text->SetAlignment(B_ALIGN_LEFT,
B_ALIGN_RIGHT);
ReqView->AddChild(new BStringView
(BRect(0,35,140,52),"lbl_Name","Decimals OK"));
break;
case REQ_PULSE:
DropListIndex=-1;
ReqView->AddChild(new BStringView
(BRect(0,5,110,18),"lbl_Name","Source:"));
BPopupMenu * BPM = new BPopupMenu
("Pulse Source");
BMenuItem * BMI;

//Time Out
for (int i=1; i < GetSockets
()->CountItems();i++) {
BMessage * amsg = new
BMessage('DDNS');
amsg->AddInt32("Index",
i);
BPM->AddItem(BMI = new
BMenuItem(((ChannelListItem*)GetSockets()->ItemAt(i))->Text(),
amsg,0,0));
BMI->SetTarget
(TheWorld->ILoop);
if (Req && Req-
>SourceChan==(ChannelListItem*)GetSockets()->ItemAt(i)) BMI->SetMarked
(true);
}

BMenuField * MenuOpt;
ReqView->AddChild(MenuOpt = new
BMenuField(BRect(0,18,140,33), "List_Source", NULL, BPM));

break;

```

```

        }
        ReqBox->AddChild(ReqView);
        CurrentReqType=ctemp;
    }
    lstStates->Window()->Unlock();
} else {
    cout << "Attempt to lock interface looper
FAILED!" << endl;
    TheWorld->Pulser->OneShot(TheWorld->ILoop,new
BMessage(*message));
}
break;
case 'DDNS': //something selected from one of those drop
down list menus
    if (message->FindInt32("Index", (int32 *)&itemp)
==B_OK) {
        DropListIndex=itemp;
    }
    break;
case '+Req':
    if (lstStates->Window()->LockWithTimeout(1000)
==B_OK) {
        if (lstStates->CurrentSelection() >=0 &&
lstDst->CurrentSelection() >=0 && CurrentReqType >= 0) {
            if (this->LockWithTimeout(1000)==B_OK) {
                cout << "Adding new requirement..." << endl;
                Requirement * nReq;
                //nothing is selected, so add a base-level
requirement
                if (lstReq->CurrentSelection() < 0) {
                    lstReq->AddItem(nReq=new Requirement
((AState*)lstStates->ItemAt(lstDst->CurrentSelection()),NULL));
                    ((AState*)lstStates->ItemAt(lstStates-
>CurrentSelection()))->Requirements.AddItem(nReq);
                } else {
                    Requirement * Parent = (Requirement*)
lstReq->ItemAt(lstReq->CurrentSelection());
                    lstReq->AddItem(nReq=new Requirement
((AState*)lstStates->ItemAt(lstDst->CurrentSelection()),Parent),lstReq-
>IndexOf(Parent)+1);
                    ((AState*)lstStates->ItemAt(lstStates-
>CurrentSelection()))->Requirements.AddItem(nReq, ((AState*)lstStates-
>ItemAt(lstStates->CurrentSelection()))->Requirements.IndexOf(Parent)
+1);
                }
                ((DState*)lstDst->ItemAt(lstDst-
>CurrentSelection()))->Active=true;
                lstDst->InvalidateItem(lstDst-
>CurrentSelection());
                lstReq->Select(lstReq->IndexOf(nReq));
                this->Unlock();
            } else {
                TheWorld->Pulser->OneShot(TheWorld->ILoop,
new BMessage(*message));
                break;
            }
        } else {
            lstStates->Window()->Unlock();
            break;
        }
    }
}

```

```

        }
        lstStates->Window()->Unlock();
    } else {
        TheWorld->Pulser->OneShot(TheWorld->ILoop,new
BMessage(*message));
        break;
    }
    //Note the lack of a "break", this is deliberate
    case '*Req':
        if (lstStates->CurrentSelection() >=0 && lstDst-
>CurrentSelection() >=0 && lstReq->CurrentSelection() >= 0 &&
CurrentReqType>=0) {
            if (lstStates->Window()->LockWithTimeout(1000)
==B_OK) {
                if (this->LockWithTimeout(1000)==B_OK) {
                    printf("Apply Changes!\n");

                    Requirement * Req = (Requirement*)lstReq-
>ItemAt(lstReq->CurrentSelection());

                    BMessage * Views = new BMessage();
                    BMessage * Arch = new BMessage();

                    ReqView->Archive(Arch,true);

                    int32 idx=0;
                    while (Arch->FindMessage("_views",idx++,
Views)==B_OK) {
                        cout << "View found @ IDX" << idx <<
endl;

                        Views->PrintToStream();
                        BString ViewName;
                        if (Views->FindString("_name",
&ViewName)==B_OK) {
                            //ProcessView(ViewName,Views);
                            if (ViewName.Compare
("slide_Chance")==0) {
                                Views->FindInt32("_val",&Req-
>Probability);
                                cout << "Process ing
Probability: " << Req->Probability << endl;
                                } else if (ViewName.Compare
("txtTO")==0) {
                                    ViewName=ViewText(Views);
                                    sscanf(ViewName.String(),"%
f",&Req->TimeOut);
                                    cout << "Text:" << ViewName.
String() << " Value: " << Req->TimeOut << endl;
                                } else if (ViewName.Compare
("txtVL")==0) {
                                    ViewName=ViewText(Views);
                                    bool inc=false;
                                    if (ViewName.String()[0]
=='<') {
                                        Req-
                                        inc=true;
                                    }
                                    if (ViewName.String()[0]

```

```

== '>') {
    >ActivationMode=VMODE_GTR|VMODE_SOCKET;
    Req-
    inc=true;
}
if (ViewName.String()[0]
== '=') {
    Req-
    inc=true;
}
sscanf(ViewName.String()+
(inc?1:0), "%f", &Req->ActivateValue);
cout << "Text:" << ViewName.
String() << " Value: " << Req->ActivateValue.f << endl;
} else if (ViewName.Compare
("List_Source")==0) {
    //ViewName=ViewMenu(Views);
    //sscanf(ViewName.String(), "%f", &Req->TimeOut);
    cout << "Selected Item: " <<
    DropListIndex << endl;
    if (DropListIndex>=0) {
        Req->SourceChan=
        (ChannelListItem*)GetSockets()->ItemAt(DropListIndex);
    }
}
//delete Views;
//Views=new BMessage();
delete Views;
printf("Apply'd Changes!\n");
Req->reqType=CurrentReqType;
Req->SetLabel();
//reset this item to make sure things flow
right...
Req->reset();
lstReq->InvalidateItem(lstReq->
>CurrentSelection());
this->Unlock();
} else {
    TheWorld->Pulser->OneShot(TheWorld->ILoop,
new BMessage('*Req'));
}
lstStates->Window()->Unlock();
} else {
    TheWorld->Pulser->OneShot(TheWorld->ILoop,
new BMessage('*Req'));
}
}
break;
case 'GoSt': //Jump to a specific state
    if (lstStates->Window()->LockWithTimeout(1000)

```

```

==B_OK) {
    if (lstStates->CurrentSelection() >=0) {
        if (this->LockWithTimeout(1000)==B_OK) {

            CState->LeaveState(); //leave the current
state
            CState=(AState*)lstStates->ItemAt(lstStates-
>CurrentSelection());
            CState->EnterState();

            lstStates->Invalidate();

            this->Unlock();
        } else {
            TheWorld->Pulser->OneShot(TheWorld->ILoop,
new BMessage(*message));
        }
    }
    lstStates->Window()->Unlock();
} else {
    TheWorld->Pulser->OneShot(TheWorld->ILoop,new
BMessage(*message));
}
break;
default:
    AuroranComponent::InterfaceMessage(message);
}
}

virtual void MessageReceived(BMessage * message) {

    switch (message->what) {
        case ('MkOL'):
        {
            float Value;
            if (message->FindFloat("value",&Value)==B_OK) {
                isOnline=(Value>=0.5);
                if (Value==1.0) {
                    CState->LeaveState();
                    CState=(AState*)States.ItemAt(0);
                    CState->EnterState();
                }
            }
        }
        break;
        case ('SNew'):
        case (A_PULSE):
            if (!isOnline) {
                void * Ptr = NULL;
                message->FindPointer("SocketPtr",&Ptr);
                AState * Next = CState->CheckState(Ptr);
                if (Next) {
                    CState->LeaveState();
                    CState=Next;
                    CState->EnterState();

                    if (isVisable==true && RefreshNeeded==false)
{
                        RefreshNeeded=true;

```

```

TheWorld->Pulser->OneShot(TheWorld-
>ILoop,new BMessage('Refr'));
    }
    }
    if (message->what==A_PULSE) TheWorld->Pulser->GotIt
(this);
    break;
case ('AddP'): //plug added
    if (isVisable) {
        cout << "Got add plug message" << endl;
        ChannelListItem * NewPlug;
        message->FindPointer("Item", (void**) &NewPlug);
        int Loc = GetPlugList()->IndexOf(NewPlug);
        exitView->AddItem(NewPlug, Loc);
        enterView->AddItem(NewPlug, Loc);
    }
    break;

    }
    AuroranComponent::MessageReceived(message);
}
virtual ComponentDescriptor * GetDesc() {return &MyDescriptor;}

status_t Save(BPositionIO * object) {
    //Plugs
    SaveChannelList(object, this, GetPlugList(), ParentPlug(),
NULL, 0);

    //Sockets
    SaveChannelList(object, this, GetSockets(),
(ChannelListItem*)GetSockets()->ItemAt(0), NULL, 0);

    //Internals
    SaveChannelList(object, this, &InternalValues, NULL, NULL, 0);

    //Save States
    int itemp;
    unsigned short int usitemp;
    usitemp=itemp=States.CountItems();
    object->Write(&usitemp, 2);
    for (int i =0;i<itemp;i++) {

        usitemp=strlen(((AState*)States.ItemAt(i))->Text());
        object->Write(&usitemp, 2);
        object->Write(((AState*)States.ItemAt(i))->Text(),
usitemp);

    }
    cout << "Afdter States: " << (int32) object->Position() << endl;
    for (int i =0;i<itemp;i++) {

        ((AState*)States.ItemAt(i))->SaveState(object, this,
&States, &InternalValues);

    }
}

```



```

        return B_OK;
    }

    status_t Load(BPositionIO * object, uint32 version) {
        //Plugs
        LoadChannelList(object, this, GetPlugList(), ParentPlug(),
NULL, 0);

        //Sockets
        LoadChannelList(object, this, GetSockets(),
(ChannelListItem*)GetSockets()->ItemAt(0), NULL, 0);

        //Internals
        LoadChannelList(object, this, &InternalValues, NULL, NULL,
0);

        //Load States
        LockLooper();
        delete
States.RemoveItem((int32)0); //delete the start state briefly
        int itemp;
        unsigned short int usitemp;
        char StringBufferA[256];
        object->Read(&usitemp, 2);
        itemp=usitemp;
        for (int i =0;i<itemp;i++) {

            object->Read(&usitemp, 2);
            if (usitemp>=127) return B_ERROR;
            object->Read(StringBufferA, usitemp);
            StringBufferA[usitemp]=0;
            States.AddItem(new AState(StringBufferA));

        }
        cout << "Afdter States: "<<(int32)object->Position()<<endl;
        for (int i =0;i<itemp;i++) {

            ((AState*)States.ItemAt(i))->LoadState(object, this,
&States, &InternalValues);

        }
        CState = (AState*)States.ItemAt(0); //set active state
        CState->Active=true;
        UnlockLooper();

        return B_OK;
    }

protected:
    virtual char * MyName() {return "State Machine";}
    virtual char * DefaultCustom() {return "State";}
};

AuroranComponent * MakeComponent(ComponentDescriptor * Who,
AuroranParameters * Params) {return new StateMachine(Params);}

extern "C" _EXPORT ComponentDescriptor* GetDesc();

```

```
ComponentDescriptor* GetDesc() {  
    return &MyDescriptor;  
}
```

Filename: Mixer.cpp

```
#include "Component.h"
#include "ResourceSet.h"
// #include <stdio.h>
#include <Debug.h>
#include <iostream.h>
#include "PrettySlider.h"

//*****
//this resource management stuff is taken from one issue of the
BMessage newsletter...
//http://www.be.com/developers/bmessage/issue09.html
#include <Locker.h>
#include <TranslatorFormats.h>

static int32 gResourcesCreated = 0;
static BResourceSet gResources;

static BResourceSet& Resources()
{
    // Return the BResourceSet object for this image, initializing
    // if needed.
    if ((gResourcesCreated&2) != 0) {
        // Quick return if already initialized.
        return gResources;
    } else if (atomic_or(&gResourcesCreated, 1) == 0) {
        // Not yet initialized -- do so.
        gResources.AddResources(&gResourcesCreated);
        // Mark that we are done.
        atomic_or(&gResourcesCreated, 2);
    } else {
        // Wait for resources to be initialized.
        while ((gResourcesCreated&2) == 0) snooze(50000);
    }
    return gResources;
}

//*****

AuroranComponent * MakeComponent(ComponentDescriptor * Who,
AuroranParameters * Params);

static ComponentDescriptor MyDescriptor = {
    A_DESC_VER,
    A_NORMAL_DESC,
    NULL,
    "Mixer",
    'TKS_',
    'MX',
    0,
    ACC_PASSIVE,
    &MakeComponent
};

const int8 SAVE_Values = 1;
```

```

class SimpleMixer : public AuroranComponent {
private:
    SocketListItem * Values[40];
    SocketListItem * Masters[4];
    SocketListItem * Master;
    PlugListItem * CHPlugs[10];
    PlugListItem * SCPlugs[4];
    PlugListItem * MasterPlug;
    BSlider * Slide[20];
    uint32 OldSlides;
    uint8 MyFlags;

    int SceneA,SceneB;

    bool AmVisable;
    bool PendingUpdate;
public:
    SimpleMixer(AuroranParameters * Pams) : AuroranComponent
(Pams, DefaultCustom()) {
        AutoSetColors();
        MyIcon=(BBitmap *)Resources().FindBitmap(B_PNG_FORMAT,
2001);

        AmVisable=false;
        BMessage * mt;

        SceneA=0;
        SceneB=1;

        RegisterSocket("Online",StdChan_Online,new BMessage());
        Master = RegisterSocket("MasterBrightness",
StdChan_Intensity,mt = new BMessage('dScn'));
        mt->AddInt32("Scene",-1);
        //Master->SetValue((CHData){0.5});
        for (int scene = 0;scene<4;scene++) {
            char SName[9] = {'S','c','e','n','e',' ',' ','A'+scene,'
',0};

            Masters[scene] = RegisterSocket(BString(SName).Append
("Fade").String(),StdChan_Intensity,mt = new BMessage('dScn'));
            mt->AddInt32("Scene",scene);
            BeginSBundle(SName);

            char name[2] = {'0',0};
            for (int i=0;i<10;i++) {
                //Values[i]=0.0;
                Values[scene*10+i] = RegisterSocket(BString("CH ").
Append(name).String(),StdChan_Intensity,mt = new BMessage('dltA'));
                mt->AddInt32("Chan",i);
                mt->AddInt32("Scene",scene);

                name[0]++;
            }

            EndSBundle();
        }

        RegisterPlug("Online",StdChan_Online);
        MasterPlug=RegisterPlug("Master Brightness",
StdChan_Intensity);

```

