Boomer:
Next Generation Audio For Solaris

Garrett D'Amore
Sun Microsystems, Inc.
February 25, 2009

Table of Contents

1. Overview

The Boomer project aims to modernize the Solaris kernel-based audio system. Specifically, we hope to modernize the framework to support the popular Open Sound System API used by several popular free operating systems, including Linux and FreeBSD.

As part of this project, we will be providing new features, such as enhanced high definition (both high frequency and high resolution sampling) and multichannel (for example, 7.1 surround sound) audio support. We will also be expanding the set of supported devices, and making it easier to incorporate work done in the open source community more easily in the future. (Specifically, it will be easier to incorporate device drivers and audio middle-ware developed for Linux and FreeBSD.)

At the same time, we are fully committed to providing first class backwards compatibility. There is a large installed base of audio applications written for Solaris, and the Boomer project team understands the importance of those applications continuing to function properly on OpenSolaris.

2. Current State of Affairs

Solaris currently ships with a 10 year old 2nd generation audio subsystem known as SADA (Sun Audio Device Architecture). SADA (and indeed, the Sun API exposed to user applications) is STREAMS driven, although most of the details of STREAMS are not exposed directly to device drivers. SADA offers mixing capabilities to audio devices and applications, and supports a rich (and in some cases unnecessarily complex) set of capabilities for audio devices. Some these features have never been utilized, and some are no longer in use by any current audio devices. (We offer as an example of the former, the "Multi-Stream Codec" support. And for the latter, one can look to the "Compatible Mode" that some drivers can use to emulate a 1st generation audio device.)

At the same time, the number of actual audio devices that Solaris supports is almost vanishingly small, with many systems unable to use Solaris audio out of the box. In fact, it is not easy to locate add-in boards that Solaris can make use of, as nearly all off-the-shelf products currently sold are unsupported. And for those devices that Solaris does support, numerous features are not accessible to end users. Perhaps the most frequently missed of these is the ability to make use of more than 2 channels (stereo) of audio data. Most modern consumer audio devices have the ability to support 6 (5.1) or even 8 (7.1) channels of audio data.

Finally, as the popularity of Open Source has grown, an increasing number of interesting multimedia applications are being made available on Linux and FreeBSD, but either do not work at all, or work sub-optimally on Solaris. These applications range from games, to consumer multimedia (DVD player applications), to simple business productivity applications (VoIP). In order to broaden our market appeal, it is important that Solaris support the most popular Linux and FreeBSD audio API so that applications can easily be supported on Solaris.

3. Alternative Implementation (4Front)

However, Sun is not the only game in town for audio on Solaris. A 3rd party ISV-- 4Front - the original inventors of Open Sound System -- has developed a portable audio framework that runs on Solaris as well as other operating systems. This implementation supports the Open Sound System API on a greatly expanded set of devices, but has limited support for compatibility with the legacy Solaris audio(7i) API.

This audio framework has a number of good things about it. This includes greatly expanded audio device and feature support, including multichannel high definition audio and even AC3 (Dolby Digital) pass through support. In fact, Sun management believed in this product so much that we have entered into a partnership with 4Front to license their implementation for use in Solaris. This license includes a grant to Solaris to make unlimited use of their API, although it does bind Sun contractually to minimize API divergence. Sun also believes that API divergence is something to be avoided, since increased compatibility with Linux and FreeBSD is one of our major goals!

The 4Front code has a number of problems which prevent from being used directly, however. The 4Front product was designed for primarily single user workstations, and cannot scale to a large multi-user and multi-device system like a Sun Ray server. It suffers from a number of design defects involving improper synchronization between multiple threads, improper global variable use, etc. It also

was designed as a portable framework, so it carries with it quite a lot of baggage required to support other operating systems, while its least common denominator approach fails to provide critical features for Solaris. (For example, it lacks support for suspend/resume, or driver quiesce. It also does not properly support Solaris advanced interrupt architecture or proper DMA cache synchronization.) It also only provides SADA compatibility for a single audio device at a time.

In many respects, the 4Front OSS project is a work in progress, and the interfaces used between device drivers and the framework are rather unstable (specifically *not* binary compatible) and inconsistent. There were also a number of performance related concerns with the 4Front code, as their framework in several places makes extensive use of expensive operations (such as integer divide for non-constant divisors or divisors that aren't powers of two), that showed up during extensive review of their code.

Boomer makes extensive use of code from 4Front, but has significantly improved the portions it has used, and has completely rewritten other portions.

4. Goals

In the previous sections, we gave some background. From this background, and the standard Solaris expectations, the project team derived some specific goals for this project.

1. Boomer must support the existing set of Solaris audio applications, written to audio(7I).

2. Boomer must support the existing set of Solaris audio devices on both SPARC and Intel platforms.

3. Boomer must support the Open Sound System (OSS) API, and remain as true to the 4Front specs as reasonably possible.

4. Boomer must support multichannel (4.0, 5.1, 7.1) audio in the framework and on at least one device. Azalia HD audio can express up to 16 independent channels.

5. Boomer must support high definition (192 kHz frequency, 24 bit resolution) audio on at least some devices (specifically Intel Azalia – aka audiohd -- devices.)

6. Boomer must support in-kernel audio sample rate and data format conversion.

7. Boomer must be rock-solid stable. (This is kernel software we're talking about, after all!)

8. Boomer must be performant. (Some audio frameworks are known for consuming inordinate amounts of CPU.)

9. Boomer must support suspend/resume. This feature has become increasingly critical for mobile applications.

10. Boomer must support power management. That is, power(9e) must be possible (or an equivalent functionality) to reduce power on devices that are not in use.

