

The Dreamcast Audio 64 API

Table of Contents

Preface	AUD-ix
1. Dreamcast Audio64 Overview	AUD-1
Introduction	AUD-1
AM Layer	AUD-2
amInit	AUD-2
amBank	AUD-3
amFile	AUD-3
amHeap	AUD-3
amSound	AUD-4
amStream	AUD-4
amMidi	AUD-5
The AC Layer	AUD-5
Tools Overview	AUD-6
MkScript	AUD-6
MkBank	AUD-7
MkStream	AUD-7
File Formats	AUD-8

2. The AICA Control Layer APIAUD-9

acSystemRequestArmInterrupt.....	Causes the driver to raise an ARM external interrupt.	AUD-9
acDigiPlay	Starts a buffer playing.	AUD-10
acDigiPlayWithLoopParameters.....	Starts a buffer playing Set loop points.	AUD-11
acDigiPlayWithParameters.....	Starts a buffer playing with all common parameters.	AUD-12
acDigiMultiSetMask	Sets the bit masks for acDigiMultiPlay()	AUD-14
acDigiMultiPlay.....	Sets the bit masks for acDigiMultiPlay()	AUD-15
acDigiMultiStop	Sets the bit masks for acDigiMultiPlay()	AUD-16
acDigiOpen	Open a DA Streaming Port for playback.	AUD-17
acDigiSetSampleRate.....	Set the playback rate (sample rate) of audio stream.	AUD-18
acDigiClose	Closes port previously opened.	AUD-19
acDigiSetCurrentPitch.....	Changes the playback rate of a running channel.	AUD-20
acDigiSetVolume.....	Adjusts volume of a voice channel.	AUD-21
acDigiSetPan	Adjusts the pan placement of a voice channel.	AUD-22
acDigiRequestEvent.....	Used to generate an interrupt when a certain buffer position is reached.	AUD-23
acDigiStop	Stops a voice channel playing.	AUD-24
acMidiOpen.....	Open a MIDI Port buffer for SMF format 0 playback.	AUD-25
acMidiSetTonebank	Assign a MIDI Program Bank (tonebank) to an active bank slot.	AUD-26
acMidiClose.....	Close a MIDI port.	AUD-27
acMidiPlay.....	Starts playback on opened MIDI port.	AUD-28
acMidiStop	Stops standard MIDI file playback on port.	AUD-29
acMidiRequestEvent.....	Generates interrupt to host upon MIDI port reaching specified address.	AUD-30
acMidiPause.....	Pauses an active MIDI port.	AUD-31
acMidiResume	Resumes playback on active MIDI port.	AUD-32
acMidiSetVolume	Sets scaled volume setting for MIDI port.	AUD-33
acMidiReset.....	Resets MIDI controllers on port to default values.	AUD-34
acMidiSetTempo.....	Set playback tempo of MIDI port.	AUD-35
acMidiSendMessage	Sends raw MIDI messages to ports.	AUD-36
acCdSetVolume	Sets Left & Right Channels for Redbook Volume Control (dependent on channel pan).	AUD-37
acCdSetPan.....	Sets Left & Right Channel pan position.	AUD-38
acCdInit	Resets CDDA channels to hard pan positions and maximum volume.	AUD-39
acDspSetQSoundAngle	Sets Q-Sound position.	AUD-40
acDspInstallProgram	Registers a dsp program bank with the driver.	AUD-41
acDspInstallOutputMixer	Registers an output mixer patch with the driver.	AUD-42
acDspSetMixerChannel.....	Sets DSP mixer level and channel for that port.	AUD-43
acErrorGetLast.....	Gets a pointer to the error structure.	AUD-44
acErrorExists	Checks to see if an error condition exists.	AUD-45
acErrorClear	Clears the AC error structure.	AUD-46
acIntInstallOsChainDeleteManager	Installs pointer to interrupt chain delete routine.	AUD-47
acIntInstallOsChainAddManager.....	Installs proc pointer to interrupt chain add routine.	AUD-48
acIntInstallCallbackHandler.....	Installs a callback handler into the ARM interrupt handler.	AUD-49
acIntInstallArmInterruptHandler.....	Installs an ARM interrupt handler.	AUD-50
acIntSetAicaChainId	Sets the AICA interrupt chain ID.	AUD-51
acIntShutdown	Shuts down the ac interrupt system.	AUD-52
acIntInit.....	Initializes the ac interrupt system.	AUD-53
acSystemShutdown.....	Shuts down the AC layer.	AUD-54
acSystemGetIntArrayStartOffset	Gets the interrupt array write cursor offset.	AUD-55
acSystemGetIntArrayStartPtr.....	Gets a pointer to the start of the drivers interrupt message array.	AUD-56
acSystemGetBaseOfSoundMemory.....	Gets the starting address for sound memory.	AUD-57
acSystemGetIntArray	Gets the address of the SH4 side interrupt message array.	AUD-58
acSystemGetIntArrayLength.....	Gets the length of the drivers interrupt message array.	AUD-59
acSystemCheckDriverRevision	Tests the driver version against the supplied version.	AUD-60
acSystemGetDriverRevision.....	Tests the driver version against the supplied version.	AUD-61
acSystemWaitUntilG2FifoIsEmpty	Waits until the G2 FIFO is clear.	AUD-62
acSystemDelay	Use to delay for short periods of time.	AUD-63
acSystemEnableArmInterrupts.....	Use to enable the ARM interrupt.	AUD-64

acSystemDisableArmInterrupts	Use to disable the ARM interrupt.	AUD-65
acSystemInit	Makes the ac system ready to use.	AUD-66
acGetSystemFlag	True if the system has been initialized.	AUD-67
acSystemGetFirstFreeSoundMemory	Gets the address of first free sound memory.	AUD-68
acSystemGetCommandFlag	Gets address of driver command flag register.	AUD-69
acSystemResetArmInterrupt	Resets the ARM interrupt flag.	AUD-70
acSystemInstallDriver	Installs the sound driver.	AUD-71

3. The AICA Manager API AUD-73

amBankFetchMidiUspqn	Fetches uspn from a MIDI type asset.	AUD-73
amBankFetchMidiLoop	Fetches the loop flag from a MIDI type asset.	AUD-74
amBankFetchMidiPpqn	Fetches ppqn from a MIDI type katbank asset.	AUD-75
amBankFetchMidiVolume	Fetches master volume from a MIDI type katbank asset.	AUD-76
amBankFetchMidiGmModeFlag	Fetches GM mode flag from a MIDI type katbank asset.	AUD-77
amBankLoad	Loads a katbank asset from disk into sound memory.	AUD-78
amBankFetchAssetParameters	Fetches parameters from any katbank asset.	AUD-79
amBankFetchWaveLoopFlag	Fetches the loop flag from a katbank asset.	AUD-80
amBankFetchWaveRandomPitch	Fetches random pitch amount from a katbank asset.	AUD-81
amBankFetchWaveSampleRate	Fetches the sample rate from a katbank WAVE asset.	AUD-82
amBankFetchWaveBitDepth	Fetches the bit depth of a WAVE type asset in a katbank.	AUD-83
amBankFetchUnknownParameters	Fetches one of the 7 user parameters from a katbank "unknown" type asset.	AUD-84
amBankFetchAsset	Fetches an asset from a katbank.	AUD-85
amBankGetAssetSize	Gets the size of an asset from a katbank.	AUD-86
amBankGetNumberOfAssets	Gets the number of assets in a katbank.	AUD-87
amBankGetHeaderSize	Gets the size of the header portion of a katbank.	AUD-88
amDmaMemCpy	Performs DMA copys to sound memory.	AUD-89
amDspFetchProgramBank	Fetches and installs a DSP program bank from a KatBank asset.	AUD-90
amDspFetchOutputBank	Fetches and installs a DSP output bank from a KatBank asset.	AUD-91
amErrorGetLast	Gets a pointer to the error structure.	AUD-92
amErrorExists	Checks to see if an error condition exists.	AUD-93
amErrorClear	Clears the AM error structure.	AUD-94
amHeapShutdown	Shuts down the AM heap management system.	AUD-95
amHeapGetInfo	Gets info necessary to start an audio heap.	AUD-96
amHeapGetFree	Gets the amount of free memory.	AUD-97
amHeapAlloc	Allocates aligned memory from the audio heap.	AUD-98
amHeapGetMaxPurgable	Gets amount of memory available from a full purge.	AUD-99
amHeapPurge	Purges memory marked as purgable.	AUD-100
amHeapFree	Frees purgable memory allocated using amHeapAlloc()	AUD-101
amHeapInit	Initializes the audio heap.	AUD-102
amHeapCheck	Checks the MCB fingerprints for overwrites.	AUD-103
amInitSelectDriver	Selects driver to be installed by amInit()	AUD-104
amShutdown	Shuts down the AM audio subsystem.	AUD-105
amInit	Starts up the AM audio subsystem.	AUD-106
amFileRewind	Seeks to the start of a file.	AUD-107
amFileLoad	Loads specified file into the buffer.	AUD-108
amFileRead	Reads from a file that is already open.	AUD-109
amFileOpen	Opens a file for reading.	AUD-110
amFileClose	Closes a file.	AUD-111
amFileGetSize	Gets the size of a file.	AUD-112
amFileInstallAlternateIoManager	Installs a custom Io proc.	AUD-113
amStreamIsr0 - 4	Interrupt Service Routine for the amStream subsystem.	AUD-114
amMemSh4Alloc	Sh4 memory allocation shell.	AUD-115
amMemSh4Free	Sh4 memory free shell.	AUD-116
amMemInit	Initializes the Sh4 memory shell system.	AUD-117
amMemInstallAlternateMemoryManager	Allows redirection of sh4 memory requests.	AUD-118
amMidiSetTempo	Sets the tempo of a MIDI sequence.	AUD-119

amMidiSetLoopFlag	Sets the loop flag on a MIDI sequence.	AUD-120
amMidiFetchToneBank	Installs an MTB asset from a bank file aggregate.	AUD-121
amMidiLoadToneBank	Loads a Sega tone bank asset	AUD-122
amMidiInstallCallback	Sets the callback proc for a sequence.	AUD-123
amMidiAllocateSequencePort	Allocates a MIDI port for the sequence.	AUD-124
amMidiFetchSequence	Fetches a sequence asset from a katBank.	AUD-125
amMidiPlay	Plays a MIDI sequence.	AUD-126
amMidiPlayRaw	Plays a MIDI sequence given the basic parameters.	AUD-127
amMidiStop	Stops a currently playing MIDI sequence.	AUD-128
amMidiSetVolume	Sets the master volume of a MIDI sequence.	AUD-129
amMidiPause	Pauses a currently playing MIDI sequence.	AUD-130
amMidiResume	Resumes playback of a paused MIDI sequence.	AUD-131
amMidiTransferToneBank	Transfers a Sega tone bank to sound memory and sets it as the current bank.	AUD-132
amMidiSetChannelProgram	Sets the current bank slot.	AUD-133
amMidiNoteOn	Plays a MIDI triggered sound effect.	AUD-134
amMidiNoteOff	Stops a MIDI triggered sound effect.	AUD-135
amMidiSetChannelVolume	Sets volume of a midi sound.	AUD-136
amMidiSetChannelPan	Sets the pan of a MIDI sound.	AUD-137
amSoundSetQSoundChannels	Used to identify which channels in an output bank are Q-Sound channels.	AUD-138
amSoundSetEffectsBuss	Sets the effects buss send and source mix for a sound object.	AUD-139
amSoundFetchSample	Fetches a sound and its parameters from a Katana format bank.	AUD-140
amSoundIsLooping	Tells if the given sound has a loop.	AUD-141
amSoundAllocateVoiceChannel	Allocates a hardware voice channel.	AUD-142
amSoundGetSampleRate	Gets the real world sample rate.	AUD-143
amSoundGetVolume	Gets the current volume setting.	AUD-144
amSoundGetPan	Gets the current pan position.	AUD-145
amSoundGetVoiceChannel	Gets the current voice channel assignment.	AUD-146
amSoundGetCallback	Gets the address of the user callback.	AUD-147
amSoundSetCurrentPlaybackRate	Sets the playback rate.	AUD-148
amSoundSetVolume	Sets a sounds volume.	AUD-149
amSoundSetPan	Sets a sounds pan.	AUD-150
amSoundSetCallback	Sets the user callback.	AUD-151
amSoundIsPlaying	Tells if a sound is currently playing.	AUD-152
amSoundStop	Stops a currently playing sound.	AUD-153
amSoundPlay	Plays a sound.	AUD-154
amSoundPlayRaw	Plays a sound given all of the required parameters.	AUD-155
amStreamSetMix	Sets volume and pan for all tracks in a stream.	AUD-157
amStreamInitFile	Initializes a stream object to play a file.	AUD-158
amStreamInitBuffer	Initializes a stream object to play a mono stream from a buffer.	AUD-159
amStreamInstallUserCallback	Installs a user callback for a stream.	AUD-160
amStreamRewind	Rewinds an open stream to its start.	AUD-161
amStreamGetMemoryRequirement	Gets memory sizes necessary to play the stream.	AUD-162
amStreamSetBufferSizes	Sets the sizes for the play and transfer buffers.	AUD-163
amStreamSetBuffers	Sets buffer memory pointers in a stream.	AUD-164
amStreamSetIsr	Sets the streams data transfer ISR.	AUD-165
amStreamAllocateVoiceChannels	Allocates voice channels.	AUD-166
amStreamPrimeBuffers	Primes the play buffer.	AUD-167
amStreamGetTrackLengthInFrames	Gets the length of a stream in frames.	AUD-168
amStreamGetNibblesPerFrame	Gets the number of nibbles in a frame.	AUD-169
amStreamGetSampleRate	Gets the real world sample rate of a stream.	AUD-170
amStreamGetMsPerIrq	Gets the number of milliseconds per callback.	AUD-171
amStreamSetVolume	Sets the volume on a stream.	AUD-172
amStreamSetPan	Sets the pan on a mono stream.	AUD-173
amStreamStop	Stops a currently playing stream.	AUD-174
amStreamPlaying	Monitors if a stream is currently playing.	AUD-175
amStreamGetVolume	Gets the streams current volume	AUD-176
amStreamGetPan	Gets the streams current pan	AUD-177

amStreamGetIsrCount	Gets the Interrupt Service Routine count.	AUD-178
amStreamClose	Closes a stream object.	AUD-179
amStreamStart	Starts a stream object playing.	AUD-180
amStreamIsStereo	Tells if a stream is stereo.	AUD-181
amStreamIsMono	Tells if a stream is mono.	AUD-182
amStreamServer	Serves data to a currently playing stream.	AUD-183
amStreamOpen	Opens a stream object.	AUD-184
amStreamSetTransferMethod	Selects DMA or memcpy as the data transfer method.	AUD-185
amStreamIoInstallAlternateIoManager	Installs a custom Io proc.	AUD-186
amUtilGetAicaVolume	Converts midi volume units to AICA units	AUD-187
amUtilAlignNumber	Performs numerical boundry alignment.	AUD-188
amUtilGetLengthInFrames	Gets the length of a stream in frames.	AUD-189
amUtilGetNibblesPerFrame	Gets the number of nibbles in a frame.	AUD-190
amUtilGetSampleRate	Gets the real world sample rate of a stream.	AUD-191
amUtilGetLengthInMs	Gets the length of a stream in milliseconds.	AUD-192
amUtilGetMsPerIrq	Gets the number of milliseconds per callback.	AUD-193
amUtilGetAicaSampleType	Extrapolates sample bit depth to AICA sample type.	AUD-194
amUtilGetAicaSampleRate	Makes a real world sample rate into an AICA sample rate.	AUD-195
amUtilGetMiddleOfBufferInFrames	Calculates the middle of the buffer in frames.	AUD-196
amUtilGetEndOfBufferInFrames	Calculates the end of the buffer in frames.	AUD-197
amVoiceInit	Initializes the voice pool.	AUD-198
amVoiceAllocate	Allocates a voice channel.	AUD-199

Preface

The Sega of America audio solutions (Audio64 and MidiDa) offer a different approach to using the audio hardware resources of the Dreamcast console. The main differences fall into the following areas:

- Sound memory usage
- Interrupt notification
- Control of AICA hardware
- Sound asset creation
- Layered API approach

The basic approach to memory usage is to allow a completely dynamic usage of the sound memory resource. This means that you can write any sound assets you require to any available area of the sound RAM and play them. You can also dynamically allocate this memory and dynamically replace it. There are no static “memory maps” to restrict your usage of this memory. Along with the freedom to use this memory as you see fit comes the possibility of unpredictable results if you misuse this resource.

Interrupt notification is implemented for sound (PCM or ADPCM) playback and MIDI playback, and sound driver command processing. This means that you can know immediately when sound resources become available and reuse the resource, thereby increasing the bandwidth of the audio system.

Control of the AICA hardware is intended to be as exposed as possible, implementation time permitting. This means that you can directly allocate all 64 audio channels (in the case of the Audio64 sound driver) and control each channels pan, volume control and sample rate playback directly. You can also gang multiple channels for phase-locked audio playback (up to 64 channels), and can dynamically control DSP effects on each.

All audio assets can be bundled together into banks of multiple or individual type data. The collection into banks is as easy as copying standard assets (.wav or .fpx or .fob or .mid or .mpb files, etc.) into a directory and running a DOS-level utility. Streams can be built in a like manner, and multiple track streams are accommodated by the tools.

Providing a layered API means developers can work at the level they are most comfortable with, and that multiple approaches are accommodated. We believe that full access to the hardware will yield the best results, that developers can adapt their current game development environments more easily and be immediately more productive. The AC (AICA Control) layer allows direct control of the AICA sound driver and hardware, while the AM (AICA Manager) layer provides higher level control with an architecture that provides dynamic resource allocation and stream playback control, among other things.

The Dreamcast Audio 64 API

The AICA hardware is an audio subsystem that supports 2 MB of sound RAM, has 64 audio playback channels which can play 16, 8 or 4 bit data at sample rates from > 11 megaHertz (theoretically) down to 172 hz (approximately). It also contains a built-in DSP unit which can provide high quality reverb and Qsound and a multitude of DSP effects which can be flexibly configured. It has a built-in digital 16 channel mixer, and can route Redbook audio through the DSP. Each voice channel also has a hardware 5 stage resonant Low Pass Filter, wave selectable amplitude and pitch LFOs, and an amplitude ADSR envelope. The sound subsystem has an embedded RISC ARM7 CPU running at 25 MHZ, and the AICA sound registers are controlled by one of the sound drivers (Audio64 or MidiDa) written in ARM7 assembler. SH4 CPU usage is minimized by utilizing ARM7 control.

1. Dreamcast Audio64 Overview

1.1 Introduction

The Audio64/MidiDa API is designed to be a logically named, easy to use, fault tolerant, componentized audio system.

It is broken down into layers. Use of the AC layer will not involve any AM layer components.

The examples all include some boilerplate code that will allow the inclusion of sound into a game in under an hour.

There are two drivers available, Audio64, and MidiDa. One is 64 channels of digital audio while the other is 16 channels of digital audio 16 MIDI ports with 48 note MIDI poliphony.

Note: The MidiDa driver is not an official part of R8 but can be obtained from Sega DTS.

The naming convention is [layer][subsystem][function] such as amSoundPlay or acDigiOpen. There are NO abbreviations used anywhere in the library and the Capping scheme is strictly adhered to, this should put to rest the problem of strange and absurd API naming conventions.

Fault tolerance is provided through extensive error trapping and reporting through an error messaging system that gives both an error number as well as a text message that gives the name of the failing procedure and the likely cause of the failure. Each layer has its own error reporting system.

The overall system obtains its necessary OS services through procedural vectors that allow it to be totally OS independent. This allows developers the ability to customize and control the file IO and interrupt handling capabilities to fit their individual needs. Application programmers will appreciate the componentized nature of the system allowing them to replace sub-systems with their own custom code.

Neither the AM nor the AC layer allocates or frees SH4 side memory; there is a SH4 memory allocation shell in place but that is not used.

The mid-level layer, AM, is implemented using only AC layer calls, at no point does it “go to the metal” and use a functionality that is not exposed in the AC layer. That insures that it is possible for an audio oriented developer to match or exceed the AM layers capabilities using the AC layer calls.

The AM system is designed to use industry standard assets such as .wav files and Type 0 MIDI (.mid) files. This will allow audio artists to work in the tools that they are familiar with rather than having to use a proprietary tool to make what should be standard assets. It also allows the full range of commercial DSP plugins to be used with a minimum of difficulty in the audio art production process. Certain assets that are Sega specific i.e. DSP Program banks, DSP output banks and MIDI tone banks, can be manufactured using the Sega Macintosh tools set.

An asset aggregation system is supplied that knows about the types of assets that will be used for sound and allows the audio programmer access to some frequently tweaked parameters in pre-built script files. The script file is completely written by a script file builder so the learning curve for building assets is very minimal. Basically throw the assets in a directory, run the script builder then the bank builder. This system also allows for the inclusion of parameterized developer specific assets.

The AM layer streamer will stream files that have multiple tracks to allow for a degree of interactivity to be implemented in a streaming audio presentation via track muting and scene mixing. If you look in the samples you will find a sample that streams, in phase coherent sync, 8 tracks of full bandwidth (44.1/16 bit) audio.

1.2 AM Layer

The AM layer controls system resources to a degree that the AC layer does not, it doesn't own these resources. They still must be given to it by the application.

All volume/pan parameter ranges for the AM layer are the same as MIDI ranges 0-127.

If sound memory is needed it can be partitioned using the amHeap system or by the developers proprietary memory/heap management system. As with memory management for any audio system this memory must not be relocatable. If the memory were to be moved while the driver is playing it ugly artifacts or failures will be produced.