```

        SCPlugs[0]=RegisterPlug("Scene A Fader",StdChan_Intensity);
        SCPlugs[1]=RegisterPlug("Scene B Fader",StdChan_Intensity);
        SCPlugs[2]=RegisterPlug("Scene C Fader",StdChan_Intensity);
        SCPlugs[3]=RegisterPlug("Scene D Fader",StdChan_Intensity);
        BeginPBundle("Output Scene");
        char name[2] = {'0',0};
        for (int i=0;i<10;i++) {
            //Values[i]=0.0;
            CHPlugs[i]=RegisterPlug(BString("CH ").Append(name).
String(),StdChan_Intensity);
            name[0]++;
        }
        EndPBundle();
        Master->SetValue((CHData){0.5});
    }
    ~SimpleMixer() {printf("4CH Mixer going to sleep\n");}
    BView * GetView(BRect frame) {
        TheWorld->Pulser->DeleteClient(TheWorld->ILoop);
        //TheWorld->Pulser->AddClient(TheWorld->ILoop,100.0);
        BView * TheView = AuroranComponent::GetView(frame);
        char name[4] = {'0','0',0,'C'};
        int i=0;

        BBox * Box;
        BSlider * Slider;
        BMessage * msg;

        //Scene Faders
        Box=new BBox(BRect(0,25,140,200),"Scene_Faders");
        Box->SetLabel("Scene Faders");
        for (int scene = 0;scene<4;scene++) {
            char SName[2] = {'A'+scene,0};

            Slider = new PrettySlider(BRect
(scene*30+15,15,35+scene*30,165),BString("Slide ").Append(SName).String
(),SName,NULL,0,511,B_VERTICAL,B_BLOCK_THUMB,B_FOLLOW_TOP|
B_FOLLOW_LEFT,B_FRAME_EVENTS|B_WILL_DRAW|B_NAVIGABLE);

            Slider->SetBarThickness(5);
            Slider->SetHashMarks(B_HASH_MARKS_BOTH);
            Slider->SetHashMarkCount(11);
            Slider->SetValue(Masters[scene]->GetValue().f*511);
            msg=new BMessage('ChgC');
            msg->AddPointer("Slider",Slider);
            msg->AddInt32("Scene",scene);
            Slider->SetModificationMessage(msg);
            Slider->SetTarget(TheWorld->ILoop);
            //cout << "Snooze Amount: " << Slider->SnoozeAmount() <<

endl;
        //
            Slider->SetBarColor((rgb_color){45,255,128,255});
            Box->AddChild(Slider);
            Slide[scene] = Slider;
        }
        TheView->AddChild(Box);

        //Cross Fader
        Box=new BBox(BRect(155,25,300,200),"XFade");
        Box->SetLabel("CrossFader");

```

```

BPopupMenu * BPM;
BMenuItem * Sig;

for (int s = 0 ; s<2;s++) {
    char XName[2] = {'X'+s,0};
    BPM = new BPopupMenu("Scene Sel");
    BPM->SetFontSize(12);

    for (int scene = 0;scene<4;scene++) {
        char SName[2] = {'A'+scene,0};
        msg = new BMessage('ChSC');
        msg->AddInt8("Side",s);
        msg->AddInt8("Scene",scene);
        BPM->AddItem(Sig = new BMenuItem(BString("Scene ").
Append(SName).Append(" ").String(),msg,0,0));
        Sig->SetTarget(TheWorld->ILoop);
        if ((s==0 && SceneA==scene) || (s==1 &&
SceneB==scene)) Sig->SetMarked(true);
    }

    BMenuField * MenuOpt;
    Box->AddChild(MenuOpt = new BMenuField(BRect
(10,15+s*130,100,30+s*130), BString("List_").Append(XName).String(),
BString(XName).Append(":").String(), BPM));
    MenuOpt->SetFontSize(12);
    MenuOpt->SetDivider(20);
}

Slider = new BSlider(BRect(15+95,20,35+100,165),
"Slide_XF",NULL,NULL,0,511,B_VERTICAL,B_TRIANGLE_THUMB,B_FOLLOW_TOP|
B_FOLLOW_LEFT,B_FRAME_EVENTS|B_WILL_DRAW|B_NAVIGABLE);

Slider->SetBarThickness(5);
Slider->SetHashMarks(B_HASH_MARKS_BOTH);
Slider->SetHashMarkCount(11);
if (SceneA>=0 && SceneA<=3 && SceneB>=0 &&SceneB<=3) {
    Slider->SetValue(511*(1-Masters[SceneB]-
>GetValue().f+Masters[SceneA]->GetValue().f)/2.0);
}
msg=new BMessage('ChgX');
msg->AddPointer("Slider",Slider);
//msg->AddInt32("Scene",scene);
Slider->SetModificationMessage(msg);
Slider->SetTarget(TheWorld->ILoop);
//cout << "Snooze Amount: " << Slider->SnoozeAmount() <<
endl;

Slider->SetBarColor((rgb_color){152,152,255,255});
Box->AddChild(Slider);
Slide[4] = Slider;

TheView->AddChild(Box);

//Master Faster
Box=new BBox(BRect(315,25,375,200),"Master_Faders");
Box->SetLabel("Master");
Slider = new PrettySlider(BRect(10,20,45,165),

```

```

"Slide_Master",NULL,NULL,0,511,B_VERTICAL,B_BLOCK_THUMB,B_FOLLOW_TOP|
B_FOLLOW_LEFT,B_FRAME_EVENTS|B_WILL_DRAW|B_NAVIGABLE);

Slider->SetBarThickness(5);
Slider->SetHashMarks(B_HASH_MARKS_BOTH);
Slider->SetHashMarkCount(11);
Slider->SetValue(Master->GetValue().f*511);
msg=new BMessage('ChgC');
msg->AddPointer("Slider",Slider);
msg->AddInt32("Scene",(-1));
Slider->SetModificationMessage(msg);
Slider->SetTarget(TheWorld->ILoop);
//cout << "Snooze Amount: " << Slider->SnoozeAmount() <<

endl;
// Slider->SetBarColor((rgb_color){45,255,128,255});
Box->AddChild(Slider);
Slide[5] = Slider;
TheView->AddChild(Box);

BControl * NC;
//Save to disk checkbox
TheView->AddChild(NC = new BCheckBox(BRect
(2,36+178,400,50+178),"chk_Save","Save Scene Values to Disk",NULL));
if ((MyFlags & SAVE_Values)) NC->SetValue(B_CONTROL_ON);

AmVisable=true;
PendingUpdate=false;
return TheView;
}

void OutOfSight() {
AmVisable=false;
TheWorld->Pulser->DeleteClient(TheWorld->ILoop);
for (int i =0 ;i < 20;i++) {
Slide[i]=NULL;
}
}

virtual void InterfaceMessage(BMessage * message) {
switch (message->what) {
//case A_PULSE:
case 'ChSC': {
int8 Scene, Side;
PRINT(("Scene Change\n"));
if (message->FindInt8("Scene",&Scene)==B_OK) {
if (message->FindInt8("Side",&Side)==B_OK) {
if (Side==0) {
SceneA=Scene;
} else {
SceneB=Scene;
}
}
}
//NO BREAK!!!
}
case 'ChgX': {

```

```

        PRINT(("CROSSFADE!\n"));
        if (SceneA>=0 && SceneA<=3 && SceneB>=0
&&SceneB<=3) {
            Masters[SceneB]->SetValue((CHData){1.0-Slide
[4]->Value()/511.0});
            Masters[SceneA]->SetValue((CHData){Slide[4]-
>Value()/511.0});
        }
        break;
    }
    case 'ChgC': {
        int32 Scene;
        BSlider * Slider;
        PRINT(("Updated Interface....\n"));
        if (message->FindInt32("Scene",&Scene)==B_OK) {
            if (message->FindPointer("Slider",((void**)
&Slider))==B_OK) {
                if (Scene >= 0 && Scene <4)
                    Masters[Scene]->SetValue((CHData){Slider-
>Value()/511.0});
                else if (Scene == (-1))
                    Master->SetValue((CHData){Slider->Value()/
511.0});
            }
        }
        break;
    }
    case 'Ivld': {
        if (AmVisable) {
            PendingUpdate=false;
            if (Slide[0]->Window()->LockWithTimeout(1000)
==B_OK) { //make sure we don't deadlock!
                for (int scene = 0;scene<6;scene++) {
                    if (scene!=4) {
                        //printf("Scene %d changes: %d?\n",
scene,OldSlides);
                        Slide[scene]->SetTarget(NULL); //
disable sending messages
                        if ((OldSlides & (1<<scene))) {
                            OldSlides &= (~(1<<scene));
                            if (scene==5) {
                                Slide[scene]->SetValue
(Master->GetValue().f*511);
                            } else {
                                Slide[scene]->SetValue
(Masters[scene]->GetValue().f*511);
                            }
                        }
                        Slide[scene]->SetTarget(TheWorld-
>ILoop); //disable sending messages
                    }
                }

                //Master
                //Slide[5]->SetTarget(NULL); //disable
sending messages
                //Slide[5]->SetValue(Master->GetValue().
f*511);
            }
        }
    }
}

```



```

//Slide[5]->SetTarget(TheWorld->ILoop); //
disable sending
messages

        Slide[0]->Window()->Unlock();
    } else {
        TheWorld->Pulser->OneShot(TheWorld->ILoop,
new BMessage('Ivld')); //if attemp to lock failed, try again later
    }
}
break;
}
default:
    AuroranComponent::InterfaceMessage(message);
}
}

float GetChan(int w, int a) {
    PRINT(("Getting %d...\n",w+a*10));
    return Values[w+a*10]->GetValue().f*Masters[a]->GetValue().
f;
}

void ProcessChannel(int w) {
    float output = GetChan(w,0)+GetChan(w,1)+GetChan(w,2)
+GetChan(w,3);
    if (output > 1.0) output = 1.0;
    if (Master->GetValue().f < 0.5) {
        output*=Master->GetValue().f*2;
    } else if (Master->GetValue().f > 0.5) {
        output=1.0-(1.0-Master->GetValue().f)*(1.0-output)*2;
    }
    PRINT(("Sending result: %f\n",output));
    CHPlugs[w]->SendMessage(output);
}

virtual void MessageReceived(BMessage * message) {
    int32 Scene;
    int32 Chan;
    switch (message->what) {
        case ('dlta'): {
            PRINT(("New Data recieved"));
            if (message->FindInt32("Scene",&Scene)==B_OK) {
                if (message->FindInt32("Chan",&Chan)==B_OK) {
                    PRINT((" for ch %d.\n",Chan));
                    Values[Chan+Scene*10]->GetValueAndClean();
                    if (Chan >=0 && Chan <10) ProcessChannel(Chan);
                }
            }
            PRINT((" \n"));
            break;
        }
        case ('dScn'): {
            PRINT(("New Scene Data recieved"));
            if (message->FindInt32("Scene",&Scene)==B_OK) {
                switch (Scene) {
                    case (-1):
                        MasterPlug->SetValue(Master-

```

```

>GetValueAndClean());
                                OldSlides|=(1<<5);
                                break;
                                case (0):
                                case (1):
                                case (2):
                                case (3):
                                SCPlugs[Scene]->SetValue(Masters
[Scene]->GetValueAndClean());
                                OldSlides|=1<<Scene;
                                break;
                                }
                                if (AmVisable && !PendingUpdate) {
                                    PendingUpdate=true;
                                    TheWorld->ILoop->PostMessage(new BMessage
('Ivld'));
                                }
                                for (Chan =0;Chan<10;Chan++) { //update all
channels
                                    ProcessChannel(Chan);
                                }
                                }
                                break;
                            }
                        }
                    AuroranComponent::MessageReceived(message);
                }

void ProcessView(BString & ViewName, BMessage * msg) {
    if (ViewName=="chk_Save") {
        int32 TheValue;

        if (msg->FindInt32("_val",&TheValue)==B_OK) {
            if (TheValue==B_CONTROL_ON) {
                MyFlags|=SAVE_Values;
            } else {
                MyFlags&=(~SAVE_Values);
            }
        }
    } else {
        AuroranComponent::ProcessView(ViewName, msg);
    }
}

virtual status_t Save(BPositionIO * object) {
    object->Write(&MyFlags,1);

    //write channel values
    if ((MyFlags & SAVE_Values)) {
        for (int i =0;i<40;i++) {
            object->Write(&Values[i]->GetValue(),sizeof
(float));
        }
    }

    return B_OK;
}

```

```

virtual status_t Load(BPositionIO * object, uint32 version) {
    //if (version<1) return B_OK;
    object->Read(&MyFlags,1);

    if ((MyFlags & SAVE_Values)) {
        //object->Read(Values,sizeof(float)*40);
        for (int i =0;i<40;i++) {
            CHData Value;
            object->Read(&Value,4);
            Values[i]->SetValue(Value);
        }
    }

    return B_OK;
}

virtual ComponentDescriptor * GetDesc() {return &MyDescriptor;}
protected:
    virtual char * MyName() {return "Mixer";}
    virtual char * DefaultCustom() {return "Mix";}
};

AuroranComponent * MakeComponent(ComponentDescriptor * Who,
AuroranParameters * Params) {return new SimpleMixer(Params);}

extern "C" _EXPORT ComponentDescriptor* GetDesc();

ComponentDescriptor* GetDesc() {
    return &MyDescriptor;
}

```

Filename: KrazyCtl.h

```
#include <View.h>
#include <List.h>
struct SwitchState {
    int Start;
    int Stop;
    SwitchState* Prev;
    SwitchState* Next;
    bool Color;
    BList IOConnections;
};

class Krazy : public BView {
public:
    Krazy(SwitchState ** Map, BLooper * Loopy);
    void Draw(BRect updateRect);
    virtual void MouseMoved(BPoint point, uint32 transit, const
BMessage *message);
    virtual void MouseDown(BPoint point);
    virtual void MouseUp(BPoint point);
    virtual void MessageReceived(BMessage * msg);
    virtual void AttachedToWindow();
private:
    uint8 Tracking;
    uint8 MinTravel;
    uint8 MaxTravel;
    uint8 LastMin;
    uint8 LastMax;
    SwitchState** Mapping;//[256];
    SwitchState* Selected;
    BLooper * ILoop;
};
```

Filename: KrazyCtl.cpp

```
//*****
//TODO: Clean up BLists when Joining
//      Show in BStringView where we are...