11. Boomer must support the full expression of the Solaris DDI in device drivers, and must not limit drivers to any subset of the Solaris DDI. (For example, Boomer audio devices might take advantage of MSI, PCIe QoS enhancements, virtualization interfaces, or advanced DMA or FMA capabilities.)

12. Boomer should treat applications written using different interfaces as equal peers. That is, management applications such as mixer panels, should be able to see and manage applications equally, irrespective of whether they are using a “foreign” API or not.

13. Boomer should be extensible to support future, as yet unknown, APIs. We know that other systems use APIs we have not looked at, such as ALSA (Linux) and CoreAudio (Apple), and we'd like to avoid painting ourselves into corners.

14. Boomer should support an expanded set of audio devices. A survey was taken to determine the most popular devices amongst OpenSolaris enthusiasts, as well as the set of devices commonly included on motherboards and add-in boards available from typical retailers.

15. Boomer should support digital SPDIF interfaces. (But see notes about AC3 below.)

16. Boomer should have a device driver interface that facilitates porting devices drivers from Linux and FreeBSD.

17. Boomer should be able to scale to a future Sun Ray audio implementation. Specifically, it must be possible to support many hundreds of audio client applications, and in the future it should be possible to distinguish between “virtual” devices.

18. Boomer should be extensible to support new controls and features in audio devices. (New devices and growth here has made it clear that earlier predictions were inadequate. There are already standards for 10.2 audio, although we are not aware of any computer hardware that supports such properly. Who knows what the future may hold?)

19. Boomer should be extensible to support AC3 (Dolby Digital) pass-through in the future. This is important for playback in multimedia applications involving either DVD playback or streaming video from DVR type applications.

5. Non-Goals

In addition, there are a number of non-goals for this project, that the project team would like to make clear.

1. Boomer will not provide support for MIDI or wave table synthesis devices. These technologies have fallen out of favor, as modern systems no longer need the hardware off-load provided by

such hardware.

2. Boomer does not provide additional support (or rather it does not *expose* such support) for so-called “multi-stream codecs”. Boomer may try to make use of multiple codecs if present, but the algorithms used to choose between software mixing and hardware mixing are not visible to the user. Software mixing is always available to provide multiple stream access even when the hardware has multiple stream support.

3. Boomer does not provide support for “Traditional mode”. In fact, several key Solaris audio devices do not support this mode anymore. (Traditional mode bypasses a software mixer, and requires the device to be able to support various modes natively in an “exclusive use mode”.)

4. Boomer does not support the legacy **mixer**(7I) API. This is the legacy API is used for mixer panel applications. On Boomer, an OSS equivalent API (now documented in the **mixer**(7I) page) must be used to do device management and control. (A gstreamer plugin that uses the OSS equivalent API will be delivered along with Boomer, so that applications like mixer_applet2 and gnome-volume-control continue to work.)

5. Boomer makes no latency guarantees. While every effort to minimize latencies is made in the framework and associated drivers, there are no latency guarantees provided to applications. (Generally the actual latencies, while dependent on specific hardware and drivers, will be less than 20 ms. In some cases, far less. The project team believes that VoIP and consumer multimedia requirements are satisfied as long as the latencies do not exceed 40 ms. Note that individual device drivers can still be tuned to adjust interrupt frequencies, and hence latencies, using **driver.conf**(4) settings.)

6. As a consequence of item #3 above, Boomer is not attempting to provide support for “advanced” audio uses. Applications involving certain demanding digital signal processing, etc. may not be be satisfied with Boomer's latencies.

7. Boomer is not a “porting layer”, nor a “portable” DDI for device drivers. While device drivers should not be hard to port to Boomer from other sources, we believe developers of kernel software need to be at least somewhat versed in Solaris driver DDIs, and make use of them appropriately. It also, therefore, does not duplicate interfaces that are already provided by the Solaris DDI.

8. “Ordinary” applications (e.g. *realplayer*, *totem*, etc.) do not have the ability to manage global volume and other hardware mixer settings. Instead, “privileged” applications (e.g. *gnome-volume-control*, *mixer_applet2*, and *mixerctl*) are used to adjust global settings, while applications adjust their own levels relative to the global setting.

9. Sun Ray support is not required for Phase I. As Sun Ray is not officially supported at this time on OpenSolaris, this should not be too grievous a limitation. A future project will address this.

10. AC3 pass through is not required for Phase I. While this feature would be nice to have, practical considerations and significant challenges prevent us from delivering this in Phase I.

(Some devices might be able to generate AC3 on their output, from PCM input data. Those devices can be supported under Phase I.)

11. Boomer need not support (at least not initially) more than 8 channels of audio data, nor more than 24 bits of audio precision, nor frequencies higher than 192 kHz.

12. Boomer does not support devices that cannot do either 16-bit, 24-bit, or 32-bit linear PCM formats (either signed or unsigned).

6. Major Subsystems

The figure below shows a representation of the Boomer subsystems.