AM has a number of discreet sub-systems that you may use as desired.

Central to the operation of the AM system is its main shared code resource the voice pool manager. All of the AM systems that play sound use the amVoice system. This manager controls allocation of voices and MIDI ports, tracks their activity and closes their voice channels when the play is completed.

Because of this interrupt driven action it is of the utmost importance that sound objects, AM_SOUND, AM_STREAM and AM_SEQUENCE, persist until the sound is finished playing.

The voice pool retains a pointer to the sound object and will issue AC commands and adjust the internal state of the object on completion of the play cycle. If this object is created on the local stack and control goes out of the scope of that function, then, when the end of the sound is reached the voice manager will write and read that address... which is part of the stack of another function now. Upon further processing, the system will fail.

Needless to say this is unfortunate when it happens so we advise that sound objects be global in scope as their addresses are retained and written to at the end of the sound, stream or sequence.

In the registration of callbacks for the different sound entities the callback proc must be registered after the voices are allocated, the function can not be registered if the voice pool does not know what voice owns the pointer.

The callback handler in MyInt.c is the default AM callback handler, if you want to post ARM interrupt messages with the driver and field the response, or add functionality to voice channel/port messages, while still using the AM system, this allows you to hack into the default callback handler and add your code to it.

1.2.1 amlnit

Initialization of the AM layer is simple, install the OS service procs, select the driver and call amInit.

See the examples and the boilerplate modules for examples of the OS service proc wrappers and the initialization sequence.

1.2.2 amBank

This API is an asset picker for the DOS command line asset aggregation tools that are supplied with the development kit.

It allows you to “fetch” an asset and its parameters from a kat bank in an orderly fashion. `amSound`, `amMidi` and `amDsp` all have “fetch” functions. In the future we will supply a way to use your own aggregation tool with these API’s.

The kat bank consists of a `DWORD` that contains the number of header records/assets in the bank, then `n` header records, then the assets in the same order as the header records. See `ambnkhdr.h` for more detail on the structure of the header records.

1.2.3 amFile

This is a redirectable file system that uses a monolithic IO proc to perform all disk based functions. This is used by all AM subsystems that perform file IO operations. The `amStream` sub-system uses a similar IO proc to read in stream files. This allows you to customize the IO proc’s or change the file system being used.

Because the file system is OS dependent it is excluded from the library code and this indirect interface is supplied.

For this reason the system is initialized prior to calling `amInit` by passing proc pointers to `amFileInstallAlternateIoManager` and `amStreamIoInstallAlternateIoManager`. If the file system is not initialized `amInit` will fail.

This file system is also used in the examples to obtain assets from disk.

Boilerplate of the monolithic IO procs may be found in `MyFile.c`.

1.2.4 amHeap

This is a standalone memory manager system that can be used to partition sound memory. It has some knowledge of the special requirements of Dreamcast sound memory i.e. `DWORD` aligned writes only. Despite the fact that the memory manager will not allocate odd sized blocks of memory it is still possible to trash memory by writing an odd `((size % 4) == true)` sized asset into the block.

Each block is aligned to the starting address alignment given with the `amHeapAlloc` call, each block ends with the mark `RCTT` on a `DWORD` boundary.

Because of this alignment there is a little bit of wastage with each block, but, to obtain both head and tail alignment it is unavoidable.

Two types of memory can be allocated; fixed, from the top of the heap and purgable, from the bottom of the heap. This will allow you to allocate persistent buffers while allocating transient buffers without fragmenting the heap.

Each memory block has a callback routine that is called when the block is either purged or freed. `amHeapPurge` will perform a top down purge of the bottom of the heap to gain `sizeNeeded` blocks of memory.

The `amHeapFree` call allows a less drastic approach to this by allowing a top down releasing of the memory blocks. It will not release a block that is not the top block as that would fragment the bottom of the heap zone.

You can either use this system or your own as none of the AM sub-systems call to this API.

1.2.5 amSound

This subsystem plays one shot sound effects that are contained in aggregated bank files. Because the memory that holds the bank file has already been allocated it this sub-system does not ask for nor allocate sound memory.

The sound effects have some simple parameters that are accessed via the bank build script that is produced by the `MkScript` tool. This allows you to inflict random pitch on sound effects to prevent player burn out and monotony, volume control and loop control.

Provisions for real time control of the sounds' placement in the sound field via volume, pan, and Q-Sound are provided.

In the case of the set volume and pan functions if they are called when the sound is not playing the value will be set in the sound object and when the play call is made that value will be sent to the driver. This avoids sending the driver unnecessary messages.

For a comprehensive example of using the `amSound` system see the file `MySfx.c` in the examples.

The `amSoundPlay` call uses the `amSoundPlayRaw` call which will play a raw asset.

1.2.6 amStream

The streamer uses block interleaved files made by the `MkStream` tool. These files have a 2048 byte header area that starts with a data structure, then some plain text information about the file and possibly the contents of an abstract file that can provide further plain text information. The structure of the header can be seen in `amstrhdr.h`.

Open the samples stream file in a hex editor (MSVC or whatever) and you will see this information. This is provided so that the programmer who didn't make the file can see what the spec's are for that file without too much pain.

Stream files have `n` tracks and each track has 1 or 2 channels. Currently streams can only contain mono tracks or stereo tracks. If a mixture is needed deinterleave the stereo file(s) and make them into mono tracks panned hard left and hard right.

The file needs to be interleaved so that each block of sound data in the file is the same size as half of the play buffer size that is requested for the stream. Further this must be a number divisible by 2048 as this reduces the loading of the low level file system.

This block sizing allows the streamers data pump to fill half of each play buffer in one cycle. The transfer buffer should be at least the play buffer size * the number of channel in the file.

The streamer works by interrupt, for a mono stream two points are calculated in the play buffer, the middle in frames and the end in frames. The play buffer is primed from the transfer buffer and a callback is set for the middle of the buffer, play is initiated and control returns to the caller.

When the interrupt comes in the streams ISR is invoked by the voice pool. This ISR sets a callback for the end of the buffer and a flag that says its time to fill the front of the buffer. The `amStreamManager` sees this flag and fills the front of the buffer. At this point the "pump" is running and it will continue to run until either stopped or the number of `interruptsTillEnd` is reached. When the end is reached if a user callback was installed it will be invoked during the final interrupt.

For a comprehensive example of using the `amStream` system see the file `MyStream.c` in the examples.

1.2.7 amMidi

This API allows the playing of Type 0 Standard MIDI files. There are a number of high level functions that allow control of the MidiDa drivers sequencer and then a number of lower level functions that allow the sending of MIDI messages to the drivers MIDI parser directly.

Supplied with the examples are a General Midi instrument bank and a GM drum bank. These may be used in your products on a royalty free basis.

User callbacks are supported, note that since all callbacks are registered with the amVoice manager the MIDI port must be allocated prior to the installation of the callback.

In the case of the set volume and pan functions if they are called when the sequence is not playing the value will be set in the sequence object and when the play call is made that value will be sent to the driver. This avoids sending the driver unnecessary messages.

The amMidiPlay call uses the amMidiPlayRaw call which will play a raw MIDI asset.

1.3 The AC Layer

This layer consists of functions that fill out driver control blocks, AC_COMMAND's, and send them to the driver. The driver has a 32 entry command queue that the commands are placed in by the acWriteCommand() function. As the SH4 is so much faster then the ARM7 processor it is possible to flood the command queue if more then 32 commands are sent at a time. Certain "Meta" commands have been created to ease the bandwidth in the AC->driver command pipe.

All of the AC layer commands take the control values that are native to the system. These are called AICA values, 0-15 for volume and 0-31 for pan.

AC functions are error trapped, if NULL parameters or out of range values that can not be corrected are used the functions will not pass these bad values to the driver. In that case the function will return false and issue an error message. If the value is correctable like an out of range volume it will be corrected to the nearest in range value and a warning message will be issued via the error messaging system. All procedures that fix-up argumentary values are noted in that specific functions documentation.

The hardware platform has 64 digital voice channels, with the audio64 driver all of these are available for digital audio playback. Under the MidiDa driver 48 of these digital voices are dynamically allocated to the MIDI playback engine. These are controlled by 16 fully polyphonic midi ports. The remaining 16 voice channels, 0-15, are available for digital audio playback. The MIDI ports are numbered 0-15 as well but are reported back by the drivers interrupt messaging service as channels 16-31.

All assets that are used by the AC layer must reside in sound memory and be an even multiple of 4 in size. This is because sound memory can only be written as DWORD's, byte writes to sound memory will cause the memory system to malfunction.

The AC layer uses raw (headerless) PCM or ADPCM audio data and Standard MIDI file type 0 assets. The DSP and MIDI tonebank assets are a Sega proprietary format that is created using the Sega Macintosh tool set. A DLS Level 1 Instrument Collection to tonebank converter is also available.

When writes are made to sound memory they pass through a 32 byte deep FIFO. For critical writes to sound memory the state of this G2 buss FIFO must be observed. Failure to observe this FIFO and creating sustained writes (> 16 msec) can lead to loss of data on the Maple bus.

The acSystem sub-system is a group of slightly higher level functions that provide driver installation and system interrogatory functions. This allows the programmer access to the inner workings of the AC layer and the driver.

The acInt sub-system provides calls to install OS services for interrupt chaining and removal of interrupts as well as a default ARM interrupt handler. The default handler may be found also in the boilerplate code in MyInt.c in the examples.

By using only the AC layer and installing a custom callback handler callbacks can be fielded with out the pain of writing an interrupt handler.

The `acMidi` group of functions will only have an effect if the `MidiDa` driver is installed. Otherwise they will have no effect, in R8 they will return true but no action will be taken, in R9 they will return false.

This group of functions allow manipulation of the MIDI sequencer that is part of the driver.

The `acDigi` group of functions allow for the playing and real-time control of digital sound effects. They work with either driver but are subject to the channel restrictions stated at the start of this section.

With all Midi and Digi library sub-systems the order of calls is:

```
Open
Play
Stop
Close
```

It is necessary to call the close function prior to reopening the port. The close returns ports to a known default state, and is especially necessary if ports are dynamically allocated and re-used, where sample or bit rates may change.

The `acCd` subsystem allows the initialization of the audio output path of the CD's Redbook playback system. The actual calls to make the drive read (play) a Redbook track are part of the file system for your specific operating system. That is why the play, stop and pause calls are not found in the AC layer. If the AC CD subsystem is not set up prior to playing Redbook tracks the result is undefined.

`acDsp` allows the installation of the two part DSP program objects created by the Sega DSP editor tool. The programs consist of two parts, the actual DSP program code and, the output assignments. The DSP output system has 16 channels that can either be assigned to the program, dry, or Q-Sound.

If a channel has the Q-Sound algorithm assigned to it gains an additional 0 based index starting with the first Q-Channel found in the output bank. This is because the Q-Sound algorithm has some internally settable parameters that are not part of the standard output channels controls.

In the example banks the first 12 channels are assigned to a medium reverb patch on channel 0, the next 4 channels are Q-Sound channels; these are output on channels 12-15. They are Q-Sound channels 0-3.

1.4 Tools Overview

For specific operation instructions for each tool see that tools `readme.txt` file. Each example that uses a tool on its assets contains an `assets` directory. In this directory is a `makeit.bat` file that contains the command lines used to build the assets. Running this batch file will rebuild the assets for the example.

1.4.1 MkScript

This tool writes build scripts for the bank builder tool `MkBank`. These scripts can be edited by hand to change the audio asset parameters for each asset type. The parameters are documented in the head of each script file.

This is set up so that each bank's assets can be placed in a directory then have the script and bank builders run on them.

Because some hand work may be done on the scripts when `MkScript` is re-run it renames the last script file rather than overwriting it.

To control the order of assets in a bank either edit the file names so that the standard directory sort order is correct or change their order in the script. The assets will be placed into the bank in the same order that they appear in the script.

1.4.2 MkBank

This tool reads bank scripts made by `MkScript` and then based on the information contained within builds a concatenated bank file from the assets.

Bank files may contain up to 3 bytes of 0x00 at the end to make them be writable to sound memory (evenly divisible by 4).

Banks will recognize certain file extensions. The extensions are `.mid`, `.wav`, `.mpb`, `.fpb` and `.fob` a file with any other extension will be added to the bank as an “unknown” type asset.

It is important that these standard assets have the correct extensions so that the fetch routines will work. If the extension is incorrect the asset will be built into the bank as the wrong type and it’s respective fetch routine will reject it.

The “unknown” type is there so that you may add your own proprietary types of assets to banks. This type allows 8 user parameters, it is up to you to determine what you want these parameters to represent for your unknown assets. All of these parameters will be defaulted to 0 in the raw script file, these parameters will be represented internally as signed longs.

When a bank is built a header (`.h`) file is generated, this contains constants that may be used to fetch the assets in the bank. The constants are a synthesis of an identifier and the assets file name and type. The header also contains a constant containing the file name of the bank using this system producing the right asset at the right time should be greatly simplified.

1.4.3 MkStream

This tool builds stream (`.str`) files. This file format allows a number of tracks and each track has one or two channels. The interleave rate of the file must match the interleave rate being requested by the programmer or the streamer code will reject the file. Further the interleave rate is forced to be a multiple of 2048 by having it stated on the command line in terms of 2048 byte blocks.

An abstract can be added to the header of the file as plain text allowing the inclusion of build, copyright or other information.

The stream file details its specs in the header in plain text just prior to the abstract file. Open one in a text or hex editor and you will see the human readable header information. The first part of the header is a binary representation of the files specs that is used by the API to set up the play for that file.

1.5 File Formats

Bank File (.kat)

[numberOfAssets]	4 bytes
[headerRecord]	sizeof(AM_BANK_FILE_UNION)
[...]	sizeof(AM_BANK_FILE_UNION)
[asset 0]	variable
[...]	variable

Stream File (.str), 1 track 2 channels, play buffer size 16384 bytes

Header contains...

[binaryHeader]	sizeof(AM_STREAM_FILE_HEADER)
[textInfo]	2048 - sizeof(AM_STREAM_FILE_HEADER)

File contains...

[header]	2048 bytes
[tlc1]	8192 bytes
[tlc2]	8192 bytes
[tlc1]	8192 bytes
[tlc2]	8192 bytes
[...]	

2. The AICA Control Layer API

acSystemRequestArmInterrupt Causes the driver to raise an ARM external interrupt.

FORMAT

```
#include <ac.h>

KTBOOL acSystemRequestArmInterrupt(KTU32 interruptId)
```

PARAMETERS

KTU32 interruptId, This value will be reported into the callback handler as its arg (0-255).

RETURN VALUE

KTTRUE if successful
KTFALSE if unable to send command or interruptId is out of range (0-255).

FUNCTION

Will raise an ARM external interrupt that will be fielded by the ARM interrupt handler on the SH4 side.

Note: Under the audio64 DA driver the first 64 ID's (0-63) are taken for use by the 64 voice channels; the MidiDa driver will use the first 32 (0-31) ID's to report the voice'sports.

The remaining ID's are available for USER application purposes.

acDigiPlay

Starts a buffer playing.

FORMAT

```
#include <ac.h>
```

```
KTBOOL acDigiPlay(KTU32 port,KTU32 startOffset, KTS16 aicaLoopFlag)
```

PARAMETERS

port	Voice channel number, 0-63 for audio64 driver, 0-15 for MidiDa driver.
startOffset	Play from start of buffer assigned to port (only 0 supported for this release).
aicaLoopFlag	Play looping buffer, 0 (loop off) or 0xff (loop on). If this is out of range it will be set to AC_LOOP_OFF

RETURN VALUE

KTTRUE if successful

KTFALSE if port is out of range or startOffset != 0 or unable to send command.

FUNCTION

Plays buffer assigned to the voice channel by `acDigiOpen()`. The total length of buffer must be < 64k sample frames.

acDigiPlayWithLoopParameters

Starts a buffer playing Set loop points.

FORMAT

```
#include <ac.h>

KTBOOL acDigiPlay(KTU32 port,
KTU32 startOffset,
KTS16 aicaLoopFlag,
KTU16 loopStartOffsetInFrames,
KTU16 loopEndOffsetInFrames)
```

PARAMETERS

port	Voice channel number, 0-63 for audio64 driver, 0-15 for MidiDa driver.
startOffset	Play from start of buffer assigned to port. (only 0 supported for this release)
aicaLoopFlag	Play looping buffer, AC_LOOP_ON or AC_LOOP_OFF.

Note: If this is out of range it will be set to AC_LOOP_OFF

loopStartOffsetInFrames The 0 based loop start offset in sample frames.
loopEndOffsetInFrames The 0 based loop ending offset in sample frames.

RETURN VALUE

KTTRUE if successful.
KTFALSE if port is > 63, or startOffset != 0, or unable to send command

FUNCTION

Plays buffer assigned to the voice channel by acDigiOpen(). The total length of buffer must be < 64k sample frames. The sample loop offsets are 0 based numbers that are expressed in sample frames e.g. 16 bit data is 2 bytes, 8 bit is 1 byte and 4 bit (ADPCM) is 1 nibble per frame.

acDigiPlayWithParameters

Starts a buffer playing with all common parameters.

FORMAT

```
#include <ac.h>

KTBOOL acDigiPlayWithParameters(          KTU32 port,
KTU32 volume,
KTU32 pan,
KTU32 dspMixerChannel,
KTU32 dspSendLevel,
KTS32 frequencyOrCentsOffset,
AC_PITCH_SET_TYPE frequencyOrCentsFlag,
KTU32 callbackOffsetInFrames,
KTU16 loopStartOffsetInFrames,
KTU16 loopEndOffsetInFrames)
```

PARAMETERS

port	Voice channel number, 0-63 for audio64 driver, 0-15 for MidiDa driver.
volume	0-15, soft to loud.
pan	0-31, left to right.
dspMixerChannel	0-15, needs to match DSP algorithm mapping.
dspSendLevel	0-15, min to max.
frequencyOrCentsOffset	either the real world sample rate i.e. 44100, 32000, 22050 etc. or the pitch offset in cents.
frequencyOrCentsFlag	One of the following values from ac.h
AC_PITCH_NO_CHANGE	values in frequencyOrCentsOffset are ignored.
AC_PITCH_AS_SAMPLE_RATE	value in frequencyOrCentsOffset will be interpreted as a real world sample rate.
AC_PITCH_AS_CENT_VALUE	value in frequencyOrCentsOffset will be interpreted as a cents offset from the root pitch at which the port was opened.
callbackOffsetInFrames	The 0 based callback offset in sample frames
loopStartOffsetInFrames	The 0 based loop start offset in sample frames, ignore == 0.
loopEndOffsetInFrames	The 0 based loop ending offset in sample frames, ignore == 0.

RETURN VALUE

KTTRUE if successful.
KTFALSE if port or dspMixerChannel is out of range or command write failed,

FUNCTION

Plays buffer assigned to the voice channel by `acDigiOpen()`. The total length of buffer must be < 64k sample frames. The sample loop offsets are 0 based numbers that are expressed in sample frames e.g. 16 bit data is 2 bytes, 8 bit is 1 byte and 4 bit (ADPCM) is 1 nibble per frame.

Note: If volume, pan or `dspSendLevel` are out of range they will be set to equal the max value for the range.

Note: If the loop offsets are set to 0 they will be ignored by the driver.

acDigiMultiSetMask

Sets the bit masks for acDigiMultiPlay()

FORMAT

```
#include <ac.h>
```

```
KTBOOL acDigiMultiSetMask(KTU32 port,KTU32 *uppermask, KTU32 *lowermask)
```

PARAMETERS

port	Voice channel number, 0-63 for audio64 driver, 0-15 for MidiDa driver.
KTU32 *uppermask,	A pointer to the upper 32 channel mask, voices 32-63
KTU32 *lowermask,	A pointer to the lower 32 channel mask, voices 0-31

RETURN VALUE

KTTRUE if successful

KTFALSE if upperMask or lowerMask is NULL or port is out of range.

FUNCTION

Creates channel masks for use with the acDigiMultiPlay() function. This may be called in a loop to set multiple channels in the mask.

acDigiMultiPlay

Sets the bit masks for acDigiMultiPlay()

FORMAT

```
#include <ac.h>
```

```
KTBOOL acDigiMultiPlay(KTS32 aicaLoopFlag, KTU32 upperMask, KTU32 lowerMask)
```

PARAMETERS

KTU32 aicaLoopFlag, Start channels as looping or not, AC_LOOP_ON or AC_LOOP_OFF

KTU32 *upperMask, A pointer to the upper 32 channel mask, voices 32-63

KTU32 *lowerMask, A pointer to the lower 32 channel mask, voices 0-31

RETURN VALUE

KTTRUE if successful

KTFALSE if upperMask is 0 or the top 4 bits of lowerMask are set and MidiDa driver is in use.

if upper and lower masks are 0.

FUNCTION

Starts a group of channels as a phase locked gang.

acDigiMultiStop

Sets the bit masks for acDigiMultiPlay()

FORMAT

```
#include <ac.h>
```

```
KTBOOL acDigiMultiStop(KTU32 upperMask, KTU32 lowerMask)
```

PARAMETERS

KTU32 *upperMask, The upper 32 channel mask, voices 32-63

KTU32 *lowerMask, The lower 32 channel mask, voices 0-31

RETURN VALUE

KTTRUE if successful

KTFALSE if upperMask is 0 or the top 4 bits of lowerMask are set and MidiDa driver is in use.

 if upper and lower masks are 0.

FUNCTION

Stops a group of channels as a phase locked gang.

acDigiOpen

Open a DA Streaming Port for playback.