#include "KrazyCtl.h"
#include <iostream.h>

#include "ResourceSet.h"
#include <Button.h>
#include <PictureButton.h>
#include <Looper.h>

//this resource management stuff is taken from one issue of the
BMessage newsletter...
//http://www.be.com/developers/bmessage/issue09.html
#include <Locker.h>
#include <TranslatorFormats.h>

//*****

static int32 gResourcesCreated = 0;
static BResourceSet gResources;

static BResourceSet& Resources()
{
    // Return the BResourceSet object for this image, initializing
    // if needed.
    if ((gResourcesCreated&2) != 0) {
        // Quick return if already initialized.
        return gResources;
    } else if (atomic_or(&gResourcesCreated, 1) == 0) {
        // Not yet initialized -- do so.
        gResources.AddResources(&gResourcesCreated);
        // Mark that we are done.
        atomic_or(&gResourcesCreated, 2);
    } else {
        // Wait for resources to be initialized.
        while ((gResourcesCreated&2) == 0) snooze(50000);
    }
    return gResources;
}

class BBitmapButton : public BButton {
    const BBitmap * up;
    const BBitmap * down;
public:
    BBitmapButton(BRect frame, const char * name, const BBitmap *
up, const BBitmap * down, BMessage * msg) : BButton(frame, name, NULL,
msg, B_FOLLOW_LEFT | B_FOLLOW_TOP, B_WILL_DRAW | B_NAVIGABLE), up(up), down
(down) {}

    void Draw(BRect updateRect) {
        SetDrawingMode(B_OP_ALPHA);
    }
}
```

```

        if ((Value())) { //pressed
            DrawBitmap(down,BPoint(0,0));
        } else { //released
            DrawBitmap(up,BPoint(0,0));
        }
    }
};

Krazy::Krazy(SwitchState ** Map, BLooper * Loopy) : BView(BRect
(0,22,555,55),"slient bob",B_FOLLOW_LEFT_RIGHT,B_WILL_DRAW) {

    SetViewColor(ui_color(B_PANEL_BACKGROUND_COLOR));

    ILoop=Loopy;

    //default starting values
    LastMin=0;
    LastMax=255;
    MinTravel=0;
    MaxTravel=255;
    Mapping=Map;
    Selected=Mapping[0];
    Tracking=0;
}

void Krazy::AttachedToWindow() { //have to do this after the view is
attached

    //set up controls
    BControl * x;
    AddChild(x=new BButton(BRect(0,5,32,29),"btn_JL","J · L",new
BMessage('J__L')));
    x->SetTarget(this);
    AddChild(x=new BBitmapButton(BRect(40,5,52,29),"btn_L",(BBitmap *)
Resources().FindBitmap(B_PNG_FORMAT, 5002),(BBitmap *)Resources().
FindBitmap(B_PNG_FORMAT, 5003),new BMessage('JmpL')));
    x->SetTarget(this);
    AddChild(x=new BBitmapButton(BRect(68+256,5,68+256+12,29),"btn_R",
(BBitmap *)Resources().FindBitmap(B_PNG_FORMAT, 5004),(BBitmap *)
Resources().FindBitmap(B_PNG_FORMAT, 5005),new BMessage('JmpR')));
    x->SetTarget(this);
    AddChild(x=new BButton(BRect(68+256+8+10,5,68+256+8+32+10,29),
"btn_JR","J · R",new BMessage('J__R')));
    x->SetTarget(this);
    AddChild(x=new BButton(BRect(500,5,555,29),"btn_Split","Split",new
BMessage('Splt')));
    x->SetTarget(this);

}

void Krazy::Draw(BRect updateRect) {
    SetLowColor(ui_color(B_PANEL_BACKGROUND_COLOR));
    FillRect(updateRect,B_SOLID_LOW);
    SetDrawingMode(B_OP_ALPHA);
    //DrawBitmap((BBitmap *)Resources().FindBitmap(B_PNG_FORMAT, 5001),
BPoint(0,5));

    DrawBitmap((BBitmap *)Resources().FindBitmap(B_PNG_FORMAT, 5006),

```

```

BPoint(60-6+Selected->Start,21));
    DrawBitmap((BBitmap *)Resources().FindBitmap(B_PNG_FORMAT, 5007),
BPoint(60-6+Selected->Stop,4));
    //Flush();
    //DrawBitmap((BBitmap *)Resources().FindBitmap(B_PNG_FORMAT, 5004),
BPoint(68+255,5));

    SetDrawingMode(B_OP_COPY);

    bool WhichColor=false;

    SwitchState * TheState = Mapping[0];
    while (TheState!=NULL) {
        if ((WhichColor=!WhichColor)) {
            SetHighColor((rgb_color){152,152,255,255});
        } else {
            SetHighColor((rgb_color){152/2,152/2,255/2,255});
        }
        TheState->Color=WhichColor;
        FillRect(BRect(60+TheState->Start,14,60+TheState->Stop,19),
B_SOLID_HIGH);
        TheState=TheState->Next;
    }

    //BeginLineArray(4);
    SetHighColor((rgb_color){96,96,96,255});
    StrokeLine(BPoint(59,13),BPoint(315,13),B_SOLID_HIGH);
    StrokeLine(BPoint(59,14),BPoint(59,20),B_SOLID_HIGH);
    SetHighColor((rgb_color){250,250,250,255});
    StrokeLine(BPoint(59,20),BPoint(315,20),B_SOLID_HIGH);
    StrokeLine(BPoint(316,13),BPoint(316,20),B_SOLID_HIGH);
    //EndLineArray();
}

void Krazy::MouseMoved(BPoint point, uint32 transit, const BMessage
*message) {

    int WhereTo=(point.x-60);

    if (Tracking==0) return;

    // if (point.x >= 60 && point.x <= 60+255) { //do we care?
        //cout << "MM:" << transit << endl;

        if (Tracking==2) { //bottom (min)

            if (WhereTo > Selected->Stop) WhereTo = Selected->Stop;

            if (WhereTo <= Selected->Prev->Start) WhereTo = Selected-
>Prev->Start+1;

            if (WhereTo!=Selected->Start) {
                SetLowColor(ui_color(B_PANEL_BACKGROUND_COLOR));
                FillRect(BRect(Selected->Start-6+60,21,Selected-
>Start+6+60,28),B_SOLID_LOW);
                SetDrawingMode(B_OP_ALPHA);
                DrawBitmap((BBitmap *)Resources().FindBitmap
(B_PNG_FORMAT, 5006),BPoint(WhereTo-6+60,21));

```

```

        if (WhereTo < Selected->Start) {
            if (Selected->Color) {
                SetHighColor((rgb_color){152,152,255,255});
            } else {
                SetHighColor((rgb_color){152/2,152/2,255/
2,255});
            }
            FillRect(BRect(60+WhereTo,14,60+Selected-
>Start,19),B_SOLID_HIGH);
            for (int i = WhereTo; i <=Selected->Start; i++) {
                Mapping[i]=Selected;
            }
        } else {
            if (Selected->Prev->Color) {
                SetHighColor((rgb_color){152,152,255,255});
            } else {
                SetHighColor((rgb_color){152/2,152/2,255/
2,255});
            }
            FillRect(BRect(60+Selected->Prev-
>Stop,14,60+WhereTo-1,19),B_SOLID_HIGH);
            for (int i = Selected->Prev->Stop; i <=WhereTo-1;
i++) {
                Mapping[i]=Selected->Prev;
            }
        }
        Selected->Start=WhereTo;
        Selected->Prev->Stop=WhereTo-1;
    }
} else { //top (max)

    if (WhereTo < Selected->Start) WhereTo = Selected->Start;

    if (WhereTo >= Selected->Next->Stop) WhereTo = Selected-
>Next->Stop-1;

    if (WhereTo!=Selected->Stop) {
        SetLowColor(ui_color(B_PANEL_BACKGROUND_COLOR));
        FillRect(BRect(Selected->Stop-6+60,4,Selected-
>Stop+6+60,11),B_SOLID_LOW);
        SetDrawingMode(B_OP_ALPHA);
        DrawBitmap((BBitmap*)Resources().FindBitmap
(B_PNG_FORMAT, 5007),BPoint(WhereTo-6+60,4));

        if (WhereTo < Selected->Stop) {
            if (Selected->Next->Color) {
                SetHighColor((rgb_color){152,152,255,255});
            } else {
                SetHighColor((rgb_color){152/2,152/2,255/
2,255});
            }
            FillRect(BRect(60+WhereTo+1,14,60+Selected->Next-
>Start,19),B_SOLID_HIGH);
            for (int i = WhereTo+1; i <=Selected->Next->Start;
i++) {
                Mapping[i]=Selected->Next;
            }
        } else {

```



```

        if (Selected->Color) {
            SetHighColor((rgb_color){152,152,255,255});
        } else {
            SetHighColor((rgb_color){152/2,152/2,255/
2,255});
        }
        FillRect(BRect(60+Selected->Stop,14,60+WhereTo,19),
B_SOLID_HIGH);
        for (int i = Selected->Stop; i <=WhereTo; i++) {
            Mapping[i]=Selected;
        }
        Selected->Stop=WhereTo;
        Selected->Next->Start=WhereTo+1;
    }
}
SetDrawingMode(B_OP_COPY);

}

void Krazy::MouseDown(BPoint point) {
    cout << "MD" << endl;
    if (point.x >= 60 && point.x <= 60+255) { //do we care?
        if (point.y <14 && Selected->Next!=NULL) { //above
            Tracking=1;
            SetMouseEventMask(B_POINTER_EVENTS,
                B_LOCK_WINDOW_FOCUS | B_NO_POINTER_HISTORY);
        } else if (point.y > 20 && Selected->Prev!=NULL) { //below
            Tracking=2;
            SetMouseEventMask(B_POINTER_EVENTS,
                B_LOCK_WINDOW_FOCUS | B_NO_POINTER_HISTORY);
        } else {
            Selected = Mapping[(int)point.x-60];
            BMessage * x = new BMessage('SlCh');
            x->AddPointer("Selected",Selected);
            ILoop->PostMessage(x);
            delete x;
            Invalidate();
        }
    }
}

void Krazy::MouseUp(BPoint point) {
    cout << "MU" << endl;
    // BView::MouseUp(point);
    if (Tracking) {
        Tracking =0;
    }
}

void Krazy::MessageReceived(BMessage * msg) {
    switch (msg->what) {
        case ('Splt'): {
            if (Selected->Start!=Selected->Stop) { //there is room to

```

```

split...
        SwitchState * NewSeg = new SwitchState;
        *NewSeg = (SwitchState){(Selected->Stop+Selected-
>Start)/2+1, Selected->Stop, Selected, Selected->Next};
        Selected->Next=NewSeg;
        Selected->Stop=NewSeg->Start-1;
        if (NewSeg->Next!=NULL) NewSeg->Next->Prev=NewSeg;
        for (int i = NewSeg->Start ; i<= NewSeg->Stop; i++) { //
set up mapping
            Mapping[i]=NewSeg;
        }
        for (int i = 0 ; i < Selected->IOConnections.CountItems
()); i++) {
            NewSeg->IOConnections.AddItem(new BList(*(BList *)
(Selected->IOConnections.ItemAt(i))));
        }
        Invalidate();
    }
    break;
}
case ('JmpL'): { //jump left
    if (Selected->Prev!=NULL) {
        Selected=Selected->Prev;

        BMessage * x = new BMessage('SlCh');
        x->AddPointer("Selected", Selected);
        ILoop->PostMessage(x);
        delete x;

        Invalidate();
    }
    break;
}
case ('JmpR'): { //jump right
    if (Selected->Next!=NULL) {
        Selected=Selected->Next;

        BMessage * x = new BMessage('SlCh');
        x->AddPointer("Selected", Selected);
        ILoop->PostMessage(x);
        delete x;

        Invalidate();
    }
    break;
}
case ('J__L'): { //Join Left
    if (Selected->Prev!=NULL) {
        SwitchState * NewPrev = Selected->Prev->Prev;
        for (int i = Selected->Prev->Start; i <= Selected->Prev-
>Stop; i ++ ) {
            Mapping[i]=Selected;
        }
        Selected->Start=Selected->Prev->Start;
        delete Selected->Prev;
        Selected->Prev=NewPrev;
        if (NewPrev!=NULL) NewPrev->Next=Selected;
        Invalidate();
    }
}

```

```

        break;
    }
    case ('J__R'): { //Join Right
        if (Selected->Next!=NULL) {
            SwitchState * NewNext = Selected->Next->Next;
            for (int i = Selected->Next->Start; i <= Selected->Next-
>Stop; i ++) {
                Mapping[i]=Selected;
            }
            Selected->Stop=Selected->Next->Stop;
            delete Selected->Next;
            Selected->Next=NewNext;
            if (NewNext!=NULL) NewNext->Prev=Selected;
            Invalidate();
        }
        break;
    }
}
}
}

```

Filename: Component.cpp

```
#include "Component.h"
#include "ResourceSet.h"
#include <stdio.h>
#include <iostream.h>
#include <SerialPort.h>
#include "KrazyCtl.h"
#include <InterfaceKit.h>
#include <stdlib.h>

//this resource management stuff is taken from one issue of the
BMessage newsletter...
//http://www.be.com/developers/bmessage/issue09.html
#include <Locker.h>
#include <TranslatorFormats.h>

//*****

static int32 gResourcesCreated = 0;
static BResourceSet gResources;

static BResourceSet& Resources()
{
    // Return the BResourceSet object for this image, initializing
    // if needed.
    if ((gResourcesCreated&2) != 0) {
        // Quick return if already initialized.
        return gResources;
    } else if (atomic_or(&gResourcesCreated, 1) == 0) {
        // Not yet initialized -- do so.
        gResources.AddResources(&gResourcesCreated);
        // Mark that we are done.
        atomic_or(&gResourcesCreated, 2);
    } else {
        // Wait for resources to be initialized.
        while ((gResourcesCreated&2) == 0) snooze(50000);
    }
    return gResources;
}

AuroranComponent * MakeComponent(ComponentDescriptor * Who,
AuroranParameters * Params);

#define World (*TheWorld)

static ComponentDescriptor SwitchDescriptor = {
    A_DESC_VER,
    A_NORMAL_DESC,
    NULL,
    "Switch",           //Menu Name
    'TKS_',             //Creator
    'SW_',              //ID
    0,                  //Version
    ACC_PASSIVE,        //Class
    &MakeComponent
};
```

```

static ComponentDescriptor ForkDescriptor = {
    A_DESC_VER,
    A_NORMAL_DESC,
    &SwitchDescriptor,
    "Fork",          //Menu Name
    'TKS_',          //Creator
    'FK_',           //ID
    0,               //Version
    ACC_PASSIVE,     //Class
    &MakeComponent
};

struct IOPair {
    SocketListItem * Input;
    PlugListItem * Output;
};

const uint32 Immediate_Values = 1;
const int MaxSockets = 256;

class Component : public AuroranComponent {
    bool IsFork;
    SwitchState* Mapping[256];
    SwitchState* CurrentState; //for display
    uint8 SendingState; //for data
    // BList * Pairs[256]; //set of mappings between inputs and
    outputs
    BListView * Sockets;
    float SocketValues[256];
    BListView * Plugs;
    BTextControl * SrcName;
    BTextControl * DestName;
    uint32 MyFlags;
    SocketListItem * mainsw;
public:
    //Don't Touch
    virtual ComponentDescriptor * GetDesc() {return (IsFork)?
    (&ForkDescriptor):(&SwitchDescriptor);}