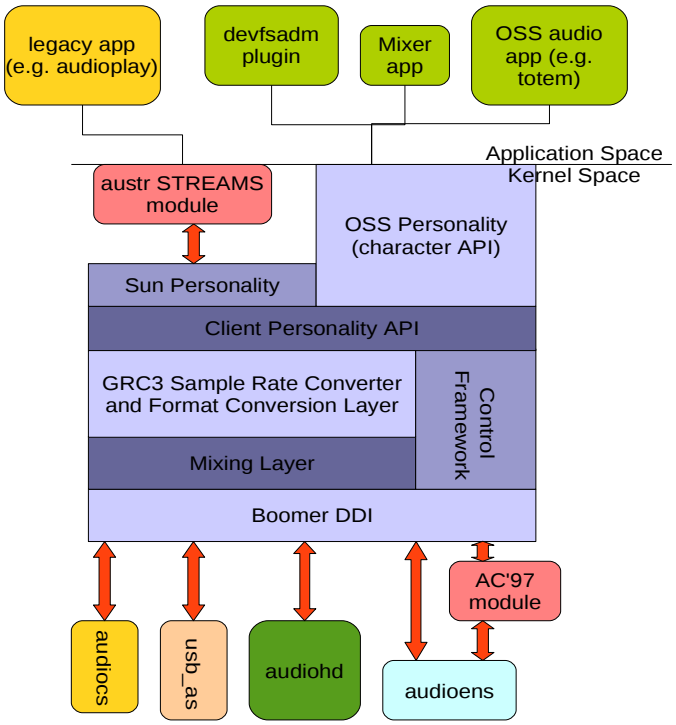

Illustration 1: Boomer Subsystems

The Boomer project delivers several significant subsystems, which together provide a complete kernel framework for audio on Solaris. In the illustration above, the various different colors represent different kernel modules. The bottom of illustration shows individual drivers, with various support layers in the middle (the square made up purple boxers represents the Audio Core, the main Boomer module). The following sections detail them one by one.

7. Audio Core

At the heart of the Boomer subsystem is the audio core. This subsystem, which is implemented in

the **audio**(7D) driver, provides all of the internal glue for the framework. It includes the logic to perform software mixing, audio format conversion (including sample rate and channel conversions), the client personality API implementation, and the device driver interfaces for audio. The **audio**(7D) module contains within it several other major subsystems. For many Boomer-native drivers (**audiopci**(7D), **audiohd**(7D), **audiocs**(7D)), the **audio**(7D) module is the only external dependency (other than the kernel itself!) that is required.

8. Client Personality Interface

A significant design goal of Boomer is to support multiple different interfaces equivalently. For example, applications using OSS should not behave substantially differently from applications using legacy Sun audio(7i). (Note, however, that the legacy interfaces for Solaris may not be able to express all of the features of an audio device properly. For example, there is currently no way that a legacy Solaris application can access volume controls for a rear pair of speakers independently from a front pair.) We also desired to leave our options open for future possible developments. For example, some day it may be desirable to support an API developed for MacOS, or the Linux ALSA API.

In order to facilitate this, Boomer is developed with the notion of multiple "Client Personalities" surrounding a common core. The client personalities communicate with the core using a relatively narrow well-structured internal (Project Private) interface. (This is enforced through the use of a specific header file, "audio_client.h", that only provides declarations for objects which the audio personalities need to access. Audio personalities must never #include other private audio header files.)

The framework chooses which audio personality a client application is using based on the minor node number that the application opens. (Note that the minor nodes themselves are "cloned", to allow for multiple simultaneous opens.)

A key component of this interface is the use of a "stream-specific" circular buffer. The buffer is used to exchange data between the core and personalities. Personalities are free to provide data in any "convertible" format (specified by use of specific function calls), and the core will consume data from the shared buffer (or for recording, produce it into the shared buffer), performing any necessary conversions as it goes. (Aside: the project team is currently considering options to allow the personalities to perform their own format conversions directly, and thereby avoid certain other complications in the framework). The buffers consist of head and tail counters, which are maintained as 64-bit non-wrapping integers. (Actual indexes are calculated from these pointers by performing the logical equivalent of a modulo operation against the configured buffer size, although for performance reasons an actual mathematical modulo operation is not performed.) These indexes are monotonically increasing, and are completely independent of any underlying physical counters or counters used in other streams.

The Boomer core will call into the client personality when a client application calls open(2), and will provide to the client application an **audio_client_t** structure corresponding to the actual client application. Each audio_client_t also has with it a pair of **audio_stream_t** structures, corresponding the playback and record directions. The audio streams operate independently of each other, and each has its own buffer.

Of course, there is also a master audio device structure for each real device. Client personalities can register private data for the device, or the client. There are also ways for one personality to access the private data of another personality. This is useful when two personalities collaborate – such as the personality used for the Sun audio interface and the personality used for the Sun audioctl interface.

The Boomer core handles minor node management, and the various DDI interfaces for the client personalities, although there are callbacks provided to personalties to trigger on certain DDI events or provide customization. (This is very important for some entry points – the handler for ioctl for example.)

9. Character API

Note that all minor nodes exported by the audio core are character devices, and with only two notable exceptions (the generic **/dev/sndstat** and **/dev/mixer** devices) are associated with the actual device node for the audio hardware in the device tree.

This holds true for nodes for all personalities, and is a significant departure from 4Front OSS in implementation. (4Front uses normal character devices, and uses a layered approach to provide a single STREAMs SADA device.) This implementation choice was made because it is easy to implement normal character devices in terms of STREAMs without loss of semantics, but it would not have been possible to do the reverse since Sun audio relies on STREAMs semantics for some of its behavior – particularly flow control and non-blocking semantics.

In order to simplify client personality implementation, Boomer common code implements much of the transport logic for client personalities, including a framework for dealing with ioctl, M_COPYIN, M_COPYOUT, and **put**(9e) and **srv**(9e) routines.

Note that the common **open**(9e) handler for Boomer does not enforce exclusive access (F_EXCL) opens. Since each open gets a “virtual” device of its own, there is no need to do such enforcement.