FORMAT

```
#include <ac.h>
```

```
KTBOOL acDigiOpen(KTU32 port,KTU32 address,KTU32 sizeInBytes,AC_AUDIO_TYPE  
aicaAudioType,KTS32 aicaSampleRate)
```

PARAMETERS

port	Voice channel number, 0-63 for audio64 driver, 0-15 for MidiDa driver.
address	the address in sound memory
sizeInBytes	buffer length in bytes, maximum 128k for 16bit data,64k for 8bit data, 32k for 4bit (ADPCM) data.
aicaAudioType	format type 4, 8, or 16 bit. See AC_AUDIO_TYPE data type enumeration in ac.h
<pre>typedef enum { AC_16BIT, AC_8BIT, AC_ADPCM AC_ADPCM_LOOP } AC_AUDIO_TYPE;</pre>	
sampleRate	Base real world sample rate. Further play commands on this open port will start at this rate unless changed by a call to acSetSampleRate().

RETURN VALUE

KTTRUE	if successful
KTFALSE	if port is out of range, address is 0, sizeInBytes is 0, audioType is out of range, sampleRate exceeds 1128900 or command write fails.

FUNCTION

Opens a digital voice channel and assigns a buffer, root pitch and loop status to the voice.

acDigiSetSampleRate

Set the playback rate (sample rate) of audio stream.

FORMAT

```
#include <ac.h>
```

```
KTBOOL acDigiSetSampleRate(KTU32 port, KTS32 sampleRate)
```

PARAMETERS

<code>port</code>	Voice channel number, 0-63 for audio64 driver, 0-15 for MidiDa driver.
<code>sampleRate</code>	The real world sample rate to set for the indicated the voice channel.

RETURN VALUE

<code>KTTRUE</code>	if successful
<code>KTFALSE</code>	if port is out of range, sampleRate exceeds 1128900, or command write fails.

FUNCTION

This changes sample rate (playback rate) of a currently running voice channel.

The voice will be set to the closest approximation of that sample rate the hardware is capable of reproducing.

acDigiClose

Closes port previously opened.

FORMAT

```
#include <ac.h>
```

```
KTBOOL acDigiClose(KTU32 port)
```

PARAMETERS

port	Voice channel number, 0-63 for audio64 driver, 0-15 for MidiDa driver.
------	--

RETURN VALUE

KTTRUE	if successful
KTFALSE	if port is out of range or command write fails.

FUNCTION

Closes voice channels opened with `acDigiOpen()`. It is important to close a channel and to not iteratively open the results of that type of methodology are undefined.

acDigiSetCurrentPitch

Changes the playback rate of a running channel.

FORMAT

```
#include <ac.h>
```

```
KTBOOL acDigiSetCurrentPitch(KTU32 port, KTS32 pitchOffsetInCents)
```

PARAMETERS

port	Voice channel number, 0-63 for audio64 driver, 0-15 for MidiDa driver.
pitchOffsetInCents	Pitch in cents. (-8400 to 8400)

RETURN VALUE

KTTRUE	if successful
KTFALSE	if port is out of range or command write fails.

FUNCTION

Changes the pitch of a currently running voice channel in increments of cents.

Cents is a musical measurement of pitch, one octave (frequency double or half) is 1200 cents.

Making this call will not change the default setting of the voice channel but it will change the pitch of a sound that is currently playing on that channel.

If the currently playing sound stops and is retriggered with a call to `acDigiPlay()` it will play at the sample rate that the voice channel was set up for in the `acDigiOpen()` call.

Calling this with an arg of 1200 will make the sound play up one octave, a second call to this with an arg of 0 will make the sound play at the setting to which the voice was initialized in the `acDigiOpen()` call.

acDigiSetVolume

Adjusts volume of a voice channel.

FORMAT

```
#include <ac.h>
```

```
KTBOOL acDigiSetVolume(KTU32 port,KTU32 aicaVolume) aicaVolume is 0-15
```

PARAMETERS

port	Voice channel number, 0-63 for audio64 driver, 0-15 for MidiDa driver.
aicaVolume	0-15, soft to loud.

RETURN VALUE

KTTRUE if successful

KTFALSE if the port is out of range or command write fails.

FUNCTION

Changes the direct output volume of an open voice channel, if a sound is currently playing on the channel the volume of that sound will be changed, if the channel is not playing this will set the volume used when that channel is started with a call to `acDigiPlay()`.

Note: If aicaVolume is out of range it will be set to AC_MAX_VOLUME.

acDigiSetPan

Adjusts the pan placement of a voice channel.

FORMAT

```
#include <ac.h>
```

```
KTBOOL acDigiSetPan(KTU32 port,KTU32 aicaPan)
```

PARAMETERS

port	Voice channel number, 0-63 for audio64 driver, 0-15 for MidiDa driver.
aicaPan	0-31, left to right.

RETURN VALUE

KTTRUE if successful
KTFALSE if the port is out of range or command write fails.

FUNCTION

Changes the direct output pan of an open voice channel, if a sound is currently playing on the channel the pan of that sound will be changed, if the channel is not playing this will set the pan used when that channel is started with a call to `acDigiPlay()`.

Note: If `aicaPan` is out of range it will be set to `AC_MAX_PAN`.

acDigiRequestEvent Used to generate an interrupt when a certain buffer position is reached.

FORMAT

```
#include <ac.h>
```

```
KTBOOL acDigiRequestEvent(KTU32 port,KTU32 offsetFromBeginningInFrames)
```

PARAMETERS

<code>port</code>	Voice channel number, 0-63 for audio64 driver, 0-15 for MidiDa driver.
<code>offsetFromBeginningInFrames</code>	0 based offset from start of buffer in sample frames. (0-65535)

RETURN VALUE

KTTRUE if successful
KTFALSE if the port is out of range or command write fails.

FUNCTION

When channel playback position reaches the indicated offset the driver will raise an ARM external interrupt causing the ARM interrupt handler to be invoked. The channel number of the calling channel will be placed into the drivers interrupt array. If multiple channels are reporting event requests at the same time there will be multiple entries in the drivers interrupt array. The number of channels reporting may be observed by measuring the incrementation of the interrupt array start offset which is available via the acSystem call `acSystemGetIntArrayStartOffset()`

Note: The parameter `offsetFromBeginningInFrames` is not error trapped.

acDigiStop

Stops a voice channel playing.

FORMAT

```
#include <ac.h>
```

```
KTBOOL acDigiStop(KTU32 port)
```

PARAMETERS

port Voice channel number, 0-63 for audio64 driver, 0-15 for MidiDa driver.

RETURN VALUE

KTTRUE if successful

KTFALSE if the port is out of range or command write fails.

FUNCTION

Stops playback of a previously started voice channel.

acMidiOpen

Open a MIDI Port buffer for SMF format 0 playback.

FORMAT

```
#include <ac.h>

KTBOOL acMidiOpen(KTU32 port,
KTU8 gmMode,
KTU32 address,
KTU32 sizeInBytes,
KTU32 pulsesPerQuarterNote)
```

PARAMETERS

port	MIDI port number, 0-15.
gmMode	AC_GM_ON or AC_GM_OFF, selects General MIDI mode on or off. Allows use of a Bank 0 General MIDI instrument and drumset.
address	address in sound ram of the start of buffer.
MidiBufferSize	buffer length in bytes.
TicksPerQNote	time base in ticks per quarter note (ppqn).

RETURN VALUE

KTTRUE if successful
KTFALSE if the port is out of range, address is 0, sizeInBytes is 0 or command write fails.

FUNCTION

Opens a MIDI port. Midi ports are fully polyphonic 16 channel ports for MIDI streams. This call sets the default set of parameters for a MIDI port, i.e. GM mode, the address of the Standard MIDI Type 0 asset in sound memory and the PPQN (pulses per quarter note) for that asset.

Note: If gmMode is out of range it will be set to AC_GM_OFF

acMidiSetTonebank

Assign a MIDI Program Bank (tonebank) to an active bank slot.

FORMAT

```
#include <ac.h>

KTBOOL acMidiSetTonebank(KTU8 toneBank,
AC_BANK_TYPE bankType,
KTU32 address,
KTU32 sizeInBytes,
KTU32 mttPtr)
```

PARAMETERS

toneBank	tone bank slot number (0-15)
bankType	AC_MELODIC_BANK for melodic banks or AC_DRUM_BANK for drum banks.
address	offset in sound memory of start of tone bank.
offset=	(addressInSoundMemory & 0x003ffff)
sizeInBytes	size of tone bank in bytes
mttPtr	MIDI translate table pointer (not implemented yet)

RETURN VALUE

KTRUE	if successful
KFALSE	if the toneBank or bankType is out of range, address is 0, sizeInBytes is 0 or command write fails.

FUNCTION

Sets a tonebank for active playback. Assigns a bank number to a tonebank slot that will be accessible via MIDI Bank Select messages in the sequence data.

acMidiClose

Close a MIDI port.

FORMAT

```
#include <ac.h>
```

```
KTBOOL acMidiClose(KTU32 port)
```

PARAMETERS

`port` MIDI port number, 0-15.

RETURN VALUE

`KTTRUE` if successful

`KTFALSE` if the port is out of range, or command write fails.

FUNCTION

Closes the indicated MIDI port and sends an `All Notes Off` message to the drivers midi parser.

acMidiPlay

Starts playback on opened MIDI port.

FORMAT

```
#include <ac.h>
```

```
KTBOOL acMidiPlay(KTU32 port,KTU32 startOffset, KTS16 aicaLoopFlag)
```

PARAMETERS

port	MIDI port number, 0-15.
startOffset	Start playback layback from buffer start position + offset.
loopFlag	Loop MIDI playback buffer, AC_LOOP_ON or AC_LOOP_OFF.

RETURN VALUE

KTTRUE if successful
KTFALSE if the port is out of range, or command write fails.

FUNCTION

Starts Standard MIDI File Type 0 playback on the given port from the start of the port's buffer plus startOffset. The default tempo is 120 BPM, until a MIDI tempo message is parsed.

Note: If aicaLoopFlag is out of range it will be set to AC_LOOP_OFF

acMidiStop

Stops standard MIDI file playback on port.

FORMAT

```
#include <ac.h>
```

```
KTBOOL acMidiStop(KTU32 port)
```

PARAMETERS

<code>port</code>	MIDI port number, 0-15.
-------------------	-------------------------

RETURN VALUE

KTTRUE if successful

KTFALSE if the port is out of range, or command write fails.

FUNCTION

Stops MIDI playback on the indicated port and sends an "All Notes Off" message to the drivers midi parser.

acMidiRequestEvent

Generates interrupt to host upon MIDI port reaching specified address.

FORMAT

```
#include <ac.h>
```

```
KTBOOL acMidiRequestEvent(KTU32 port,KTU32 offsetFromBeginningInBytes)
```

PARAMETERS

port	MIDI port number, 0-15.
portEventAddress	Sound memory event address.

RETURN VALUE

KTTRUE if successful
KTFALSE if the port is out of range, or command write fails.

FUNCTION

When MIDI playback position reaches the indicated offset the driver will raise an ARM external interrupt causing the ARM interrupt handler to be invoked. The channel (port + 16) number of the caller will be placed into the drivers interrupt array. If multiple channelsports are reporting event requests at the same time there will be multiple entries in the drivers interrupt array. The number of channelsports reporting may be observed by measuring the incrementation of the interrupt array start offset which is available via the acSystem call `acSystemGetIntArrayStartOffset()`.

Note: The parameter `offsetFromBeginningInBytes` is not error trapped.

acMidiPause

Pauses an active MIDI port.

FORMAT

```
#include <ac.h>
```

```
KTBOOL acMidiPause(KTU32 port)
```

PARAMETERS

port	MIDI port number, 0-15.
------	-------------------------

RETURN VALUE

KTTRUE	if successful
KTFALSE	if the port is out of range, or command write fails.

FUNCTION

Pauses playback on an actively playing MIDI port. Sends an `All Notes Off` message to the drivers midi parser.

acMidiResume

Resumes playback on active MIDI port.

FORMAT

```
#include <ac.h>
```

```
KTBOOL acMidiResume(KTU32 port)
```

PARAMETERS

port	MIDI port number, 0-15.
------	-------------------------

RETURN VALUE

KTTRUE if successful

KTFALSE if the port is out of range, or command write fails.

FUNCTION

Resumes playback on the given MIDI port. When playback is resumed running status mode is retained from the point at which the sequence was paused.

acMidiSetVolume

Sets scaled volume setting for MIDI port.

FORMAT

```
#include <ac.h>
```

```
KTBOOL acMidiSetVolume(KTU32 port,KTU32 portMasterVolume)
```

PARAMETERS

port	MIDI port number, 0-15.
portMasterVolume	Global volume setting for port, (0-127).

RETURN VALUE

KTTRUE if successful
KTFALSE if the port is out of range, or command write fails.

FUNCTION

Sets global volume setting for the given MIDI port. This will cause the driver to scale all MIDI CC7 (volume) messages accordingly. This will affect ALL channels in the sequence running on the port.

Note: If portMasterVolume is out of range it will be set to AC_MAX_MIDI_VOLUME.

acMidiReset

Resets MIDI controllers on port to default values.

FORMAT

```
#include <ac.h>
```

```
KTBOOL acMidiReset(KTU32 port)
```

PARAMETERS

port MIDI port number, 0-15.

RETURN VALUE

KTTRUE if successful

KTFALSE if the port is out of range, or command write fails.

FUNCTION

Resets controller values to standard defaults. The bank select per channel is set to 0.

The MIDI continuous controller settings affected are as follows:

CC7	= 100
CC11	= 127
CC10, CC71, CC74	= 64
CC20-CC28, CC88	= 32
CC0, CC52-CC56, CC70	= 0
Pitch Bend	= 0 (center).

acMidiSetTempo

Set playback tempo of MIDI port.

FORMAT

```
#include <ac.h>
```

```
KTBOOL acMidiSetTempo(KTU32 port,KTU32 microSecondsPerQuarterNote)
```

PARAMETERS

port	MIDI port number, 0-15.
microSecondsPerQuarterNote	Sets the Microseconds per Quarter Note for MIDI port.

RETURN VALUE

KTTRUE if successful

KTFALSE if the port is out of range, or command write fails.

FUNCTION

Allows real-time control of tempo for port.

Note: The parameter microSecondsPerQuarterNote is not error trapped.

acMidiSendMessage

Sends raw MIDI messages to ports.

FORMAT

```
#include <ac.h>

KTBOOL acMidiSendMessage(KTU32 port,
KTU32 channel,
KTU32 midiMessage,
KTU32 value1,
KTU32 value2)
```

PARAMETERS

port	MIDI port number, 0-15.
channel	MIDI channel number, 0-15.
midiMessage	MIDI command number (channel nibble ignored), Channel voice messages 0x80-0xe0.
midiValue1	First MIDI voice message data byte.
midiValue2	Second MIDI voice message data byte.

RETURN VALUE

KTTRUE if successful
KTFALSE if the port, channel, value1 or value2 is out of range, or command write fails.

FUNCTION

Immediately sends raw MIDI message to port by channel number. Allows real-time dynamic control of note-on and controller messages, etc.

Note: The parameter midiMessage is not error trapped.

acCdSetVolume

Sets Left & Right Channels for Redbook Volume Control (dependent on channel pan).

FORMAT

```
#include <ac.h>
```

```
KTBOOL acCdSetVolume(KTU32 leftVolume,KTU32 rightVolume)
```

PARAMETERS

leftVolume Volume level for left audio channel, 0-127.

rightVolume Volume level for right audio channel, 0-127.

RETURN VALUE

KTTRUE if successful

KTFALSE if the leftVolumerightVolume is out of range, or command write fails.

FUNCTION

Sets volume level for CD-DA (Redbook) playback channels.

Note: Left and right depend on CD-DA Pan position.

See: gdfsgdda documentation for calls to play and stop tracks.

acCdSetPan

Sets Left & Right Channel pan position.

FORMAT

```
#include <ac.h>
KTBOOL acCdSetPan          (KTU32 leftPan,KTU32 rightPan)
```

PARAMETERS

leftPan	Pan Position for left audio channel, 0-127, left to right.
rightPan	Volume level for right audio channel, 0-127, left to right.

RETURN VALUE

KTTRUE if successful
KTFALSE if the leftPanrightPan is out of range, or command write fails.

FUNCTION

Sets pan position for CD-DA playback channels.

Note: This may affect the behavior of volume command since left channel can be set to the right pan position and vice versa.

acCdInit

Resets CDDA channels to hard pan positions and maximum volume.

FORMAT

```
#include <ac.h>

KTBOOL acCdInit(void)
```

PARAMETERS

void

RETURN VALUE

KTTRUE if successful
KTFALSE if command write fails.

FUNCTION

Sets default pan position and volume for CDDA playback channels.

Note: This must be called prior to playing back redbook audio from the CD.

acDspSetQSoundAngle

Sets Q-Sound position.

FORMAT

```
#include <ac.h>
```

```
KTBOOL acDspSetQSoundAngle(KTU32 qSoundChannel,KTU32 angle)
```

PARAMETERS

qSoundChannel	The 0 based Q-Sound channel number, if the effect patch has 4 channels of qsound and they are mixer channels 12-16 then for the sound on mixer channel 12 the Q-Sound channel is 0, 13 = 1 etc... range: 0-7
angle	0-127, left to right.

RETURN VALUE

KTTRUE if successful

KTFALSE if command write fails.

FUNCTION

Allows the setting of the Q-Sound angle parameter in real time.

Note: The parameter angle is trapped so that if it is out of range it will be set to 127 and the function will continue to execute.

acDspInstallProgram

Registers a dsp program bank with the driver.

FORMAT

```
#include <ac.h>
```

```
KTBOOL acDspInstallProgram(KTU32 address,KTU32 sizeInBytes)
```

PARAMETERS

KTU32 address, The address of the program bank in sound memory.

KTU32 sizeInBytes, The size in bytes of the program bank.

RETURN VALUE

KTTRUE if successful

KTFALSE if address is NULL, size in bytes is 0 or command write failed.

FUNCTION

Sets a DSP program bank as the current DSP program. This program bank is currently produced using the Mac DSP editor tool.

acDspInstallOutputMixer

Registers an output mixer patch with the driver.

FORMAT

```
#include <ac.h>
```

```
KTBOOL acDspInstallOutputMixer(KTU32 address,KTU32 sizeInBytes)
```

PARAMETERS

KTU32 address, The address of the output mixer bank in sound memory.

KTU32 sizeInBytes, The size in bytes of the output mixer bank.

RETURN VALUE

KTTRUE if successful

KTFALSE if address is NULL, size in bytes is 0 or command write failed...

FUNCTION

Sets an output mixer bank as the current output routing. This output mixer bank is currently produced using the Mac DSP editor tool.

acDspSetMixerChannel

Sets DSP mixer level and channel for that port.

FORMAT

```
#include <ac.h>
```

```
KTBOOL acDspSetMixerChannel(KTU32 port,KTU32 mixer,KTU32 level)
```

PARAMETERS

port	Voice channel number, 0-63 for audio64 driver, 0-15 for MidiDa driver.
toMixerChannel	DSP mixer channel number, 0-15.
sendLevel	Audio signal level, 0-15.

RETURN VALUE

KTTRUE if successful

KTFALSE if port is out of range or command write failed...

FUNCTION

Set port's audio signal to DSP mixer channel to enable DSP effects for stream. This allows stream to be altered by reverb, etc.

Note: The parameters mixer and level are not error trapped.

acErrorGetLast

Gets a pointer to the error structure.

FORMAT

```
#include <ac.h>
```

```
AC_ERROR_PTR acErrorGetLast(void)
```

PARAMETERS

void

RETURN VALUE

AC_ERROR_STRUCT a pointer to the AC error structure.

FUNCTION

Gets a pointer to the AC error structure. This contains an error number enumerated as an AC_ERROR_TYPE in ac.h and a more informative error message that tells the name of the function that failed as well as some descriptive text regarding the cause of the failure.

acErrorExists

Checks to see if an error condition exists.

FORMAT

```
#include <ac.h>
```

```
KTBOOL acErrorExists(void)
```

PARAMETERS

void

RETURN VALUE

KTTRUE if a error exists

KTFALSE if no error exists.

FUNCTION

Allows checking of the error state for the AC layer in a single call returning a bool.

acErrorClear

Clears the AC error structure.

FORMAT

```
#include <ac.h>

void acErrorClear(void)
```

PARAMETERS

RETURN VALUE

KTTRUE if successful
KTFALSE if unable to send command or interruptId is out of range (0-255).

FUNCTION

Clears the AC Error structure.

acIntInstallOsChainDeleteManager Installs pointer to interrupt chain delete routine.

FORMAT

```
#include <am.h>
```

```
KTBOOL acIntInstallOsChainDeleteManager(AC_INT_CHAIN_DELETE_MANAGER  
theChainDeleteManager)
```

PARAMETERS

AC_INT_CHAIN_DELETE_MANAGER theChainDeleteManager, a pointer to the wrapped routine.

The wrapper prototype is defined as follows:

```
void                                feiux(KTU32);
```

RETURN VALUE

KTTRUE if the proc was installed.

KTFALSE if the proc was not installed due to a prior initialization of the vector.

FUNCTION

Allows installation of OS specific interrupt chain removal procPointer. The procedure's wrapper must have the following prototype: `void foo(KTU32);` the argument being the chain ID to be removed.

Note: This **MUST** be done prior to calling `acInit()` or `amInit()` or init failure will result.

See Also: `MyInt.c` (a part of the samples)

acIntInstallOsChainAddManager

Installs proc pointer to interrupt chain add routine.

FORMAT

```
#include <am.h>
```

```
KTBOOL acIntInstallOsChainAddManager(AC_INT_CHAIN_ADD_MANAGER theChainAddManager)
```

PARAMETERS

AC_INT_CHAIN_ADD_MANAGER theChainAddManager, a pointer to the wrapped routine.