    Component(AuroranParameters * MyWorld, bool Fork) :
    AuroranComponent(MyWorld, ((Fork)?"Fork":"Switch")), IsFork(Fork) { //
    set up

        AutoSetColors();
        MyIcon=(Fork)?(BBitmap *)Resources().FindBitmap
        (B_PNG_FORMAT, 2001):(BBitmap *)Resources().FindBitmap(B_PNG_FORMAT,
        2002);

        SwitchState* Base = new SwitchState;
        *Base = (SwitchState){0,255,NULL,NULL};
        for (int i =0 ; i<256; i++) {
            Mapping[i]=Base;
        }

        CurrentState = Base;
        SendingState = 0;

        if (!IsFork) {
            mainsw = RegisterSocket("SwitchValue",StdChan_Intensity,

```

```

new BMessage('00SW'));
    mainsw->SetFlags(mainsw->GetFlags() | Recv_Dirty);
}

~Component() { //Clean up
}

BView * GetView(BRect frame) {
    BView * TheView = AuroranComponent::GetView(frame);

    if (!IsFork) { //Spoon...er...Switch output code
        TheView->AddChild(new Krazy(Mapping, TheWorld->ILoop));
        BControl * NC;
        TheView->AddChild(NC = new BCheckBox(BRect
(2,200,400,50),"chk_Immediate","Send values immediately upon state
change",NULL));
        if ((MyFlags & Immediate_Values)) NC->SetValue
(B_CONTROL_ON);
    }

    float Top = (IsFork)?25:60;

    Sockets = new BListView(BRect(3,Top+15,120,190),
"lst_Sockets",B_SINGLE_SELECTION_LIST,B_FOLLOW_TOP_BOTTOM);
    for (int i=(IsFork?0:1) ; i < MySockets.CountItems(); i++)
    {
        Sockets->AddItem(new BStringItem(((BStringItem*)
MySockets.ItemAt(i))>Text()));
    }
    Sockets->SetSelectionMessage(new BMessage('SlNS'));
    Sockets->SetTarget(TheWorld->ILoop); //send to interafce
messenger
    TheView->AddChild(new BScrollView("scroll_Sockets",Sockets,
B_FOLLOW_TOP_BOTTOM,0,false,true));

    TheView->AddChild(new BStringView(BRect(3,Top,120,Top+13),
"lbl_Sockets","Sources:"));

    TheView->AddChild(new BStringView(BRect(144,Top,266,
Top+13),"lbl_InName","Source Name:"));

    TheView->AddChild(SrcName = new BTextControl(BRect(144,
Top+15,266,Top+30),"txt_In", NULL,"Source",NULL));
    TheView->AddChild(DestName = new BTextControl(BRect(286,
Top+15,406,Top+30),"txt_Out", NULL,"Dest",NULL));

    BControl * x;

    TheView->AddChild(x=new BButton(BRect(144,Top+40,266,
Top+60),"btn_AddIn","Add Source",new BMessage('+Src')));
    x->SetTarget(World.ILoop);
    TheView->AddChild(x=new BButton(BRect(144,Top+70,266,
Top+90),"btn_RenIn","Rename Source",new BMessage('rSrc')));
    x->SetTarget(World.ILoop);
    TheView->AddChild(x=new BButton(BRect(144,170,266,190),
"btn_DelIn","Remove Source",new BMessage('-Src')));
    x->SetTarget(World.ILoop);

```

```

        TheView->AddChild(x=new BButton(BRect(286,Top+40,406,
Top+60),"btn_AddOut","Add Destination",new BMessage('+Dst')));
        x->SetTarget(World.ILoop);
        TheView->AddChild(x=new BButton(BRect(286,Top+70,406,
Top+90),"btn_RenOut","Rename Destination",new BMessage('rDst')));
        x->SetTarget(World.ILoop);
        TheView->AddChild(x=new BButton(BRect(286,170,406,190),
"btn_DelOut","Remove Destination",new BMessage('-Dst')));
        x->SetTarget(World.ILoop);

        Plugs = new BListView(BRect(420,Top+15,540,190),
"lst_Plugs",B_MULTIPLE_SELECTION_LIST,B_FOLLOW_TOP_BOTTOM);
        Plugs->SetSelectionMode(new BMessage('SlNP'));
        Plugs->SetTarget(World.ILoop); //send to interafce messenger
        TheView->AddChild(new BScrollView("scroll_Plugs",Plugs,
B_FOLLOW_TOP_BOTTOM,0,false,true));
        for (int i=0;i<MyPlugs.CountItems();i++) {
            Plugs->AddItem(new BStringItem(((PlugListItem *)MyPlugs.
ItemAt(i))>Text()));
        }

        TheView->AddChild(new BStringView(BRect(420,Top,540,
Top+13),"lbl_Plugs","Destinations:"));

    return TheView;
}

void ProcessView(BString & ViewName, BMessage * msg) {
    if (ViewName=="chk_Immediate") {
        int32 TheValue;

        if (msg->FindInt32("_val",&TheValue)==B_OK) {
            if (TheValue==B_CONTROL_ON) {
                MyFlags|=Immediate_Values;
            } else {
                MyFlags&=(~Immediate_Values);
            }
        }
    } else {
        AuroranComponent::ProcessView(ViewName, msg);
    }
}

virtual void MessageReceived(BMessage * message) {
    if (message->what == '00SW') {
        float Value;
        if (message->FindFloat("value",&Value)==B_OK) {
            cout << "NEW SWITCH VALUE!!!" << endl;
            mainsw->GetValueAndClean();
            if (Value < 1.0 && Value >= 0) {
                SendingState = (uint8)(rint(Value * 255));
            }
        }
    }
}

```

```

        } else if (message->what == 'xyzz') {
            float Value;
            SocketListItem* WhichSocket;
            if (message->FindFloat("value",&Value)==B_OK && message-
>FindPointer("SocketPtr", (void**) &WhichSocket)==B_OK) {
                BList * WhereTo = (BList *) (Mapping[SendingState])-
>IOConnections.ItemAt(MySockets.IndexOf(WhichSocket)-(IsFork?0:1));
                int y = WhereTo->CountItems();
                for (int i = 0 ; i < y ; i++) {
                    ((PlugListItem *) MyPlugs.ItemAt((int)WhereTo-
>ItemAt(i)))->SendMessage(Value);
                }
            }
        } else {
            AuroranComponent::MessageReceived(message);
        }
    }
    virtual void InterfaceMessage(BMessage * message) {
        switch (message->what) {
            case '+Src': {
                if (Sockets->CountItems() < MaxSockets) {
                    if (TheWorld->CList->Window()->LockWithTimeout(200)
==B_OK) {
                        //set initial value to off
                        SocketValues[Sockets->CountItems()]=0.0;
                        //add to socket list
                        BMessage* msg = new BMessage('xyzz');//
+MySockets.CountItems()-(IsFork?0:1))
                        RegisterSocket(SrcName->Text(),msg);
                        //add to our list
                        Sockets->AddItem(new BStringItem(SrcName->Text
()));
                        //add to all states
                        SwitchState * TheState = Mapping[0];
                        while (TheState!=NULL) {
                            TheState->IOConnections.AddItem(new BList
());
                            TheState=TheState->Next;
                        }
                        TheWorld->CList->Window()->Unlock();
                    } else {
                        TheWorld->Pulser->OneShot(TheWorld->ILoop, new
BMessage('+Src'));
                    }
                }
                break;
            }
            case '+Dst': {
                if (World.CList->Window()->LockWithTimeout(200)
==B_OK) {
                    RegisterPlug(DestName->Text(),
StdChan_Intensity);
                    Plugs->AddItem(new BStringItem(DestName->Text
()));
                    World.CList->Window()->Unlock();
                } else {
                    World.Pulser->OneShot(World.ILoop, new BMessage
('+Dst'));
                }
            }
        }
    }

```



```

        }
        break;
    }
    case '-Src': {
        if (Sockets->CurrentSelection()>=0) {
            if (World.CList->Window()->LockWithTimeout(200)
==B_OK) {
                if (this->LockWithTimeout(200)==B_OK) {
                    //which socket?
                    int ThisSocket = Sockets->CurrentSelection
                );

                    //remove Socket from all states
                    SwitchState * TheState = Mapping[0];
                    while (TheState!=NULL) {
                        delete TheState->IOConnections.
                    }
                    TheState=TheState->Next;

                    //remove from list box
                    delete Sockets->RemoveItem(ThisSocket);

                    //official removal
                    RemoveSocket(ThisSocket+(IsFork?0:1));

                    this->Unlock();
                } else {
                    World.Pulser->OneShot(World.ILoop, new
                BMessage('-Src'));
                }
                World.CList->Window()->Unlock();
            } else {
                World.Pulser->OneShot(World.ILoop, new
                BMessage('-Src'));
            }
        }
        break;
    }
    case '-Dst': {
        if (Plugs->CurrentSelection(0)>=0) {
            if (World.CList->Window()->LockWithTimeout(200)
==B_OK) {
                if (this->LockWithTimeout(200)==B_OK) {
                    int ThisPlug=Plugs->CurrentSelection(0);
                    SwitchState * TheState = Mapping[0]; //the
destination from whatever mappings
                    while (TheState!=NULL) { //cycle through all
possible states
                        int temp;
                        int i = TheState->IOConnections.
                    }
                    CountItems()-1;
                    for (;i>=0;i--) { //cycle though
sources in that state
                        BList

```

```

    * Dests = (BList*)TheState->IOConnections.ItemAt(i);
    for (int j=Dests->CountItems()-1; j
    >=0 ;j--) { //cycle though selected items for that source
        if (ThisPlug < (int)Dests-
    >ItemAt(j)) {
            temp = (int)Dests-
    >RemoveItem(j);
            Dests->AddItem((void *)
    (temp-1));
        } else if (ThisPlug == (int)
    Dests->ItemAt(j)) {
            Dests->RemoveItem(j);
        }
    }
    TheState=TheState->Next;
}
delete Plugs->RemoveItem(ThisPlug);
DeletePlug(ThisPlug);

this->Unlock();
} else {
    World.Pulser->OneShot(World.ILoop, new
BMessage('-Dst'));
}
World.CList->Window()->Unlock();

} else {
    World.Pulser->OneShot(World.ILoop, new
BMessage('-Dst'));
}
}
break;
}
case 'SlNS': { //socket selected
    if (World.CList->Window()->LockWithTimeout(200)
==B_OK) { //we need to lock the view
        Plugs->DeselectAll(); //clear list...

        if (Sockets->CurrentSelection()>=0) {
            SrcName->SetText(((BStringItem*)Sockets-
    >ItemAt(Sockets->CurrentSelection()))->Text());
            //cout << "sockets: " << CurrentState-
    >IOConnections.CountItems() << endl;
            BList * Dests = (BList *) (CurrentState-
    >IOConnections.ItemAt(Sockets->CurrentSelection()));
            for (int i=0; i < Dests->CountItems(); i++)
            {
                Plugs->Select((int)Dests->ItemAt(i),
    true);
            }
        }

        World.CList->Window()->Unlock();
    } else { //if we can't get a lock now, try again
        later
        World.Pulser->OneShot(World.ILoop, new BMessage
    ('SlNS'));
    }
}

```

```

        }
        break;
    }
    case 'SlNP': { //plug selected
        if (Plugs->CurrentSelection()>=0) { //update text
displayed
            if (World.CList->Window()->LockWithTimeout(200)
==B_OK) { //we need to lock the view
                DestName->SetText(((BStringItem*)Plugs-
>ItemAt(Plugs->CurrentSelection()))->Text());
                World.CList->Window()->Unlock();
            } else { //if we can't get a lock now, try
again later
                World.Pulser->OneShot(World.ILoop, new
BMessage('SlNP'));
                break;
            }
        }
        if (Sockets->CurrentSelection()>=0) {
            int i =0;
            int who;
            BList * Dests = (BList *) (CurrentState-
>IOConnections.ItemAt(Sockets->CurrentSelection()));
            Dests->MakeEmpty();
            while ((who = Plugs->CurrentSelection(i++)) >=
0) {
                Dests->AddItem((void *)who);
            }
        }
        break;
    }
    case 'rSrc': { //rename current source socket
        if (Sockets->CurrentSelection()>=0) {
            if (World.CList->Window()->LockWithTimeout(200)
==B_OK) { //we need to lock the view
                ((BStringItem*)Sockets->ItemAt(Sockets-
>CurrentSelection()))->SetText(SrcName->Text());
                Sockets->Invalidate();
                RenameSocket(Sockets->CurrentSelection()+1,
SrcName->Text());
                World.CList->Window()->Unlock();
            } else { //if we can't get a lock now, try
again later
                World.Pulser->OneShot(World.ILoop, new
BMessage('rSrc'));
                //break;
            }
        }
        break;
    }
    case 'rDst': { //rename current destination plug
        int i=0;
        if (Plugs->CurrentSelection(i)>=0) {
            if (World.CList->Window()->LockWithTimeout(200)
==B_OK) { //we need to lock the view
                while (Plugs->CurrentSelection(i)>=0) {
                    ((BStringItem*)Plugs->ItemAt(Plugs-

```

```

>CurrentSelection(i))->SetText(DestName->Text());
                                Plugs->Invalidate();
                                RenamePlug(Plugs->CurrentSelection
(i++),DestName->Text());
                                }
                                World.CList->Window()->Unlock();
                                } else { //if we can't get a lock now, try
again later
                                World.Pulser->OneShot(World.ILoop, new
BMessage('rDst'));
                                //break;
                                }
                                }
                                break;
                                }

                                case 'SlCh': { //state changed (?)
                                message->FindPointer("Selected", (void **))
                                &CurrentState);
                                Sockets->DeselectAll();
                                Plugs->DeselectAll();
                                }

                                default:
                                AuroranComponent::InterfaceMessage(message);
                                }
                                }

virtual status_t Save(BPositionIO * object) {
    int o;

    //Store plugs...
    o = MyPlugs.CountItems();
    object->Write(&o,4);
    for (int i=0;i<o;i++) {
        const char * l;
        int k;
        l = ((BStringItem *)MyPlugs.ItemAt(i))->Text();
        k = strlen(l);
        object->Write(&k,4);
        object->Write(l,k);
    }

    //Store states
    SwitchState * TheState = Mapping[0];
    while (TheState!=NULL) {
        object->Write(&(TheState->Stop),4);
        TheState=TheState->Next;
    }

    //Store sockets...
    o = MySockets.CountItems()-(IsFork?0:1);
    object->Write(&o,4);
    for (int i=0;i<o;i++) {
        const char * l;
        int k;

```

```

        //write out name
        l = ((BStringItem *)MySockets.ItemAt(i+(IsFork?0:1)))-
>Text();
        k = strlen(l);
        object->Write(&k,4);
        object->Write(l,k);