It is worth restating that the Boomer framework creates minor nodes on behalf of the driver, but the minor nodes are attached to the driver's dev_info node in the device tree. This architecture makes it easier for the Solaris DDI to recognize when a device is in use and do proper reference counting. (Note that the exceptions here are the pseudo nodes that are created for /dev/sndstat and /dev/mixer.)

10. STREAMs Module (austr)

In order to properly support the STREAMs semantics that are part of the audio(7I) API, and support a character device for the core as well, it was necessary to introduce a new STREAMs module, **austr**(7D). This is a STREAMs pseudo driver, and it exists primarily to provide a node to hang cloneable STREAMs minor nodes from. As typical with pseudo drivers, there is only one instance in the device tree.

The actual functionality for the austr module is located in the Sun personality in the audio core itself. Thus, the austr module is extremely lightweight, containing only an **_init**(9E) and **_fini**(9E) that call

into the audio core.

The book-keeping to ensure that minor nodes are properly tracked to physical drivers is done by having the STREAMs code perform an **ldi_open_by_dev**(9F) on a special internal node created by the framework for each device.

Internally, only the open and close operations make use of the LDI API. For all actual data transfer, simple function calls are used to ensure that a minimum performance penalty is incurred.

11. GRC3 Sample Rate Converter & Format Conversion Layer

Boomer has taken the sample rate conversion from OSS, and adapted it to our use. We have simplified and optimized the sample rate conversion somewhat, in an effort to further improve the performance of the code. As a result, the GRC3 converter we have bundled can only deal with 24-bit native-endian signed linear PCM data.

However, the format conversion layer converts from a client personality's preferred format to this internal native 24-bit format, and also deals with converting the data to the number of channels on the physical device, before passing it to GRC3. This code was heavily modified from OSS. The code can convert from ULAW, ALAW, and the various 8, 16, 24, and 32 bit PCM formats (signed, unsigned, native or reverse endian, etc.) Other formats are not supported natively in the kernel interface, and must be converted in user-land by application code.

12. Mixing Layer

Once audio data has been converted to 24 bit linear PCM, and has been sample rate converted to a common rate (determined by the audio device itself) the mixing layer mixes the audio streams together.

As part of this process, per-stream & per-channel adjustments attenuation adjustments are made to the samples. This allows an application to adjust its own volume levels, without disturbing master settings for the audio device.

Mixing streams normally results in adding their samples together (using signed arithmetic.) However, it is possible for multiple applications to drive the audio output beyond what is representable in the sample format. Therefore, at this point, an extra set of checks are also made to ensure that the process of mixing the audio streams does not cause this to happen. If it should be noticed, a global attenuation is made to the master volume. This attenuation is temporary, and fades away over time.

For recording, the mixing layer does a “fanout” of the audio data, and while per-stream attenuation is still applied, there is no additive volume adjustment that is necessary.

13. Sun Personality

The Sun Personality implements the specific semantics that are documented in **audio**(7I), and represent the legacy interface to Boomer.

There were however some significant problems, with this. Some of the worst of these problems were bugs that were discovered in applications and in the legacy APIs themselves.

The biggest problem is the fact that the API was designed for a single exclusive-use device. When the 2nd generation mixer was added, some new ioctls were added that applications could use to help with sharing the device, but most applications don't make use of them. Two legacy ioctls, AUDIO_GETINFO and AUDIO_SETINFO, might need to access either settings associated with hardware, or settings associated with the "application" (in some cases even within the same ioctl), and it is not always possible to tell which the application intended. The 2nd generation API used heuristics to figure this out, based on the order that /dev/audio and /dev/audioctl files were opened, but this heuristic does not always operate properly.

As a result, it is not always easy to predict when an application changes a setting, such as the volume or balance, whether the setting affects the master hardware volume, or a virtual setting (such as a sample count) associated only with the application. Making matters worse, sometimes which setting is affected *changes* during the life of the application.

For Boomer, we took a hard look at this, and decided that the best thing to do would be to separate applications into two classes. In the first class are applications that just do normal recording and playback, such as *realplayer*, *audioplay*, etc. In the second class are applications that are used to manage the master settings, such as *gnome-volume-control*, *mixerctl*, *sdtaudiocontrol*, and *mixer_applet2*.

The stance we've taken is that applications shall be assumed to be in the first class, and therefore are unable to change master settings (including volume or port configuration settings). Their settings are applied only to the local application. To facilitate this, each application gets a "virtual device" created for it (applications determined by process id), and the settings a tracked for that application as long as it has either a **/dev/audio** or **/dev/audioctl** file open. It does not matter what order these are opened, or whether more than one is open at a time.

The virtual device that the application sees is rather limited – it does not support changing the physical port configuration or monitor gain, and only has monophonic volume control for playback volume and record gain. (These controls will affect the volume of all channels for a given stream.) Put another way, the virtual balance is locked to AUDIO_MID_BALANCE. (The end user can still control the individual physical gains associated with each individual channel using an OSS compliant mixer application.

Applications which fall into the 2nd class are not supported by the legacy **audio**(7I) API in Boomer. Instead, these applications should use the new style OSS mixer API. A gstreamer plugin is provided so that Gnome mixer panel applications (e.g. *gnome-volume-control*) work as expected. Indeed they get the ability to access *additional* functionality, such as control for each of the audio levels in a 7.1 or 5.1 configuration.

The one application that we did not convert which falls into this 2nd class is *sdtaudiocontrol*. However, this application is ultimately scheduled to be EOF'd, as approved in LSARC case 2009/074.