The wrapper prototype is defined as follows:

```
KTU32 foo(KTS16,AC_ARM_INTERRUPT_HANDLER,KTU32,void *);
```

RETURN VALUE

KTTRUE if the proc was installed.

KTFALSE if the proc was not installed due to a prior initialization of the vector.

FUNCTION

This allows the app programmer to wrap a given OS's interrupt chain add routine and send it to the audio system. This provides for OS neutrality.

See Also: MyInt.c (a part of the samples)

acIntInstallCallbackHandler Installs a callback handler into the ARM interrupt handler.

FORMAT

```
#include <am.h>
```

```
KTBOOL acIntInstallCallbackHandler(AC_CALLBACK_HANDLER theCallbackHandler)
```

PARAMETERS

AC_CALLBACK_HANDLER theCallbackHandler, a pointer to a callback handler function.

The prototype of the callback handler function is as follows:

```
void        fuu(volatile KTU32);
```

RETURN VALUE

KTTRUE if the proc was installed.

KTFALSE if the proc was not installed due to a prior initialization of the vector.

FUNCTION

Installs a callback handler into the ARM interrupt handler. This allows developers wanting to work at the AC level to get callbacks from both the voice channelsmidi ports and from the interruptId arg to acSystemRequestArmInterrupt().

Please note that the audio64 driver claims the first 64 ID's (0-63) while the MidiDa driver claims the first 32 (0-31) ID's. MIDI ports report the (port + 16) so in using the MidiDa driver 0-15 are the 16 available digital voice channels and 16-31 are the 16 available MIDI ports.

The ARM interrupt handler is installed into the OS's ARM external interrupt chain. It is invoked when ever an ARM interrupt is raised. The interrupt handler parses the drivers interrupt array to determine which channels are reporting on this interrupt cycle, it then calls the callback handler once for each message it finds in the drivers interrupt array.

See Also: KTBOOL acSystemRequestArmInterrupt(KTU32 interruptId)

See Also: MyInt.c (a part of the samples)

acIntInstallArmInterruptHandler

Installs an ARM interrupt handler.

FORMAT

```
#include <am.h>
```

```
KTBOOL acIntInstallArmInterruptHandler(AC_ARM_INTERRUPT_HANDLER theInterruptHandler)
```

PARAMETERS

AC_ARM_INTERRUPT_HANDLER theInterruptHandler, a pointer to an interrupt handler function.

The prototype of the callback handler function is as follows:

```
void feeb(void *);
```

The value AC_ARM_INTERRUPT_HANDLER_ID will be incoming as the argument to this function if it is a legitimate interrupt message.

RETURN VALUE

KTTRUE if the proc was installed.

KTFALSE if the proc was not installed due to a prior initialization of the vector.

FUNCTION

Initializes the ARM interrupt handler vector with your function. A default function will be installed if this vector has not been initialized at startup time. This default handler is illustrated in MyInt.c and is described below.

The default ARM interrupt handler is installed into the OS's ARM external interrupt chain.

This is done at start up if a user handler has not been supplied via this routine.

It is invoked when ever an ARM interrupt is raised. The interrupt handler parses the drivers interrupt array to determine which channels are reporting on this interrupt cycle, it then calls the callback handler once for each message it finds in the drivers interrupt array.

See Also: KTBOOL acSystemRequestArmInterrupt(KTU32 interruptId)

See Also: MyInt.c (a part of the samples)

acIntSetAicaChainId

Sets the AICA interrupt chain ID.

FORMAT

```
#include <am.h>

void acIntSetAicaChainId(KTU32 chainId)
```

PARAMETERS

KTU32 chainId,	The OS specific ID for the AICA EXTERNAL interrupt in the case of Shinobi it is 0xb20
----------------	---

RETURN VALUE

void

FUNCTION

Sets the interrupt ID for the AICA external interrupt, this is used when installing interrupt handlers.

Note: This defaults to 0xb20

See Also: `MyInt.c` (a part of the samples)

acIntShutdown

Shuts down the ac interrupt system.

FORMAT

```
#include <am.h>
```

```
void acIntShutdown(void)
```

PARAMETERS

```
void
```

RETURN VALUE

```
void
```

FUNCTION

Shuts down the am interrupt system by removing the ARM interrupt callback from the OS using the chain delete function and clearing the OS service vectors.

See Also: `MyInt.c` (a part of the samples)

acIntInit

Initializes the ac interrupt system.

FORMAT

```
#include <am.h>

void acIntInit(void)
```

PARAMETERS

void

RETURN VALUE

KTTRUE	if the interrupt system was successfully initialized
KTFALSE	if OS based chain add or delete managers not installed, see the following functions to install these OS based services.
KTBOOL	<code>acIntInstallOsChainAddManager(AC_INT_CHAIN_ADD_MANAGER theChainAddManager);</code>
KTBOOL	<code>acIntInstallOsChainDeleteManager(AC_INT_CHAIN_DELETE_MANAGER theChainDeleteManager);</code>

FUNCTION

Initializes the am interrupt system by installing the ARM interrupt handler to the OS's ARM interrupt chain.

If user interrupt handler and or callback handlers have been installed these will not be overwritten by this function.

See Also: `MyInt.c` (a part of the samples)

acSystemShutdown

Shuts down the AC layer.

FORMAT

```
#include <ac.h>

void acSystemShutdown(void)
```

PARAMETERS

void

RETURN VALUE

void

FUNCTION

Shuts down the AC layer by removing the interrupt handler using the delete chain vector and clearing all of the OS service vectors.

acSystemGetIntArrayStartOffset

Gets the interrupt array write cursor offset.

FORMAT

```
#include <ac.h>
```

```
KTBOOL acSystemGetIntArrayStartOffset(KTU32 *interruptArrayStartOffset)
```

PARAMETERS

KTU32 *interruptArrayStartOffset, a byte pointer expressed as a KTU32 indicating the current write position within the drivers interrupt message array.

RETURN VALUE

KTTRUE if successful

KTFALSE if interruptArrayStartOffset is NULL or driver is not installed.

FUNCTION

Gets the interrupt array start offset from the driver. By comparing the movement of this number from interrupt to interrupt it can be determined how many messages are being returned and where they are located in the drivers interrupt message array.

The audio64 driver claims the first 64 ID's (0-63) while the MidiDa driver claims the first 32 (0-31) ID's. MIDI ports report the (port + 16) so in using the MidiDa driver 0-15 are the 16 available digital voice channels and 16-31 are the 16 available MIDI ports.

See Also: MyInt.c (a part of the samples)

acSystemGetIntArrayStartPtr Gets a pointer to the start of the drivers interrupt message array.

FORMAT

```
#include <ac.h>
```

```
KTBOOL acSystemGetIntArrayStartPtr(char **intArrayStartPointer)
```

PARAMETERS

char **intArrayStartPointer, a pointer to the start of the interrupt message array.

RETURN VALUE

KTTRUE on success

KTFALSE if drive is not installed or *intArrayStartPointer is NULL

FUNCTION

Gets a pointer to the start of the 64 byte interrupt message array in the driver.

This address is in SOUND memory so NO BYTE READS move it into SH4 memory before you start to dissect it in a byte wise fashion or sound memory will be turned into putty.

The audio64 driver claims the first 64 ID's (0-63) while the MidiDa driver claims the first 32 (0-31) ID's. MIDI ports report the (port + 16) so in using the MidiDa driver messages 0-15 are the 16 available digital voice channels and 16-31 are the 16 available MIDI ports.

acSystemGetBaseOfSoundMemory Gets the starting address for sound memory.

FORMAT

```
#include <ac.h>
```

```
KTBOOL acSystemGetBaseOfSoundMemory(KTU32 *baseOfSoundMemory)
```

PARAMETERS

KTU32 *baseOfSoundMemory, the address of the base of sound memory represented as a KTU32

RETURN VALUE

KTTRUE on success.

KTFALSE if the baseOfSoundMemory is NULL or driver is not installed.

FUNCTION

Gets the address of the base of sound memory represented as a KTU32.

acSystemGetIntArray

Gets the address of the SH4 side interrupt message array.

FORMAT

```
#include <ac.h>
```

```
KTBOOL acSystemGetIntArray(char **interruptArray)
```

PARAMETERS

RETURN VALUE

KTTRUE	if
KTFALSE	*interruptArray is NULL or driver is not installed.

FUNCTION

Gets the address of the SH4 side interrupt message array buffer that is contained in the `acSystem` structure.

Note: This is broken in R8 as it gets the SH4 side array but does not fill it from the driver.

acSystemGetIntArrayLength Gets the length of the drivers interrupt message array.

FORMAT

```
#include <ac.h>
```

```
KTBOOL acSystemGetIntArrayLength(KTU32 *interruptArrayLength)
```

PARAMETERS

KTU32 *interruptArrayLength,	the length of the interrupt message array is returned via this pointer.
------------------------------	---

RETURN VALUE

KTTRUE	on success
KTFALSE	if interruptArrayLength is NULL or the driver is not installed.

FUNCTION

Gets the length of the drivers interrupt message array.

Note: This is a vestigial function from when the Midi driver had a shorter message array then the DA driver. Now they both use 64 byte arrays.

acSystemCheckDriverRevision Tests the driver version against the supplied version.

FORMAT

```
#include <ac.h>
```

```
KTBOOL acSystemCheckDriverRevision(KTU8 *driver,KTU8 major,KTU8 minor,KTCHAR local)
```

PARAMETERS

KTU8 *driver,	An image in memory of the driver
KTU8 major,	The major revision desired
KTU8 minor,	The minor revision desired
KTCHAR local	The local version desired

RETURN VALUE

KTTRUE if it is the same version
KTFALSE if it is not the same version

FUNCTION

Used in `acSystemInstallDriver()` to test the driver revision.

See: The top of `ac.h` for the constants that it uses to test the driver.

acSystemGetDriverRevision

Tests the driver version against the supplied version.

FORMAT

```
#include <ac.h>
```

```
KTBOOL acSystemGetDriverRevision(KTU8 *driver,KTU8 *major,KTU8 *minor,KTCHAR *local)
```

PARAMETERS

KTU8 *driver,	An image in memory of the driver
KTU8 *major,	The major revision is returned via this pointer
KTU8 *minor,	The minor revision is returned via this pointer
KTCHAR *local	The local version is returned via this pointer

RETURN VALUE

KTTRUE the version was returned intact

KTFALSE if driver was NULL

FUNCTION

Called by acSystemCheckDriverRevision to obtain the driver revision.

acSystemWaitUntilG2FifolsEmpty

Waits until the G2 FIFO is clear.

FORMAT

```
#include <ac.h>

void acSystemWaitUntilG2FifoIsEmpty(void)
```

PARAMETERS

void

RETURN VALUE

void

FUNCTION

The G2 FIFO is 32 bytes deep, when writing critical messages to sound RAM the FIFO status must be checked to determine when the write is complete. For each check that it makes of the FFST bits it increments a counter to allow real time observation of the amount of waiting required.

acSystemDelay

Use to delay for short periods of time.

FORMAT

```
#include <ac.h>

void acSystemDelay(KTU32 delay)
```

PARAMETERS

KTU32 delay, the number of NOP's of delay.

RETURN VALUE

void

FUNCTION

Uses a loop with a no-op in it to delay for short periods of time, used to allow memory to “settle” or for ARM writes to take place fully when critical values are read from sound memory.

acSystemEnableArmInterrupts

Use to enable the ARM interrupt.

FORMAT

```
#include <ac.h>

void acSystemEnableArmInterrupts(void)
```

PARAMETERS

```
void
```

RETURN VALUE

```
void
```

FUNCTION

Enables the ARM external interrupt. The driver does not enable or disable the interrupt this allows the interrupt to be enabled\disabled in critical sections.

acSystemDisableArmInterrupts

Use to disable the ARM interrupt.

FORMAT

```
#include <ac.h>
```

```
void acSystemDisableArmInterrupts(void)
```

PARAMETERS

void

RETURN VALUE

void

FUNCTION

Disables the ARM external interrupt. The driver does not enable or disable the interrupt this allows the interrupt to be enabled\disabled in critical sections.

acSystemInit

Makes the ac system ready to use.

FORMAT

```
#include <ac.h>
```

```
KTBOOL acSystemInit(void)
```

PARAMETERS

```
void
```

RETURN VALUE

```
KTBOOL
```

FUNCTION

Makes the ac system ready to use, must be called prior to any AC lib calls.

acGetSystemFlag

True if the system has been initialized.

FORMAT

```
#include <ac.h>
```

```
KTBOOL acGetSystemFlag(void)
```

PARAMETERS

```
void
```

RETURN VALUE

KTTRUE,	If the driver has been installed and the system initialized.
Or...	
KTFALSE	If not.

FUNCTION

Returns KTTRUE if the function acInstallDriver has been run successfully.

acSystemGetFirstFreeSoundMemory Gets the address of first free sound memory.

FORMAT

```
#include <ac.h>

volatile KTU32 * acSystemGetFirstFreeSoundMemory(void)
```

PARAMETERS

void

RETURN VALUE

a pointer to the first free memory in the sound heap as obtained from the driver

FUNCTION

Gets the address of the first free memory in the sound memory area as specified by the driver

acSystemGetCommandFlag

Gets address of driver command flag register.

FORMAT

```
#include <ac.h>

volatile KTU32 * acSystemGetCommandFlag(void)
```

PARAMETERS

void

RETURN VALUE

a pointer to the command flag register

FUNCTION

Gets the command flag register address for system usage. The command flag is written after commands are placed into the drivers command queue to indicate to the driver that there are commands to be processed.

When setting the flag the value should be 0xffffffff, the driver will then start processing the command queue from its last queue position. When writing commands it is necessary to observe the state of the G2 buss FIFO to ensure that the command write has completed prior to setting the command flag.

acSystemResetArmInterrupt

Resets the ARM interrupt flag.

FORMAT

```
#include <ac.h>

void acSystemResetArmInterrupt(void)
```

PARAMETERS

void

RETURN VALUE

void

FUNCTION

Resets the ARM interrupt status flag

acSystemInstallDriver

Installs the sound driver.

FORMAT

```
#include <ac.h>
```

```
KTBOOL acSystemInstallDriver(void)
```

PARAMETERS

```
void
```

RETURN VALUE

KTBOOL, KTTRUE if the driver was successfully installed and started

FUNCTION

Installs the AICA driver image and sets the system data structure.

3. The AICA Manager API

amBankFetchMidiUspqn

Fetches uspqn from a MIDI type asset.

FORMAT

```
#include <am.h>
```

```
KTBOOL    amBankFetchMidiUspqn(AM_BANK_PTR theBank,KTU32 assetNumber,KTU32 *uspqn)
```

PARAMETERS

AM_BANK_PTR theBank,

A pointer to a .kat bank.

KTU32 assetNumber,

The number of the asset.

KTU32 *uspqn,

The the microseconds pqn is returned via this pointer.

RETURN VALUE

KTTRUE,

on success

KTFALSE,

theBank is NULL,

uspqn is NULL,

assetNumber is not in this bank

assetNumber is not a MIDI asset

FUNCTION

Fetches the microseconds per quarter note (uspqn) of a midi asset in a bank file.

amBankFetchMidiLoop

Fetches the loop flag from a MIDI type asset.

FORMAT

```
#include <am.h>
KTBOOL amBankFetchMidiLoop(AM_BANK_PTR theBank,KTU32 assetNumber,KTU32 *loop)
```

PARAMETERS

AM_BANK_PTR theBank,	A pointer to a .kat bank.
KTU32 assetNumber,	The number of the asset.
KTU32 *loop,	The loop flag is returned via this pointer.

RETURN VALUE

KTTRUE,	on success
KTFALSE,	theBank is NULL, loop is NULL, assetNumber is not in this bank assetNumber is not a MIDI asset

FUNCTION

Fetches the loop flag of a midi asset in a bank file.

amBankFetchMidiPpqn

Fetches ppqn from a MIDI type katbank asset.

FORMAT

```
#include <am.h>
KTBOOL  amBankFetchMidiPpqn(AM_BANK_PTR theBank,KTU32 assetNumber,KTU32 *ppqn)
```

PARAMETERS

AM_BANK_PTR theBank,	A pointer to a .kat bank.
KTU32 assetNumber,	The number of the asset.
KTU32 *ppqn,	The ppqn is returned via this pointer.

RETURN VALUE

KTTRUE,	on success
KTFALSE,	theBank is NULL, ppqn is NULL, assetNumber is not in this bank assetNumber is not a MIDI asset

FUNCTION

Gets the ppqn(pulses per quarter note) from a SMF Type 0 MIDI file asset in a bank.

amBankFetchMidiVolume

Fetches master volume from a MIDI type katbank asset.

FORMAT

```
#include <am.h>
KTBOOL  amBankFetchMidiVolume(AM_BANK_PTR theBank,KTU32 assetNumber,KTU32
*masterVolume)
```

PARAMETERS

AM_BANK_PTR theBank,	A pointer to a .kat bank.
KTU32 assetNumber,	The number of the asset.
KTU32 *masterVolume,	The master volume of the asset is returned via this pointer.

RETURN VALUE

KTTRUE,	on success
KTFALSE,	theBank is NULL, masterVolume is NULL, assetNumber is not in this bank assetNumber is not a MIDI asset

FUNCTION

Fetches the master volume setting from a MIDI type katbank asset. This setting is set in the katbank build script file (.oss) via the "Volume" tag and is used to set the overall starting volume of a MIDI sequence. This allows the volumes of the sequences used in a game to be balanced against each other.

amBankFetchMidiGmModeFlag Fetches GM mode flag from a MIDI type katbank asset.

FORMAT

```
#include <am.h>
KTBOOL  amBankFetchMidiGmModeFlag(AM_BANK_PTR theBank,KTU32 assetNumber,KTU32
*gmModeFlag)
```

PARAMETERS

AM_BANK_PTR theBank,	A pointer to a .kat bank.
KTU32 assetNumber,	The number of the asset.
KTU32 *gmModeFlag,	The GM mode of the asset is returned via this pointer.

RETURN VALUE

KTTRUE,	on success
KTFALSE,	theBank is NULL, gmModeFlag is NULL, assetNumber is not in this bank assetNumber is not a MIDI asset

FUNCTION

This fetches the value set via the GmMode tag in the katbank build script file. This should be set to 1 if it is a GM sequence or 0 if it is not.

amBankLoad

Loads a `katbank` asset from disk into sound memory.

FORMAT

```
#include <am.h>
KTBOOL amBankLoad(KTSTRING fileName,AM_BANK_PTR buffer)
```

PARAMETERS

<code>KTSTRING fileName,</code>	The filename and path of the bank to load.
<code>AM_BANK_PTR buffer,</code>	A 32 byte aligned buffer in sound memory big enough to hold the asset.

RETURN VALUE

<code>KTTRUE,</code>	on success
<code>KTFALSE,</code>	fileName is NULL, File not found buffer is NULL, buffer is not 32 byte aligned.

FUNCTION

Loads a `katbank` asset from disk into sound memory. This calls the redirectable file system (`amFile...`) to do the loading operation.

amBankFetchAssetParameters

Fetches parameters from any `katbank` asset.

FORMAT

```
#include <am.h>
KTBOOL amBankFetchAssetParameters      (AM_BANK_PTR theBank,
KTU32 assetNumber,
AM_BANK_FILE_UNION_PTR parameters
)
```

PARAMETERS

<code>AM_BANK_PTR theBank,</code>	A pointer to a <code>.kat</code> bank.
<code>KTU32 assetNumber,</code>	The number of the asset.
<code>AM_BANK_FILE_UNION_PTR parameters</code>	The parameter block is returned via this pointer.

RETURN VALUE

<code>KTTRUE,</code>	on success
<code>KTFALSE,</code>	<code>theBank</code> is NULL, <code>parameters</code> is NULL, <code>assetNumber</code> is not in this bank

FUNCTION

This will fetch the parameter block from any type of `katbank` asset.

amBankFetchWaveLoopFlag

Fetches the loop flag from a katbank asset.

FORMAT

```
#include <am.h>
KTBOOL  amBankFetchWaveLoopFlag(AM_BANK_PTR theBank,KTU32 assetNumber,KTBOOL
*loopFlag)
```

PARAMETERS

AM_BANK_PTR theBank,	A pointer to a .kat bank.
KTU32 assetNumber,	The number of the asset.
KTBOOL *loopFlag,	The loop flag value is returned via this pointer.

RETURN VALUE

KTTRUE,	on success
KTFALSE,	theBank is NULL, loopFlag is NULL, assetNumber is not in this bank assetNumber is not a MIDI asset

FUNCTION

Fetches the loop flag from a WAVE type katbank asset. The loop flag is set in the katbank build script via the “Loop” tag. If the wave is to loop the value is set to 1 if not it is set to 0.

amBankFetchWaveRandomPitch Fetches random pitch amount from a katbank asset.

FORMAT

```
#include <am.h>
KTBOOL  amBankFetchWaveRandomPitch(AM_BANK_PTR theBank,KTU32 assetNumber,KTU32
*randomPitchAmount)
```

PARAMETERS

AM_BANK_PTR theBank,	A pointer to a .kat bank.
KTU32 assetNumber,	The number of the asset.
KTBOOL *randomPitchAmount,	The random pitch amount is returned via this pointer.

RETURN VALUE

KTTRUE,	on success
KTFALSE,	theBank is NULL, randomPitchAmount is NULL, assetNumber is not in this bank assetNumber is not a MIDI asset

FUNCTION

Fetches the random pitch amount from a WAVE type katbank asset. This amount will be applied as a random percentage of change from the root pitch of the sound when it is played using the amSound... interface.

amBankFetchWaveSampleRate

Fetches the sample rate from a katbank WAVE asset.