        //store connections for each state
        TheState=Mapping[0]; //reset current state
        while (TheState!=NULL) {
            BList* L;
            L=(BList*) (TheState->IOConnections.ItemAt(i));
            k=L->CountItems();
            object->Write(&k,4);
            for (int j=0;j<k;j++) {
                int p = (int) (L->ItemAt(j));
                object->Write(&p,4);
            }
            TheState=TheState->Next;
        }

        return B_OK;
    }

virtual status_t Load(BPositionIO * object, uint32 version) {
    int MaxString=64;
    char * StrTemp = (char *)malloc(64);
    int o; //object count

    //Read plugs...
    object->Read(&o,4);
    cout << dec << o << " Plugs found..." << endl;
    for (int i=0;i<o;i++) {
        int k;
        object->Read(&k,4); //string length
        if (k>MaxString) {
            free(StrTemp);
            StrTemp=(char *)malloc(k);
            MaxString=k;
        }
        object->Read(StrTemp,k);
        StrTemp[k]=0;

        RegisterPlug(StrTemp,StdChan_Intensity);
    }

    //Read states
    object->Read(&o,4);
    Mapping[0]->Stop=o;
    SwitchState* Recent = Mapping[0];
    SwitchState* New;
    while (o<255) {
        object->Read(&o,4);
        New = new SwitchState;
        *New = (SwitchState){Recent->Stop+1,o,Recent,NULL,!
(Recent->Color)};

```

```

        for (int i=Recent->Stop+1;i<=o;i++) {
            Mapping[i]=New;
        }
        Recent->Next=New;
        Recent=New;
    }

    //Read sockets...
    cout << "Pos: " << hex << object->Position() << endl;
    object->Read(&o,4);
    cout << dec << o << " Sockets found..." << endl;
    for (int i=0;i<o;i++) {
        cout << "In for: " << i << " at: " << hex << object-
>Position() << endl;
        int k;
        object->Read(&k,4); //string length
        if (k>MaxString) {
            free(StrTemp);
            StrTemp=(char *)malloc(k+1);
            MaxString=k;
        }
        object->Read(StrTemp,k);
        StrTemp[k]=0;

        RegisterSocket(StrTemp,new BMessage('xyzz'));

        //restore connections for each state
        SwitchState* TheState=Mapping[0]; //reset current state
        while (TheState!=NULL) {
            BList* L;
            TheState->IOConnections.AddItem(L=new BList);
            object->Read(&k,4);
            for (int j=0;j<k;j++) {
                int p;
                object->Read(&p,4);
                L->AddItem((void*)p);
            }
            TheState=TheState->Next;
        }
    }
    cout << "Pos: " << hex << object->Position() << endl;

    free(StrTemp); //clean up

    return B_OK;
}
protected:
    virtual const char * MyName() {return (IsFork)?
"Fork":"Switch";}
    virtual const char * DefaultCustom() {return (IsFork)?
"Fork":"Switch";}
};

AuroranComponent * MakeComponent(ComponentDescriptor * Who,
AuroranParameters * Params) {
    if (Who->What=='SW') {

```

```
        return new Component(Params,false);
    } else {
        return new Component(Params,true);
    }
}

extern "C" _EXPORT ComponentDescriptor* GetDesc();

ComponentDescriptor* GetDesc() {
    return &ForkDescriptor;
}
```

Filename: Component.cpp

```
#include "Component.h"
#include "ResourceSet.h"
#include <stdio.h>
#include <iostream.h>
#include <SerialPort.h>

//this resource management stuff is taken from one issue of the
BMessage newsletter...
//http://www.be.com/developers/bmessage/issue09.html
#include <Locker.h>
#include <TranslatorFormats.h>

//*****

static int32 gResourcesCreated = 0;
static BResourceSet gResources;

static BResourceSet& Resources()
{
    // Return the BResourceSet object for this image, initializing
    // if needed.
    if ((gResourcesCreated&2) != 0) {
        // Quick return if already initialized.
        return gResources;
    } else if (atomic_or(&gResourcesCreated, 1) == 0) {
        // Not yet initialized -- do so.
        gResources.AddResources(&gResourcesCreated);
        // Mark that we are done.
        atomic_or(&gResourcesCreated, 2);
    } else {
        // Wait for resources to be initialized.
        while ((gResourcesCreated&2) == 0) snooze(50000);
    }
    return gResources;
}

AuroranComponent * MakeComponent(ComponentDescriptor * Who,
AuroranParameters * Params);

static ComponentDescriptor MyDescriptor = {
    A_DESC_VER,
    A_NORMAL_DESC,
    NULL,
    "Polar Coordinates", //Menu Name
    'TKS_',              //Creator
    'PC_',               //ID
    0,                   //Version
    ACC_PASSIVE,         //Class
    &MakeComponent
};

class Component : public AuroranComponent {
private:
    double theta;
```



```

        double radius;
        PlugListItem * X;
        PlugListItem * Y;
public:
    //Don't Touch
    virtual ComponentDescriptor * GetDesc() {return &MyDescriptor;}

    Component(AuroranParameters * Parms) : AuroranComponent(Parms,
DefaultCustom()) { //set up
        AutoSetColors();
        MyIcon=(BBitmap *)Resources().FindBitmap(B_PNG_FORMAT,
2001);
        theta=0;
        radius=0;
        RegisterSocket("Theta",StdChan_Position,new BMessage
('thta'));
        RegisterSocket("Radius",StdChan_Position,new BMessage
('radi'));
        X=RegisterPlug("X",StdChan_Position);
        Y=RegisterPlug("Y",StdChan_Position);
    }
    ~Component() { //Clean up
    }
    BView * GetView(BRect frame) {
        BView * TheView = AuroranComponent::GetView(frame);
        return TheView;
    }
    virtual void MessageReceived(BMessage * message) {
        switch (message->what) {
            case 'thta': {
                float Value;
                if (message->FindFloat("value",&Value)==B_OK) {
                    if (Value < 1.0 && Value >= 0) {
                        theta = Value*3.1415*2;
                        X->SendMessage(cos(theta)*radius+0.5);
                        Y->SendMessage(sin(theta)*radius+0.5);
                    }
                }
                break;
            }
            case 'radi': {
                float Value;
                if (message->FindFloat("value",&Value)==B_OK) {
                    if (Value < 1.0 && Value >= 0) {
                        radius = Value-0.5;
                        X->SendMessage(cos(theta)*radius+0.5);
                        Y->SendMessage(sin(theta)*radius+0.5);
                    }
                }
                break;
            }
            default:
                AuroranComponent::MessageReceived(message);
        }
    }
protected:
    virtual char * MyName() {return "Polar Coords";}
    virtual char * DefaultCustom() {return "Polar";}
};

```

```
AuroranComponent * MakeComponent(ComponentDescriptor * Who,  
AuroranParameters * Params) {return new Component(Params);}
```

```
extern "C" _EXPORT ComponentDescriptor* GetDesc();
```

```
ComponentDescriptor* GetDesc() {  
    return &MyDescriptor;  
}
```

Filename: Component.cpp

```
#include "Component.h"
#include "ResourceSet.h"
#include "colorspace.h"
#include <stdio.h>
#include <iostream.h>
// #include <SerialPort.h>
#include <OS.h>
#include <malloc.h>
#include <math.h>
#include <stdlib.h>
#include "PrettySlider.h"
#include "ChannelTools.h"
#include "AColorControl.h"

//this resource management stuff is taken from one issue of the
BMessage newsletter...
//http://www.be.com/developers/bmessage/issue09.html
#include <Locker.h>
#include <TranslatorFormats.h>

//*****

static int32 gResourcesCreated = 0;
static BResourceSet gResources;

static BResourceSet& Resources()
{
    // Return the BResourceSet object for this image, initializing
    // if needed.
    if ((gResourcesCreated & 2) != 0) {
        // Quick return if already initialized.
        return gResources;
    } else if (atomic_or(&gResourcesCreated, 1) == 0) {
        // Not yet initialized -- do so.
        gResources.AddResources(&gResourcesCreated);
        // Mark that we are done.
        atomic_or(&gResourcesCreated, 2);
    } else {
        // Wait for resources to be initialized.
        while ((gResourcesCreated & 2) == 0) snooze(50000);
    }
    return gResources;
}

AuroranComponent * MakeComponent(ComponentDescriptor * Who,
AuroranParameters * Params);

static ComponentDescriptor MyDescriptor = {
    A_DESC_VER,
    A_NORMAL_DESC,
    NULL,
    "Lamp Group",    //Menu Name
    'TKS_',          //Creator
    'LG_',           //ID
```

```

    0,          //Version
    ACC_ABSTRACT, //Class
    &MakeComponent
};

enum {
    LG_MODE_OFFLINE=0,
    LG_MODE_BLACKOUT=1,
    LG_MODE_SOLOTEST=2,
    LG_MODE_COLOR=3,
    LG_MODE_RANDOMIZE=4,
    LG_MODE_CHASE=5,
    LG_MODE_1HOT=6
};

class Component : public AuroranComponent {
private:
    int OpMode;
    bool OpLock;
    int SelectedChan;
    double theta;
    double radius;
    SocketListItem * Mode;
    SocketListItem * Hue;
    SocketListItem * Saturation;
    SocketListItem * Intensity;
    SocketListItem * Tolerance;
    SocketListItem * Activity;
    SocketListItem * Blur;
    SocketListItem * Pulse;
    float MasterFader;
    bigtime_t LastBeat;
    bigtime_t LastPulse;
    float Period;
    typedef struct {
        float OldValue;
        float NextValue;
        float CurrentValue;
        hsv_fcolor Color;
    } ValSet;
    BList ChanData;
    int ChaseStep;
    PlugListItem * AAA;
    BRadioButton * ModeSelections[7];
    AColorControl * ChanColors;
    AColorControl * MasterColor;
    bool UpdatingView;
    bool Visable;
public:
    //Don't Touch
    virtual ComponentDescriptor * GetDesc() {return &MyDescriptor;}

    Component(AuroranParameters * Params) : AuroranComponent(Params,
DefaultCustom()) { //set up

        AutoSetColors();

```

```

2001);
    MyIcon=(BBitmap *)Resources().FindBitmap(B_PNG_FORMAT,
    ApplyLoc=AC_APPLY_HIDDEN;

    //start up into offline mode by default.
    OpMode=0;
    OpLock=false;
    SelectedChan=-1;
    UpdatingView=false;
    Visable=false;

    Mode=RegisterSocket("Mode",new BMessage('pchg'));
    BeginSBundle("Color");
    Hue=RegisterSocket("Hue",new BMessage('pchg'));
    Saturation=RegisterSocket("Saturation",new BMessage
('pchg'));
    Intensity=RegisterSocket("Intensity",new BMessage('pchg'));
    Tolerance=RegisterSocket("Tolerance",new BMessage('pchg'));
    EndSBundle();
    Activity=RegisterSocket("Activity",new BMessage('pchg'));
    Hue->SetValue((CHData){0.0});
    Saturation->SetValue((CHData){0.0});
    Intensity->SetValue((CHData){0.3333});
    Tolerance->SetValue((CHData){1.0});
    Activity->SetValue((CHData){0.0});
    Blur=RegisterSocket("Blur",new BMessage('pchg'));
    Pulse=RegisterSocket("Pulse",new BMessage('boom'));

    Parms->Pulser->AddClient(this,25.0);
    LastBeat=0;
    LastPulse=0;

    Parms->Pulser->OneShot(this,new BMessage('pchg'));
}
~Component() { //Clean up
    ValSet * x;
    while (x=(ValSet*)ChanData.RemoveItem((int32)0)) {
        delete x;
    }
}
BView * GetView(BRect frame) {
    BView * TheView = AuroranComponent::GetView(frame);
    BTabView * megaTabView = new BTabView(BRect(0,25,frame.
Width(),235),"megatab");

    BBox * Box;
    BSlider * Slider;
    BMessage * msg;
    BView * view;
    BButton * b;
    BTextControl * Text;

    BTab * aTab;

    //aTab = new BTab();
    BView * behaveView = new BView(megaTabView->Bounds(),
"Behavior",B_FOLLOW_LEFT|B_FOLLOW_TOP,B_WILL_DRAW);
    behaveView->SetViewColor(kViewGray);

```

```

        BControl * x;
        behaveView->AddChild(MasterColor=new AColorControl
(ACC_PREVIEW|ACC_VALUE|ACC_TOLERANCE,BRect(350,5,550,180),"Color
Picker",new BMessage('Colr'),B_FOLLOW_LEFT|B_FOLLOW_TOP));
        MasterColor->SetColor(hsv_fcolor(Hue->GetValue().
f*2*M_PI,Saturation->GetValue().f,Intensity->GetValue().f));
        MasterColor->SetTarget(TheWorld->ILoop);

        //Set up the mode selection radio buttons
        behaveView->AddChild(new BStringView(BRect(5,5,100,15),
"lbl_OpMode","Operating Mode:"));
        behaveView->AddChild(ModeSelections[LG_MODE_OFFLINE]=new
BRadioButton(BRect(10,20,100,30),"rad_Off", "Offline",new BMessage
('OpMd'+LG_MODE_OFFLINE)));
        behaveView->AddChild(ModeSelections[LG_MODE_BLACKOUT]
=new BRadioButton(BRect(10,20+18*1,150,30+18*1),"rad_Black", "Blackout
(all 0%)",new BMessage('OpMd'+LG_MODE_BLACKOUT)));
        behaveView->AddChild(ModeSelections[LG_MODE_SOLOTEST]
=new BRadioButton(BRect(10,20+18*2,100,30+18*2),"rad_Solo", "Solo
Test",new BMessage('OpMd'+LG_MODE_SOLOTEST)));
        behaveView->AddChild(ModeSelections[LG_MODE_COLOR]=new
BRadioButton(BRect(10,20+18*3,150,30+18*3),"rad_Color", "Solid Color",
new BMessage('OpMd'+LG_MODE_COLOR)));
        behaveView->AddChild(ModeSelections[LG_MODE_RANDOMIZE]
=new BRadioButton(BRect(10,20+18*4,100,30+18*4),"rad_Random",
"Randomize",new BMessage('OpMd'+LG_MODE_RANDOMIZE)));
        behaveView->AddChild(ModeSelections[LG_MODE_CHASE]=new
BRadioButton(BRect(10,20+18*5,150,30+18*5),"rad_Chase", "Sequential
Chase",new BMessage('OpMd'+LG_MODE_CHASE)));
        behaveView->AddChild(ModeSelections[LG_MODE_1HOT]=new
BRadioButton(BRect(10,20+18*6,100,30+18*6),"rad_Hot", "Random 1-Hot",
new BMessage('OpMd'+LG_MODE_1HOT)));
        for (int i=0;i<=LG_MODE_1HOT;i++) {
            ModeSelections[i]->SetTarget(TheWorld->ILoop);
        }
        ModeSelections[OpMode]->SetValue(B_CONTROL_ON);