As a consequence of the above changes, we are removing the ability to use the CLI *audioplay* and *audiorecord* features to administer port selection or master settings. These applications will be relegated into the 1st class of "normal" audio applications. New features will be supplied in the *mixerctl* application to provide alternative ways to administer port configuration and master settings, including volume and monitor gain, from the command line. The related command line switches will be accepted and silently ignored.

Because of the various limitations inherent in the **audio**(7I) API, we would like to declare it Obsolete as part of this project. New applications should use the OSS API.

14. Open Sound System (OSS) Personality

The Open Sound System (OSS) personality provides the API found on Linux and FreeBSD, and represents a substantial new interface to audio for applications on Solaris.

There are a number of issues with this API as well. There are actually several different sub-APIs used by OSS, and 4Front does not consistently support all of these in different drivers. Additionally, there are parts of the API that are intended for use by applications, and parts that are intended for "internal private use". The distinction between these is not always clear. There are also parts of the API that are completely irrelevant for this project, such as support for wave table synthesizers and MIDI devices.

While we will adhere to the primary parts of the API as documented at 4Front's web site, there are some minor deviations that we will make that should not impact "properly designed" applications. Furthermore, until we gain confidence in our implementation, and in the maturity of the API itself, we will be limiting our commitment for the OSS API to Uncommitted. This will allow application developers to use it, without guaranteeing it. We fully expect this API to become the API of choice over time, and therefore expect to increase commitment level to Committed once we are certain in the stability and the Sun Ray audio project has integrated.

The project will deviate in a few key areas from 4Front.

1. There is no support for MIDI, sequencers, timers, or wave table synthesizers.

2. There will not be separate device instances for "vmix" (software mixer) devices. The Boomer core doesn't require separate devices for software mixing, since the functionality of "vmix" is an integral part of the Boomer core. (We use clone opens to give each open file its own unique minor number, instead.)

3. Support for certain diagnostic use or optional ioctls may not be present.

4. The actual paths to device names may vary. Although applications that correctly use the /dev/sndstat or /dev/mixer common pseudo devices to discover what devices are present will function properly. (Conversely, applications that rely on /dev/pcm or /dev/dsp might not.)

5. Applications that use the 3.x version of the OSS mixer API will be assumed to be ordinary audio applications. As such, when they attempt to use this to adjust hardware gain levels, the changes will be made to a virtual view of the device for the process, and not applied to the physical hardware. This is similar to what we are doing for the legacy Sun applications.

6. The 4.x version of the mixer API is reserved for use by mixer panel type applications. A gstreamer plugin will be supplied to use this, which will enable the proper operation of the gnome desktop. See the section titled *Gstreamer and Gnome Desktop* below for sample screen shots made with gnome-volume-control using this plugin on a Sun Ultra 20.

7. Applications making use of the mixer API are required to issue ioctls against the actual mixer device node. The exception to this rule are the information queries supported by SNDCTL_SYSINFO, SNDCTL_AUDIOINFO, SNDCTL_CARDINFO, and SNDCTL_MIXERINFO. These ioctls (particularly SNDCTL_AUDIOINFO and SNDCTL_MIXERINFO) report the physical device node names (in /dev, e.g. /dev/sound/audio810:0mixer) which should be used to access the mixer feature. (This change is required to ensure that permissions associated with a mixer devices device node are not bypassed by using a different devices node, which might have reduced permissions.) Note that 4Front has agreed to document our recommendation that applications be coded this way in their official API documentation.

8. Unlike 4Front OSS, the names used for both controls and values in the 4.x mixer API will be predictable, so that they can be used by applications which need to support globalization, or need to determine the purpose of a control for other reasons.

9. The ability to change interrupt frequency or latency policies using ioctls is not be supported. (This is not guaranteed to be supported in OSS anyway, and we believe it would be a gross error for an application to depend on specific latencies.) These are intended to be hints only, at the API layer, and while we accept the hint, we ignore it. (Note that the latency policy is typically configurable for the device drivers using the **driver.conf**(4) mechanism to configure the interrupt rate. This mechanism is still available in Boomer, although we believe there will be little need for end-users to ever change these from the default. Most drivers are configured for 175Hz as the interrupt frequency.

10. Not all OSS ioctls are supported equally. In particular, there are a number of ioctls which are required for legacy applications, but which should probably not be used by new code, since there are often cleaner or better ways of achieving the same thing. While Boomer will provide support for many of these legacy support ioctls, the project does not intend to document the in full detail; they will be given an Obsolete Uncommitted stability level, and will only be noted as being present for legacy application support in documentation. Other, “preferred” API calls will be fully documented and should be given Uncommitted stability level. The precise details are supplied in the **dsp**(7I) and **mixer**(7I) man pages.

The OSS API is documented at 4Front's web site, although we will be developing our own documentation for Boomer. Draft man pages are included in the case materials.

15. Gstreamer and Gnome Desktop

The Gstreamer subsystem is used by many multimedia applications, and is the preferred multimedia framework for Gnome. Gstreamer is a plugin based subsystem, and supports a variety of different back-end plugins for different audio subsystems.