FORMAT

```
#include <am.h>
KTBOOL amBankFetchWaveSampleRate(AM_BANK_PTR theBank,KTU32 assetNumber,KTU32
*sampleRate)
```

PARAMETERS

AM_BANK_PTR theBank,	A pointer to a .kat bank.
KTU32 assetNumber,	The number of the asset.
KTBOOL *sampleRate,	The real world sample rate is returned via this pointer.

RETURN VALUE

KTTRUE,	on success
KTFALSE,	theBank is NULL, sampleRate is NULL, assetNumber is not in this bank assetNumber is not a MIDI asset

FUNCTION

Fetches the sample rate from a katbank WAVE asset. This is the real world sample rate number that is set in the katbank build script (.oss) using the “SampleRate” tag.

amBankFetchWaveBitDepth Fetches the bit depth of a WAVE type asset in a katbank.

FORMAT

```
#include <am.h>
KTBOOL amBankFetchWaveBitDepth(AM_BANK_PTR theBank,KTU32 assetNumber,KTU32 *bitDepth)
```

PARAMETERS

AM_BANK_PTR theBank,	A pointer to a .kat bank.
KTU32 assetNumber,	The number of the asset.
KTBOOL *bitDepth,	The bit depth is returned via this pointer.

RETURN VALUE

KTTRUE,	on success
KTFALSE,	theBank is NULL, bitDepth is NULL, assetNumber is not in this bank assetNumber is not a MIDI asset

FUNCTION

Fetches the bit depth of a WAVE type asset in a katbank.

amBankFetchUnknownParameters

Fetches one of the 7 user parameters from a katbank "unknown" type asset.

FORMAT

```
#include <am.h>
KTBOOL amBankFetchUnknownParameters(AM_BANK_PTR theBank,
KTU32 assetNumber,
KTU32 parameterNumber,
KTS32 *parameterValue
)
```

PARAMETERS

AM_BANK_PTR theBank,	A pointer to a .kat bank.
KTU32 assetNumber,	The number of the asset.
KTU32 parameterNumber,	The parameter to fetch (0-7)
KTBOOL *parameterValue,	The parameter value is returned via this pointer.

RETURN VALUE

KTTRUE,	on success
KTFALSE,	theBank is NULL, parameterValue is NULL, assetNumber is not in this bank, parameterNumber is out of range, assetNumber is not a UNKNOWN asset

FUNCTION

Fetches one of the seven user parameters from a katbank asset. These parameters are defined in the katbank build script using the Parameter0 to Parameter7 tags.

amBankFetchAsset

Fetches an asset from a katbank.

FORMAT

```
#include <am.h>
KTBOOL amBankFetchAsset(
    AM_BANK_FILE_UNION_PTR parameters,
    KTU32 assetNumber,
    KTU32 **theAsset,
    KTU32 *assetSize
)
```

PARAMETERS

AM_BANK_PTR theBank,	A pointer to a .kat bank.
AM_BANK_FILE_UNION_PTR parameters,	The parameter block is returned via this pointer.
KTU32 assetNumber,	The number of the asset.
KTU32 **theAsset,	A pointer to the asset is returned via this handle.
KTU32 *assetSize	The assets size is returned via this pointer.

RETURN VALUE

KTTRUE,	on success
KTFALSE,	theBank is NULL, parameters is NULL, assetSize is NULL, theAsset is NULL, assetNumber is not in this bank,

FUNCTION

Fetches an asset from a katbank aggregation. Returns the size, parameters and a pointer to data via the arguments.

amBankGetAssetSize

Gets the size of an asset from a katbank.

FORMAT

```
#include <am.h>
KTBOOL amBankGetAssetSize(AM_BANK_PTR theBank,KTU32 assetNumber,KTU32 *assetSize)
```

PARAMETERS

AM_BANK_PTR theBank,	A pointer to a .kat bank.
KTU32 assetNumber,	The number of the asset.
KTU32 *assetSize,	The size of the asset is returned via this pointer.

RETURN VALUE

KTTRUE,	on success
KTFALSE,	theBank is NULL, assetSize is NULL, assetNumber is not in this bank,

FUNCTION

Fetches the size of an asset from a katbank.

amBankGetNumberOfAssets

Gets the number of assets in a katbank.

FORMAT

```
#include <am.h>
KTBOOL amBankGetNumberOfAssets(AM_BANK_PTR theBank,KTU32 *numberOfAssets)
```

PARAMETERS

KTU8 *theBank,	A pointer to either the header from a bank file or an entire bank file.
KTU32 *numberOfAssets,	The number of assets in the katbank is returned via this pointer.

RETURN VALUE

KTTRUE,	on success
KTFALSE,	theBank is NULL, assetSize is NULL, assetNumber is not in this bank,

FUNCTION

Gets the number of assets in a katbank file.

amBankGetHeaderSize

Gets the size of the header portion of a `katbank`.

FORMAT

```
#include <am.h>
KTBOOL amBankGetHeaderSize(AM_BANK_PTR theBank,KTU32 *headerSize)
```

PARAMETERS

KTU8 *theBank,	A pointer to either the header from a bank file or an entire bank file.
KTU32 *headerSize,	The size of the katbank header is returned via this pointer.

RETURN VALUE

KTRUE,	on success
KFALSE,	theBank is NULL, headerSize is NULL, assetNumber is not in this bank,

FUNCTION

Gets the size of the header portion of a `katbank`.

amDmaMemCpy

Performs DMA copys to sound memory.

FORMAT

```
#include <am.h>
KTBOOL amDmaMemCpy(KTU32 *target, KTU32 *source, KTU32 size,KTU32 bytesPerTransfer,KTU32
dmaChannel)
```

PARAMETERS

KTU32 *target,	The target buffer, must be large enough to hold size bytes.
KTU32 *source,	The source buffer
KTU32 size,	The number of bytes to transfer
KTU32 bytesPerTransfer,	The number of bytes to transfer in one DMA frame
KTU32 dmaChannel	AM_DMA_CHANNEL only for now.

RETURN VALUE

KTTRUE,	On success
KTFALSE,	target or source is NULL, size is 0 bytesPerTransfer is not 1,2,4,8 or 32 dmaChannel is not AM_DMA_CHANNEL

FUNCTION

Note: This is not implemented in R8, the function will simply return false with an AC error condition.

Copies memory from one place to the other starting at the bottom of the block. The source target and size must be multiples of bytesPerTransfer or failure will result. The transfer is made in burst mode rather then cycle steal mode as timeliness is important to streaming audio processes.

amDspFetchProgramBank

Fetches and installs a DSP program bank from a KatBank asset.

FORMAT

```
#include <am.h>
KTBOOL amDspFetchProgramBank(AM_BANK_PTR theBank,KTU32 assetNumber)
```

PARAMETERS

AM_BANK_PTR theBank,	A pointer to a .kat bank.
KTU32 assetNumber,	The number of the asset.

RETURN VALUE

KTRUE,	On success
KFALSE,	theBank is NULL, assetNumber is not in this bank, unable to send driver command.

FUNCTION

Fetches and installs a DSP program bank from a KatBank asset.

amDspFetchOutputBank Fetches and installs a DSP output bank from a `KatBank` asset.

FORMAT

```
#include <am.h>
KTBOOL amDspFetchOutputBank(AM_BANK_PTR theBank,KTU32 assetNumber)
```

PARAMETERS

<code>AM_BANK_PTR theBank,</code>	A pointer to a <code>.kat</code> bank.
<code>KTU32 assetNumber,</code>	The number of the asset.

RETURN VALUE

<code>KTTRUE,</code>	On success
<code>KTFALSE,</code>	<code>theBank</code> is NULL, <code>assetNumber</code> is not in this bank, unable to send driver command

FUNCTION

Fetches and installs a DSP output bank from a `KatBank` asset.

amErrorGetLast

Gets a pointer to the error structure.

FORMAT

```
#include <ac.h>
AC_ERROR_PTR amErrorGetLast(void)
```

PARAMETERS

void

RETURN VALUE

AC_ERROR_STRUCT a pointer to the AM error structure.

FUNCTION

Gets a pointer to the AM error structure. This contains an error number enumerated as an AC_ERROR_TYPE in ac.h and a more informative error message that tells the name of the function that failed as well as some descriptive text regarding the cause of the failure.

amErrorExists

Checks to see if an error condition exists.

FORMAT

```
#include <ac.h>
KTBOOL amErrorExists(void)
```

PARAMETERS

void

RETURN VALUE

KTTRUE	if a error exists
KTFALSE	if no error exists.

FUNCTION

Allows checking of the error state for the AM layer in a single call returning a bool.

amErrorClear

Clears the AM error structure.

FORMAT

```
#include <ac.h>
void amErrorClear(void)
```

PARAMETERS

RETURN VALUE

KTTRUE	if successful
KTFALSE	if unable to send command or interruptId is out of range (0-255).

FUNCTION

Clears the AM Error structure.

amHeapShutdown

Shuts down the AM heap management system.

FORMAT

```
#include <am.h>
void amHeapShutdown(void)
```

PARAMETERS

void

RETURN VALUE

void

FUNCTION

Shuts down the AM heap management system.

amHeapGetInfo

Gets info necessary to start an audio heap.

FORMAT

```
#include <am.h>
KTBOOL amHeapGetInfo(volatile KTU32 **freeSoundMemory,KTU32 *size)
```

PARAMETERS

volatile KTU32 **freeSoundMemory,	The pointer to the first free sound memory is returned via this handle.
KTU32 *size	The size of the free portion of sound memory.

RETURN VALUE

KTTRUE,	on success
KTFALSE,	freeSoundMemory is NULL
	size is NULL
	the sound driver has not been successfully installed

FUNCTION

Gets the necessary information for the amHeapInit() call from the sound driver.

Note: The driver must have been successfully installed prior to this call.

amHeapGetFree

Gets the amount of free memory.

FORMAT

```
#include <am.h>
KTBOOL  amHeapGetFree(KTU32 *freeMemory)
```

PARAMETERS

KTU32 *freeMemory,	The amount of free memory is returned via this pointer.
--------------------	---

RETURN VALUE

KTTRUE,	On success.
KTFALSE,	If the heap has not been initialized, freeMemory is NULL.

FUNCTION

Gets the amount of free memory remaining in the heap.

amHeapAlloc

Allocates aligned memory from the audio heap.

FORMAT

```
#include <am.h>
KTBOOL amHeapAlloc(
volatile KTU32 **buffer,
KTU32 size,KTU32 alignment,
AM_HEAP_MEMORY_TYPE memoryType,
AM_MEMORY_CALLBACK callback
)
```

PARAMETERS

volatile KTU32 **buffer,	A pointer to the block of memory is returned via this handle.
KTU32 size,KTU32 alignment,	The desired alignment for the block (4 or 32)
AM_HEAP_MEMORY_TYPE memoryType,	The type of memory desired AM_FIXED_MEMORY or AM_PURGABLE_MEMORY.
AM_MEMORY_CALLBACK callback,	A pointer to a callback function for the memory

RETURN VALUE

KTTRUE	if the operation was successful
KTFALSE,	buffer is NULL, size is 0 size exceeds available free memory alignment is not 4 or 32 memoryType is not AM_FIXED_MEMORY or AM_PURGABLE_MEMORY

FUNCTION

Allocates aligned memory from the audio heap zone. The memory can be allocated in alignments of either 4 or 32 bytes. Non DWORD aligned writes to the audio memory area are illegal and will corrupt the audio memory area severely. If the type is AM_FIXED_MEMORY the blocks will be allocated from the top of the heap progressing downwards, if the type is AM_PURGABLE_MEMORY the blocks are allocated from the bottom of the heap progressing upwards.

There is a variable amount of block overhead, this is applied as a fixed amount of $((\text{alignment}-1) * 2) + 4$ when the parameters are tested so it is not possible to call for the amount of free memory remaining and allocate all of it. Depending on the alignment value the maximum allocation would be: alignment=4, maxMem - 10; or alignment=32, maxMem-66; The callback function will be invoked when the block is either purged or freed. The argument of the function is the address of the block that owned the callback.

Prototype for callback: void MyCallback(KTU32 blockAddress)

Note: All GD file system calls currently require that the buffer be aligned on a 32 byte boundry.

This may only be called post a successful call to amHeapInit()

amHeapGetMaxPurgable

Gets amount of memory available from a full purge.

FORMAT

```
#include <am.h>
KTBOOL amHeapGetMaxPurgable(KTU32 *maxPurgable)
```

PARAMETERS

KTU32 *maxPurgable,	The free memory size is returned via this pointer.
---------------------	--

RETURN VALUE

KTTRUE,	on success
KTFALSE,	heap is not initialized maxPurgable is NULL

FUNCTION

Gets the amount of memory available from the free memory pool + all AM_PURGABLE_MEMORY type blocks. This amount of memory is only available if a call is made to the function `amHeapClear(AM_PURGABLE_MEMORY)` or a call to `amHeapPurge(sizeNeeded)`.

amHeapPurge

Purges memory marked as purgable.

FORMAT

```
#include <am.h>
KTBOOL amHeapPurge(KTU32 sizeNeeded)
```

PARAMETERS

KTU32 sizeNeeded,	The size of the block of memory needed.
-------------------	---

RETURN VALUE

KTTRUE,	If the memory is now available.
KTFALSE	If the heap has not been initialized, sizeNeeded is 0, sizeNeeded exceeds free + purgable,

FUNCTION

Will purge (if necessary) blocks of purgable memory in a top down fashion until sufficient memory is available to fill the requested size. If there is sufficient free memory to fill the request the function returns KTTRUE and does nothing. When a block is purged its callback (if installed) is invoked. This returns the address of the block to the application.

This function will not alter blocks of memory allocated as AM_FIXED_MEMORY.

amHeapFree

Frees purgable memory allocated using amHeapAlloc()

FORMAT

```
#include <am.h>
KTBOOL amHeapFree(volatile KTU32 *buffer)
```

PARAMETERS

volatile KTU32 *buffer,

A pointer to the buffer to be freed.

RETURN VALUE

KTTRUE,
KTFALSE,

On success
If buffer is NULL,
buffer does not point to an allocated block
buffer is not the highest address allocated in purgable
memory.

FUNCTION

This will free purgable memory from the top down by block address. If there is a block allocated with a higher address the call will fail, this prevents fragmentation. On freeing a block, if the block has a callback, it will be executed.

amHeapInit

Initializes the audio heap.

FORMAT

```
#include <am.h>
KTBOOL amHeapInit(volatile KTU32 *memoryPool,KTU32 size)
```

PARAMETERS

volatile KTU32 *memoryPool,	The start of the audio heap zone
KTU32 size,	The size of the heap

RETURN VALUE

KTRUE	If the operation was successful
KFALSE,	If memoryPool is NULL, size is 0, heap is already open,

FUNCTION

Initializes the heaps data structures

Note: A warning will be issued if size is not a multiple of 4, in this case size will be rounded down to the next multiple of 4.

amHeapCheck

Checks the MCB fingerprints for overwrites.

FORMAT

```
#include <am.h>
KTBOOL amHeapCheck(void)
```

PARAMETERS

void

RETURN VALUE

KTTRUE,	If the heap fingerprints are intact
KTFALSE,	If the fingerprints are corrupted or the heap is not open.

FUNCTION

Checks the MCB fingerprints in the heap to detect overwrites in that memory zone. Use this liberally to detect corruption or its possibility it will disappear in non-DEBUG versions.

Note: This is a MACRO that is expanded to the heap check function if DEBUG is defined.

If DEBUG is not defined it will become `((void)0);` a null statement.

amInitSelectDriver

Selects driver to be installed by amInit()

FORMAT

```
#include <am.h>
KTBOOL amInitSelectDriver(AM_DRIVER_TYPE driverType)
```

PARAMETERS

AM_DRIVER_TYPE driverType,	Either AM_DA_DRIVER or AM_MIDI_DRIVER.
----------------------------	--

RETURN VALUE

KTTRUE	on success
KTFALSE on fail,	Driver is already installed or bad arg for driverType.

FUNCTION

Allows selection of the type of driver to be loaded by the amInit () call. The default driver is the audio64 driver so if this call is not made the system will be set up as audio64.

Note: This must be called **PRIOR** to the call to amInit ().

amShutdown

Shuts down the AM audio subsystem.

FORMAT

```
#include <am.h>
void amShutdown(void)
```

PARAMETERS

void

RETURN VALUE

void

FUNCTION

Shuts down the AM audio subsystem by stopping all sounds and closing their voice channels, releasing all OS service vectors and closing the amHeap subsystem.

amInit

Starts up the AM audio subsystem.

FORMAT

```
#include <am.h>
KTBOOL amInit(void)
```

PARAMETERS

void

RETURN VALUE

KTTRUE	on success
KTFALSE	on fail, Driver file not found Driver startup fail

FUNCTION

Starts up the AM audio subsystem. This will load the driver into the middle of the audio heap then install that image using `acInstallDriver`. It then starts up the am interrupt and heap management systems. This also calls `acCdInit()` to initialize the redbook playback mechanism.

amFileRewind

Seeks to the start of a file.

FORMAT

```
#include <am.h>
KTBOOL amFileRewind(ACFILE fd)
```

PARAMETERS

ACFILE fd,	A GD system file descriptor
------------	-----------------------------

RETURN VALUE

KTBOOL, KTTRUE on success,	KTFALSE on fail
----------------------------	-----------------

FUNCTION

Seeks to the head (byte 0) of the file. This operates through the am lib IO shell and is redirectable to the applications file system.

See: `amFileInstallAlternateIoManager()`

An example of this redirection is available in `MyFile.c` as well as a boilerplate copy of the IO proc for modification.

amFileLoad

Loads specified file into the buffer.

FORMAT

```
#include <am.h>
KTBOOL amFileLoad(KTSTRING fileName,KTU8 * buffer)
```

PARAMETERS

KTSTRING fileName,	The name of the file to load
KTU8 * buffer,	A buffer large enough to hold the file

RETURN VALUE

KTBOOL,	KTTRUE on success, KTFALSE on fail
---------	---------------------------------------

FUNCTION

Loads a file given the file name and a buffer to load it into. This operates through the am lib IO shell and is redirectable to the applications file system.

See: `amFileInstallAlternateIoManager()`

An example of this redirection is available in `MyFile.c` as well as a boilerplate copy of the IO proc for modification.

amFileRead

Reads from a file that is already open.

FORMAT

```
#include <am.h>
KTBOOL amFileRead(ACFILE fd,KTU8 * buffer,KTU32 size)
```

PARAMETERS

ACFILE fd,	A GD system file descriptor
KTU8 * buffer,	A pointer to a buffer into which to read
KTU32 size,	The size of the data to be read

RETURN VALUE

KTBOOL,	KTTRUE on success,
	KTFALSE on fail

FUNCTION

Reads from an open file. This operates through the am lib IO shell and is redirectable to the applications file system.

See: `amFileInstallAlternateIoManager()`

An example of this redirection is available in `MyFile.c` as well as a boilerplate copy of the IO proc for modification.

amFileOpen

Opens a file for reading.

FORMAT

```
#include <am.h>
KTBOOL amFileOpen(KTSTRING fileName,ACFILE *fd)
```

PARAMETERS

KTSTRING fileName,	The name of the file to load
ACFILE fd,	A GD system file descriptor

RETURN VALUE

KTBOOL,	KTTRUE on success, KTFALSE on fail
---------	---------------------------------------

FUNCTION

Loads a file given the file name and a buffer to load it into. This operates through the am lib IO shell and is redirectable to the applications file system.

See: amFileInstallAlternateIoManager()

An example of this redirection is available in `MyFile.c` as well as a boilerplate copy of the IO proc for modification.

amFileClose

Closes a file.

FORMAT

```
#include <am.h>
KTBOOL amFileClose(ACFILE fd)
```

PARAMETERS

ACFILE fd,	A GD system file descriptor
------------	-----------------------------

RETURN VALUE

KTBOOL,	KTTRUE on success, KTFALSE on fail
---------	---------------------------------------

FUNCTION

Closes a file. This operates through the am lib IO shell and is redirectable to the applications file system.

See: `amFileInstallAlternateIoManager()`

An example of this redirection is available in `MyFile.c` as well as a boilerplate copy of the IO proc for modification.

amFileGetSize

Gets the size of a file.

FORMAT

```
#include <am.h>
KTBOOL amFileGetSize(KTSTRING fileName, KTU32 * size)
```

PARAMETERS

KTSTRING fileName,	The name of the file to load
KTU32 * size,	The size of the asset is returned via this pointer.

RETURN VALUE

KTBOOL,	KTTRUE on success, KTFALSE on fail
---------	---------------------------------------

FUNCTION

Gets the size of a file. This operates through the am lib IO shell and is redirectable to the applications file system.

See: `amFileInstallAlternateIoManager()`

An example of this redirection is available in `MyFile.c` as well as a boilerplate copy of the IO proc for modification.

amFileInstallAlternateIoManager

Installs a custom Io proc.

FORMAT

```
#include <am.h>
void amFileInstallAlternateIoManager(AM_IO_PROC ioProc)
```

PARAMETERS

AM_IO_PROC ioProc,	A pointer to a custom Io proc, see the example in <code>MyFile.c</code>
--------------------	---

RETURN VALUE

void

FUNCTION

Installs a custom Io proc into the Io shell, this allows all file system calls to be intercepted by the applications file system.

The prototype for the IO proc is as follows:

```
KTBOOL MyCustomIoProc(KTSTRING fileName,
ACFILE * fd,
KTU8 * buffer,
KTU32 * size,
AM_FILE_OPERATION_MODE mode
)
```

An example of this redirection is available in `MyFile.c` as well as a boilerplate copy of the IO proc for modification.

amStreamIsr0 - 4

Interrupt Service Routine for the amStream subsystem.