        //allow the user to lock the current mode
        behaveView->AddChild(x=new BCheckBox(BRect
(5,20+18*8,150,30+18*8),"chk_Lock", "Lock Operating Mode",new BMessage
('OpLk')));
        x->SetValue(OpLock?B_CONTROL_ON:B_CONTROL_OFF);
        x->SetTarget(TheWorld->ILoop);

        megaTabView->AddTab(behaveView);

        BView * outView = new BView(megaTabView->Bounds(),
"Channels",B_FOLLOW_LEFT|B_FOLLOW_TOP,B_WILL_DRAW);
        outView->SetViewColor(kViewGray);
        BView * outcView = new ChannelTools(this, CHT_PLUGS,
GetPlugList(), ParentPlug(),new BMessage('AddC'),new BMessage('AltC'),
new BMessage('DelC'),new BMessage('SelC'), BRect(0,0,megaTabView-
>Bounds().Width()/2,megaTabView->Bounds().Height()-megaTabView-
>TabHeight()),"Channel",B_FOLLOW_LEFT|B_FOLLOW_TOP,B_WILL_DRAW);
        outcView->SetViewColor(kViewGray);

        outView->AddChild(ChanColors=new AColorControl

```

```

(ACC_PREVIEW_LEFT,BRect(megaTabView->Bounds().Width()/2,10,megaTabView-
>Bounds().Width()-10,megaTabView->Bounds().Height()-megaTabView-
>TabHeight()-10),"Color Picker",NULL,B_FOLLOW_LEFT|B_FOLLOW_TOP));

    outView->AddChild(outcView);
    //aTab->SetView(outView);
    megaTabView->AddTab(outView);

    TheView->AddChild(megaTabView);
    Visable=true;

    return TheView;
}

void OutOfSight() {
MasterColor->Halt();
TheWorld->ILoop->LockLooper();
LockLooper();
Visable=false;
UpdatingView=false;
UnlockLooper();
TheWorld->ILoop->UnlockLooper();
}

virtual void MessageReceived(BMessage * message) {
    float Value;
    bigtime_t now;
    ValSet * VS;
    //cout <<"LG got BMessage..." << endl;
    switch (message->what) {
        case 'SelC': //channel selected
            //pass this on to the interface looper - don't want
to waste time here.
            TheWorld->ILoop->PostMessage(new BMessage
(*message));

            break;
        case 'AddC': //component is being added
            //pass this on to the interface looper - don't want
to waste time here.
            {
                //cout << "Got add plug message" << endl;
                ChannelListItem * NewPlug;
                if (message->FindPointer("Item",(void**))
&NewPlug)==B_OK) {
                    int Loc = GetPlugList()->IndexOf(NewPlug);

                    //initialize supplementary information
                    ChanData.AddItem(malloc(sizeof(ValSet)),
Loc);

                    ((ValSet*)ChanData.ItemAt(Loc))-
>Color=ChanColors->Color();
                }
            }

            break;
        case 'AltC': //component is being altered
            //pass this on to the interface looper - don't want

```

```

to waste time here.
    {
        //cout << "Got alter plug message" << endl;
        ChannelListItem * NewPlug;
        if (message->FindPointer("Item", (void**)
&NewPlug)==B_OK) {
            int Loc = GetPlugList()->IndexOf(NewPlug);
            //set color appropriately
            ((ValSet*)ChanData.ItemAt(Loc))-
>Color=ChanColors->Color();
        }
    }

    break;
case 'pchg': //some parameter changed
    Period=log10f(0.01+Activity->GetValueAndClean().
f*0.9);
    Period*=Period*2;

    //Mode stuff - will be fixed later...
    Mode->GetValueAndClean(); //clear dirty bit whether
we use it or not
    if (!OpLock) { //only change the mode if we are
allowed to
        int OldMode=OpMode;
        float NowMode = Mode->GetValue().f;
        if (NowMode > 0.8) {
            OpMode = LG_MODE_1HOT;
        }
        else if (NowMode >0.5) {
            OpMode = LG_MODE_CHASE;
        } else if (NowMode > 0.25) {
            OpMode = LG_MODE_COLOR;
        } else if (NowMode > 0.05) {
            OpMode = LG_MODE_RANDOMIZE;
        } else {
            OpMode = LG_MODE_OFFLINE;
        }
    }

    if (!UpdatingView && Visable) {
        UpdatingView=true;
        TheWorld->ILoop->PostMessage('UpVw');
    }
    //Period=fabs(Period);
    if (OpMode==LG_MODE_COLOR) Pulse->SetValue((CHData)
{0.0});
    break;

case A_PULSE:
    TheWorld->Pulser->GotIt(this);
    if (OpMode==LG_MODE_OFFLINE ||
OpMode==LG_MODE_BLACKOUT || OpMode==LG_MODE_SOLOTEST) break;
    now = real_time_clock_usecs();

    //fader stuff
    Value=(now-LastBeat)/(Period*1000*1000);
    if ((Value < 1.0 && fabs(Value - MasterFader)
>0.0001) || (MasterFader < 1.0 && Value >= 1.0)) {

```



```

MasterFader=Value;
if (MasterFader > 1.0) MasterFader=1.0;
//cout << "Fade"<<(now-LastBeat)<<"!" <<
MasterFader << endl;
for (int i = 0; i<GetPlugList()->CountItems();
i++) {
    VS = (ValSet*) ChanData.ItemAt(i);
    VS->CurrentValue = VS->OldValue*(1.0-
MasterFader) + VS->NextValue*MasterFader;
    ((PlugListItem *)GetPlugList()->ItemAt(i))-
>SetValue((CHData){VS->CurrentValue});
}
}

if ((1.01 - Activity->GetValue().f)*8*1000*1000 >
(now-LastBeat)) {
    break;
}
case 'boom': //we got the beat
    if (message->what=='boom') Pulse->GetValueAndClean
()); //acklowdge event (needed to avoid overload)
    if (OpMode==LG_MODE_OFFLINE) break;
    LastBeat=real_time_clock_usecs();
    //cout << "Boom! " << OpMode << endl;
    srand(LastBeat);

    if (OpMode==LG_MODE_SOLOTEST) {
        ChaseStep=SelectedChan;
    } else if (OpMode==LG_MODE_1HOT) {
        int NewStep, ICount;
        if ((ICount=GetPlugList()->CountItems())>1)
{ //We don't want to repeat the same step twice if we can avoid it
            while (ChaseStep==(NewStep=rand() %
GetPlugList()->CountItems())) {}
            } else {
                NewStep=1;
            }
            ChaseStep=NewStep;
        } else {
            ChaseStep++;
            if (ChaseStep < 0 || ChaseStep >= GetPlugList
()->CountItems()) ChaseStep=0;
        }

        ValSet * CVals;
        for (int i = 0; i<GetPlugList()->CountItems(); i++)
{
            CVals=(ValSet*)ChanData.ItemAt(i);
            CVals->OldValue = CVals->CurrentValue;
            if (OpMode==LG_MODE_COLOR) {
                CVals->NextValue=1.0;
            } else if (OpMode==LG_MODE_BLACKOUT) {
                CVals->NextValue=0;
            } else if (OpMode==LG_MODE_CHASE ||
OpMode==LG_MODE_SOLOTEST || OpMode==LG_MODE_1HOT) {
                if (i==ChaseStep) {
                    CVals->NextValue = 1.0;
                } else {

```

```

        if (OpMode==LG_MODE_SOLOTEST) {
            CVals->NextValue=0;
        } else {
            CVals->NextValue = ((ValSet*)
ChanData.ItemAt(i))->CurrentValue * 0.9 * Blur->GetValue().f;
        }
    }
    } else {
        CVals->NextValue = (rand() % 256)/255.0;
    }
    if ((OpMode==LG_MODE_COLOR ||
OpMode==LG_MODE_RANDOMIZE) && Tolerance->GetValue().f < 0.95) {
        float m = fabs(1.0-ColorMatch(CVals->Color.
h,CVals->Color.s,Hue->GetValue().f*2.0*M_PI,Saturation->GetValue().f));
        m=pow(m,pow(2,(511.0/2-Tolerance->GetValue
().f*511.0)/80.0));
        CVals->NextValue*=m;

        //CVals->NextValue=fabs(1.0-ColorMatch(CVals-
>Color.h,CVals->Color.s,Hue->GetValue().f*2.0*M_PI,Saturation->GetValue
().f));
    }

    //CVals->NextValue=fabs(1.0-ColorMatch(CVals-
>Color.h,CVals->Color.s,Hue->GetValue().f*2.0*M_PI,Saturation->GetValue
().f));

    //cout << "Match["<<i<<"]: "<<fabs(1.0-
ColorMatch(CVals->Color.h,CVals->Color.s,Hue->GetValue().f*2.0*M_PI,
Saturation->GetValue().f))<<endl;
}
    if (GetPlugList()->CountItems()>2 &&
(OpMode==LG_MODE_RANDOMIZE || OpMode==LG_MODE_COLOR)) { //Enforce
requested intensity...
        float avg,scale,dest;
        float max,min,pk;
        int fails;
        scale=1.0;
        dest=Intensity->GetValue().f;
        bool cont;
        cont=true;
        int counter=0;
        int icount = GetPlugList()->CountItems();
        while (cont && counter<10) {
            counter++;
            fails=0;
            avg=0.0;
            max=0.0;
            min=1.0;
            for (int i = 0; i<icount; i++) {
                VS = (ValSet*) ChanData.ItemAt(i);
                VS->NextValue*=scale;
                if (VS->NextValue>1.0) {
                    VS->NextValue=1.0;
                    fails++;
                }
                if (VS->NextValue>max) {
                    max=VS->NextValue;
                }
                if (VS->NextValue<min) {

```

```

        min=VS->NextValue;
    }
    avg+= VS->NextValue;
}
avg/=icount;
scale = dest/avg;
//printf("Dest: %f, Fails: %d, Scale: %f,
Avg: %f\n",dest,fails,scale,avg);
    if (fails*3>icount*2 || fabs(avg-dest)<0.03)
cont=false;
    }
    scale=avg/fabs(avg-min);
    if ((1-avg)/fabs(max-avg)<scale) scale =(1-
avg)/fabs(max-avg);
    if (scale>1.05 && OpMode==LG_MODE_RANDOMIZE) {
        //cout << "fixing values by " << scale <<
endl;
        for (int i = 0; i<icount; i++) {
            VS = (ValSet*) ChanData.ItemAt(i);
            VS->NextValue=(VS->NextValue-avg)
*scale+avg;
            if (VS->NextValue>1.0) VS-
>NextValue=1.0;
            if (VS->NextValue<0.0) VS-
>NextValue=0.0;
        }
    }
}

    if (OpMode==LG_MODE_BLACKOUT ||
OpMode==LG_MODE_SOLOTEST || OpMode==LG_MODE_COLOR) {
        for (int i = 0; i<GetPlugList()->CountItems();
i++) {
            VS = (ValSet*) ChanData.ItemAt(i);
            VS->CurrentValue = VS->NextValue;
            ((PlugListItem *)GetPlugList()->ItemAt(i))-
>SetValue((CHData){VS->CurrentValue});
        }
        MasterFader=0.0;
        break;
    default:
        AuroranComponent::MessageReceived(message);
    }
}

virtual void InterfaceMessage(BMessage * message) {
    void * ptemp;
    hsv_fcolor TheColor;
    BWindow * hWnd;
    if ((hWnd=MasterColor->Window())==NULL) return;
    if (!Visable) return;
    if (hWnd->LockWithTimeout(5000)!=B_OK) {
        TheWorld->Pulser->OneShot(TheWorld->ILoop,new BMessage
(*message));
        return;
    }
}

```

```

    }

    switch (message->what) {
        case 'Colr': //New Color selected...

            //printf("New color!!!\n");
            TheColor = MasterColor->Color();
            Hue->SetValue((CHData){TheColor.h/(2*M_PI)});
            Saturation->SetValue((CHData){TheColor.s});
            Intensity->SetValue((CHData){TheColor.v});
            Tolerance->SetValue((CHData){MasterColor->Tolerance

        ));

            break;
        case 'OpLk': //Lock Operating mode toggled
            OpLock=!OpLock;
            break;
        case 'OpMd'+0: //Lock Operating mode changed
        case 'OpMd'+1: //Lock Operating mode changed
        case 'OpMd'+2: //Lock Operating mode changed
        case 'OpMd'+3: //Lock Operating mode changed
        case 'OpMd'+4: //Lock Operating mode changed
        case 'OpMd'+5: //Lock Operating mode changed
        case 'OpMd'+6: //Lock Operating mode changed
        case 'OpMd'+7: //Lock Operating mode changed
        case 'OpMd'+8: //Lock Operating mode changed
            OpMode=message->what-'OpMd';
            PostMessage('boom');
            break;
        case 'SelC': //channel selected...
            //printf("Whee channel selected!\n");
            {
                ChannelListItem * Item;
                if (message->FindPointer("Item", (void**)&Item)

==B_OK) {

                    //cout << "Ptr: " << Item << endl;
                    int Loc = GetPlugList()->IndexOf(Item);
                    //printf("Item: %d\n", Loc);
                    if (Loc>=0) {
                        hsv_fcolor c=((ValSet*)ChanData.ItemAt

(Loc))->Color;

                        ChanColors->SetColor(c);
                    }
                    SelectedChan=Loc;
                    if (OpMode==LG_MODE_SOLOTEST) {
                        PostMessage('boom');
                    }
                }
            }
            break;
        case 'CHTM': //the channel tools stuff got a message...
            if (message->FindPointer("Owner", &ptemp)==B_OK) {
                ((ChannelTools *)ptemp)->InterfaceMessage

(message);

            }
            break;
        case 'UpVw':
            UpdatingView=false;
            if (Visable) {

```

```

        MasterColor->SetTolerance(Tolerance->GetValue
    ()).f);
        MasterColor->SetColor(hsv_fcolor(Hue->GetValue
    ()).f*2*M_PI,Saturation->GetValue().f,Intensity->GetValue().f));
        //usleep(5000);
    }
    break;
default:
    AuroranComponent::InterfaceMessage(message);
}
hWnd->Unlock();
}

status_t Save(BPositionIO * object) {
//ChannelTools(this, CHT_PLUGS, GetPlugList(), ParentPlug(),new
BMessage('AddC'),new BMessage('AltC'),new BMessage('DelC'),new BMessage
('SelC'), BRect(0,0,megaTabView->Bounds().Width()/2,megaTabView-
>Bounds().Height()-megaTabView->TabHeight()),"Channel",B_FOLLOW_LEFT,
B_FOLLOW_TOP,B_WILL_DRAW);
    SaveChannelList(object, this,GetPlugList(),ParentPlug(),
&ChanData, sizeof(ValSet));

    return B_OK;
}

status_t Load(BPositionIO * object, uint32 version) {

    LoadChannelList(object, this, GetPlugList(),ParentPlug(),
&ChanData, sizeof(ValSet));

    return B_OK;
}
protected:
    virtual char * MyName() {return "Lamp Group";}
    virtual char * DefaultCustom() {return "LG";}
};