We will be supplying a Gstreamer plugin (working in conjunction with the JDS team) that provides full support for the Boomer framework using the OSS API. This will include Contracted support for the OSSv4 mixer API, so that applications like gnome-volume-control can work properly. We're also working to enhance gnome-volume-control to work well with Boomer (as well as other implementations of the OSSv4 API). Here are screen shots of gnome-volume-control, taken on a Sun Ultra 20 running Boomer code (note that normally all these controls are not displayed by default, but we've enabled them all here for illustrative purposes):

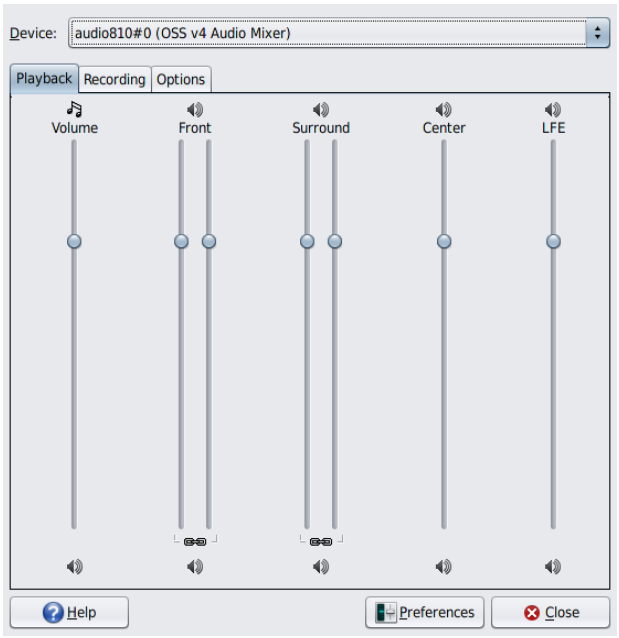

Illustration 2: Playback Pane

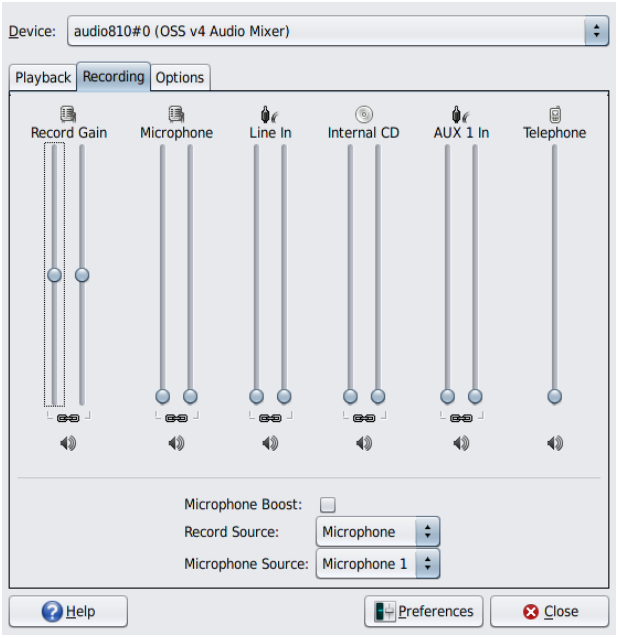

Illustration 3: Recording Pane

16. Audio DDI

As part of Boomer's goal to facilitate porting of device drivers from 4Front and other sources (including Linux and FreeBSD), as well as just generally making it easier to write audio device drivers in general, this project introduces a new kernel DDI for audio device drivers. Our limited experience so far is that the average device driver written using the Boomer DDI will be roughly half as many lines of code as a SADA driver, and significantly simpler to understand. We estimate that it will take an average kernel engineer about a week to port an average driver from SADA or 4Front OSS to Boomer, after spending some initial time to learn the new DDI. (More senior engineers can accomplish this in significantly less time. Some audio hardware might be significantly more complicated, resulting in a more expensive effort. Two such examples are the audiohd and the USB audio subsystems.)

It should be noted that at this point in the project, and for initial delivery, the Audio DDI is intended

to remain Consolidation Private to the ON consolidation. This will allow the team to get more experience with the DDI, before we open it up to third party device driver authors. We believe that in the near future it will be possible to increase the commitment level to Committed, once we are more confident in our interfaces. This will probably follow after any work for Sun Ray audio.

The device driver DDI is fundamentally quite different from the legacy SADA DDI. Device drivers need include only one audio-specific header, <sys/audio/audio_driver.h>, which brings in all of the audio functionality required. (Solaris DDI headers, such as <sys/ddi.h> are still required, however.)

Device drivers exchange audio data with the framework using a shared circular DMA buffer. The contents of this buffer are similar to the buffer used between audio personalities and the core framework, and likewise these buffers contain head and tail indexes. However, in this case one of the indexes is maintained by the device driver. DMA cache synchronization of the buffer is performed by the framework, using an entry point supplied by the device driver.

Whether a device has a single DMA engine or a hundred of them, the management is the same. Devices register DMA engines (which can also be thought of as “voices”, or input streams for a hardware mixer) along with the capabilities of the engine (such as playback or record) and the supported audio formats for the engine. (At this time, the supported formats are 16, 24, and 32 bit signed PCM in either endianness. In the future, AC3 and other formats might be supported. This list was arrived at by looking at what current devices support, and eliminating any formats that were redundant or not useful – such as 8-bit ULAW.)

The framework manages the allocation (assignment) of engines to open audio clients. (At the moment the load balancing algorithm is rather primitive, and devices with more than a few DMA engines will be under utilized. This is a limitation in the internal policy engine, and should be fixed.)

Once an engine is needed to perform some functionality, it is “opened”. At this time the engine will inform the framework the number of hardware channels in use, the sample rate in use, and the “fragment” sizes in use (the number of frames between interrupts) as well as critical information about the DMA buffer itself (namely its kernel virtual address and size). There are provisions in the DDI for the framework to request overrides for some of these settings, but they are not currently used.

When the last audio application using a device closes the device, the engine itself is closed using the close entry point for the engine ops vector.