FORMAT

```
#include <am.h>
void _amStreamIsr0(KTU32 streamPtr)
void _amStreamIsr1(KTU32 streamPtr)
void _amStreamIsr2(KTU32 streamPtr)
void _amStreamIsr3(KTU32 streamPtr)
void _amStreamIsr4(KTU32 streamPtr)
```

PARAMETERS

RETURN VALUE

KTTRUE,	On success
KTFALSE,	

FUNCTION

ISR routine for the amStream subsystem. These routines are used as the theIsr argument to the amStreamSetIsr() call.

See Also: KTBOOL amStreamSetIsr(AM_STREAM_PTR theStream,AM_STREAM_ISR theIsr)

amMemSh4Alloc

Sh4 memory allocation shell.

FORMAT

```
#include <am.h>
KTBOOL amMemSh4Alloc(volatile KTU32 ** base,
volatile KTU32 ** aligned,
KTU32 size,
KTU32 alignment)
```

PARAMETERS

volatile KTU32 ** base,	an unaligned pointer to the block allocated
volatile KTU32 ** aligned, address	a pointer to the first aligned address after the base
KTU32 size,KTU32 alignment,	the alignment desired

RETURN VALUE

KTTRUE	if a block was successfully allocated
KTFALSE	if insufficient memory available.

FUNCTION

This is a shell that verifies a proc pointer then calls it to invoke whatever malloc proc is currently installed.

Note: Neither the AM nor AC layers allocate or free SH4 memory.

amMemSh4Free

Sh4 memory free shell.

FORMAT

```
#include <am.h>
void amMemSh4Free(volatile KTU32 * block)
```

PARAMETERS

volatile KTU32 * block,	a pointer to the unaligned base address of the block to be freed
-------------------------	--

RETURN VALUE

void

FUNCTION

This is a shell that verifies a proc pointer then calls it to invoke whatever free proc is currently installed.

Note: Neither the AM nor AC layers allocate or free SH4 memory.

amMemInit

Initializes the Sh4 memory shell system.

FORMAT

```
#include <am.h>
void amMemInit(void)
```

PARAMETERS

void

RETURN VALUE

void

FUNCTION

Initializes the memory manager shell proc pointers with the default routines if they have not been previously initialized. Called by `amInit()`

Note: Neither the AM nor AC layers allocate or free SH4 memory.

amMemInstallAlternateMemoryManager Allows redirection of sh4 memory requests.

FORMAT

```
#include <am.h>
void amMemInstallAlternateMemoryManager(AM_SH4_ALLOC_PROC allocProc,AM_SH4_FREE_PROC
freeProc)
```

PARAMETERS

AM_SH4_ALLOC_PROC allocProc,	a pointer to an correctly prototyped malloc proc
AM_SH4_FREE_PROC freeProc,	a pointer to an correctly prototyped free proc

RETURN VALUE

void

FUNCTION

Initializes the malloc and free proc pointers in the audio engines memory allocation shell.

Note: This **MUST** be called prior to the call to `amInit()`

Note: Neither the AM nor AC layers allocate or free SH4 memory.

amMidiSetTempo

Sets the tempo of a MIDI sequence.

FORMAT

```
#include <am.h>
KTBOOL amMidiSetTempo(AM_SEQUENCE_PTR theSequence, KTS32 percentOfChange)
```

PARAMETERS

AM_SEQUENCE_PTR theSequence,	A pointer to an AM_SEQUENCE object.
KTS32 percentOfChange,	The percent of change over or under the root tempo.

RETURN VALUE

KTTRUE, on success
or...
KTFALSE on fail, theSequence is NULL.

FUNCTION

Changes the tempo of a currently playing midi sequence to the new tempo. This is expressed as a percentage of change from the root (original) tempo. i.e. the tempo of the file is 120, a +10% change is applied, the sequence is now playing at tempo 132. If a change of 0 is specified the sequence will play at its root tempo.

Note: This group of functions will only work with the MidiDa driver, they will not work with the Audio64 driver.

amMidiSetLoopFlag

Sets the loop flag on a MIDI sequence.

FORMAT

```
#include <am.h>
KTBOOL amMidiSetLoopFlag(AM_SEQUENCE_PTR theSequence,KTBOOL onOrOff)
```

PARAMETERS

AM_SEQUENCE_PTR theSequence, a pointer to an AM_SEQUENCE object.

KTBOOL onOrOff, KTTRUE to loop, KTFALSE to not.

RETURN VALUE

KTTRUE, on success

or...

KTFALSE on fail, bad arguments, theSequence is NULL.

FUNCTION

Sets the loop flag in an AM_SEQUENCE object.

Note: if onOrOff is out of range it will be set to KTTRUE.

Note: This group of functions will only work with the MidiDa driver, they will not work with the Audio64 driver.

amMidiFetchToneBank

Installs an MTB asset from a bank file aggregate.

FORMAT

```
#include <am.h>
KTBOOL amMidiFetchToneBank(AM_BANK_PTR theBank,KTU32 assetNumber,KTU8 toneBankSlot)
```

PARAMETERS

AM_BANK_PTR theBank,	A pointer to a .kat bank type asset aggregation.
KTU32 assetNumber,	The number of the tone bank asset in the bank.
KTU8 toneBankSlot,	The slot number of the bank 0-15

RETURN VALUE

KTTRUE, on success

or...

KTFALSE on fail, asset is wrong type, asset number not in bank, unable to post command to driver.

FUNCTION

Installs an MTB asset that is contained in a .kat bank aggregate file.

Note: This group of functions will only work with the MidiDa driver, they will not work with the Audio64 driver.

amMidiLoadToneBank

Loads a Sega tone bank asset

FORMAT

```
#include <am.h>
KTBOOL amMidiLoadToneBank(KTSTRING fileName,KTU8 gmMode,volatile KTU32 * buffer,KTU32
bankSize,KTU8 toneBankSlot)
```

PARAMETERS

KTSTRING fileName,	The name of the bank file to be loaded from the GD system.
KTU8 gmMode,	AC_GM_ON or AC_GM_OFF, enables or disables general midi mode
volatile KTU32 * buffer,	A 32 byte aligned buffer in sound memory
KTU32 bankSize,	The size of the bank to be loaded
KTU8 toneBankSlot,	The slot number of the bank 0-15

RETURN VALUE

KTTRUE, on success
or...
KTFALSE on fail, buffer not 32 byte aligned,file not found or unable to send driver command.

FUNCTION

This loads a midi tonebank made by the SOJ mac tool from the GD-ROM using the redirectable file system.

Note: If gmMode is out of range it will be set to AC_GM_ON

Note: This group of functions will only work with the MidiDa driver, they will not work with the Audio64 driver.

amMidiInstallCallback

Sets the callback proc for a sequence.

FORMAT

```
#include <am.h>
KTBOOL amMidiInstallCallback(AM_SEQUENCE_PTR theSequence, AC_MIDI_CALLBACK theCallback)
```

PARAMETERS

AM_SEQUENCE_PTR theSequence,	A properly initialized sequence object.
AC_MIDI_CALLBACK theCallback,	The callback proc.

RETURN VALUE

KTTRUE, on success	
or...	
KTFALSE on fail,	unable to send command to driver theSequence is NULL

FUNCTION

Sets the callback proc for a sequence.

The voice channel number is returned to the callback, however, please note that this is not the same as the midiPort number. The midiPort number is 16 less then the voice channel number.

The format of the callback is:

```
void MyCallbackProc(KTU32 voiceChannelNumber)
```

Note: This must be called prior to the amMidiAllocateSequencePort() and amMidiPlay() calls.

Note: This group of functions will only work with the MidiDa driver, they will not work with the Audio64 driver.

amMidiAllocateSequencePort

Allocates a MIDI port for the sequence.

FORMAT

```
#include <am.h>
KTBOOL amMidiAllocateSequencePort(AM_SEQUENCE_PTR theSequence)
```

PARAMETERS

AM_SEQUENCE_PTR theSequence, A properly initialized sequence object.

RETURN VALUE

KTTRUE,	on success
or...	
KTFALSE on fail,	unable to send command to driver theSequence is NULL port allocation failed (all voices busy)

FUNCTION

Allocates a MIDI port for the sequence. This calls `amVoiceAllocate()` and allocates a `AM_MIDI_VOICE` type channel. This voice channel number is the `midiPort` number + 16.

Note: This sets the user callback in the voice management system so the callback proc must be installed prior to making this call.

Note: This group of functions will only work with the `MidiDa` driver, they will not work with the `Audio64` driver.

amMidiFetchSequence

Fetches a sequence asset from a `katBank`.

FORMAT

```
#include <am.h>

KTBOOL amMidiFetchSequence(AM_SEQUENCE_PTR theSequence,KTU8 *theBank,KTU32
sequenceNumber)
```

PARAMETERS

<code>AM_SEQUENCE_PTR theSequence,</code>	A properly initialized sequence object.
<code>AM_BANK_PTR theBank,</code>	A pointer to a <code>katBank</code> in sound memory.
<code>KTU32 sequenceNumber,</code>	The bank asset number to fetch, see the <code>banks.h</code> file for bank and asset info.

RETURN VALUE

<code>KTTRUE,</code>	on success
or...	
<code>KTFALSE</code> on fail,	unable to send command to driver theSequence is NULL theBank is NULL the asset fetch failed (asset not present in bank) the requested asset was not a MIDI asset the bank header is corrupt

FUNCTION

Fetches a standard MIDI type 0 sequence asset from a `kat` type bank using the `amBank...()` API. This type of bank is manufactured with the `mkscript` and `mkbank` utilities.

Note: This group of functions will only work with the `MidiDa` driver, they will not work with the `Audio64` driver.

amMidiPlay

Plays a MIDI sequence.

FORMAT

```
#include <am.h>
KTBOOL amMidiPlay(AM_SEQUENCE_PTR theSequence)
```

PARAMETERS

AM_SEQUENCE_PTR theSequence,	A properly initialized sequence object.
------------------------------	---

RETURN VALUE

KTTRUE,	on success
or...	
KTFALSE on fail,	unable to send command to driver theSequence is NULL

FUNCTION

Plays a standard MIDI type 0 asset obtained from a kat bank using `amMidiPlayRaw()`. This type of bank is manufactured with the `mkscript` and `mkbank` utilities.

Note: This group of functions will only work with the `MidiDa` driver, they will not work with the `Audio64` driver.

amMidiPlayRaw

Plays a MIDI sequence given the basic parameters.

FORMAT

```
#include <am.h>
KTBOOL amMidiPlayRaw(KTU32 midiPort,KTU8 gmMode,KTU32 ticksPQN,KTU32 sequenceSize,
KTU32 *sequenceAddress,KTU32 midiVolume,AC_MIDI_CALLBACK callback)
```

PARAMETERS

KTU32 midiPort,	The MIDI port number (0-15)
KTU8 gmMode,	AC_GM_ON or AC_GM_OFF, enables or disables general midi mode
KTU32 ticksPQN,	The number of ticks per quarter note. (often 480)
KTU32 sequenceSize,	The size in bytes of the MIDI sequence data
KTU32 *sequenceAddress,	The address of a MIDI type 0 asset in sound memory
KTU32 midiVolume,	The MIDI volume at which to start the sequence (0-127)
AC_MIDI_CALLBACK callback,	The address of a callback proc or KTNUL for no callback

RETURN VALUE

KTRUE,	on success
or...	
KFALSE on fail,	unable to send command to driver sequenceAddress is NULL sequenceSize is 0

FUNCTION

Plays a MIDI type 0 asset in sound memory at the given volume with an optional callback that will be raised at the end of the sequences play. The voice channel number is returned to the callback however please note that this is not the same as the midiPort number. The midiPort number is 16 less then the voice channel number.

Note: If midiVolume is out of range it will be set to 127

If gmMode is out of range it will be set to AC_GM_ON

The format of the callback is:

```
void MyCallbackProc(KTU32 voiceChannelNumber)
```

Note: This group of functions will only work with the MidiDa driver, they will not work with the Audio64 driver.

amMidiStop

Stops a currently playing MIDI sequence.

FORMAT

```
#include <am.h>
KTBOOL amMidiStop(AM_SEQUENCE_PTR theSequence)
```

PARAMETERS

AM_SEQUENCE_PTR theSequence,	A properly initialized sequence object.
------------------------------	---

RETURN VALUE

KTTRUE,	on success
or...	
KTFALSE on fail,	unable to send command to driver theSequence is NULL

FUNCTION

This call stops a currently playing standard MIDI type 0 sequence. This releases the midi port back to the voice pool post this call another call must be made to `amMidiAllocateSequencePort()` to acquire a new midi port for playback. The callback, if one has been set using `amMidiInstallCallback()`, is still in place.

Note: This group of functions will only work with the `MidiDa` driver, they will not work with the `Audio64` driver.

amMidiSetVolume

Sets the master volume of a MIDI sequence.

FORMAT

```
#include <am.h>
KTBOOL amMidiSetVolume(AM_SEQUENCE_PTR theSequence,KTU32 newAicaVolume)
```

PARAMETERS

AM_SEQUENCE_PTR theSequence, A properly initialized sequence object.
KTU32 newMidiVolume, the MIDI volume for the port master (0-127).

RETURN VALUE

KTTRUE, on success
or...
KTFALSE on fail, unable to send command to driver
 theSequence is NULL

FUNCTION

This call sets the MASTER volume of a MIDI sequence. The MASTER volume is the overall volume of the sequence as opposed to the CHANNEL volume which would affect only one of the 16 possible MIDI channels in the sequence.

If the newMidiVolume value is out of range it will be set to 127

Note: This group of functions will only work with the MidiDa driver, they will not work with the Audio64 driver.

amMidiPause

Pauses a currently playing MIDI sequence.

FORMAT

```
#include <am.h>
KTBOOL amMidiPause(AM_SEQUENCE_PTR theSequence)
```

PARAMETERS

AM_SEQUENCE_PTR theSequence,	A properly initialized sequence object.
------------------------------	---

RETURN VALUE

KTTRUE,	on success
or...	
KTFALSE on fail,	unable to send command to driver
	theSequence is NULL

FUNCTION

Pauses a currently playing MIDI sequence. This will silence all currently sounding notes.

Note: This group of functions will only work with the MidiDa driver, they will not work with the Audio64 driver.

amMidiResume

Resumes playback of a paused MIDI sequence.

FORMAT

```
#include <am.h>
KTBOOL amMidiResume(AM_SEQUENCE_PTR theSequence)
```

PARAMETERS

AM_SEQUENCE_PTR theSequence,	A properly initialized sequence object.
------------------------------	---

RETURN VALUE

KTTRUE,	on success
or...	
KTFALSE on fail,	unable to send command to driver
	theSequence is NULL

FUNCTION

Resumes playback of a previously paused MIDI sequence.

Note: This group of functions will only work with the MidiDa driver, they will not work with the Audio64 driver.

amMidiTransferToneBank

Transfers a Sega tone bank to sound memory and sets it as the current bank.

FORMAT

```
#include <am.h>
KTBOOL amMidiTransferToneBank(volatile KTU32 *destination,KTU32 *source,KTU8 gmMode,KTU32
bankSize,KTU8 toneBankSlot)
```

PARAMETERS

volatile KTU32 * destination,	A dword aligned buffer in sound memory.
KTU32 *source,	A buffer that contains the bank to be transferred.
KTU8 gmMode,	AC_GM_ON or AC_GM_OFF, enables or disables general midi mode
KTU32 bankSize,	The size of the bank.
KTU8 toneBankSlot,	The slot number of the bank 0-15

RETURN VALUE

KTTRUE,	on success
Or...	
KTFALSE on fail,	destination is not 32 byte aligned, unable to send driver command

FUNCTION

This transfers a midi tonebank made by the SOJ mac tool from any memory to sound memory and sets it as the current bank.

Note: If gmMode is out of range it will be set to AC_GM_ON

Note: This group of functions will only work with the MidiDa driver, they will not work with the Audio64 driver.

amMidiSetChannelProgram

Sets the current bank slot.

FORMAT

```
#include <am.h>
KTBOOL amMidiSetChannelProgram(KTU32 midiPort,KTU32 midiChannel,KTU32
midiProgramNumber)
```

PARAMETERS

KTU32 midiPort,	The MIDI port number 0-15
KTU32 midiChannel,	The MIDI channel number 1-16
KTU8 midiProgramNumber,	The slot number of the program to be played for the midi channel

RETURN VALUE

KTTRUE, on success
or...
KTFALSE on fail, unable to send command to driver

FUNCTION

Prior to playing a sound effect from a midi bank the bank slot must be made the current bank slot this allows the setting of a current bank for a given portchannel configuration.

Note: This group of functions will only work with the MidiDa driver, they will not work with the Audio64 driver.

amMidiNoteOn

Plays a MIDI triggered sound effect.

FORMAT

```
KTBOOL amMidiNoteOn(KTU32 midiPort,KTU32 midiChannel,KTU8 midiNoteNumber,KTU32
midiNoteOnVelocity)
#include <am.h>
```

PARAMETERS

KTU32 midiPort,	The MIDI port number 0-15
KTU32 midiChannel,	The MIDI channel number 1-16
KTU8 midiNoteNumber,	The MIDI note number of the sound to be played. 0-127
KTU32 midiNoteOnVelocity,	The MIDI note on velocity 0-127

RETURN VALUE

KTTRUE,	on success
or...	
KTFALSE on fail,	unable to send command to driver
	midiPort out of range (0-15)
	midiChannel out of range (1-16)
	midiNoteNumber out of range (0-127)

FUNCTION

Plays a MIDI triggered sound effect from a Sega Tonebank type asset loaded with the `amMidiLoadBank()` call.

Note: If `midiNoteOnVelocity` is out of range it will be set to `AC_MAX_MIDI_VELOCITY` (127).

Note: This group of functions will only work with the `MidiDa` driver, they will not work with the `Audio64` driver.

amMidiNoteOff

Stops a MIDI triggered sound effect.

FORMAT

```
#include <am.h>
KTBOOL amMidiNoteOff(KTU32 midiPort,KTU32 midiChannel,KTU8 midiNoteNumber)
```

PARAMETERS

KTU32 midiPort,	The MIDI port number 0-15
KTU32 midiChannel,	The MIDI channel number 1-16
KTU8 midiNoteNumber,	The MIDI note number of the sound to be played.

RETURN VALUE

KTTRUE,	on success
or...	
KTFALSE on fail,	unable to send command to driver
	midiPort out of range (0-15)
	midiChannel out of range (1-16)
	midiNoteNumber out of range (0-127)

FUNCTION

This will stop a MIDI triggered sound effect if it is currently playing.

Note: This group of functions will only work with the MidiDa driver, they will not work with the Audio64 driver.

amMidiSetChannelVolume

Sets volume of a midi sound.

FORMAT

```
#include <am.h>
KTBOOL amMidiSetChannelVolume(KTU32 midiPort,KTU32 midiChannel,KTU32 midiVolume)
```

PARAMETERS

KTU32 midiPort,	The MIDI port number 0-15
KTU32 midiChannel,	The MIDI channel number 1-16
KTU32 midiVolume,	The MIDI volume to set 0-127

RETURN VALUE

KTTRUE,	on success
or...	
KTFALSE on fail,	unable to send command to driver
	midiPort out of range (0-15)
	midiChannel out of range (1-16)

FUNCTION

Sets CHANNEL volume of a currently playing MIDI triggered sound. This sends a MIDI Control Change 7 value ? to the driver.

Note: If midi volume is out of range it will be set to AC_MAX_MIDI_VOLUME (127)

Note: This group of functions will only work with the MidiDa driver, they will not work with the Audio64 driver.

amMidiSetChannelPan

Sets the pan of a MIDI sound.

FORMAT

```
#include <am.h>
KTBOOL amMidiSetChannelPan(KTU32 midiPort,KTU32 midiChannel,KTU32 midiPan)
```

PARAMETERS

KTU32 midiPort,	The MIDI port number 0-15
KTU32 midiChannel,	The MIDI channel number 1-16
KTU32 midiPan,	The MIDI pan to set 0-127

RETURN VALUE

KTTRUE,	on success
or...	
KTFALSE on fail,	unable to send command to driver
	midiPort out of range (0-15)
	midiChannel out of range (1-16)

FUNCTION

Sets pan (position) of a currently iterating MIDI triggered sound. This sends a MIDI Control Change 10 value ? to the driver.

Note: If midi pan is out of range it will be set to AC_MAX_MIDI_PAN (127)

Note: This group of functions will only work with the MidiDa driver, they will not work with the Audio64 driver.

amSoundSetQSoundChannels

Used to identify which channels in an output bank are Q-Sound channels.

FORMAT

```
#include <am.h>
KTBOOL amSoundSetQSoundChannels(KTU32 firstQChannel,KTU32 numberOfQChannels)
```

PARAMETERS

KTU32 firstQChannel,	The first Q-Sound channel in the output bank (.fob) asset.
KTU32 numberOfQChannels	The number of Q-Sound channels in the output bank (.fob) asset.

RETURN VALUE

KTTRUE,	On success
KTFALSE,	firstQChannel is out of range
	numberOfQChannels > AM_MAX_Q_CHANNELS

FUNCTION

Used to identify which channels in an output bank are Q-Sound channels. If this is called with numberOfQChannels==0 then the Q channel identification system is cleared.

amSoundSetEffectsBuss Sets the effects buss send and source mix for a sound object.