AuroranComponent * MakeComponent(ComponentDescriptor * Who,
AuroranParameters * Params) {return new Component(Params);}

extern "C" _EXPORT ComponentDescriptor* GetDesc();

ComponentDescriptor* GetDesc() {
    return &MyDescriptor;
}

```

Filename: COMMThread.h

```
#include "ThreadPrimitive.h"
#include <SerialPort.h>
#include <Locker.h>

class COMMThread : public ThreadPrimitive {
public:
    COMMThread();
    ~COMMThread();
    void RunThread();
    void SetPort(const char * portname);
    void SetResolution(bool HiRes) {HiResMode=HiRes;}
    void NewData(short int chan, float value);
    float GetValue(short int chan) {return Values[chan];}
    float * GetValues() {return Values;}
private:
    bool HiResMode;
    bool PortSet;
    BLocker Start;
    BLocker Safety;
    BLocker QueueLock;
    BSerialPort PMXPort;
    // QD * UpdateQueue;
    int8 PendingUpdate;
    uint8 LastSentValue;
    float Values[120];
    int8 Updates[120];
};
```

Filename: COMMThread.cpp

```
#include "COMMThread.h"
#include <iostream.h>

COMMThread::COMMThread(): ThreadPrimitive(THREAD_NORM,"COM Port
Output") {
    LastSentValue=119;
    //UpdateQueue=NULL;
    for (int i =0;i<120;i++) {
        Updates[i]=(-1);
    }
    PendingUpdate=(-1);
    Start.Lock();
    PortSet=false;
    Go();
}

COMMThread::~~COMMThread() {

}

void COMMThread::SetPort(const char * portname) {
    if (PortSet=true) {
        PMXPort.ClearOutput();
        PMXPort.Close();
    }

    Safety.Lock();

    PMXPort.Open(portname);
    PMXPort.SetDataRate(B_9600_BPS);
    PMXPort.SetBlocking(0); //none
    PMXPort.SetTimeout(250000); //1/4 sec
    PMXPort.SetFlowControl(0);
    PMXPort.SetDTR(1);
    PMXPort.SetRTS(1);
    PortSet=true; //the user has set a port, so we can send stuff there
    Safety.Unlock();

    Start.Unlock();
}

void COMMThread::NewData(short int chan, float value) {
    int height = 0;
    if (Values[chan]==value) return; //no need to do anything - no
change
    Values[chan]=value;
    QueueLock.Lock();

    if (PendingUpdate<0) {
        PendingUpdate=chan;
        Updates[chan]=(-1);
    } else if (PendingUpdate!=chan) {
        //cout << "Other Data pending" << endl;
        int8 next = Updates[PendingUpdate];
        if (next<0) {
```

```

        Updates[PendingUpdate]=chan;
        Updates[chan]=(-1);
    } else {
        while (Updates[next]>=0 && next!=chan) { //set through
pending updates...
            //cout << ":" << (int)next << ":";
            next=Updates[next];
            height++;
        }
        if (next!=chan) { //we aren't already in the queue
            Updates[next]=chan;
            Updates[chan]=(-1);
        }
    }
}
QueueLock.Unlock();

//if (Updates[chan]==(-1)) cout << "Height: " << height << endl;
}

void COMMThread::RunThread() {
    char data[3];
    short int chan;
    char steps;

    Start.Lock();

    while (kill==false) {
        if (PortSet) { //only do stuff if we are sending stuff somewhere
            Safety.Lock();

            LastSentValue++;
            LastSentValue%=120;
            chan = LastSentValue;

            steps=6;
            while(PendingUpdate<0 && (steps) && kill==false) { //no new
data
                //snooze(1145); //time to send 1 character
                steps--;
            }
            if (PendingUpdate>=0) { //Queue not empty
                QueueLock.Lock();

                chan = PendingUpdate;
                PendingUpdate=Updates[chan];
                //cout << "Sending CH: " << chan << " next is " <<
(int)PendingUpdate << endl;
                QueueLock.Unlock();
            }

            data[0]=chan+128;
            chan=((short int) (Values[chan]*0x3FFF)) % 0x4000;
            data[1]=chan>>7;
            data[2]=chan&127;
            if (data[1]> 127) data[1]=127; //note: this used to be ">124"

```



```
        PMXPort.Write(&data, HiResMode?3:2);
        Safety.Unlock();
        snooze(HiResMode?3438:2292); //give time for 3 characters to
get sent...
        //snooze(3438/2); //give time for 1.5 characters to get sent...
    } else snooze(8192);
}

    Start.Unlock();
}
```

Filename: PMXOutput.cpp

```
#include "Component.h"
#include "ResourceSet.h"
#include <stdio.h>
#include <iostream.h>
#include <SerialPort.h>
#include <CheckBox.h>
#include "COMMThread.h"

//this resource management stuff is taken from one issue of the
BMessage newsletter...
//http://www.be.com/developers/bmessage/issue09.html
#include <Locker.h>
#include <TranslatorFormats.h>

//*****

static int32 gResourcesCreated = 0;
static BResourceSet gResources;

static BResourceSet& Resources()
{
    // Return the BResourceSet object for this image, initializing
    // if needed.
    if ((gResourcesCreated&2) != 0) {
        // Quick return if already initialized.
        return gResources;
    } else if (atomic_or(&gResourcesCreated, 1) == 0) {
        // Not yet initialized -- do so.
        gResources.AddResources(&gResourcesCreated);
        // Mark that we are done.
        atomic_or(&gResourcesCreated, 2);
    } else {
        // Wait for resources to be initialized.
        while ((gResourcesCreated&2) == 0) snooze(50000);
    }
    return gResources;
}

AuroranComponent * MakeComponent(ComponentDescriptor * Who,
AuroranParameters * Params);

static ComponentDescriptor MyDescriptor = {
    A_DESC_VER,
    A_NORMAL_DESC,
    NULL,
    "PMX Out",
    'TKS_',
    'PX',
    0,
    ACC_OUTPUT,
    &MakeComponent
};

const uint32 SAVE_Values = 1;
const uint32 PMX_HIRES = 2;
```

```

class PMX : public AuroranComponent {
private:
    uint32 MyFlags;
    BSerialPort PMXPort;
    COMMThread COMM;
public:
    PMX(AuroranParameters * Params) : AuroranComponent(Params,
DefaultCustom()) {
        AutoSetColors();
        MyIcon=(BBitmap *)Resources().FindBitmap(B_PNG_FORMAT,
2001);
        MyFlags=0;

        BeginSBundle("World 0");
        char name[4] = {'0','0','0',0};
        BMessage * msg;
        for (short int i=0;i<120;i++) {
            msg=new BMessage('CDat');
            msg->AddInt16("CH",i);
            RegisterSocket(BString("CH ").Append(name).String(),
StdChan_Intensity,msg);
            if (name[2]=='9') {
                name[2]='0';
                if (name[1]=='9') {
                    name[0]++;
                    name[1]='0';
                } else {
                    name[1]++;
                }
            } else {
                name[2]++;
            }
        }
        EndSBundle();
        char devName[B_OS_NAME_LENGTH];
        PMXPort.GetDeviceName(0, devName);
        //printf("PMX Output using port: %s\n",devName);
        /*PMXPort.Open(devName);
        PMXPort.SetDataRate(B_9600_BPS);
        PMXPort.SetBlocking(0); //none
        PMXPort.SetDTR(1);
        PMXPort.SetRTS(1);*/
        //COMM.SetPort(devName);

    }
    ~PMX() {
        //printf("%s: Closing serial port\n", CustomName);
        PMXPort.ClearOutput();
        PMXPort.Close();
        //printf("%s: Serial port closed\n", CustomName);
    }

    virtual void MessageReceived(BMessage * message) {
        SocketListItem * MsgSrc;
        //printf("px\n");
        switch (message->what) {
            case ('CDat'): {
                //char data[3];

```

```

        short int chan=0;
        float value=0.0;
        //printf("pxmsg \n");
        if (message->FindPointer("SocketPtr", (void**))
&MsgSrc)==B_OK) {
        //printf("sktfound \n");
        value=MsgSrc->GetValueAndClean().f;
        if (message->FindInt16("CH",&chan)==B_OK) {
            //if (message->FindFloat("value",&value)==B_OK)
{
            //cout << "To: " << chan << " Value: " <<
value << endl;

            /*data[0]=chan+128;
            chan=((short int) (value*0x3FFF)) % 0x4000;
            data[1]=chan>>7;
            data[2]=chan&127;
            if (data[1]> 124) data[1]=127;
            PMXPort.Write(&data,3);*/
            if (value >= 0.0) {
                if (value > 1.0) value=1.0;
                COMM.NewData(chan,value);
            } else { //reply time
                BMessage reply('Rply');
                reply.AddDouble("returnval",COMM.
GetValue(chan));
                //cout << "Sending " << COMM.GetValue
(chan) << "for: " << chan << endl;
                message->SendReply(&reply);
                //cout << "Sent" << endl;
            }
        }
    }
} else {
    printf("Fint skt ptr failed!!!\n");
}

break;
}
case 'Rply': {
    cout << "REply!" << endl;
}
}
AuroranComponent::MessageReceived(message);
}

BView * GetView(BRect frame) {
    BView * TheView = AuroranComponent::GetView(frame);

    BControl * NC;

    //Save to disk checkbox
    TheView->AddChild(NC = new BCheckBox(BRect
(2,36+50,400,50+50),"chk_Save","Save Output Levels to Disk",NULL));
    if ((MyFlags & SAVE_Values)) NC->SetValue(B_CONTROL_ON);

    //Hi-Res PMX
    TheView->AddChild(NC = new BCheckBox(BRect

```

```

(2,36+50+20,400,50+50+20),"chk_HiRes","Use Hi-Res (14- instead of 7-
bit) PMX output",NULL));
    if ((MyFlags & PMX_HIRES)) NC->SetValue(B_CONTROL_ON);

    //Serial port selector
    BPopupMenu * BPM;
    BMenuItem * Sig;
    BMessage * msg;

    BPM = new BPopupMenu("Port");
    BPM->SetFontSize(12);
    char devName[B_OS_NAME_LENGTH];

    printf("Active Serial Port: %d\n", ((MyFlags>>8)&0xFF));
    for (int s = 0; s<PMXPort.CountDevices(); s++) {
        PMXPort.GetDeviceName(s, devName);
        msg = new BMessage('SerP');
        msg->AddInt8("Port",s);
        BPM->AddItem(Sig = new BMenuItem(devName,msg,0,0));
        Sig->SetTarget(TheWorld->ILoop);
        if (((MyFlags>>8)&0xFF)==(s+1)) {
            Sig->SetMarked(true);
        }
    }

    BMenuField * MenuOpt;
    TheView->AddChild(MenuOpt = new BMenuField(BRect
(2,36,180,57), "List_Serial", "Serial Port:", BPM,true));
    MenuOpt->SetFontSize(12);
    //MenuOpt->SetDivider(20);

    return TheView;
}

virtual void InterfaceMessage(BMessage * message) {
    int8 ctemp;
    switch (message->what) {
        case ('SerP'):
            if (message->FindInt8("Port",&ctemp)==B_OK) {
                printf("Selected Serial Port %d!\n",ctemp);
                MyFlags&=~(0xFF<<8);
                MyFlags|=((ctemp+1)<<8);
                printf("Active Serial Port: %d\n", ((MyFlags>>8)
&0xFF));

                char devName[B_OS_NAME_LENGTH];
                if (PMXPort.GetDeviceName(ctemp, devName)
==B_OK) {
                    printf("PMX Output using port: %s\n",
devName);
                    COMM.SetPort(devName);
                }
            }
            break;
        default:
            AuroranComponent::InterfaceMessage(message);
    }
}

```

```

void ProcessView(BString & ViewName, BMessage * msg) {
    if (ViewName=="chk_Save") {
        int32 TheValue;

        if (msg->FindInt32("_val",&TheValue)==B_OK) {
            if (TheValue==B_CONTROL_ON) {
                MyFlags|=SAVE_Values;
            } else {
                MyFlags&=(~SAVE_Values);
            }
        }
    } else if (ViewName=="chk_HiRes") {
        int32 TheValue;

        if (msg->FindInt32("_val",&TheValue)==B_OK) {
            if (TheValue==B_CONTROL_ON) {
                MyFlags|=PMX_HIRES;
            } else {
                MyFlags&=(~PMX_HIRES);
            }
            COMM.SetResolution(MyFlags & PMX_HIRES);
        }
    } else {
        AuroranComponent::ProcessView(ViewName, msg);
    }
}

virtual status_t Save(BPositionIO * object) {
    int itemp = MyFlags | 0xFF00;
    uint8 ctemp;

    //write flags
    object->Write(&itemp,4);

    //write channel values
    if ((MyFlags & SAVE_Values)) {
        object->Write(COMM.GetValues(),sizeof(float)*120);
    }

    //save device name
    char devName[B_OS_NAME_LENGTH | 256];
    PMXPort.GetDeviceName(((MyFlags>>8)&0xFF)-1, devName);
    ctemp=strlen(devName)+1;
    object->Write(&ctemp,sizeof(uint8));
    object->Write(devName,ctemp);
    return B_OK;
}

virtual status_t Load(BPositionIO * object, uint32 version) {
    object->Read(&MyFlags,4);

    COMM.SetResolution(MyFlags & PMX_HIRES);

    if ((MyFlags & SAVE_Values)) {
        object->Read(COMM.GetValues(),sizeof(float)*120);
    }
}

```

```

//Set Serial Port
char devName[B_OS_NAME_LENGTH | 256];
if ((MyFlags & 0xFF00)==0xFF00) {
    printf("Got port byname\n");
    int8 size;
    bool PortFound;
    object->Read(&size,sizeof(int8));
    size=object->Read(devName,size);
    if (size <0) size =0;
    devName[size]=0;

    char devName2[B_OS_NAME_LENGTH];
    for (int s = 0;s<PMXPort.CountDevices();s++) {
        PMXPort.GetDeviceName(s, devName2);
        if (strcmp(devName,devName2)==0) {
            MyFlags&=0xFFFF00FF;
            MyFlags|=((s+1)<<8);
            PortFound=true;
        }
    }

    if (PortFound) COMM.SetPort(devName);

} else if ((MyFlags>>8)&0xFF > 0) {
    printf("Got port by number\n");

    if (PMXPort.GetDeviceName((MyFlags>>16)&0xFF-1, devName)
==B_OK)
        COMM.SetPort(devName);
}
printf("PMX Output using port: %s\n",devName);

return B_OK;
}

virtual ComponentDescriptor * GetDesc() {return &MyDescriptor;}
protected:
virtual char * MyName() {return "PMX Out";}
virtual char * DefaultCustom() {return "PMX";}
};

AuroranComponent * MakeComponent(ComponentDescriptor * Who,
AuroranParameters * Params) {return new PMX(Params);}

extern "C" _EXPORT ComponentDescriptor* GetDesc();