Note that engines each have their own optional engine private data, as well as their own entry points. This allows a device driver that has multiple engines that behave differently (e.g. an AC3 engine which requires special programming, or different programming between record and playback engines) to easily be supported.

Audio drivers also support physical controls, which are “out-of-band” operations used to affect device operation. Typically these are port selection and gain settings. They are discussed more fully in in the section titled Audio Control Framework.

Note that the older SADA DDI is not supported, in any form, in Boomer. The Boomer project team

has converted all of the existing device drivers to use the new DDI instead.

17. Audio Control Framework

The audio control framework provides a mechanism for audio drivers to export control objects and operations for use by mixer panel applications. Some other operating systems call their equivalent interfaces “mixer APIs” or similar. These objects are usually used to control the behavior of hardware mixing devices on audio hardware. (For example, directing which output ports should be active, which input ports should be active, and volume or attenuation settings for individual channels or streams.)

Boomer defines a broad list of “well-known” controls, which are identified by constant string values. These controls are listed in <sys/audio/audio_common.h>. Well known controls will have a well understood purpose, such that their behavior is predictable from one driver to the next. These controls also will have specific “types” associated with them, which shall not vary. (For example, the AUDIO_CONTROL_INPUTS control shall always be an enumeration of ports (AUDIO_CTRL_TYPE_PORTS), while the AUDIO_CONTROL_LINEOUT will always be a stereo gain setting of AUDIO_CTRL_TYPE_STEREO.)

In addition, Boomer allows for device drivers to supply their own “device specific” controls (tunable parameters). The driver will uniquely identify these controls using a name. For example, “ac97_stereo_simulate” is an AC'97 specific control that determines whether stereo output should be simulated when only a single monaural output is available. Generally it will not be possible for applications to know in advance what device specific controls might be present, and therefore we do not expect these controls to be readily accessible in GUI applications.

This design pattern should be familiar to anyone who is familiar with the Brussels framework used for NIC driver parameters. Indeed, it was in part inspired by Brussels.

18. Common AC'97 Module

It turns out that many of the devices in common use are based around the popular Intel AC'97 specification. This specification defines the interface between audio codecs, and host controllers. However, it says nothing about the host controller interface itself, so there are a large number of variations requiring a large number of drivers. (This is not unlike the world of Ethernet drivers, where the 802.3u standard spells out the interface between host controllers and transceivers, but says nothing about how the host controller itself is accessed.)

As a result, it is desirable to provide as much of the common AC'97 functionality (which can be thought of as “mixer controls”) in a common module, so that device drivers need supply only generic access routines to read and write codec registers, and can leave the bulk of the work in implementing the control API to common shared code.

Device drivers that desire to do this use the common misc/ac97 module, and access the declarations through the <sys/audio/ac97.h> header file. They can still supplement AC'97 with their own controls, or override the default definitions of controls provided by AC'97.

This design also allows for a greater range of devices to be supported, since probably the most common axis for variation in a family of audio devices is the selection of the codecs used on the device. By putting the logic in a common core, we gain the ability to autodetect different codec behaviors, as well as provide for codec-specific overrides.

19. High Definition Audio

Perhaps the most common device on new motherboards these days is the Intel Azalia (High Definition Audio). Intel released a new specification for audio devices, to replace AC'97. Unlike AC'97, this is a complete specification for the codecs and the controller, including a PCI class definition. Thus, there are controllers from other parties than Intel, but which also conform to this new specification.

Going forward, for PCI (and PCIe) based motherboards at least, we expect this part to be the audio device of choice.

The specification is very rich, and allows for a mind-boggling set of possible features, including up to 16 channels of 192 kHz 32-bit audio. (This is far in excess of any current needs, or the ability of most human ears to discern.) It also provides for very flexible codec configuration, and up to 15 separate DMA channels for playback and 15 separate DMA channels for record.

We believe therefore, that this device represents the “high bar” for capabilities of audio subsystems for the foreseeable future. (At least the next decade. And quite possibly beyond. Although there have been some demonstrations of a system with 24 separate output channels for experimental Ultra High Definition Video.)

For Boomer, we will be supplying a “converted” High Definition audio (audiohd) driver, that is updated to take advantage of Boomers additional features. This device driver shall support *at least* 24-bit audio, with up to 8 channels of output, and up to 96 kHz sampling rate. It *may* support additional features as well. (One limitation is that actual implementations of the specification do not implement all of the *possible* features of the specification, limiting our ability to properly verify our implementation.

In addition, work on this driver by the Boomer team has already resulted in a SADA driver that supports a broader set of possible codec configurations (and hence a broader set of motherboards) than was previously available. This work has been delivered in Solaris Nevada already, but further improvements here are likely.

20. USB Audio

Boomer provides support for USB audio, but this subsystem is one of the areas that has been changed significantly from the legacy stack.

In the early stages of implementation, Boomer had a separate shim layer which allowed legacy SADA drivers to be used with Boomer. After all of the other drivers were converted, USB still

remained. Rather than retain a separate module, the shim layer (formerly a greatly changed **audiosup**(7D) module) was merged into the USB audio drivers directly. The **audiosup** module itself (as well as other related modules **amsrc1**, **amsrc2**, and **mixer**) has been removed in the Boomer implementation.

The other significant change is that since Boomer drivers are no longer STREAMs devices, it was no longer appropriate to use DACF to configure the various portions of the USB audio framework. As a result, the USB audio DACF module has been removed. The configuration of the multiple device nodes is now accomplished using the layered driver interfaces (LDI). The related functionality is part of the **usb_ac**(7D) module now.