FORMAT

```
#include <am.h>
KTBOOL amSoundSetEffectsBuss(AM_SOUND_PTR theSound,KTU32 dspMixerChannel,KTU32 sourceMix)
```

PARAMETERS

AM_SOUND_PTR theSound,	A pointer to a properly initialized sound object
KTU32 dspMixerChannel,	The DSP mixer channel to route the dry send into.
KTU32 sourceMix,	The percentage of the dry volume to route to wet volume (1-100)

RETURN VALUE

KTTRUE,	On success
KTFALSE,	if theSound is NULL
	if dspMixerChannel > AM_MAX_DSP_MIXER_CHANNELS
	if sourceMix > AM_MAX_DSP_SOURCE_MIX

FUNCTION

This will set the effects send and source mix of the given sound. The argument `sourceMix` is how much of a DSP program is added to the dry send. If a source mix of 100% is selected and the sound has a volume of 90 then the wet level will be 90 and the dry level will be 90, if a sourceMix of 50% is selected the the wet level will be 45 and the dry level 90.

amSoundFetchSample

Fetches a sound and its parameters from a Katana format bank.

FORMAT

```
#include <am.h>
KTBOOL amSoundFetchSample(AM_BANK_PTR theBank,KTU32 soundNumber,AM_SOUND_PTR theSound)
```

PARAMETERS

AM_BANK_PTR theBank,	A pointer to a katbank containing the sound to be fetched.
KTU32 soundNumber,	The sound number to be fetched.
AM_SOUND_PTR sound,	A pointer to an AM_SOUND structure, this will contain all needed information on the sound on successful return from this function.
	On fail this structure will be filled with 0x00.

RETURN VALUE

KTTRUE,	on success
KTFALSE,	theSound is NULL
	theBank is NULL
	soundNumber is out of range
	the bank asset is not of the right type

FUNCTION

Fetches a digital sound from a given Katbank.
Calls: amBankFetchAsset ()

amSoundIsLooping

Tells if the given sound has a loop.

FORMAT

```
#include <am.h>
KTBOOL amSoundIsLooping(AM_SOUND_PTR theSound,KTBOOL *loopFlag)
```

PARAMETERS

AM_SOUND_PTR theSound,	A pointer to a properly initialized sound object
KTBOOL *loopFlag,	The loop flag is returned via this pointer

RETURN VALUE

KTTRUE	on success
KTFALSE	theSound is NULL
	loopFlag is NULL

FUNCTION

Queries weather a given sound has a loop or not.

Note: The sound structure must have been initialized with the `amBankFetchSound` function for it to contain valid data.

amSoundAllocateVoiceChannel

Allocates a hardware voice channel.

FORMAT

```
#include <am.h>
KTBOOL amSoundAllocateVoiceChannel(AM_SOUND_PTR theSound)
```

PARAMETERS

AM_SOUND_PTR theSound,	A pointer to a properly initialized sound object
------------------------	--

RETURN VALUE

KTBOOL, KTTRUE	on success
KTFALSE	can't allocate voice (all channels busy)
	theSound is NULL

FUNCTION

This allocates a hardware voice channel (an ac lib "port") for playback by the amSound subsystem. The channel is freed via the system callback mechanism when the sound has been stopped prior to the end or has finished playing.

Note: The sound structure must have been initialized with the amBankFetchSound function for it to contain valid data.

amSoundGetSampleRate

Gets the real world sample rate.

FORMAT

```
#include <am.h>
KTBOOL amSoundGetSampleRate(AM_SOUND_PTR theSound,KTU32 *realWorldSampleRate)
```

PARAMETERS

AM_SOUND_PTR theSound,	A pointer to a properly initialized sound object
KTU32 *realWorldSampleRate,	The real world sample rate is returned via this pointer.

RETURN VALUE

KTBOOL, KTTRUE	on success
KTFALSE	can't allocate voice (all channels busy)
	theSound is NULL
	realWorldSampleRate is NULL

FUNCTION

This will return the real world sample rate of the given sound. Real world rates are 44100, 22050 etc.

Note: The sound structure must have been initialized with the amBankFetchSound function for it to contain valid data.

amSoundGetVolume

Gets the current volume setting.

FORMAT

```
#include <am.h>
KTBOOL amSoundGetVolume(AM_SOUND_PTR theSound,KTU32 *volume)
```

PARAMETERS

AM_SOUND_PTR theSound,	A pointer to a properly initialized sound object
KTU32 *volume,	The volume (0-127) is returned via this pointer.

RETURN VALUE

KTBOOL, KTTRUE	on success
KTFALSE	theSound is NULL
	volume is NULL

FUNCTION

This returns the current volume of the sound in normal volume units (0-127).

Note: The sound structure must have been initialized with the `amBankFetchSound` function for it to contain valid data.

amSoundGetPan

Gets the current pan position.

FORMAT

```
#include <am.h>
KTBOOL amSoundGetPan(AM_SOUND_PTR theSound,KTU32 *aicaPan)
```

PARAMETERS

AM_SOUND_PTR theSound,	A pointer to a properly initialized sound object
KTU32 *pan,	The pan (0-127) is returned via this pointer.

RETURN VALUE

KTBOOL, KTTRUE	on success
KTFALSE	theSound is NULL pan is NULL

FUNCTION

This returns the current pan of the sound in normal pan units (0-127).

Note: The sound structure must have been initialized with the amBankFetchSound function for it to contain valid data.

amSoundGetVoiceChannel

Gets the current voice channel assignment.

FORMAT

```
#include <am.h>
KTBOOL amSoundGetVoiceChannel(AM_SOUND_PTR theSound,KTU32 *voiceChannel)
```

PARAMETERS

AM_SOUND_PTR theSound,	A pointer to a properly initialized sound object
KTU32 *voiceChannel,	The voice channel is returned via this pointer.

RETURN VALUE

KTTRUE	on success
KTFALSE	theSound is NULL voiceChannel is NULL

FUNCTION

This gets the current voice channel assignment of a sound. If the sound has not yet been initialized with a voice channel assignment the value `AM_UNINITIALIZED_VOICE_CHANNEL` will be returned.

Note: The sound structure must have been initialized with the `amBankFetchSound` function for it to contain valid data.

amSoundGetCallback

Gets the address of the user callback.

FORMAT

```
#include <am.h>
KTB00L amSoundGetCallback(AM_SOUND_PTR theSound,AM_USER_CALLBACK *theCallback)
```

PARAMETERS

AM_SOUND_PTR theSound,	A pointer to a properly initialized sound object
AM_USER_CALLBACK *theCallback,	A pointer to the callback is returned via this handle.

RETURN VALUE

KTRUE	on success
KTFALSE	theSound is NULL
	theCallback is NULL

FUNCTION

This gets the address of the user callback proc assigned to a sound, if no callback has been assigned KTNUL will be returned.

Note: The sound structure must have been initialized with the amBankFetchSound function for it to contain valid data.

amSoundSetCurrentPlaybackRate

Sets the playback rate.

FORMAT

```
#include <am.h>
KTBOOL amSoundSetCurrentPlaybackRate(AM_SOUND_PTR theSound,KTU32 sampleRate)
```

PARAMETERS

AM_SOUND_PTR theSound,	A pointer to a properly initialized sound object
------------------------	--

RETURN VALUE

KTTRUE	on success
KTFALSE	theSound is NULL sampleRate is > 1128900 can't send a command to the driver

FUNCTION

If called prior to playing a sound this will set the sounds initial playback rate. If called while the sound is playing the current playback rate will be set.

Note: The sound structure must have been initialized with the `amBankFetchSound` function for it to contain valid data.

amSoundSetVolume

Sets a sounds volume.

FORMAT

```
#include <am.h>
KTBOOL amSoundSetVolume(AM_SOUND_PTR theSound,KTU32 newVolume)
```

PARAMETERS

AM_SOUND_PTR theSound,	A pointer to a properly initialized sound object
KTU32 newVolume,	The volume to set (0-127)

RETURN VALUE

KTTRUE	on success
KTFALSE	can't send a command to the driver
	theSound is NULL

FUNCTION

If called prior to playing a sound this will set the sounds initial playback volume. If called while a sound is playing it will set the current playback volume.

Note: The sound structure must have been initialized with the `amBankFetchSound` function for it to contain valid data.

If `newVolume > AM_MAX_VOLUME (127)` `newVolume` will be set to `AM_MAX_VOLUME`
Further the aica volume range is 0-15 so the 0-127 range is quantitized into 15 steps.

amSoundSetPan

Sets a sounds pan.

FORMAT

```
#include <am.h>
KTBOOL amSoundSetPan(AM_SOUND_PTR theSound,KTU32 newPan)
```

PARAMETERS

AM_SOUND_PTR theSound,	A pointer to a properly initialized sound object
KTU32 newPan,	The pan to set (0-127)

RETURN VALUE

KTRUE	on success
KFALSE	can't send a command to the driver
	theSound is NULL

FUNCTION

If called prior to playing a sound this will set the sounds initial playback pan position.
If called while a sound is playing it will set the current playback pan position.

Note: The sound structure must have been initialized with the `amBankFetchSound` function for it to contain valid data.

If `pan > AM_MAX_PAN (127)` pan will be set to `AM_MAX_PAN`.
Because the AICA pan scale is 0-31 the normal pan numbers of 0-127 are quantitized to 31 steps.

amSoundSetCallback

Sets the user callback.

FORMAT

```
#include <am.h>
KTB00L amSoundSetCallback(AM_SOUND_PTR theSound,AM_USER_CALLBACK callback)
```

PARAMETERS

AM_SOUND_PTR theSound,	A pointer to a properly initialized sound object
KTU32 callback,	The address of a user callback function.

RETURN VALUE

KTTRUE	on success
KTFALSE	theSound is NULL
	theSound is playing

FUNCTION

Sets the user callback for a sound. This function will be called when a sound has finished playing. The callback function will need to be protyped as void foo(KTU32 voiceChannel).

Note: The sound structure must have been initialized with the amBankFetchSound function for it to contain valid data.

amSoundIsPlaying

Tells if a sound is currently playing.

FORMAT

```
#include <am.h>
KTBOOL amSoundIsPlaying(AM_SOUND_PTR theSound)
```

PARAMETERS

AM_SOUND_PTR theSound,	A pointer to a properly initialized sound object
------------------------	--

RETURN VALUE

KTTRUE	if the sound is playing.
KTFALSE	if theSound is NULL the sound is not playing.

FUNCTION

Note: The sound structure must have been initialized with the amBankFetchSound function for it to contain valid data.

amSoundStop

Stops a currently playing sound.

FORMAT

```
#include <am.h>
KTBOOL amSoundStop(AM_SOUND_PTR theSound)
```

PARAMETERS

AM_SOUND_PTR theSound,	A pointer to a properly initialized sound object
------------------------	--

RETURN VALUE

KTTRUE	if the sound was stopped
KTFALSE	if the sound was not playing. can't send a command to the driver theSound is NULL

FUNCTION

This stops a currently playing sound and releases its voice channel.

Note: The sound structure must have been initialized with the `amBankFetchSound` function for it to contain valid data.

amSoundPlay

Plays a sound.

FORMAT

```
#include <am.h>
KTBOOL amSoundPlay(AM_SOUND_PTR theSound)
```

PARAMETERS

AM_SOUND_PTR theSound,	A pointer to a properly initialized sound object
------------------------	--

RETURN VALUE

KTTRUE	if the sound was played
KTFALSE	can't send a command to the driver theSound is NULL a voice channel had not been allocated

FUNCTION

This will start a properly initialized sound object playing.

Note: The sound structure must have been initialized with the `amBankFetchSound` function for it to contain valid data.

On Failure, due to failing `amSoundPlayRaw()` or internal error, this will release the voice channel that was allocated for the sound to prevent a failed call from leaking a voice channel.

The member `theSound->voiceChannel` will be set to `AM_UNINITIALIZED_VOICE_CHANNEL` if it has been released.

amSoundPlayRaw

Plays a sound given all of the required parameters.

FORMAT

```
#include <am.h>
KTBOOL amSoundPlayRaw(      KTS32 voiceChannel,
KTU32 sizeInBytes,
KTU32 address,
KTU32 sampleRate,
AC_AUDIO_TYPE aicaAudioType,
KTU32 pitchOffsetInCents,
KTS32 aicaLoopFlag,
AM_USER_CALLBACK userCallbackProc,
KTU32 dryVolume,
KTU32 wetVolume,
KTU32 pan,
KTU32 mixerChannel,
KTBOOL effectsOnOrOff
)
```

PARAMETERS

KTS32 voiceChannel,	The DA port number to use for the sound playback.
KTU32 sizeInBytes,	The size of the sound in bytes
KTU32 address,	The address of the sound in sound memory
KTU32 sampleRate,	The real world sample rate, i.e. 44100, 22050, 16000 etc
AC_AUDIO_TYPE aicaAudioType,	The audio type, i.e. AC_16BIT, AC_8BIT, AC_ADPCM_LOOP, see ac.h
KTU32 pitchOffsetInCents,	The amount to offset the pitch (positive offset only)
KTS32 aicaLoopFlag,	The aica loop flag, either AC_LOOP_ON or AC_LOOP_OFF, see ac.h
AM_USER_CALLBACK userCallbackProc,	A pointer to a user callback proc or KTNNULL
KTU32 dryVolume,	The normal volume (0-127)
KTU32 wetVolume,	The effects volume (0-127)
KTU32 pan,	The normal pan (0-127)
KTU32 mixerChannel	The effects bank mixer channel to use
KTBOOL effectsOnOrOff	True if mixer channel supplied is valid, turns effects on and off

RETURN VALUE

void

FUNCTION

Plays a raw PCM intel byte order sound from sound memory. If a user callback proc is supplied the callback will be invoked when the sound is finished with its play.

The proc will need to have the following prototype:

```
void MyCallbackProc(KTU32 voiceChannel);
```

The voice channel (DA port #) that raised the interrupt will be passed up in the arg `voiceChannel`.

amStreamSetMix

Sets volume and pan for all tracks in a stream.

FORMAT

```
#include <am.h>
KTBOOL amStreamSetMix(AM_STREAM_PTR theStream, AM_STREAM_MIX_PTR theMix)
```

PARAMETERS

AM_STREAM_PTR theStream,	The stream object to be set
AM_STREAM_MIX_PTR theMix,	The new scene mix to be set.

RETURN VALUE

KTBOOL, KTTRUE	if the mix was successfully set.
----------------	----------------------------------

FUNCTION

Sets the volume and pan of all tracks in a stream to new values. If a value in the new scene is the same as the current value the command is not sent.

amStreamInitFile

Initializes a stream object to play a file.

FORMAT

```
#include <am.h>
KTBOOL amStreamInitFile(      AM_STREAM_PTR theStream,KTSTRING fileName)
```

PARAMETERS

AM_STREAM_PTR theStream,	the stream object to be initialized
KTSTRING fileName,	the file name of the .str file to stream

RETURN VALUE

KTBOOL, KTTRUE	if the stream object was successfully initialized
----------------	---

FUNCTION

Sets the members of the stream object necessary to the preparation for a call to amStreamOpen ()

Note: if the length of the filename is in excess of AM_STREAM_FILENAME_LEN the call will fail.

amStreamInitBuffer

Initializes a stream object to play a mono stream from a buffer.

FORMAT

```
#include <am.h>
KTBOOL amStreamInitBuffer(      AM_STREAM_PTR theStream,
volatile KTU32 *buffer,
KTU32 size,
KTU32 sampleRate,
KTU32 bitDepth)
```

PARAMETERS

AM_STREAM_PTR theStream,	The stream object to be initialized
volatile KTU32 *buffer,	A buffer in either sh4 memory or sound memory
KTU32 size,	The size of the buffer, NOTE: the buffer MUST be a multiple of the play buffer size
KTU32 sampleRate,	The real world integral sample rate of the file, 44100, 22050, or 11025
KTU32 bitDepth,	4,8 or 16

RETURN VALUE

KTBOOL, KTTRUE if the stream object was successfully initialized

FUNCTION

Sets the members of the stream object necessary to the preparation for a call to `amStreamOpen()`, this allows the playback of a chunk of headerless raw sound data. This is the way to play a stream that is to be constructed at play time. When preparing the buffer it should be sized to be an even multiple of the playbuffer size, allocate the buffer wherever you want it, sound or sh4 memory, then fill it with silence.

For ADPCM data silence is 0x80, for 8 and 16 bit data silence is 0x00.

For a 16 bit44.1k memory stream a 4096 byte play buffer is sufficient.

Note: This does not use the `streamIO` subsystem, it is also possible to play multitrack buffers using that subsystem.

amStreamInstallUserCallback

Installs a user callback for a stream.

FORMAT

```
#include <am.h>
KTBOOL amStreamInstallUserCallback(AM_STREAM_PTR theStream, AM_USER_CALLBACK userCallback)
```

PARAMETERS

AM_STREAM_PTR theStream,	The stream object.
AM_USER_CALLBACK userCallback,	The address of the callback function, see <code>am.h</code>

RETURN VALUE

KTBOOL, KTTRUE	If the callback was installed
KTFALSE	If the stream object has not been opened or is corrupt.

FUNCTION

This function will call `_amVoiceInstallUserCallback` to install a user callback into the interrupt handling system.

The callback will be issued when the stream is stopped via `amStreamStop` or the stream reaches the end.

Note: This must be called post the call to `amStreamAllocateVoiceChannels()` and the call to `amStreamOpen()`.

amStreamRewind

Rewinds an open stream to its start.

FORMAT

```
#include <am.h>
KTBOL amStreamRewind(AM_STREAM_PTR theStream)
```

PARAMETERS

AM_STREAM_PTR theStream,	The stream object to rewind.
--------------------------	------------------------------

RETURN VALUE

KTBOL, KTTRUE,	If the stream was successfully rewound.
KTFALSE,	If the seekrewind call failed or the stream is not open.

FUNCTION

Allows an open stream to be rewound to its start with out closing and reopening the file to get to its beginning.

amStreamGetMemoryRequirement

Gets memory sizes necessary to play the stream.

FORMAT

```
#include <am.h>
KTBOOL amStreamGetMemoryRequirement(          AM_STREAM_PTR theStream,
KTU32 *transferBufferSize,
KTU32 *playBufferSize
)
```

PARAMETERS

AM_STREAM_PTR theStream,	The stream object to get the requirement from.
KTU32 *transferBufferSize,	The size of the transfer buffer is returned via this pointer.
KTU32 *playBufferSize,	The size of the play buffer(s) are returned via this pointer.

RETURN VALUE

KTBOOL, KTTRUE,	If the memory requirements were returned.
KTFALSE,	If the stream was not open or is corrupt.

FUNCTION

Gets the minimum amount of memory that will need to be passed into the `amStreamSetBuffers()` call.

Note: This must be called post the calls to `amStreamOpen()` and `amSetBufferSizes()`

amStreamSetBufferSizes

Sets the sizes for the play and transfer buffers.

FORMAT

```
#include <am.h>
void amStreamSetBufferSizes(      AM_STREAM_PTR theStream,
KTU32 transferBufferSize,
KTU32 playBufferSize)
```

PARAMETERS

AM_STREAM_PTR theStream,	The stream object to be set.
KTU32 transferBufferSize,	The size of the transfer buffer.
KTU32 playBufferSize,	The size of a play buffer.

RETURN VALUE

KTBOOL, KTTRUE,	If the sizes were set.
KTFALSE,	If the stream is already open.

FUNCTION

Sets the buffer sizes in a stream that is not currently open.

Currently the basic recommendations for buffer sizes are as follows:

- 1) Play buffer size must be a multiple of 2048
- 2) Mono streams play well with a 2048 byte play buffer, stereo with a 4096 byte play buffer.
- 3) Transfer buffer size should be as follows: $\text{transferBufferSize} = (\text{playBufferSize} * 2)$

Note: This must be called **PRIOR** to the call to `amStreamOpen()`

amStreamSetBuffers

Sets buffer memory pointers in a stream.

FORMAT

```
#include <am.h>
KTBOOL amStreamSetBuffers(          AM_STREAM_PTR theStream,
volatile KTU32 *transferBuffer,
volatile KTU32 *playBuffer)
```

PARAMETERS

AM_STREAM_PTR theStream,	The stream object to be set.
volatile KTU32 *transferBuffer,	The transfer buffer memory
volatile KTU32 *playBuffer,	The play buffer memory

RETURN VALUE

KTBOOL, KTRUE,	If the buffer pointers were set.
KFALSE,	If amStreamSetBufferSizes() has not been called successfully
	If amStreamOpen() was not called successfully
	If the stream is corrupt.

FUNCTION

Sets the buffers necessary to run a stream, stereo and multi track streams require a play buffer per channel. This routine will subdivide the buffer passed in for the play buffer as necessary for the given stream.

Note: This must be called post the call to amStreamSetBufferSizes() and the call to amStreamOpen()

amStreamSetIsr

Sets the streams data transfer ISR.

FORMAT

```
#include <am.h>
KTBOOL amStreamSetIsr(AM_STREAM_PTR theStream, AM_STREAM_ISR theIsr)
```

PARAMETERS

AM_STREAM_PTR theStream,	The stream object to be set.
AM_STREAM_ISR theIsr,	A pointer to a data transfer ISR

Library ISR identifiers:

void _amStreamIsr0	(KTU32 streamPtr)
void _amStreamIsr1	(KTU32 streamPtr)
void _amStreamIsr2	(KTU32 streamPtr)
void _amStreamIsr3	(KTU32 streamPtr)
void _amStreamIsr4	(KTU32 streamPtr)

RETURN VALUE

KTBOOL, KTTRUE,	If the ISR was installed successfully.
KTFALSE,	If the stream was not open

FUNCTION

Sets the streams ISR that pumps data from the transfer buffer to the `play buffer(s)`. To determine if a stream is mono or stereo, post `amStreamOpen()`, use the calls:

`amStreamIsMono()` or `amStreamIsStereo()`.

Note: This must be called post the calls to `amStreamOpen()` and `amStreamAllocateVoiceChannels()`

amStreamAllocateVoiceChannels

Allocates voice channels.