ComponentDescriptor* GetDesc() {
    return &MyDescriptor;
}

```

Filename: LightBar.h

```
#include <Window.h>
#include <View.h>
#include <Locker.h>
#include <iostream.h>

class LightView : public BView {
    BBitmap * m_bitmap;           // bitmap to drag
    BView * m_bitmapView;        // for drawing into bitmap
public:
    float Values[10];
    rgb_color Colors[11];
    BLocker ProtectValues;
    bool ChangingValues[10];
    bool LockedValues[11];
    LightView();
    ~LightView();
    void Draw(BRect updateRect);
};

class LightBar : public BWindow {
private:
public:
    LightView * MyView;
    LightBar() : BWindow(BRect(50,500,480+50,500+47),"Light Bar",
        B_FLOATING_WINDOW,B_NOT_CLOSABLE|B_NOT_ZOOMABLE|B_NOT_RESIZABLE|
        B_ASYNCHRONOUS_CONTROLS) {AddChild(MyView = new LightView());}
    virtual void MessageReceived(BMessage * message);
};
```


Filename: Lightbar.cpp

```
#include "LightBar.h"
#include <iostream.h>
#include <Message.h>
#include <Bitmap.h>

LightView::LightView() : BView(BRect(0,0,480,48),"view_Lights",
B_FOLLOW_NONE,B_WILL_DRAW) {
    SetViewColor((rgb_color){0,0,0,255});

    BRect r(0,0,47,47);
    m_bitmap = new BBitmap(r, B_RGB32, true);
    m_bitmapView = new BView(r, "_bm", B_FOLLOW_NONE, B_WILL_DRAW);
    m_bitmap->Lock();
    m_bitmap->AddChild(m_bitmapView);
    m_bitmapView->SetLowColor((rgb_color){0,0,0,255});
    m_bitmapView->FillRect(BRect(0,0,47,47), B_SOLID_LOW);
    m_bitmapView->Sync();
    m_bitmap->Unlock();
    for (int i = 0 ; i<10;i++) {Values[i]=0.0;ChangingValues[i]=false;}
}

LightView::~LightView() {
    delete m_bitmap;
    //delete m_bitmapView;
}

void LightView::Draw(BRect updateRect) {

    int Left = updateRect.left/48;
    int Right = updateRect.right/48;

    //cout << "Draw! " << Left << " :: "<<Right <<endl;
    for (int i = Left; i<=Right; i++) {
        m_bitmap->Lock();
        m_bitmapView->SetHighColor((rgb_color){Values[i]*Colors[i].red,
Values[i]*Colors[i].green,Values[i]*Colors[i].blue,255});
        m_bitmapView->FillEllipse(BRect(1,1,46,46),B_SOLID_HIGH);
        //m_bitmapView->Sync();
        m_bitmapView->SetHighColor((rgb_color){Values[i]*Colors[i].red/
2,Values[i]*Colors[i].green/2,Values[i]*Colors[i].blue/2,255});
        m_bitmapView->StrokeEllipse(BRect(1,1,46,46),B_SOLID_HIGH);
        m_bitmapView->Sync();
        m_bitmap->Unlock();
        //Sync();
        DrawBitmap(m_bitmap,BPoint(i*48,0));
    }
    //Flush();
    //Sync();
}

void LightBar::MessageReceived(BMessage * message) {
    switch (message->what) {
        case 'ReDr': { //new value
            int8 WhichLight;
            if (message->FindInt8("CH",&WhichLight)==B_OK) {
                MyView->ProtectValues.Lock();
            }
        }
    }
}
```

```

        MyView->ChangingValues[WhichLight]=false;
        MyView->ProtectValues.Unlock();
        MyView->Draw(BRect(WhichLight*48,0,
WhichLight*48+47,48));
    }
    break;
}
default:
    BWindow::MessageReceived(message);
}
}

```

Filename: Component.cpp

```
#include "Component.h"
#include "ResourceSet.h"
#include <stdio.h>
#include <iostream.h>
#include <SerialPort.h>
#include "Pulse.h"
#include "MaxMinCalculator.h"
#include "PercentControl.h"
#include <alloc.h>
#include "LightBar.h"
#include <OS.h>
#include <ColorControl.h>
#include <RadioButton.h>
#include <Screen.h>

//this resource management stuff is taken from one issue of the
BMessage newsletter...
//http://www.be.com/developers/bmessage/issue09.html
#include <Locker.h>
#include <TranslatorFormats.h>

//*****

static int32 gResourcesCreated = 0;
static BResourceSet gResources;

static BResourceSet& Resources()
{
    // Return the BResourceSet object for this image, initializing
    // if needed.
    if ((gResourcesCreated&2) != 0) {
        // Quick return if already initialized.
        return gResources;
    } else if (atomic_or(&gResourcesCreated, 1) == 0) {
        // Not yet initialized -- do so.
        gResources.AddResources(&gResourcesCreated);
        // Mark that we are done.
        atomic_or(&gResourcesCreated, 2);
    } else {
        // Wait for resources to be initialized.
        while ((gResourcesCreated&2) == 0) snooze(50000);
    }
    return gResources;
}

AuroranComponent * MakeComponent(ComponentDescriptor * Who,
AuroranParameters * Params);

static ComponentDescriptor MyDescriptor = {
    A_DESC_VER,
    A_NORMAL_DESC,
    NULL,
    "Light Bar",      //Menu Name
    'TKS_',           //Creator
    'LB',              //ID
```

```

        1, //Version
        ACC_OUTPUT, //Class
        &MakeComponent
    };

#define World (*TheWorld)

class Component : public AuroranComponent {
private:
    LightBar * MyLB;
    BColorControl * ColorPicker;
    int CurrentLamp;
public:
    //Don't Touch
    virtual ComponentDescriptor * GetDesc() {return &MyDescriptor;}

    Component(AuroranParameters * Parms) : AuroranComponent(Parms,
DefaultCustom()) { //set up
        HighlightBG=(rgb_color){250,110,120,128};
        NormalBG= (rgb_color){210,90,100,128};
        MyIcon=(BBitmap *)Resources().FindBitmap(B_PNG_FORMAT,
2001);

        char name[2] = {'0',0};
        BMessage * msg;
        for (char i=0;i<10;i++) {
            msg=new BMessage('NVal');
            msg->AddInt8("CH",i);
            RegisterSocket(BString("Light ").Append(name).String(),
StdChan_Intensity, msg);
            //if (name[1]=='0') {
                name[0]++;
            // name[1]='9';
            //} else {
            // name[1]--;
            //}
        }

        MyLB = new LightBar;
        MyLB->SetTitle(CustomName);
        //cout << "My Lightbar is: " << MyLB << endl;
        CurrentLamp=10;
        for (int i = 0;i<10;i++) { //set initial colors to white
            MyLB->MyView->Colors[i]=(rgb_color){255,255,255,255};
            MyLB->MyView->Values[i]=0.0;
            MyLB->MyView->LockedValues[i]=false;
            MyLB->MyView->ChangingValues[i]=false;
        }
        MyLB->Show();
        set_thread_priority(MyLB->Thread(),B_LOW_PRIORITY);

    }
    ~Component() { //Clean up
        //cout << "I am deleting: " << MyLB << endl;
        MyLB->Lock();
        MyLB->Quit();
    }
}

```

```

BView * GetView(BRect frame) {
    BView * TheView = AuroranComponent::GetView(frame);

    char radio_name[8] = "radio_0";
    char radio_lbl[8] = "Light 0";
    BRadioButton * RB;
    int i;

    for (i = 0; i < 10; i++) {
        TheView->AddChild(RB = new BRadioButton(BRect
(1, 30+16*i, 60, 45+16*i), radio_name, radio_lbl, new BMessage('NSel'+i)));
        RB->SetTarget(World.ILoop);
        radio_name[6]++;
        radio_lbl[6]++;
    }
    TheView->AddChild(RB = new BRadioButton(BRect
(1, 30+16*i, 60, 45+16*i), radio_name, "(none)", new BMessage('NSel'+i)));
    RB->SetTarget(World.ILoop);
    RB->SetValue(1);

    TheView->AddChild(new BStringView(BRect(120, 30, 300, 45),
"lbl_Colors", "Choose Gel Color:"));
    TheView->AddChild(ColorPicker = new BColorControl(BPoint
(120, 46), B_CELLS_32x8, 8, "BarColors", new BMessage('NClr'), false));
    ColorPicker->SetTarget(World.ILoop);
    ColorPicker->SetEnabled(false);

    return TheView;
}

void OutOfSight() { // BView about to go away...
    MyLB->MyView->LockedValues[CurrentLamp]=false; //unlock
current lamp
    CurrentLamp=10;
}

virtual void InterfaceMessage(BMessage * message) { //interface
messages
    if (message->what >= 'NSel' && message->what <='NSel'+10) {
        if (ColorPicker->Window()->LockWithTimeout(1000)==B_OK)
        {

            MyLB->MyView->LockedValues[CurrentLamp]=false; //
unlock current lamp
            CurrentLamp=message->what - 'NSel';
            if (CurrentLamp==10) { //none selected
                ColorPicker->SetEnabled(false);
            } else {
                MyLB->MyView->ProtectValues.Lock();
                MyLB->MyView->LockedValues[CurrentLamp]
=1; //lock the new lamp
                MyLB->MyView->Values[CurrentLamp]=1.0;
                MyLB->MyView->ProtectValues.Unlock();
                ColorPicker->SetValue(MyLB->MyView->Colors
[CurrentLamp]);
                ColorPicker->SetEnabled(true);
                if (MyLB->MyView->ChangingValues[CurrentLamp]
==false) {

                    BMessage * msg = new BMessage('ReDr');

```

```

        msg->AddInt8("CH",CurrentLamp);
        MyLB->PostMessage(msg);
        delete msg;
    }
}

    ColorPicker->Window()->Unlock();
} else { //lock failed
    World.Pulser->OneShot(World.ILoop,new BMessage
(*message));
}

    } else {
        switch (message->what) {
            case 'NClr': {
                MyLB->MyView->Colors[CurrentLamp] =
ColorPicker->ValueAsColor();
                if (MyLB->MyView->ChangingValues[CurrentLamp]
==false) {
                    BMessage * msg = new BMessage('ReDr');
                    msg->AddInt8("CH",CurrentLamp);
                    MyLB->PostMessage(msg);
                    delete msg;
                }
            }
            case A_APPLY_CHANGES: { //Apply changes
                MyLB->SetTitle(CustomName);
            } //DO NOT BREAK - we want to flow into the
standard Apply Changes
            default: {
                AuroranComponent::InterfaceMessage(message);
            }
        }
    }
}

virtual void MessageReceived(BMessage * message) {
    switch (message->what) {
        case 'NVal': { //new value
            int8 WhichLight;
            float Value;
            if (message->FindInt8("CH",&WhichLight)==B_OK) {
                if (message->FindFloat("value",&Value)==B_OK) {
                    if (Value >= 0.0 && Value <= 1.0) {
                        if (MyLB->MyView->LockedValues
[WhichLight]==false) {
                            MyLB->MyView->ProtectValues.Lock();
                            MyLB->MyView->Values[WhichLight]
=Value;
                            if (MyLB->MyView->ChangingValues
[WhichLight]==false) {
                                MyLB->MyView->ChangingValues
[WhichLight]=true;
                                BMessage * msg = new BMessage
('ReDr');
                                msg->AddInt8("CH",WhichLight);
                                MyLB->PostMessage(msg);
                                delete msg;
                            }
                            MyLB->MyView->ProtectValues.Unlock();

```

```

        }
        } else {
            BMessage reply('Rply');
            reply.AddDouble("returnval", MyLB->MyView->Values[WhichLight]);
            message->SendReply(&reply);
        }
    }
}

default:
    AuroranComponent::MessageReceived(message);
}

virtual status_t Save(BPositionIO * object) {

    //save window position
    BPoint WindowPos = MyLB->Frame().LeftTop();
    object->Write(&WindowPos, sizeof(WindowPos));

    //future use variable size matrix
    short int x=10;
    object->Write(&x, 2);
    x=0;
    object->Write(&x, 2);

    //colors
    for (int i = 0 ; i<10 ; i++) {
        object->Write(&(MyLB->MyView->Colors[i]), sizeof
(rgb_color));
    }

    //future use, channel names or other
    object->Write(&x, 2);
    return B_OK;
}

virtual status_t Load(BPositionIO * object, uint32 version) {
    if (version > 0) {
        BPoint WindowPos;
        object->Read(&WindowPos, sizeof(WindowPos));
        BPoint Limit = BScreen().Frame().RightBottom();
        if ( 0 <= WindowPos.x && WindowPos.x <= Limit.x && 0 <=
WindowPos.y && WindowPos.y <= Limit.y )
            MyLB->MoveTo(WindowPos);

        object->Seek(4, SEEK_CUR); //skip the next four bytes

        for (int i = 0 ; i<10 ; i++) {
            object->Read(&(MyLB->MyView->Colors[i]), sizeof
(rgb_color));
        }

    }
    MyLB->SetTitle(CustomName);
    return B_OK;
}

protected:
    virtual char * MyName() {return "Light Bar";}

```

```

        virtual char * DefaultCustom() {return "Bar";}
};

AuroranComponent * MakeComponent(ComponentDescriptor * Who,
AuroranParameters * Params) {return new Component(Params);}

extern "C" _EXPORT ComponentDescriptor* GetDesc();

ComponentDescriptor* GetDesc() {
    return &MyDescriptor;
}

```


Glossary

Desk: This refers to the desks in lighting booths containing large arrays of individual dimmer slides and other controls.

DMX512: Digital MultipleX – A digital lighting control protocol developed by the Entertainment Services & Technology Association, supporting 512 channels of 8 bit lighting values transmitted via twisted pair RS485 at 250kbps. Currently it the most widely used digital lighting protocol. Details of the standard may be found here: <http://www.dmx512-online.com/>.

Gel: This refers to the color filters used for theatrical lighting applications, such as those made by Rosco (<http://www.rosco-ca.com/>).

Gobo: Another lighting industry term, referring to patterns (not slides, but rather usually cut out of a piece of metal or printed on glass to sustain the high temperatures within high wattage lights) projected by lights.

Intelligent Light: Any light that may be controlled directly by some digital control system. This does not mean the light is capable of doing anything on its own, simply that it has a built in microcontroller or ASIC to process digital lighting data.

Moving Head: A light whose direction can be adjusted – generally by motors moving the light itself, though may also include lights whose direction is adjusted via a mirror with motors.

PMX: Pulsar MultipleX – A digital lighting control protocol developed by Pulsar and Clay Paky, supporting 15,360 channels of 7 or 14 bit lighting values transmitted via RS232 at 9600bps. It is primarily used by Pulsar and Clay Paky equipment, but provides a convenient standard for easy computer control via the serial port. That being the case, it has been adopted by some others as a means of transmitting data from a computer to an external device that then translates the data into DMX512 signals. This avoids the need for DMX512 specific hardware to be installed within a computer.