Eventually, we would like to see the various separate USB audio modules (**usb_as**, **usb_ac**, and **usb_ah**) combined into a single unified driver, and the code restructured to act more like a natural native Boomer device. However, there was not time to accomplish this in Boomer Phase I.

21. Additional Drivers

In addition to the existing SADA drivers, this project will expand upon the set of supported devices for audio in Solaris. Already, the Boomer team has implemented a driver for the older Creative AudioPCI (Ensoniq ES1370) device. (The effort to develop this from the existing audioens driver took only about a day.) This driver should help certain virtual environments such as VMware guests.

We've also expanded the set of devices supported by our existing devices. For example, the audio810 driver now supports a variety of additional chips, such as the SiS 7012 and additional parts from Intel and nVidia.

Other common commodity hardware is definitely on our list of things to do. The exact set of devices will be supporting has not been fixed yet, but is likely to include the latest offerings from Creative such as Audigy, SoundBlaster Live!, and SB-XFi products. These additional drivers are not being delivered as part of the Boomer Phase I project, but will be handled via follow-on RFEs after Boomer integrates. These drivers can be imported from external sources, such as 4Front or FreeBSD, and we expect the effort to port them to Boomer be fairly small. (Actual engineering effort can be expected to be a small number days for each driver, for an engineer familiar with audio devices in general and Boomer in particular.)

22. Linux Branded Zones

As a result of the inclusion of the OSS API, the Boomer project team feels that it may be possible to remove the **lx_audio** compatibility driver for Linux branded zones. In order to accomplish this, Boomer will have to retain identical numbering schemes for certain constants used in OSS. It may also impose additional constraints on device naming and file paths.

Furthermore, lx_audio provides an **mmap**(2) capability which many Linux applications require. In order to remove lx_audio, the same feature will be required from Boomer. More on mmap is described below. As a result, removal of lx_audio is *not* planned for the Phase I delivery of Boomer.

While there are likely additional improvements that could be made here even without mmap support, the project team is electing to defer those improvements until after Boomer Phase I. However, Linux Branded Zones should receive the same quality of support in Boomer Phase I that they receive in Solaris today.

23. Future Directions: Sun Ray

One of the projects for the Boomer project team, though not necessarily phase I of the project, is to resolve some of the long-standing deficiencies in the Sun Ray audio support. Currently, Sun Ray DTUs behave as a 1st generation exclusive use audio device by default. While there are ways to provide for simultaneous audio application use, these are rather clunky, and difficult to use.

A significant challenge for Sun Ray audio will be determining which DTU an application's data should go to, and the fact that there can be a great many of audio sessions, each needing a different virtual device. (There can be so many of these, that normal minor numbering schemes may not be scalable enough.)

Additionally, applications which expect to be able to use the “default” audio device for a system may incorrectly wind up sending audio data to the output device associated with the server (perhaps in the machine room) rather than the DTU.

We anticipate a future project to address these considerations by introducing a “virtual” audio device that finds the appropriate physical device based on some parameters including the Sun Ray session id. This may require enhancements to the Solaris DDI to facilitate identification of Sun Ray sessions. (Such enhancements could also be used for things like VNC or other remote desktop technologies.)

Notably, the project is *not* aiming to solve these problems all in this phase, but is merely designing things in such a way that we believe it will be possible to extend the framework to meet these future needs.

24. Future Directions: mmap

Linux and FreeBSD audio frameworks support the use of mmap to allow applications to directly map the audio buffer, and use ioctls to deal with head/tail synchronization. This allows for a much lower overhead for audio data, and allows for mixing to occur in user space.

In Linux and 4Front's code, the support for mmap is not universal (not all devices are mappable), and incurs some limitations. Specifically, the application must use a data format and rate that is supported by the device, and once the device mapped no other applications can use the device.

Ultimately, we hope to provide support for mmap, but rather than mapping memory shared with the device driver itself (or the DMA region) directly, the buffer used for audio client personalities will be mapped instead. This will eliminate the restrictions on exclusive use and data format, while allowing applications to avoid an expensive data copy.

Note that this is *not* planned for Boomer Phase I. A follow on project to provide this project will be done, and Boomer Phase I has been designed internally to facilitate this work.

Notably, mmap support may be required to eliminate lx_audio.

25. Future Directions: Dolby Digital (AC3) Pass Through

In order to support multimedia applications involving certain video formats (DVD, DVR), it may be desirable to support feeding Dolby Digital (AC3) encoded audio directly from the application to the SPDIF digital output of an audio device, where it can be decoded by an external Dolby Digital receiver.

We have designed the framework as much as possible to support this in a future update, but are not planning on delivering this functionality in Phase I.

Notably, not all hardware is physically capable of performing this AC3 pass through. For audio playback of digital formats which support AC3, generally alternative formats (linear PCM analog data) are possible as well. Such applications may have to settle for either analog or reduced quality PCM output in the mean time.

Note that Dolby Digital is a patent encumbered format, so software support for decoding this is unlikely to exist in Solaris in any immediate future, although nothing prevents an application from doing the decode in user-space and providing PCM encoded data to the Boomer core.

26. Future Directions: Virtual Audio & Hotplug Automagic

In order to facilitate the Sun Ray project, the Boomer project will need to create a device which has the property of being able to detect whether the process is associated with a Sun Ray or not. The project team believes it would be useful to extend the project to include support for a “virtual audio device”, that reroutes audio to real devices in response to dynamic hot plug events, based on user configuration. (For example, when a USB headset is plugged, it might be desirable to reroute the main system audio there.) This project is outside of the scope of Boomer Phase I.