FORMAT

```
#include <am.h>
KTBOOL amStreamAllocateVoiceChannels(AM_STREAM_PTR theStream)
```

PARAMETERS

AM_STREAM_PTR theStream,	The stream object to get voices.
--------------------------	----------------------------------

RETURN VALUE

KTBOOL, KTTRUE,	If the voice(s) were successfully allocated.
KTFALSE,	If the allocation failed.
	If the stream was not open.

FUNCTION

This calls `amVoiceAllocate()` to allocate voices for the given stream, playback requires one voice per channel of program. A mono stream is one channel, a single track of stereo is two channels.

Note: This must be called post to call to `amStreamOpen()`.

amStreamPrimeBuffers

Primes the play buffer.

FORMAT

```
#include <am.h>
KTBOOL amStreamPrimeBuffers(AM_STREAM_PTR theStream)
```

PARAMETERS

AM_STREAM_PTR theStream, the stream object to be primed.

RETURN VALUE

KTBOOL, KTTRUE,	If the stream was successfully primed.
KTFALSE,	If the stream was not open.
	If the read failed.
	If the stream is corrupt.

FUNCTION

Moves the first load of data into the transfer and play buffer(s) for the given stream.

Note: This must be called post the call to amStreamOpen ().

amStreamGetTrackLengthInFrames

Gets the length of a stream in frames.

FORMAT

```
#include <am.h>

KTBOOL      amStreamGetTrackLengthInFrames(AM_STREAM_PTR theStream,KTU32 trackNumber,KTU32
*trackLengthInFrames)
```

PARAMETERS

AM_STREAM_PTR theStream,	the stream object to get the length from.
KTU32 trackNumber,	the number of the track.
KTU32 *trackLengthInFrames,	the length of a stream in frames is returned via this pointer.

RETURN VALUE

KTBOOL

FUNCTION

Gets the length of a stream in frames.

Note: This must be called post the call to `amStreamOpen()` or it will return `0` `amUtilGetNibblesPerFrame()`

amStreamGetNibblesPerFrame

Gets the number of nibbles in a frame.

FORMAT

```
#include <am.h>
KTBOOL  amStreamGetNibblesPerFrame(AM_STREAM_PTR theStream,KTU32 *nibblesPerFrame)
```

PARAMETERS

AM_STREAM_PTR theStream,	The stream object
KTU32 *nibblesPerFrame,	The number of nibbles in a frame of data.

RETURN VALUE

KTBOOL

FUNCTION

Gets the number of nibbles in a frame for the sample format of the given stream.

Note: This must be called post the call to amStreamOpen () or it will return 0

amStreamGetSampleRate

Gets the real world sample rate of a stream.

FORMAT

```
#include <am.h>
KTBOOL  amStreamGetSampleRate(AM_STREAM_PTR theStream,KTU32 *sampleRate)
```

PARAMETERS

AM_STREAM_PTR theStream,	The stream object
KTU32 *sampleRate,	The real world sample rate of a stream.

RETURN VALUE

KTBOOL

FUNCTION

Returns the real world (44100, 22050, 11025, ...) sample rate of a stream. In the stream object the sample rate is AICA encoded and bears no resemblance to the real world rate. This allows access to a meaningful value for sample rate.

Note: This must be called post the call to amStreamOpen () or it will return 0

amStreamGetMsPerIrq

Gets the number of milliseconds per callback.

FORMAT

```
#include <am.h>
KTBOOL  amStreamGetMsPerIrq(AM_STREAM_PTR theStream,KTU32 *millisecondsPerIrq)
```

PARAMETERS

AM_STREAM_PTR theStream,	The stream object
KTU32 *millisecondsPerIrq,	The number of milliseconds per callback.

RETURN VALUE

vKTBOOL

FUNCTION

Gets the number of milliseconds per callback. There are two callbacks per iteration of the play buffer which is playing at sample rate in frames, this resolves all of the variables to produce the number of milliseconds per callback.

Note: This must be called post the call to amStreamOpen () or it will return 0

amStreamSetVolume

Sets the volume on a stream.

FORMAT

```
#include <am.h>
KTBOOL amStreamSetVolume(AM_STREAM_PTR theStream,KTU8 newVolume)
```

PARAMETERS

AM_STREAM_PTR theStream,	The stream object of the stream to have its volume set
KTU8 newVolume,	The new volume to set (0-127)

RETURN VALUE

KTBOOL, KTTRUE	if the pan was successfully set
KTFALSE	if the stream is not playing.

FUNCTION

Sets the volume of the currently playing mono or stereo stream.

amStreamSetPan

Sets the pan on a mono stream.

FORMAT

```
#include <am.h>
KTBOOL amStreamSetPan(AM_STREAM_PTR theStream,KTU8 newPan)
```

PARAMETERS

AM_STREAM_PTR theStream,	The stream object of the stream to be panned
KTU8 newPan,	The new setting for the pan (0-127)

RETURN VALUE

KTBOOL,	KTTRUE if the pan was successfully set
KTFALSE	if the stream is not playing or the stream is stereo

FUNCTION

Sets the pan of a currently playing MONO stream, if a stereo stream is submitted the call will return KTFALSE as stereo streams can not be panned.

amStreamStop

Stops a currently playing stream.

FORMAT

```
#include <am.h>
KTBOOL amStreamStop(AM_STREAM_PTR theStream)
```

PARAMETERS

AM_STREAM_PTR theStream,	the stream object of the stream to be stopped
--------------------------	---

RETURN VALUE

KTBOOL,	KTTRUE if the stream was successfully stopped
---------	---

FUNCTION

Used to stop a currently playing stream, this routine is called by `amStreamServer()` at the end of a stream. This closes and frees the port, removes and releases the callback and releases the port and the buffers from the stream.

amStreamPlaying

Monitors if a stream is currently playing.

FORMAT

```
#include <am.h>
KTBOOL _amStreamPlaying(AM_STREAM_PTR theStream)
```

PARAMETERS

AM_STREAM_PTR theStream,	the stream object to be monitored for play activity
--------------------------	---

RETURN VALUE

KTBOOL, KTTRUE	if the stream is currently playing
----------------	------------------------------------

FUNCTION

Used to monitor the play status of a stream.

amStreamGetVolume

Gets the streams current volume

FORMAT

```
#include <am.h>
KTU32 amStreamGetVolume(AM_STREAM_PTR theStream)
```

PARAMETERS

AM_STREAM_PTR theStream,
KTU32 *volume,

The stream object to get the volume from
The volume is returned via this pointer.

RETURN VALUE

KTBOOL,
KTFALSE

KTTRUE on success
on NULL parameter or track number out of range.

FUNCTION

Gets the current volume of a stream.

amStreamGetPan

Gets the streams current pan

FORMAT

```
#include <am.h>
KTU32 amStreamGetVolume(AM_STREAM_PTR theStream)
```

PARAMETERS

AM_STREAM_PTR theStream,
KTU32 *pan,

The stream object to get the volume from
The pan is returned via this pointer.

RETURN VALUE

KTBOOL,
KTFALSE

KTTRUE on success
on NULL parameter or track number out of range.

FUNCTION

Gets the current pan of a stream.

amStreamGetIsrCount

Gets the Interrupt Service Routine count.

FORMAT

```
#include <am.h>
KTU32 amStreamGetIsrCount(AM_STREAM_PTR theStream)
```

PARAMETERS

AM_STREAM_PTR theStream, the stream object to get the ISR count from

RETURN VALUE

KTU32, The number of times the ISR has been invoked for this stream.

FUNCTION

Gets the number of times that the ISR has been invoked in this play cycle.

amStreamClose

Closes a stream object.

FORMAT

```
#include <am.h>
KTBOOL amStreamClose(AM_STREAM_PTR theStream)
```

PARAMETERS

AM_STREAM_PTR theStream,	the stream object to be closed
--------------------------	--------------------------------

RETURN VALUE

KTBOOL, KTTRUE	if the stream object was (or is) closed
----------------	---

FUNCTION

Used to close a stream file. This closes the file but does not release the resources.

amStreamStart

Starts a stream object playing.

FORMAT

```
#include <am.h>
KTBOOL amStreamStart(AM_STREAM_PTR theStream)
```

PARAMETERS

AM_STREAM_PTR theStream,	the stream object to be started
--------------------------	---------------------------------

RETURN VALUE

KTBOOL, KTTRUE	if the stream was successfully started
----------------	--

FUNCTION

Starts the given stream object playing, acquires callback procs and acquires and configures the ports based on the type of stream contained in the `.str` file.

amStreamIsStereo

Tells if a stream is stereo.

FORMAT

```
#include <am.h>
KTB00L amStreamIsStereo(AM_STREAM_PTR theStream)
```

PARAMETERS

AM_STREAM_PTR theStream, the stream object to be queried.

RETURN VALUE

KTB00L, KTTRUE if the stream is stereo.

FUNCTION

If the given stream is stereo this will return KTTRUE.

amStreamIsMono

Tells if a stream is mono.

FORMAT

```
#include <am.h>
KTBOOL amStreamIsMono(AM_STREAM_PTR theStream)
```

PARAMETERS

AM_STREAM_PTR theStream, the stream object to be queried.

RETURN VALUE

KTBOOL, KTTRUE if the stream is mono

FUNCTION

If the given stream is mono this will return KTTRUE.

amStreamServer

Serves data to a currently playing stream.

FORMAT

```
#include <am.h>
KTB00L amStreamServer(AM_STREAM_PTR theStream)
```

PARAMETERS

AM_STREAM_PTR theStream, the stream object to be served

RETURN VALUE

KTB00L, KTTRUE if the stream is currently playing, KTFALSE if it is finished

FUNCTION

Fills the transfer buffer of a given stream when its contents have been completely transferred by the ISR.

amStreamOpen

Opens a stream object.

FORMAT

```
#include <am.h>
KTBOOL amStreamOpen(AM_STREAM_PTR theStream)
```

PARAMETERS

AM_STREAM_PTR theStream,	the stream object to be opened
--------------------------	--------------------------------

RETURN VALUE

KTBOOL, KTTRUE	if the stream was successfully started
----------------	--

FUNCTION

Opens the stream file named in the `InitStream()` call, acquires buffers and “primes” the play buffer(s) with data from the transfer buffer.

amStreamSetTransferMethod Selects DMA or memcpy as the data transfer method.

FORMAT

```
#include <am.h>
KTBOOL amStreamSetTransferMethod(          AM_STREAM_PTR theStream,
AM_STREAM_TRANSFER_METHOD transferMethod,
KTU32 dmaChannel,
KTU32 dmaFrameSize)
```

PARAMETERS

AM_STREAM_PTR theStream,	the stream object.
AM_STREAM_TRANSFER_METHOD transferMethod,	either AM_STREAM_DMA or AM_STREAM_NON_DMA
KTU32 dmaChannel,	AM_DMA_CHANNEL
KTU32 dmaFrameSize,	4,8 or 32 bytes per frame

RETURN VALUE

KTTRUE	on success
KTFALSE	if theStream is open
	if theStream is NULL
	if transferMethod is not AM_STREAM_DMA or AM_STREAM_NON_DMA
	if dmaChannel is not AM_DMA_CHANNEL
	if dmaFrameSize is not 4,8 or 32

FUNCTION

Causes the stream to be transfered via DMA rather then using the foreground memcpy process.
dmaFrameSize controls how many bytes are transferred in a single dma transfer.

Note: This must be called prior to StreamOpen ()

amStreamIoInstallAlternateIoManager

Installs a custom Io proc.

FORMAT

```
#include <am.h>
void amStreamIoInstallAlternateIoManager(AM_STREAM_IO_PROC ioProc)
```

PARAMETERS

AM_STREAM_IO_PROC ioProc,	A pointer to a custom Io proc, see the example in <code>MyFile.c</code>
---------------------------	---

RETURN VALUE

void

FUNCTION

Installs a custom Io proc into the Io shell, this allows all file system calls to be intercepted by the applications file system.

The prototype for the IO proc is as follows:

```
KTBOOL MyCustomIoProc(KTSTRING fileName,
KTU32 * sd,
KTU8 * buffer,
KTU32 * size,
AM_FILE_OPERATION_MODE mode
)
```

An example of this redirection is available in `MyFile.c` as well as a boilerplate copy of the IO proc for modification.

amUtilGetAicaVolume

Converts midi volume units to AICA units

FORMAT

```
#include <am.h>
KTBOOL amUtilGetAicaVolume(KTU32 midiVolume,KTU32 *aicaVolume)
```

PARAMETERS

KTU32 midiVolume,	the midi volume to be converted (0-127)
KTU32 *aicaVolume,	the AICA volume (0-15) is returned via this pointer

RETURN VALUE

KTTRUE	on success
KTFALSE	if aicaVolume is NULL

FUNCTION

Converts from midi volume units (0-127) to AICA (0-15) volume units.

Note: If midiVolume is > 127 it will be set to 127 and a debug warning issued.

amUtilAlignNumber

Performs numerical boundry alignment.

FORMAT

```
#include <am.h>
KTBOOL amUtilAlignNumber(KTU32 theNumber,KTU32 theAlignment,KTU32 *theResult)
```

PARAMETERS

KTU32 theNumber,	the number to be aligned
KTU32 theAlignment,	the desired boundry
KTU32 *theResult,	the aligned number is returned via this pointer

RETURN VALUE

KTTRUE	on success
KTFALSE	if alignment is 0
	if theResult is NULL

FUNCTION

Rounds a number up to the next multiple of alignment. If the number is evenly divisible by alignment it will be returned untouched.

amUtilGetLengthInFrames

Gets the length of a stream in frames.

FORMAT

```
#include <am.h>
KTU32 amUtilGetLengthInFrames(AC_AUDIO_TYPE type,KTU32 channels,KTU32 size,KTU32
*lengthInFrames)
```

PARAMETERS

AC_AUDIO_TYPE type,	the AICA type of the sound, see ac.h
KTU32 channels,	the number of channels in the sound, 1=mono 2=stereo
KTU32 size,	the size of the sound in bytes
KTU32 *lengthInFrames,	the length is returned via this pointer.

RETURN VALUE

KTU32,	The length of a stream in frames.
--------	-----------------------------------

FUNCTION

Gets the length of a stream in frames.

Note: This must be called post the call to amUtilOpen() or it will return 0

amUtilGetNibblesPerFrame

Gets the number of nibbles in a frame.

FORMAT

```
#include <am.h>
KTBOOL  amUtilGetNibblesPerFrame(AC_AUDIO_TYPE type,KTU32 *nibblesPerFrame)
```

PARAMETERS

AC_AUDIO_TYPE type,	the AICA type of the sound, see ac.h
KTU32 *nibblesPerFrame,	the number of nibbles in a frame is returned via this pointer

RETURN VALUE

KTBOOL

FUNCTION

Gets the number of nibbles in a frame for the sample format of the given stream.

amUtilGetSampleRate

Gets the real world sample rate of a stream.

FORMAT

```
#include <am.h>
KTBOOL  amUtilGetSampleRate(KTU32 aicaSampleRate,KTU32 *sampleRate)
```

PARAMETERS

KTU32 aicaSampleRate,	the AICA sample rate as defined in <code>ac.h</code>
KTU32 *sampleRate,	The real world sample rate of a stream is returned via this pointer.

RETURN VALUE

KTBOOL,	fails if sampleRate is NULL or aicaSampleRate is not a correct value.
---------	---

FUNCTION

Returns the real world (44100, 22050, 11025, ...) sample rate of a stream. In the stream object the sample rate is AICA encoded and bears no resemblance to the real world rate. This allows access to a meaningful value for sample rate.

Note: This must be called post the call to `amUtilOpen()` or it will return 0

amUtilGetLengthInMs

Gets the length of a stream in milliseconds.

FORMAT

```
#include <am.h>

KTBOOL      amUtilGetLengthInMs(AC_AUDIO_TYPE type,KTU32 channels,KTU32 size,KTU32
aicaSampleRate,KTU32 *lengthInMs)
```

PARAMETERS

AC_AUDIO_TYPE type,	the AICA type of the sound, see ac.h
KTU32 channels,	the number of channels in the sound, 1=mono 2=stereo
KTU32 size,	the size of the sound in bytes
KTU32 aicaSampleRate,	the AICA sample rate of the sound, see ac.h
KTU32 *lengthInMs,	the length of a stream in milliseconds.

RETURN VALUE

KTBOOL

FUNCTION

Gets the length of a stream in milliseconds.

Note: This must be called post the call to amUtilOpen() or it will return 0

amUtilGetMsPerIrq

Gets the number of milliseconds per callback.

FORMAT

```
#include <am.h>
KTBOOL  amUtilGetMsPerIrq(AC_AUDIO_TYPE type,KTU32 aicaSampleRate,KTU32
playBufferSizeInBytes,KTU32 *msPerIrq)
```

PARAMETERS

AC_AUDIO_TYPE type,	the AICA type of the sound, see ac.h
KTU32 aicaSampleRate,	the AICA sample rate of the sound, see ac.h
KTU32 playBufferSizeInBytes,	the size of the playback buffer in bytes
KTU32 *msPerIrq,	the number of milliseconds per callback.

RETURN VALUE

FUNCTION

Gets the number of milliseconds per callback. There are two callbacks per iteration of the play buffer which is playing at sample rate in frames, this resolves all of the variables to produce the number of milliseconds per callback.

Note: This must be called post the call to amUtilOpen() or it will return 0

amUtilGetAicaSampleType

Extrapolates sample bit depth to AICA sample type.

FORMAT

```
#include <am.h>
KTBOOL amUtilGetAicaSampleType(KTU32 bitDepth, AC_AUDIO_TYPE_PTR aicaSampleType)
```

PARAMETERS

KTU32 bitDepth,	The bit depth of the sample.
AC_AUDIO_TYPE_PTR aicaSampleType,	The returned AICA sample type.

RETURN VALUE

KTTRUE on success

Or...

KTFALSE on fail

FUNCTION

Extrapolates sample bit depth to AICA sample type. See the AC_AUDIO_TYPE enum in `ac.h` for the types returned by this function.

Note: This will always identify 4 bit data as the type AC_ADPCM_LOOP

amUtilGetAicaSampleRate Makes a real world sample rate into an AICA sample rate.

FORMAT

```
#include <am.h>
KTBOOL amUtilGetAicaSampleRate(KTU32 realWorldSampleRate, KTS32 *aicaSampleRate)
```

PARAMETERS

KTU32 realWorldSampleRate,	The real world sample rate, i.e. 44100, 22050, 11025, 5012
KTS32 *aicaSampleRate,	The returned AICA sample rate.

RETURN VALUE

KTTRUE on success
Or...
KTFALSE on fail

FUNCTION

Extrapolates from real world sample rates to AICA sample rates, the only allowed AICA rates are 44100, 22050, 11025 and 5012. Using any other rate will cause this function to fail.

amUtilGetMiddleOfBufferInFrames

Calculates the middle of the buffer in frames.

FORMAT

```
#include <am.h>
KTBOOL amUtilGetMiddleOfBufferInFrames(KTU32 bitDepthOfSample,KTU32
sizeofBufferInBytes,KTU32 * middleInFrames)
```

PARAMETERS

KTU32 bitDepthOfSample,	the bit depth of the sample data
KTU32 sizeofBufferInBytes,	the size of the buffer in bytes
KTU32 * middleInFrames,	the offset of the middle in frames is returned via this value

RETURN VALUE

KTBOOL, KTTRUE if the calculation was successful, KTFALSE if the bit depth is unsupported

FUNCTION

Calculates the middle of the buffer in frames.

amUtilGetEndOfBufferInFrames

Calculates the end of the buffer in frames.

FORMAT

```
#include <am.h>
KTBOOL amUtilGetEndOfBufferInFrames(KTU32 bitDepthOfSample,KTU32
sizeofBufferInBytes,KTU32 * middleInFrames)
```

PARAMETERS

KTU32 bitDepthOfSample,	the bit depth of the sample data
KTU32 sizeofBufferInBytes,	the size of the buffer in bytes
KTU32 * endInFrames,	the offset of the end in frames is returned via this value

RETURN VALUE

KTBOOL, KTTRUE if the calculation was successful, KTFALSE if the bit depth is unsupported

FUNCTION

Calculates the end of the buffer in frames.

amVoiceInit

Initializes the voice pool.

FORMAT

```
#include <am.h>
void amVoiceInit(void)
```

PARAMETERS

void

RETURN VALUE

void

FUNCTION

Initializes the voice pool and voice (port) allocation functionality.

amVoiceAllocate

Allocates a voice channel.

FORMAT

```
#include <am.h>
KTBOOL amVoiceAllocate(KTU32 * voiceChannel, AM_VOICE_TYPE voiceType, void * owner)
```

PARAMETERS

KTU32 * voiceChannel,	The voice channel allocated is returned via this pointer.
AM_VOICE_TYPE voiceType,	The type, AM_ONESHOT_VOICE or AM_STREAM_VOICE, see am.h
void * owner,	The address of the AM_SOUND or AM_STREAM object that holds the channel,

RETURN VALUE

KTTRUE	If a channel was available.
or...	
KTFALSE,	If no channels are available.

FUNCTION

Allocates voice channels (DA port #'s) needed for playing sounds with the ac layer. If the type is AM_ONESHOT_VOICE the owner arg is the address of the AM_SOUND structure that holds the sound to be played on that channel.

If the type is AM_STREAM_VOICE the owner arg is the address of the AM_STREAM structure. If the type is AM_MIDI_VOICE the midi port number that must be used is the following:

```
(voiceChannel - AM_FIRST_MIDI_VOICE)
```

The midi ports are 0-15 but the driver reports them as event numbers 16-31 in the interrupt message array.

